Recherche approximative de plus proches voisins avec contrôle probabiliste de la précision ; application à la recherche d'images par le contenu

Sid-Ahmed Berrani

▶ **To cite this version:**

Sid-Ahmed Berrani. Recherche approximative de plus proches voisins avec contrôle probabiliste de la précision ; application à la recherche d'images par le contenu. Interface homme-machine [cs.HC]. Université Rennes 1, 2004. Français. NNT : . tel-00532854

HAL Id: tel-00532854
https://theses.hal.science/tel-00532854

Submitted on 4 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : 2953

THÈSE

présentée

devant l'université de Rennes 1

pour obtenir le grade de :

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1
Mention INFORMATIQUE

par

Sid-Ahmed Berrani

Équipes d'accueil : Thomson Multimédia R&D France / Projet TEXMEX (IRISA)

École doctorale : Mathématiques, Informatique, Signal, Électronique et Télécommunications

Composante Universitaire : IFSIC, université de Rennes 1

Titre de la thèse :

Recherche approximative de plus proches voisins avec contrôle probabiliste de la précision ; application à la recherche d'images par le contenu.

Soutenue le 6 février 2004, devant la commission d'examen

Composition du jury

M.	Patrice	QUINTON	Président
MM.	Amr	EL ABBADI	Rapporteurs
	Roger	MOHR	
MM.	Patrick	VALDURIEZ	Examinateurs
	François	LE CLERC	
	Patrick	GROS	
	Laurent	AMSALEG	

Remerciements

Ce travail a été réalisé dans le cadre d'une convention CIFRE entre Thomson Multimédia R&D France et l'IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires, Rennes), au sein de l'équipe TEXMEX.

Je tiens à remercier Patrice Quinton, professeur à l'université de Rennes 1, pour avoir accepté de participer à mon jury de thèse et à le présider. Je suis également reconnaissant à Amr El Abbadi, professeur à l'université de Californie, Santa Barbara aux USA, ainsi qu'à Roger Mohr, professeur et directeur de l'ENSIMAG à Grenoble, d'avoir apporté leur point de vue sur ce travail en acceptant d'en être les rapporteurs. Je remercie aussi Patrick Valduriez, directeur de recherche INRIA, d'avoir accepté d'examiner ce travail de thèse. Je lui suis reconnaissant de s'être penché sur ce travail avec enthousiasme et rigueur.

Je remercie très chaleureusement François Le Clerc, ingénieur R&D à Thomson, d'avoir suivi ce travail avec intérêt, pour ses conseils et ses remarques constructives, sa disponibilité et sa relecture minutieuse du manuscrit.

Je souhaite exprimer ma gratitude à Patrick Gros, chargé de recherche CNRS et responsable de l'équipe TEXMEX, et Laurent Amsaleg, chargé de recherche CNRS, pour m'avoir proposé un sujet de recherche original et fort intéressant, pour leur disponibilité, pour m'avoir encadré et guidé avec clairvoyance et permis de mener ce travail dans des conditions les meilleures qui soient. Je tiens à les remercier aussi pour leur indéfectible bonne humeur, leurs très grandes qualités humaines et pour toutes les discussions souvent très animées que l'on a pu avoir. Je les remercie aussi de m'avoir ouvert le chemin de nombreux thèmes de recherche et permis des rencontres enrichissantes et variées. Merci.

Je remercie aussi Gérard Briand, responsable de l'ancien laboratoire AVP de Thomson d'avoir initié avec Patrick Gros cette thèse, et Nour-Eddine Tazine, responsable du laboratoire CSA de Thomson, d'avoir veillé à la continuation de cette collaboration. Je le remercie également pour la confiance qu'il m'a accordée et pour l'intérêt qu'il a porté à ce travail.

Je souhaite aussi remercier Christine Guillemot et Patrick Bouthemy de m'avoir accueilli dans leurs équipes TEMICS et VISTA en début de thèse. Je salue également tous les membres de ces équipes que j'ai pu côtoyer.

Merci aussi aux relecteurs et correcteurs de cette thèse : Gwen, Vincent, François R., Fabienne, François T., Thomas et Yahia.

Merci à toutes les personnes qui m'ont entouré et aidé (directement ou indirectement) ces trois années durant.

Enfin, un grand merci à mes parents, à ma très chère sœur et à mes frères...

À mes parents
À mes frères et sœurs
À Meriem

Table des matières

Introduction générale **5**

I État de l'art des méthodes de recherche de plus proches voisins **11**

1 Indexation multidimensionnelle **13**
1.1 Introduction . . . 13
1.2 Définitions de base . . . 14
1.3 Principe des index multidimensionnels . . . 15
1.3.1 Formation des cellules . . . 15
1.3.2 Algorithmes de recherche . . . 16
1.4 Principales techniques basées sur le partitionnement des données . . . 18
1.4.1 Les R-Tree, R^+-Tree et R^*-Tree . . . 18
1.4.2 Le X-Tree . . . 22
1.4.3 Le SS-Tree . . . 23
1.4.4 Le SR-Tree . . . 23
1.4.5 Le TV-Tree . . . 24
1.4.6 Le M-Tree . . . 24
1.5 Principales techniques basées sur le partitionnement de l'espace . . . 27
1.5.1 Le K-D-B-Tree . . . 27
1.5.2 Le LSD-Tree et le LSD^h-Tree . . . 28
1.6 Synthèse . . . 31

2 Indexation et malédiction de la dimension **33**
2.1 Introduction . . . 33
2.2 Origine . . . 33
2.3 Quelques propriétés surprenantes des espaces de grande dimension . . . 35
2.4 La dimension intrinsèque des données . . . 40
2.5 Techniques de réduction de la dimension . . . 44
2.5.1 Techniques linéaires . . . 46
2.5.1.1 Analyse en composantes principales . . . 46
2.5.1.2 Analyse en composantes indépendantes . . . 48
2.5.1.3 Limites des techniques linéaires . . . 51
2.5.2 Techniques non linéaires . . . 54
2.5.2.1 *Multidimensional Scaling* . . . 55
2.5.2.2 Analyse en composantes curvilinéaires . . . 57

2.5.2.3 *FastMap* . . . 58
2.6 Synthèse et discussions . . . 61

3 Nouvelles approches pour la recherche de plus proches voisins 63
3.1 Introduction . . . 63
3.2 Le Pyramid-Tree . . . 63
3.3 Amélioration de la recherche séquentielle . . . 65
3.3.1 VA-File . . . 66
3.3.2 LPC-File . . . 68
3.4 La recherche approximative . . . 70
3.4.1 Approximations sans contrôle de précision . . . 71
3.4.1.1 Approche par réduction de la dimension . . . 71
3.4.1.2 Arrêts prématurés . . . 71
3.4.1.3 Hachage multidimensionnel . . . 72
3.4.2 Approximations avec contrôle de précision . . . 72
3.4.2.1 Approximations géométriques . . . 73
3.4.2.2 Approximations probabilistes . . . 74
3.5 Synthèse . . . 77

II Méthode proposée pour la recherche de plus proches voisins 79

Introduction 81

4 Regroupement des données hors-ligne 83
4.1 *Clustering :* principe et applications . . . 83
4.2 Application à la l'indexation . . . 84
4.3 État de l'art des méthodes de *clustering* . . . 85
4.3.1 Classification des techniques existantes . . . 85
4.3.2 Approches basées sur le partitionnement des données . . . 85
4.3.3 Approches hiérarchiques . . . 88
4.3.3.1 Méthodes traditionnelles . . . 88
4.3.3.2 BIRCH . . . 90
4.3.4 Approches basées sur l'estimation de la densité . . . 92
4.3.4.1 Approches paramétriques . . . 93
4.3.4.2 Approches non paramétriques . . . 95
4.3.5 Approches basées sur le partitionnement de l'espace . . . 99
4.3.6 Synthèse . . . 100
4.4 Algorithme proposé . . . 101
4.4.1 Retour sur la première phase de BIRCH . . . 101
4.4.2 Améliorations et modifications apportées . . . 102
4.4.3 Détails de l'algorithme proposé . . . 103
4.5 Évaluation de la méthode . . . 105
4.6 Synthèse . . . 108

5 Recherche approximative de plus proches voisins - Calculs des rayons approximatifs 109
5.1 Introduction . . . 109
5.2 Présentation de la méthode de recherche approximative . . . 110
5.2.1 La recherche en ligne à l'aide des règles de filtrage . . . 111
5.3 Calcul des rayons approximatifs et contrôle de la précision . . . 114
5.3.1 De l'imprécision globale à l'imprécision locale . . . 114
5.3.2 Calcul des rayons approximatifs . . . 115
5.3.3 Retour sur l'hypothèse d'isotropie . . . 122
5.3.4 Estimation de $P(H_i)$ . . . 124
5.4 Extension de la méthode de base : deux autres modes de recherche possibles 127
5.5 Synthèse . . . 130

6 Résultats expérimentaux 131
6.1 Introduction . . . 131
6.2 Contexte des expérimentations . . . 132
6.3 Nature des données . . . 132
6.4 Expérience 1 : étude du paramètre $P(H)$ . . . 134
6.5 Expérience 2 : impact du nombre de *clusters* . . . 135
6.6 Expérience 3 : précision de la recherche . . . 141
6.7 Expérience 4 : influence de la dimension des données . . . 142
6.8 Expérience 5 : influence du nombre de vecteurs . . . 146
6.9 Expérience 6 : sensibilité à l'algorithme de *clustering* . . . 148
6.10 Expérience 7 : étude comparative avec CLINDEX . . . 150
6.11 Expérience 8 : influence du nombre k de plus proches voisins . . . 152
6.12 Synthèse . . . 155

III Application à la recherche d'images par le contenu 157

Introduction 159

7 Recherche d'images par le contenu pour la détection des copies 161
7.1 Introduction . . . 161
7.2 Reconnaissance d'images par le contenu . . . 163
7.2.1 La famille des descripteurs locaux différentiels . . . 163
7.2.1.1 Extraction de points d'intérêt . . . 164
7.2.1.2 Calcul des descripteurs locaux . . . 164
7.2.2 Schéma de recherche basé sur les descripteurs locaux . . . 166
7.2.3 Recherche de plus proches voisins . . . 168
7.3 Recherche approximative et descripteurs locaux . . . 169
7.4 Optimisation en utilisant la multiplicité des requêtes . . . 172
7.5 Synthèse . . . 173

8 Évaluation des performances **175**
8.1 Introduction 175
8.2 Méthodologie d'évaluation 176
8.3 Contexte des expérimentations 180
8.4 Plan des expérimentations 180
8.5 Résultats expérimentaux 181
8.5.1 Expérience 1 : la multiplicité des requêtes 181
8.5.2 Expérience 2 : évaluation de la robustesse en utilisant la base MOVI 183
8.5.3 Expérience 3 : évaluation de la robustesse face aux attaques générées par StirMark 185
8.5.4 Expérience 4 : évaluation de la robustesse face au recadrage 189
8.6 Synthèse 191

Conclusion générale **193**

Bibliographie **198**

Références de l'auteur **210**

Introduction générale

Jusqu'à présent, l'exploitation d'un fond photographique faisait largement appel à des techniques de descriptions manuelles et textuelles : un documentaliste saisissait à la main un ensemble de mots clés ou une courte notice décrivant le contenu des images avant archivage. Ceci étant fait, il était alors possible d'interroger la collection d'images à partir d'un ensemble de mots décrivant l'image recherchée. Ceci permettait de retrouver, par exemple, les images relatives à un thème particulier ou bien celles où apparaît telle personnalité.

Ce procédé de description porte beaucoup de sémantique de par la description textuelle des images par un humain. Il est cependant subjectif à cause du très grand nombre d'interprétations pouvant être prêtées au contenu d'une même image et aux différents niveaux de descriptions possibles. Deux documentalistes peuvent ainsi donner deux descriptions différentes d'une même image. De plus, ce mode de description étant basé uniquement sur le texte, il est intrinsèquement limité au contenu exprimable à l'aide de mots. Ce schéma est également laborieux et onéreux à mettre en œuvre lorsque l'on doit décrire de très grandes collections d'images. Ces limites rendent, par conséquent, son utilisation de plus en plus difficile dans une application réelle, gérant une très grande collection d'images.

Il existe d'autres techniques de description qui sont, elles, automatiques. Se passant d'intervention humaine, elles peuvent servir à décrire de très larges collections d'images. Ces techniques sont basées sur l'extraction de propriétés visuelles des images (couleur, texture, forme...) et, ainsi, tendent à en traduire le contenu. Décrire automatiquement le contenu visuel des images permet alors l'émergence d'un nouveau mode d'interrogation : l'interrogation par l'exemple. Dans ce cas, le principe est de retrouver les images qui ressemblent fortement à l'image donnée en exemple, c'est-à-dire celles qui ont une forte similarité visuelle avec l'exemple. Retrouver ainsi des images où apparaît telle personnalité ne requiert plus d'avoir par avance saisit son nom, mais demande alors de fournir un exemplaire du visage recherché pour initier la recherche.

Clairement, la nature des propriétés visuelles à choisir pour la description automatique dépend du type de ressemblance que l'on désire : trouver des visages similaires fait appel à des propriétés différentes de celles mises en jeu si l'on cherche à retrouver les images ressemblantes à des images de paysages par exemple.

Mettre en œuvre ces techniques de description automatique basée sur la ressemblance visuelle entre images dans un contexte de très grandes collections d'images fait appel à des techniques développées dans deux domaines différents : l'analyse d'images et les bases de données. Cette mise en œuvre s'effectue dans le cadre d'un système de recherche d'images par le contenu (SRIC) offrant la possibilité d'archiver, de décrire automatiquement une collection d'images hors ligne, et d'exécuter, en ligne, des interrogations par l'exemple.

Les méthodes d'analyse d'images permettent d'extraire à partir des images, des descripteurs permettant de décrire certaines de leurs propriétés visuelles. L'enjeu principal de ces méthodes est de mettre au point des schémas de description permettant de traduire, le plus fidèlement possible, la notion de similarité visuelle requise par l'application en une mesure de similarité entre descripteurs. Généralement, les descripteurs se présentent sous la forme de vecteurs numériques de grande dimension auxquels est associée une mesure de similarité (typiquement une distance). Cette mesure permet de quantifier la proximité entre descripteurs et, de manière indirecte, la similarité visuelle entre images. Ainsi, la proximité entre descripteurs se veut proportionnelle à la similarité visuelle entre images.

Lors d'une interrogation, un descripteur est tout à bord extrait à partir de l'image fournie en exemple (image requête). Ce descripteur requête est ensuite utilisé pour retrouver les descripteurs stockés dans la base qui lui sont les plus proches en terme de distance. Une fois retrouvés, ces descripteurs permettent d'obtenir les images auxquelles ils sont associés et qui sont censées être similaires à l'image requête. Ce mode de recherche se traduit le plus souvent par la recherche des voisins les plus proches du descripteur requête au sens de la mesure de similarité associée aux descripteurs. On parle ainsi de *recherche de plus proches voisins*. Ces recherches sont cependant extrêmement coûteuses lorsqu'elles s'appliquent à de très grandes collections d'images. Il est donc fait appel à des méthodes issues du domaine des bases de données pour tenter d'accélérer les interrogations.

Cette accélération repose sur le principe très général de limiter au maximum la quantité des descripteurs à comparer avec le descripteur requête. Confiner la recherche permet de réduire la quantité des données à lire depuis le disque et de diminuer le nombre d'opérations de calcul de similarité à effectuer. Ceci réduit les temps de réponse de la recherche. Les stratégies permettant de limiter l'étendue des recherches sont issues des algorithmes proposés pour l'indexation multidimensionnelle.

Ainsi, un SRIC complet requiert l'intégration dans un même système, de techniques issues de ces deux domaines. Or, il faut constater que peu de SRICs réels font appel aux techniques d'indexation multidimensionnelle.

On donne généralement deux raisons à cela. D'abord, la dimension élevée des descripteurs pose de sérieux problèmes de performances aux algorithmes d'indexation et de recherche. Brièvement, les propriétés particulières des espaces de grande dimension et le problème de la « malédiction de la dimension » font que leurs performances se dégradent exponentiellement lorsque la dimension augmente. Il a même été montré qu'en grande dimension, leur temps de réponse est supérieur à celui d'une simple recherche séquentielle et exhaustive dans la base [Weber *et al.*, 1998, Beyer *et al.*, 1999, Amsaleg & Gros, 2001].

Ensuite, les algorithmes d'indexation et de recherche développés en bases de données sont souvent inadaptés aux descripteurs d'images réels car ils sont souvent validés en utilisant des schémas de description simplistes par rapport aux schémas développés dans le domaine de l'analyse d'images. De plus, nombre de ces algorithmes reposent sur des hypothèses irréalistes concernant la distribution des descripteurs. En particulier, ils supposent que les descripteurs suivent une distribution particulière, notamment uniforme, ou que leur dimension peut être facilement réduite par des méthodes d'analyse de données.

Partant de ce constat, nous nous sommes intéressés dans ce travail de thèse au problème de la réduction du coût de la recherche par similarité dans les très grandes bases de descripteurs. Nous nous sommes positionnés à la frontière entre le domaine des bases de données

et celui de l'analyse d'images afin de mieux intégrer dans les stratégies d'indexation les particularités des méthodes de description d'images, particularités souvent mal prises en compte.

Ainsi, notre objectif est de développer une méthode *efficace* de recherche de plus proches voisins qui soit adaptée aux applications d'indexation par le contenu et aux propriétés des descripteurs d'images (grand volume, grande dimension, aucune hypothèse sur la distribution des descripteurs...).

Pour cela, nous nous sommes appuyés sur des travaux récents qui ont montré que le problème de la malédiction de la dimension est particulièrement manifeste lorsqu'il s'agit de retrouver les plus proches voisins « exacts » du descripteur requête [Bennett *et al.*, 1999, Weber & Böhm, 2000, Li *et al.*, 2002]. Ces travaux suggèrent d'échanger une petite imprécision lors de la recherche contre une forte réduction du temps de la recherche. Plusieurs travaux ont adopté cette approche et de nombreuses méthodes pour la recherche approximative ont été proposées.

Cependant, ces approximations imposent à ces méthodes de doter leurs stratégies de recherche de mécanismes permettant de contrôler la précision des résultats. En pratique, ce contrôle est primordial car il sert à adapter la précision de la recherche aux contraintes imposées par l'application considérée. Or, les méthodes existantes ne permettent, pour certaines, aucun contrôle de la précision, et pour les autres, ce contrôle est mis en œuvre au travers de nombreux paramètres difficiles à manipuler par un utilisateur non spécialisé. Par ailleurs, nombre de ces méthodes ne permettent de rechercher que *le* plus proche voisin.

En reprenant le principe de la recherche approximative et en se basant sur une étude approfondie des propriétés des espaces de grande dimension, nous proposons une méthode originale de recherche approximative de plus proches voisins. Cette méthode permet de contrôler la précision de la recherche de façon probabiliste à l'aide d'un seul paramètre appelé le niveau d'imprécision. Ce paramètre étant une probabilité portant sur l'ensemble des résultats (plus proches voisins recherchés), il est donc intuitif et peut être fixé même en n'ayant aucune connaissance sur le fonctionnement interne de la méthode. De plus, ce paramètre est fixé en ligne, ce qui permet à l'utilisateur d'adapter la recherche en fonction de ses contraintes. Il peut, par exemple, choisir d'effectuer tout d'abord une recherche grossière mais très rapide en fixant un niveau d'imprécision élevé. Dans le cas où les résultats retournés ne sont pas satisfaisants, il peut alors relancer la recherche avec un niveau d'imprécision plus faible. Par ailleurs, cette méthode est fiable et permet d'effectuer des recherches de k plus proches voisins.

Nous proposons également deux autres modes de recherche incrémentielle. Ce sont deux modes de recherche interactive dans lesquels les résultats de la recherche sont affichés à l'utilisateur au fur à mesure de l'avancement de la recherche. Celui-ci peut alors interrompre la recherche dès que les résultats retournés sont satisfaisants.

Nous nous sommes intéressés dans un deuxième temps à l'application de la méthode de recherche approximative de plus proches voisins à la recherche d'images par le contenu. L'objectif est d'intégrer puis d'évaluer notre méthode pour la recherche de plus proches voisins dans le cadre d'un système réel de recherche d'images par le contenu. Cette évaluation concerne à la fois l'efficacité et l'impact de l'imprécision de notre méthode sur les performances du système de recherche en terme de reconnaissance d'images. Il importait donc

de choisir une application dont l'évaluation des performances en terme de reconnaissance peut être effectuée de façon automatique.

Le choix s'est porté sur un SRIC pour la détection de copies. Le but de ce système est de vérifier si des images de propriétés « douteuses » ne proviennent pas de la collection d'un propriétaire. Celui-ci peut être une agence de photos qui utilise le système pour vérifier si des images présentes sur des sites tiers sur le Web, ne proviennent pas de sa propre collection de photos. La vérification des images douteuses s'effectue dans ce cas uniquement sur la base de la similarité visuelle entre images. Dans ce système, de fortes contraintes sont imposées à la fois sur le schéma de description des images et sur l'efficacité du système. En effet, les images piratées sont le plus souvent retouchées, ce qui rend leur identification extrêmement difficile. Ce système doit également pouvoir traiter un très grand nombre d'images récupérées du Web et de les confronter aux images de la base.

Par ailleurs, cette application se base sur un schéma de description d'images très robuste mais extrêmement coûteux. Il s'agit des descripteurs locaux différentiels [Schmid, 1996, Amsaleg & Gros, 2001]. Elle constitue, par conséquent, un environnement approprié pour une évaluation rigoureuse de notre méthode de recherche.

Résumé des contributions

La première contribution de ce travail est l'analyse des difficultés liées aux espaces de grande dimension et leur impact sur les performances des techniques de recherche de plus proches voisins. Cette analyse permet notamment d'expliquer les causes de la dégradation des performances des techniques traditionnelles d'indexation multidimensionnelle lorsque la dimension augmente (problème de la malédiction de la dimension). Elle étudie également les différentes méthodes de réduction de la dimension et leurs applications à la recherche de plus proches voisins. Ces méthodes sont généralement présentées comme une solution permettant de pallier les difficultés liées aux grandes dimensions. Cette étude montre que leur utilisation n'est pas triviale et qu'elle est sujette à discussion.

La seconde et principale contribution est la conception d'une méthode efficace de recherche de plus proches voisins. Cette méthode permet d'accélérer considérablement le temps de recherche en échange de petites imprécisions dans les résultats. Cette imprécision est toutefois contrôlée de façon intuitive, en ligne et à l'aide d'un seul paramètre.

Enfin, l'application de la méthode de recherche de plus proches voisins que nous avons proposée, à la recherche d'images par le contenu constitue notre dernière contribution. Un système de recherche d'images par le contenu pour la détection de copies a été construit et notre méthode de recherche y a été intégrée. Cette application a permis une évaluation rigoureuse de la méthode de recherche.

Plan de la thèse

Ce document décrit tour à tour différents aspects des travaux concernant la recherche de plus proches voisins et son application à la recherche d'images par le contenu. Il est organisé en trois parties de la manière suivante.

- La première partie est composée de trois chapitres. Le premier dresse un panorama des techniques traditionnelles d'indexation multidimensionnelle. Le deuxième analyse

le problème de la malédiction de la dimension, détaille quelques propriétés des espaces de grande dimension et leurs effets sur les index multidimensionnels. Les techniques de réduction de la dimension sont ensuite présentées, le but étant de montrer l'insuffisance de ces techniques à pallier le problème de la malédiction de la dimension. Dans le dernier chapitre de cette partie, sont présentées les nouvelles approches pour la recherche de plus proches voisins. Il est notamment question des approches pour la recherche approximative.

- La deuxième partie est consacrée à la méthode de recherche approximative de plus proches voisins que nous proposons. Cette partie est également composée de trois chapitres. Dans le premier, est présenté un algorithme de *clustering* que nous avons développé pour le regroupement des descripteurs dans une phase préliminaire de structuration des descripteurs (hors ligne). Ensuite, notre méthode pour la recherche approximative de plus proches voisins est détaillée dans le deuxième chapitre. Enfin, dans le troisième, l'évaluation des performances de cette méthode est présentée.

- Dans la troisième et dernière partie, l'application de la méthode de recherche approximative à la recherche d'images par le contenu pour la détection de copies est présentée. Cette partie est composée de deux chapitres. Dans le premier, le contexte et les contraintes de l'application considérée sont discutés. Ensuite, la méthode utilisée pour la description des images est décrite. Enfin, l'impact de l'imprécision de la recherche de plus proches voisins sur le résultat final est analysé. Dans le deuxième chapitre, la méthodologie d'évaluation du système proposé est décrite puis les résultats expérimentaux obtenus sont présentés.

- Enfin, la conclusion générale présente une synthèse des travaux menés dans le cadre de cette thèse ainsi que les perspectives liées à ce travail.

Première partie

État de l'art des méthodes de recherche de plus proches voisins

Chapitre 1

Indexation multidimensionnelle

Ce chapitre présente les techniques traditionnelles de base de données pour la recherche par similarité dans les bases de données multidimensionnelles. Il s'agit des index multidimensionnels. Après la définition des éléments de base liés à ces index et les différents modes de recherche associés, nous en donnerons le principe général et nous dresserons un panorama des différentes techniques existantes. Nous classerons ces techniques en deux grandes familles selon la stratégie adoptée pour la construction de l'index. Nous conclurons ce chapitre par une synthèse des propriétés des techniques présentées.

1.1 Introduction

Les index multidimensionnels ont été proposés pour permettre un accès et une interrogation efficaces dans des bases de données multidimensionnelles. Ils reposent sur le même principe que les index traditionnels tel que le B-Tree [Beyer & McCreight, 1972] mais permettent de gérer des données multidimensionnelles et d'exécuter des recherches par similarité. Ils *structurent* les données de la base pour accélérer les recherches en limitant autant que possible les entrées-sorties (E/S) et le coût de calcul CPU. Leur utilisation est recommandée dès que la base de données devient conséquente et qu'il est obligatoire de stocker les données sur disque.

L'une des premières applications à laquelle étaient destinés ces index est la gestion des systèmes d'information géographique. Les données manipulées sont bi ou tri dimensionnelles et représentent des cartes et des données géographiques [Samet, 1990]. Ces données sont décrites par des formes et des vecteurs d'attributs, et l'objectif est de pouvoir exécuter efficacement des requêtes d'adjacence, d'inclusion, de calcul du plus court chemin...

Ces index ont été ensuite utilisés dans d'autres domaines comme la recherche de motifs dans les bases de données génomiques et la recherche par similarité dans les bases de données multimédia. Dans ce chapitre, nous présentons ces index dans le cadre de cette dernière application, c'est-à-dire la recherche par similarité dans les bases de données multimédia et plus précisément pour des applications de recherche d'images par le contenu.

Dans la suite, nous utilisons indifféremment les termes descripteurs ou vecteurs pour désigner les entités caractérisant le contenu des images.

Nous utilisons également les expressions base de données, base de vecteurs ou base de descripteurs pour désigner l'ensemble des descripteurs calculés pour la collection des images gérées. Celle-ci est appelée quant à elle base d'images. Dans le cadre de la recherche par similarité, on s'intéresse uniquement à la base des descripteurs car le but est de retrouver les descripteurs similaires au descripteur requête. La base des images est nécessaire uniquement pour le calcul des descripteurs une seule fois au départ, et pour l'affichage éventuel des résultats de la recherche sous forme d'images à l'utilisateur.

Par ailleurs, notons que l'utilisation de l'expression « base de données » est abusive car aucun SGBD n'est associé à ces données et aucune fonctionnalité relative aux bases de données telles que la gestion des pannes, la gestion des concurrences d'accès ou bien la définition d'un langage d'interrogation (type SQL) ne sont nécessaires à ce niveau. La base de vecteurs est tout simplement un fichier dans lequel sont déposés les descripteurs séquentiellement et l'objectif est de le structurer afin d'accélérer les exécutions des recherches par similarité.

1.2 Définitions de base

Bien que les index multidimensionnels permettent également d'effectuer des recherches exactes et des recherches par intervalle, exécuter efficacement des requêtes de recherche par similarité reste leur principal objectif. Comme nous le verrons dans la suite de ce chapitre, ce mode de recherche est le plus coûteux et le plus complexe.

Mais avant, nous définissons de manière précise la recherche par similarité et les différentes formes qu'elle peut prendre.

Définition 1 (Recherche par similarité) *Ce mode de recherche permet de retrouver les vecteurs les plus similaires à un vecteur requête au sens d'une mesure de similarité donnée. La recherche par similarité peut être exécutée soit par une recherche de plus proches voisins soit par une recherche à ε-près.*

Définition 2 (Recherche de k-plus proches voisins) *Il s'agit de rechercher les k vecteurs les plus proches du vecteur requête au sens de la mesure de similarité associée aux vecteurs.*

Définition 3 (Recherche à ε-près) *Il s'agit de rechercher l'ensemble des vecteurs qui se trouvent à une distance ε du vecteur requête au sens de la mesure de similarité associée aux vecteurs. Cet ensemble est défini par :*

$$\{v \in BD \mid sim(q, v) < \varepsilon\}.$$

où BD est la base de vecteurs, v est un vecteur de la base, q est le vecteur requête et sim est la mesure de similarité associée aux vecteurs.

Chacune de ces deux stratégies possède des avantages et des inconvénients. La recherche de k-plus proches voisins garantit systématiquement k vecteurs dans l'ensemble résultat. Cependant, certains de ces vecteurs peuvent être très éloignés du vecteur requête et ne peuvent être considérés comme similaires. La recherche à ε-près permet à un utilisateur

expérimenté qui maîtrise la distribution de ses vecteurs d'éviter ce problème en choisissant une valeur appropriée de ε. Mais dans le cas général, le choix du paramètre ε reste problématique : une trop petite valeur impose une recherche très restrictive et peut donner lieu à des ensembles résultats vides ; une trop grande valeur peut engendrer, à l'inverse, des ensembles résultats de très grande taille.

En pratique, c'est la recherche de plus proches voisins qui est souvent privilégiée car elle ne nécessite aucune connaissance sur la distribution et sur l'ordre de grandeur des vecteurs. Par ailleurs, une étude *a posteriori* de la distribution des distances entre le vecteur requête et ses k-plus proches voisins permet d'éliminer les vecteurs les plus éloignés et de remédier ainsi à l'inconvénient majeur de ce mode de recherche. Par conséquent, nous considérons dans la suite de ce travail uniquement la recherche de plus proches voisins .

1.3 Principe des index multidimensionnels

Les techniques traditionnelles d'indexation multidimensionnelles reposent sur le principe de regrouper en paquets les vecteurs de la base, puis de les englober dans des cellules ayant une forme simple à manipuler. L'idée est de considérer les données par paquets au lieu de les considérer vecteur par vecteur. Lors de la recherche, ceci permet de travailler d'abord sur les paquets pour sélectionner les plus pertinents et de n'accéder, dans un deuxième temps, qu'aux vecteurs qui appartiennent aux paquets sélectionnés. Cette façon de rechercher en deux étapes permet de réduire le nombre de vecteurs à lire et à comparer au vecteur requête, ce qui permet de réduire le nombre d'entrées/sorties et le nombre de calculs de distance à effectuer, et, par conséquent, le temps global de la recherche. Dans ce mode de recherche, le coût de la recherche est principalement composé de calculs CPU (calculs de distance). Ceci est l'une des principales différences entre cette application et les applications d'indexation traditionnelles dans le domaine des bases de données. Il est donc important de réduire à la fois le nombre d'E/S et le coût des calculs de distance.

Cette section est organisée en deux sous-sections. Dans la première, est présenté le principe de base pour la formation des paquets. Dans la deuxième, nous décrivons les algorithmes de recherche de plus proches voisins associés aux index multidimensionnels.

1.3.1 Formation des cellules

Il existe principalement deux stratégies de création de cellules. L'une basée sur le partitionnement des vecteurs et l'autre basée sur le partitionnement de l'espace. Les algorithmes utilisant la première stratégie partitionnent les vecteurs de la base en paquets selon la distribution des vecteurs et leur proximité relative dans l'espace. À chaque paquet, ils associent une cellule qui englobe l'ensemble de ses vecteurs. On trouve dans cette catégorie des techniques comme le X-Tree [Berchtold *et al.*, 1996] ou le SS-Tree [White & Jain, 1996]. La deuxième approche divise l'espace multidimensionnel directement en cellules suivant une grille plus ou moins complexe et régulière, comme dans le cas du K-D-B-Tree [Robinson, 1981] ou du LSD^h-Tree [Henrich, 1998].

Ce principe de création de cellules est très similaire à celui des algorithmes de *clustering*. Certains états de l'art sur le *clustering* considèrent même les structures d'index multidimensionnels comme des techniques particulières de *clustering* [Keim & Hinneburg, 1999].

Les méthodes de *clustering*[1] permettent de regrouper les données en groupes homogènes de données colocalisées dans l'espace. Ces groupes sont appelés *clusters*. Pour cela, elles se basent uniquement sur la proximité entre les données. Celle-ci est mesurée par une mesure de similarité qui est généralement une distance. Ces méthodes ont été initialement développées en statistique. Leur objectif initial était d'extraire la connaissance enfouie dans de grandes bases de données. Les connaissances extraites se présentent généralement sous forme de dépendances entre les attributs constituant les vecteurs de données. Chaque *cluster* est censé représenter une notion particulière ou une relation de causes à effets. Le *clustering* a été ensuite appliqué dans d'autres domaines d'application, notamment l'indexation comme nous le verrons dans le chapitre 3, où certaines techniques de recherche de plus proches voisins s'inspirent des algorithmes de *clustering* pour la mise au point de leur stratégie de formation de cellules.

1.3.2 Algorithmes de recherche

Les algorithmes de recherche de k-plus proches voisins associés aux index multidimensionnels utilisent les propriétés géométriques des cellules pour éliminer de l'analyse celles qui contiennent des vecteurs n'ayant aucune chance d'être parmi les voisins les plus proches du vecteur requête. Cette élimination n'altère donc en rien l'exactitude du résultat. Elle permet, cependant, d'accélérer la recherche.

Les index sont le plus souvent des structures arborescentes. Ils permettent d'organiser les vecteurs en paquets, puis en paquets de paquets et ainsi de suite. Les propriétés géométriques des cellules sont typiquement employées par les algorithmes de recherche dans tous les niveaux de l'arbre pour élaguer les branches non pertinentes.

Règles de filtrage. Les propriétés géométriques des cellules sont utilisées par les algorithmes de recherche au travers de filtres utilisés à deux étapes du processus de recherche. Le premier filtre est appliqué au tout début du processus. Il considère uniquement les cellules et permet d'identifier celles à éliminer de la recherche à l'aide de la règle suivante[2] :

$$\text{si } \mathrm{dmin}(q, C_i) \geqslant \mathrm{dmax}(q, C_j) \text{ alors éliminer } C_i. \tag{1.1}$$

Ici, $\mathrm{dmin}(q, C_i)$ est la distance minimale entre le vecteur requête q et la cellule C_i et $\mathrm{dmax}(q, C_j)$ est la distance maximale entre q et la cellule C_j.

La suite de la recherche ordonne puis traite les cellules non éliminées selon leurs distances dmin par rapport à q. Chaque cellule est ensuite chargée en mémoire et toutes les distances entre les vecteurs qu'elle contient et le vecteur requête sont calculées. Cela peut modifier l'ensemble actuel des k-plus proches voisins.

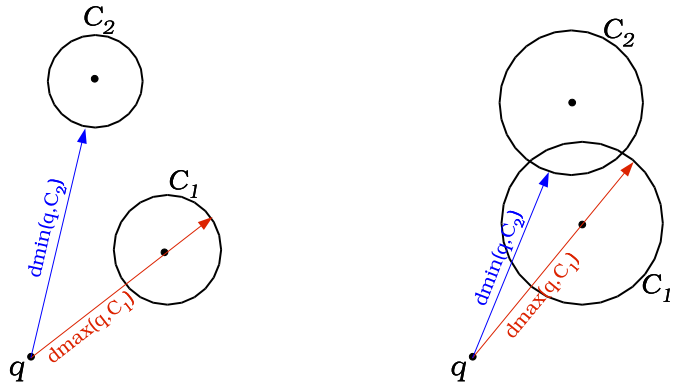
La seconde règle de filtrage sert à arrêter la recherche dès qu'il est détecté que les vecteurs de la prochaine cellule à traiter sont tous plus loin de q que ne l'est le k^e plus proche voisin courant. Cette règle stipule :

[1] Appelées aussi méthodes de classification non supervisée.

[2] Cette règle suppose implicitement que C_j contienne au moins k vecteurs, sans quoi cette règle ne s'applique telle qu'elle qu'à partir du moment où suffisamment de cellules ont été retenues pour garantir l'existence de k vecteurs dans le résultat en cours d'élaboration.

$$\text{si } \mathrm{dmin}(q, C_i) \geqslant d(q, ppv_k) \text{ alors arrêter la recherche} \tag{1.2}$$

où C_i est la prochaine cellule dans la liste des cellules à traiter et $d(q, ppv_k)$ est la distance entre q et le k^e plus proche voisin courant.



(a) (b)

FIG. 1.1: Deux cas de figure de l'application de la première règle de filtrage sur deux cellules sphériques et un vecteur requête.

L'efficacité de ces règles, c'est-à-dire leur capacité à éliminer beaucoup de cellules, est liée à la disposition des cellules et à leur compacité : plus les cellules sont compactes et éloignées les unes des autres, meilleure est l'efficacité des règles de filtrage. Si, par contre, les cellules se recouvrent (se chevauchent), il est impossible pour ces règles de déduire celles qui n'ont aucune chance de contenir les plus proches voisins recherchés.

La figure 1.1 montre un exemple de deux cellules sphériques et d'un vecteur requête dans deux cas de figure :

1. (a) les deux cellules sont complètement séparées ; il est facile pour la règle de filtrage (1.1) de déduire que la cellule C_2 ne peut contenir des vecteurs plus proches de q que les vecteurs de la cellule C_1 ($\mathrm{dmin}(q, C_2) > \mathrm{dmax}(q, C_1)$) ;
2. (b) les deux cellules se chevauchent ; même si, visuellement, il apparaît que la cellule C_2 ne peut contenir des vecteurs plus proches de q que certains des vecteurs de la cellule C_1 (car le centre de C_1 correspond au centre de gravité des vecteurs contenus dans cette cellule), la règle 1.1 ne peut pas décider d'éliminer la cellule C_2 car $\mathrm{dmin}(q, C_2) \not> \mathrm{dmax}(q, C_1)$.

Nous verrons dans la section 2.3 du chapitre suivant comment ce phénomène s'amplifie lorsque la dimension des vecteurs augmente.

Algorithmes RKV et HS. Ces règles de filtrage peuvent être reprises et affinées lorsqu'elles sont appliquées aux algorithmes de recherche associés à certaines structures d'index particulières (les structures arborescentes par exemple). Il s'agit principalement de l'algorithme de recherche de plus proches voisins incrémentiel HS [Hjaltason & Samet, 1995]

et l'algorithme RKV [Roussopoulos *et al.*, 1995]. Ce dernier, par exemple, tire profit des particularités des hyper-rectangles englobants minimaux pour renforcer la sévérité du filtrage. La première règle de filtrage est rendue plus sévère parce que RKV utilise une distance plus petite que dmax. Cette distance, appelée MinMaxDist, garantit au processus de recherche qu'il existe au moins un vecteur de la cellule situé à une distance inférieure ou égale à MinMaxDist. Dans le cas des recherches de k-plus proches voisins, la k^e plus petite MinMaxDist est utilisée. Comme MinMaxDist $<$ dmax, ceci permet de filtrer plus de cellules lors de la première étape de la recherche. RKV a été appliqué pour la recherche de plus proches voisins dans les index de la famille du R-Tree (*cf.* section 1.4.1).

L'algorithme HS, quant à lui, propose une stratégie globale pour la sélection de la branche suivante à explorer dans le cas des structures arborescentes. Pour cela, il maintient une liste globale de branches candidates avec leurs distances, et passe d'un nœud à un autre sans qu'il ne soit nécessaire d'explorer toutes les (sous-)branches du nœud de départ. Le parcours de l'arbre « saute » ainsi de branche en branche, selon un critère de distance, et est alors complètement indépendant de la structure de l'arbre, s'affranchissant de parcours typiques de type profondeur ou largeur d'abord.

1.4 Principales techniques basées sur le partitionnement des données

Ces techniques, toutes dérivées du R-Tree, procèdent par regroupement des vecteurs selon leur proximité relative dans l'espace. Chaque groupe de vecteurs est englobé dans une forme géométrique simple à manipuler (hyper-rectangle, hypersphère, 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 nœuds internes. Une forme englobante « couvre » tous les vecteurs compris dans le sous-arbre pointé par le nœud correspondant. La construction de l'arbre s'effectue soit par insertions successives soit par une construction *bottom-up*. Les algorithmes de recherche souvent associés à ces techniques sont HS et RKV. On trouve dans cette famille des méthodes comme le R*-Tree [Beckmann *et al.*, 1990], le X-Tree [Berchtold *et al.*, 1996], le SS-Tree [White & Jain, 1996], le SR-Tree [Katayama & Satoh, 1997], le TV-Tree [Lin *et al.*, 1994] et le M-Tree [Ciaccia *et al.*, 1997]. Nous les détaillons ci-après.

1.4.1 Les R-Tree, R$^+$-Tree et R*-Tree

Le R-Tree. Le R-Tree [Guttman, 1984] est l'extension directe du B-Tree [Beyer & McCreight, 1972] aux espaces multidimensionnels. C'est un arbre équilibré dans lequel les vecteurs, regroupés selon leur proximité relative, sont stockés dans les feuilles, alors que les nœuds sont constitués par une hiérarchie d'hyper-rectangles englobants, les nœuds du niveau supérieur englobant les données des niveaux inférieurs. Un hyper-rectangle est appelé rectangle englobant minimal (REM) et correspond au plus petit hyper-rectangle pouvant englober l'ensemble des données du niveau inférieur. Chaque entrée d'un nœud pointe vers une feuille ou vers un nœud du niveau inférieur et mémorise les caractéristiques de l'hyper-rectangle englobant tous les objets (REMs ou vecteurs) stockés

dans le sous-arbre pointé. On peut dire que les REMs d'un niveau inférieur appartiennent au REM du niveau supérieur, et, par extension, un vecteur est dit appartenir au REM qui l'englobe. La taille des nœuds et des feuilles est limitée et fixée *a priori*. Elle correspond à la taille d'une page du disque. La figure 1.2 montre un exemple de R-Tree. Par simplicité, seuls quelques points (ronds noirs sur la figure) sont représentés. Les REMs A, B, C et D sont englobés dans le REM 1. Par construction, les REMs peuvent se chevaucher.

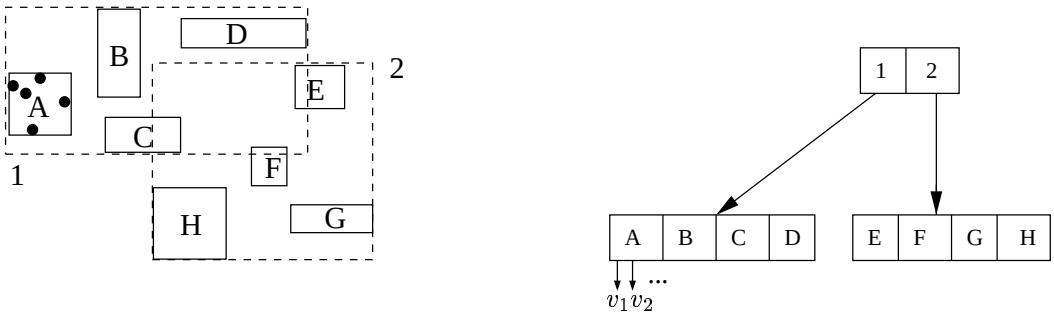


FIG. 1.2: Structure du R-Tree.

Selon l'algorithme présenté par [Guttman, 1984], la construction de l'arbre s'effectue de manière dynamique par insertions successives. Cependant, d'autres techniques de construction dites *bottom-up* ont été proposées [Kamel & Faloutsos, 1993, Bercken *et al.*, 1997, Berchtold *et al.*, 1998a].

De façon analogue au B-Tree, les insertions peuvent causer des fractionnements de feuilles, de nœuds, et éventuellement augmenter la hauteur de l'arbre.

Pour déterminer le cheminement à suivre dans l'arbre pour insérer un élément, trois cas sont considérés au niveau de chaque nœud :

- l'élément à insérer appartient à un seul des (sous-)REMs mémorisés par le nœud courant ; dans ce cas l'entrée correspondante est suivie ;
- l'élément à insérer appartient à plusieurs (sous-)REMs mémorisés par le nœud courant ; dans ce cas l'entrée correspondante au plus petit REM est suivie ;
- l'élément à insérer n'appartient à aucun des (sous-)REMs mémorisés par le nœud courant. Dans ce cas, l'entrée correspondante au REM qui nécessite le moins d'être agrandi est suivie. Si plusieurs REMs sont dans ce cas, alors le plus petit REM est choisi.

Pour gérer les dépassements de capacité des nœuds, les REMs du nœud saturé sont partitionnés en deux sous-ensembles de même cardinalité, avec comme objectif de minimiser le volume de l'espace couvert par les formes englobantes des deux sous-ensembles qui viennent d'être créés. Le fait d'avoir deux sous-ensembles de même cardinalité permet de préserver l'équilibre de l'arbre. La figure 1.3 illustre deux partitionnements possibles des REMs A, B, C et D. Le partitionnement (b) génère deux REMs qui occupent moins de surface que ceux du partitionnement (a). Minimiser les volumes des REMs permet d'éviter de gérer les zones d'espace vides entre les REMs, ce qui permet d'accroître l'efficacité des règles de filtrage des algorithmes de recherche.

[Guttman, 1984] présente trois algorithmes de partitionnement qui se différencient par leur complexité. L'algorithme exhaustif trouve le meilleur partitionnement après

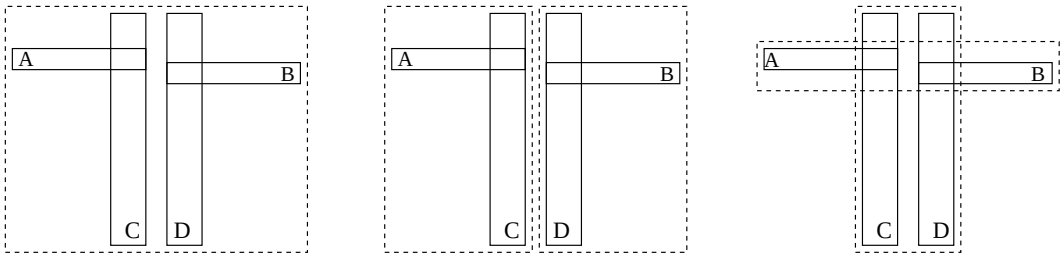


FIG. 1.3: Deux exemples de partitionnement d'un ensemble de 4 REMs dans un espace de dimension 2.

l'examen de toutes les possibilités, mais son coût est prohibitif. L'algorithme quadratique (en $O(n^2)$) et l'algorithme linéaire trouvent chacun un bon partitionnement qui peut toutefois ne pas être globalement le meilleur. La recherche de plus proches voisins dans les R-Tree utilise le plus souvent les algorithmes RKV ou HS (*cf.* section 1.3). Il est important de remarquer que le R-Tree reste très sensible à l'ordre d'insertion des vecteurs. Les mêmes vecteurs insérés dans un ordre différent donneront deux R-Tree différents.

Le R$^+$-Tree. Le R$^+$-Tree a été proposé par [Sellis *et al.*, 1987] comme une extension du R-Tree afin de résoudre le problème du chevauchement entre les REMs d'un R-Tree.

Dans [Sellis *et al.*, 1987], deux conditions ont été proposées pour qu'un R-Tree soit efficace: il faut, d'une part, minimiser le volume de l'espace couvert par les REMs afin d'éviter de gérer les régions vides, et, d'autre part, minimiser le taux de chevauchement entre les entrées d'un même nœud. Un taux de chevauchement important rend les règles de filtrage incapables d'élaguer les branches de l'arbre ne pouvant convenir. Par conséquent, une grande partie de l'arbre est explorée (souvent en vain) entraînant ainsi une dégradation des temps de réponse.

Il a été montré dans [Roussopoulos & Leifker, 1985] qu'il est impossible, dans certains cas, de partitionner un ensemble de REMs en deux sans qu'il n'y ait de chevauchement entre les deux formes qui englobent les deux sous-ensembles résultants. La figure 1.4 montre qu'il est impossible de partitionner les rectangles A, B, C et D en deux ensembles sans que ce partitionnement ne coupe un des rectangles. Par conséquent, il est impossible de trouver un partitionnement qui permette d'avoir deux sous-ensembles de même cardinalité dont les REMs soient complètement disjoints.

Éviter le chevauchement suggère d'autoriser à « couper » (en 2 dans le cas du rectangle D de la figure 1.4) et à « mettre de chaque côté » un objet à cheval sur la frontière du fractionnement à opérer. Ce principe est repris dans le R$^+$-Tree. Un objet ainsi coupé est référencé par chacun de ses morceaux, ce qui entraîne la duplication de l'identifiant de l'objet autant de fois qu'il y a de fractions le constituant. Aucun chevauchement n'existe alors entre les formes englobantes. Un inconvénient de cette technique est le coût relativement élevé des opérations d'insertion puisque le fractionnement d'un hyper-rectangle se répercute sur tous les niveaux inférieurs où doivent être dupliquées certaines informations.

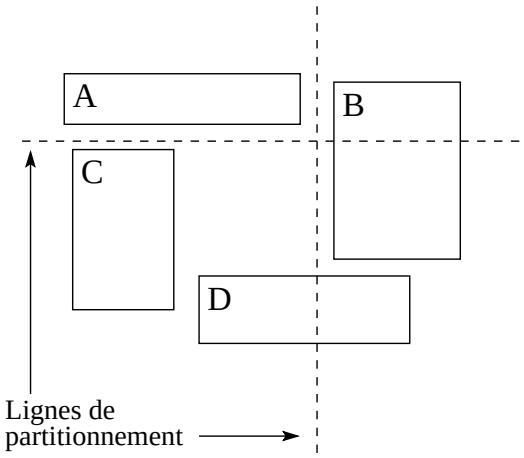


FIG. 1.4: Partitionnement d'un ensemble de REMs dans un espace de dimension 2.

Le R*-Tree. Dans [Beckmann *et al.*, 1990], Beckmann *et al.* remarquent que la réinsertion des données stockées dans un R-Tree (suppression de données déjà insérées puis nouvelle insertion) permet d'améliorer ses performances considérablement et de limiter l'indéterminisme et la sensibilité des R-Tree à l'ordre d'insertion des vecteurs. Pour cela, ils proposent dans le R*-Tree, de supprimer puis de réinsérer dans le même niveau de l'arbre un sous-ensemble des entrées d'une page saturée avant de la fractionner. Il s'est avéré que, dans plusieurs cas, le fractionnement envisagé est finalement évité, et, par conséquent, le taux d'exploitation de l'espace alloué est amélioré. De plus, cette réinsertion permet de réorganiser l'arbre continuellement et diminue les effets d'indéterminisme dus aux changements dans l'ordre d'insertion des données. Par ailleurs, le R*-Tree utilise des procédures d'insertion et de fractionnement différentes de celles du R-Tree. Pour l'insertion, le choix du chemin d'insertion prend en compte deux contraintes : minimiser le volume des REMs lorsqu'il s'agit d'une insertion dans un nœud et minimiser le taux de chevauchement lorsqu'il s'agit d'une insertion dans une feuille. Concernant la procédure de partitionnement des REMs d'un nœud saturé, un critère mixte prenant en compte le taux de chevauchement, le périmètre des hyper-rectangles ainsi que leurs volumes doit être minimisé. L'introduction du critère « périmètre » dans cette procédure a pour but de produire des hyper-rectangles de petite diagonale ce qui permet d'améliorer la *co-localisation* des vecteurs appartenant à une même région, et donc d'améliorer les performances des procédures de recherche.

Résumé. En résumé, la famille des techniques d'indexation multidimensionnelles 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 de niveau inférieur ;
- les REMs permettent d'optimiser les règles de filtrage par l'utilisation d'une distance plus précise que dmax (MinMaxDist) ;
- 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 génèrent des régions de petit volume mais dont les diagonales risquent d'être très grandes [Katayama & Satoh, 1997] ; par conséquent, les vecteurs appartenant à une même région risquent d'être très peu semblables ce qui détériore les performances des procédures de recherche par similarité ;
- le nombre d'éléments que l'on peut stocker dans les nœuds et les feuilles dépend de la dimension des données ; en effet, représenter un REM demande d'utiliser deux points multidimensionnels, et, par conséquent, plus la dimension est élevée, plus la mémorisation de chaque point consomme de l'espace, ce qui diminue la capacité de chaque nœud ; cela entraîne la construction d'arbres de grande hauteur ;
- la contrainte de partitionnement équilibré des REMs des nœuds saturés entraîne des partitionnements qui ne correspondent pas toujours aux partitionnements optimaux ;
- la complexité spatiale du R-Tree est calculable en fonction de la taille des pages et du facteur de remplissage choisi ;
- l'algorithme de construction du R-Tree par insertion successive est non déterministe : deux R-Tree construits en utilisant le même ensemble de données mais avec des ordres d'insertion différents ne sont pas forcément identiques.

1.4.2 Le X-Tree

Le partitionnement de REMs génère un chevauchement croissant avec la dimension des données. Aussi, Berchtold *et al.* proposent, dans [Berchtold *et al.*, 1996], le X-Tree, un index multidimensionnel similaire au R-Tree, mais équipé d'une nouvelle stratégie de partitionnement de nœuds rejetant le fractionnement si celui-ci génère un taux de chevauchement trop élevé. Cette stratégie, *overlap-free split*, est associée à la notion de *super-nœud*. Trois étapes ponctuent l'insertion dans un nœud saturé. Un algorithme de fractionnement similaire à celui du R-Tree utilisant les propriétés géométriques des formes englobantes est d'abord employé. Si le fractionnement proposé entraîne un chevauchement inférieur à un seuil fixé, le fractionnement a lieu et l'insertion se termine. Sinon, un autre fractionnement est calculé, mais dans ce cas, l'historique des fractionnements qui ont touché les formes englobantes du nœud est pris en compte : le but est de choisir la dimension selon laquelle la majorité des formes englobantes a été fractionnée, car cela minimise le taux de chevauchement. Un taux nul est atteint s'il existe une dimension selon laquelle toutes les formes englobantes ont été précédemment fractionnées. L'insertion s'arrête si le partitionnement obtenu est équilibré, ou, du moins, s'il garantit l'existence d'un minimum d'entrées dans les pages créées. Sinon, le fractionnement du nœud est annulé et une nouvelle page disque est allouée et concaténée au nœud saturé. Le rejet du fractionnement et l'ajout d'une nouvelle page forme un *super-nœud*. Notons que, dans un cas extrême où le chevauchement est très élevé, le X-Tree n'est alors constitué que d'un unique *super-nœud* dans lequel seule une recherche séquentielle est possible.

Bien que le X-Tree repose sur des principes assez simples à mettre en œuvre, les paramètres de l'algorithme sont très difficiles à fixer et en affectent les performances globales. En effet, la structure globale du X-Tree dépend essentiellement du seuil maximal du taux de chevauchement. Si la valeur de ce paramètre est très grande, des taux de chevauchement élevés sont tolérés, et, par conséquent, le X-Tree devient un R-Tree. Dans le cas où cette valeur est très petite, peu de chevauchements sont tolérés, et, par conséquent, le X-Tree

risque d'être un seul *super-nœud* dans lequel tous les vecteurs de la base sont stockés. De plus, étant donné que la taille des nœuds et des feuilles du X-Tree est variable, ni la taille de l'arbre ni sa complexité ne peuvent être calculées.

1.4.3 Le SS-Tree

Le SS-Tree [White & Jain, 1996] est une structure d'index qui utilise les mêmes principes que le R*-Tree. Il utilise toutefois des hyper-sphères englobantes au lieu des hyper-rectangles. Le centre de la sphère est le centre de gravité des vecteurs englobés. La figure 1.5 illustre un exemple de SS-Tree ainsi que la représentation géométrique de ses entrées. Lors d'une insertion, un point est assigné à l'hyper-sphère dont le centre est le plus proche. Quand une page (nœud ou feuille) est saturée, le fractionnement s'effectue selon la dimension suivant laquelle la dispersion des données est la plus grande. Ceci permet de réduire le chevauchement entre hyper-sphères et d'améliorer les performances des requêtes du type plus proches voisins. Par ailleurs, une hyper-sphère est représentée par un centre (point multidimensionnel) et un rayon (valeur réelle), alors que pour représenter un hyper-rectangle deux points multidimensionnels sont nécessaires. Par conséquent, l'utilisation d'hyper-sphères permet d'accroître la capacité des nœuds *(fanout)* et de réduire la taille de l'arbre.

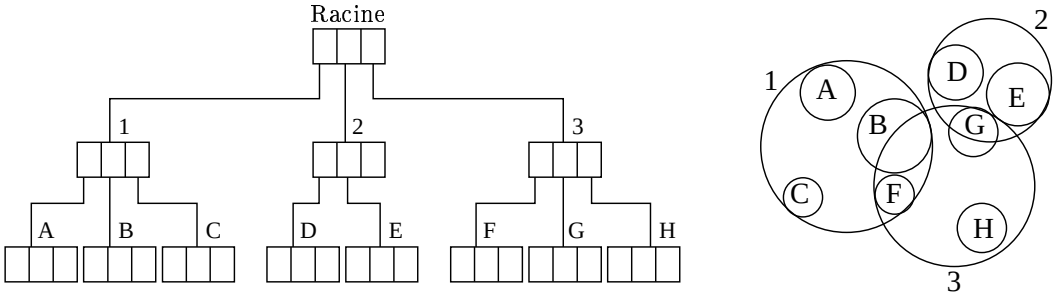


FIG. 1.5: Structure du SS-Tree et sa représentation géométrique.

De plus, le SS-Tree emploie aussi la réinsertion des entrées d'un nœud saturé avant de le fractionner. Cette stratégie est toutefois modifiée : la réinsertion d'un sous-ensemble des entrées d'un nœud saturé se répète jusqu'à ce que le fractionnement soit évité, ou bien jusqu'à ce que toutes les entrées réinsérées se retrouvent dans le même nœud. Dans ce dernier cas seulement, le nœud est fractionné.

Tout comme le R-Tree, le SS-Tree est un arbre équilibré de complexité calculable. De plus, des algorithmes de recherche (typiquement RKV ou HS) utilisant les règles de filtrage détaillées dans la section 1.3 peuvent y être associés pour effectuer des recherches de plus proches voisins.

1.4.4 Le SR-Tree

Les expérimentations menées dans [Katayama & Satoh, 1997] montrent que les hyper-rectangles divisent un espace de grande dimension en régions de petits volumes mais possédant de grandes diagonales. Par contre, les hypersphères possèdent de petits diamètres mais des volumes comparativement plus importants que ceux des hyper-rectangles. En

général, des régions avec de petits volumes diminuent le taux de chevauchement, et des régions de petites diagonales sont des régions compactes qui améliorent les performances des algorithmes de recherche. Partant de ces constats, le SR-Tree [Katayama & Satoh, 1997] propose d'utiliser des régions englobantes qui sont le résultat de l'intersection d'un hyper-rectangle et d'une hypersphère. Ceci permet de profiter des avantages des deux. Cependant, la forme résultante étant complexe, la taille qu'occupe chacune dans un nœud augmente, et, par conséquent le *fanout* du SR-Tree est réduit. De plus, la complexité des formes englobantes accroît le coût des opérations d'insertion, de mise à jour et de recherche (les algorithmes utilisés sont généralement RKV ou HS).

1.4.5 Le TV-Tree

Pour limiter les effets liés aux espaces de grande dimension, [Lin *et al.*, 1994] propose le TV-Tree *(Telescopic Vector Tree)*, dérivé du R*-Tree, mais qui utilise des méthodes de réduction de la dimension pour en améliorer les performances. Au niveau de chaque nœud, les dimensions sont d'abord ordonnées selon leur importance (contenu informatif) et seulement un sous-ensemble d'entre elles est pris en compte. Les dimensions restantes sont ignorées. Cela permet de réduire la taille globale de l'arbre. Les formes englobantes choisies sont des L_p-sphères (sphères définies dans un espace muni de la métrique L_p). Le nombre de composantes prises en compte pour chaque vecteur dépend de la cellule dans laquelle il se trouve. La méthode de réduction de la dimension étant considérée comme orthogonale au TV-Tree par les auteurs, la méthode implémentée est très simple : les dimensions selon lesquelles les vecteurs possèdent la même valeur sont éliminées. Cette approche est efficace si (1) les dimensions peuvent être ordonnées selon leur importance et (2) si les vecteurs de la base tendent à avoir les mêmes valeurs selon certaines dimensions. La première condition nécessite d'avoir des connaissances sur la distribution des données selon chaque dimension, par une étude statistique par exemple. La deuxième condition peut être extrêmement contraignante dans le cas où les vecteurs sont définis dans un espace réel. Par contre, si les vecteurs sont définis dans un espace discret au nombre de valeurs limitées, un alphabet par exemple, cette contrainte devient moins forte. Ces deux conditions limitent l'applicabilité du TV-Tree.

1.4.6 Le M-Tree

Toutes les techniques présentées jusqu'à présent ont pour objectif de réduire la quantité des données lues pendant la recherche sans se soucier explicitement du coût CPU nécessaire aux opérations de calcul de distance. Cependant, il existe des fonctions de distance assez compliquées dont le coût de calcul ne peut être négligé. Le M-Tree [Ciaccia *et al.*, 1997] est l'une des premières méthodes d'indexation qui vise à réduire, en plus du nombre d'entrées/sorties, le coût CPU des calculs de distance. Pour cela, le M-Tree confine la recherche en utilisant une structure arborescente similaire au R-Tree, et utilise l'inégalité triangulaire ainsi que quelques distances pré-calculées pour éviter certains calculs de distance inutiles.

Cette dernière idée repose sur une application de l'inégalité triangulaire dont le principe est le suivant. Soit un vecteur requête q et deux cellules C_1 et C_2 de centres o_1 et o_2 et de rayons r_1 et r_2 respectivement (voir figure 1.6). Soit, de plus, $d(q, ppv_k)$ la distance entre q et son k^e plus proche voisin courant. Afin d'appliquer les règles de filtrage

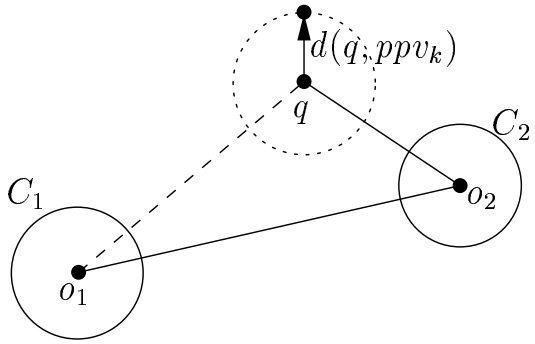


FIG. 1.6: Application de l'inégalité triangulaire dans la recherche de plus proches voisins.

lors de la recherche de plus proches voisins pour éliminer les cellules non pertinentes, il est nécessaire de calculer les distances minimales entre q et les deux cellules C_1 et C_2. Si on suppose cependant que la distance entre o_1 et o_2 est connue, il est possible d'effectuer uniquement un seul calcul de distance ($\mathrm{dmin}(q, C_2)$ par exemple).

Éliminer C_1 n'est possible que si

$$\mathrm{dmin}(q, C_1) \geqslant d(q, ppv_k),$$

ou en d'autres termes si

$$d(q, o_1) \geqslant d(q, ppv_k) + r_1.$$

Or, en appliquant l'inégalité triangulaire, on a :

$$d(q, o_1) \geqslant |d(q, o_2) - d(o_1, o_2)|.$$

Donc, si

$$|d(q, o_2) - d(o_1, o_2)| \geqslant d(q, ppv_k) + r_1$$

alors on peut affirmer que

$$d(q, o_1) \geqslant d(q, ppv_k) + r_1,$$

et, par conséquent, la cellule C_1 ne peut contenir de meilleur voisin que le k^e actuel. Dans ce cas, C_1 est éliminée sans avoir à calculer la distance entre q et o_1.

L'efficacité de l'inégalité triangulaire dépend fortement du rapport entre le coût d'une opération de calcul de distance et le coût d'une opération de comparaison. Ce rapport doit être suffisamment important pour justifier, d'une part le pré-calcul de certaines distances (idéalement les distances entre tous les vecteurs de la base sont à pré-calculer) et leur stockage, et d'autre part l'exécution de plusieurs instructions de comparaison qui ne permettent pas toujours d'éviter le calcul de la distance. [Braunmüller *et al.*, 2000a] donne quelques indications sur les coûts respectifs des calculs de distance (Euclidienne) et de l'application de l'inégalité triangulaire : il est 52 fois plus coûteux d'effectuer un calcul de distance sur des vecteurs de dimension 20 (155 pour des vecteurs de dimension 64) que d'utiliser le principe de l'inégalité.

Le M-Tree utilise ce principe. Il peut, de plus, être utilisé dans n'importe quel espace métrique (pas forcément euclidien). Dans un M-Tree, seules les distances relatives entre vecteurs sont considérées. Chaque cellule c est représentée par un doublet (o, r). o est un objet de la base. Il est appelé objet[3] de redirection *(routing object)*. r est appelé rayon de couverture *(covering radius)* et est égal à la distance entre o et le plus lointain objet compris dans le sous-arbre correspondant à la cellule c. Cette représentation correspond à une M-sphère englobante minimale. Au niveau de l'arbre, à chaque objet est associée sa distance par rapport à son objet de redirection. Cette distance, calculée lors de la construction du M-Tree, permet d'appliquer l'inégalité triangulaire lors de la recherche afin d'éviter certains calculs de distance inutiles en utilisant la méthode décrite dans le paragraphe précédent. Lorsque la recherche atteint un nœud N_i, l'inégalité triangulaire est d'abord appliquée pour essayer d'élaguer une partie des branches non pertinentes sans calculer la distance entre q et tous les objets stockés dans N_i. Cela est possible car la distance entre chaque objet stocké dans N_i et son objet de redirection est stockée dans l'arbre et la distance entre q et l'objet de redirection de N_i a été déjà calculée.

Le M-Tree est construit par insertions successives. Le critère de choix du chemin d'insertion vise à minimiser les rayons de couverture des objets de redirection. L'algorithme de fractionnement des pages saturées doit d'abord élire deux objets de redirection parmi les objets de la page, puis répartir les objets restants, de façon à minimiser le volume des deux régions obtenues ainsi que leur taux de chevauchement. Étant donné que le taux de chevauchement est généralement difficile à calculer dans les espaces métriques non Euclidiens, l'élection des deux objets de redirection se contente d'utiliser les rayons de couverture et peut être effectuée soit par une technique exhaustive, aléatoire ou en considérant uniquement un échantillon d'objets.

Notons que [Chu *et al.*, 2002] exploite la même idée pour réduire les coûts CPU d'un algorithme de *clustering* dans sa phase de recherche du plus proche centre d'un vecteur à insérer. Il utilise cependant une autre heuristique qu'il appelle distance partielle pour réduire davantage le coût CPU. Cette idée permet d'interrompre un calcul de distance dès que la distance partielle (calculée en considérant uniquement un sous-ensemble des composantes des vecteurs) est supérieure à la distance au plus proche voisin courant. Dans ce cas, il n'est pas nécessaire d'aller jusqu'au bout dans le calcul de la distance car le vecteur considéré n'a aucune chance d'être plus proche que le plus proche voisin courant. En effet, l'accumulation des distances sur les composantes restantes ne peut qu'augmenter la distance en cours de calcul.

Si on considère la distance Euclidienne sur des vecteurs de dimension d, si le calcul de la distance est interrompu à la dimension d', c'est-à-dire que la distance calculée en considérant uniquement les d' premières composantes est supérieure à la distance au plus proche voisin courant, alors $(d-d')$ opérations de soustraction, $(d-d')$ opérations d'addition et $(d-d')$ opérations de multiplication sont sauvegardées au prix de d' instructions de test.

L'extension de cette idée à la recherche de k-plus proches voisins est triviale. Il suffit d'utiliser la distance au k^e plus proche voisin courant dans les tests.

Il est à noter cependant que cette idée peut être appliquée à tout algorithme de recherche de plus proches voisins, y compris l'algorithme de recherche séquentielle.

[3]Nous avons repris la terminologie utilisée dans [Ciaccia *et al.*, 1997]. Un *objet* est tout simplement un vecteur multidimensionnel appartenant à la base de données.

1.5 Principales techniques basées sur le 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 techniques 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.

Elles reposent sur le principe qui consiste à partitionner *a 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'index faciles à construire et simples à gérer. De plus, le chevauchement entre cellules est évité. Ces techniques sont toutes dérivées du k-d-Tree [Bentley, 1979] et du GridFile [Nievergelt *et al.*, 1984]. Les cellules peuvent être gérées de deux manières différentes :

- soit organisées en arborescence d'une façon similaire à celle décrite dans la section précédente, ce qui est le cas des techniques de la famille du k-d-Tree : le K-D-B-Tree [Robinson, 1981], le LSD-Tree [Henrich *et al.*, 1989] et le LSD^{h}-Tree [Henrich, 1998] ;
- soit adressées par une fonction de hachage, ce qui est le cas des techniques de la famille du GridFile. Dans cette classe on trouve le BuddyTree [Seeger & Kriegel, 1990] et le twin GridFile [Hutflesz *et al.*, 1988].

1.5.1 Le K-D-B-Tree

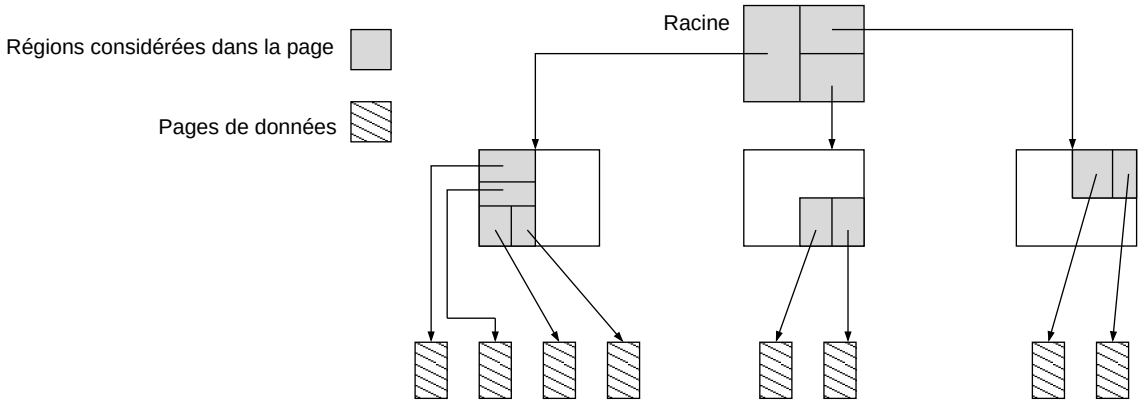


FIG. 1.7: Structure d'un K-D-B-Tree.

Le K-D-B-Tree [Robinson, 1981] est une structure d'index multidimensionnel qui se base sur le partitionnement de l'espace en régions organisées dans une arborescence similaire au R-Tree. Les nœuds et les feuilles de l'arbre correspondent à des régions de l'espace. Dans un espace de dimension d, chaque entrée dans un nœud représente un partitionnement de la région mère en deux régions adjacentes et disjointes à l'aide d'un hyperplan parallèle à $(d-1)$ axes de l'espace. Par conséquent, les régions d'un même niveau dans l'arbre sont toutes disjointes. La figure 1.7 illustre un exemple de la structure générale d'un K-D-B-Tree. L'union des régions considérées au niveau de la racine constitue la totalité

de l'espace, alors qu'au niveau de chaque nœud seule une partie de l'espace est prise en compte. Celle-ci est à son tour partitionnée sans chevauchement.

L'absence de chevauchement accroît les performances des algorithmes de recherche (typiquement RKV ou HS). Cependant, si un point requête est suffisamment proche de la frontière d'une région, il est alors nécessaire de visiter les régions voisines, ce qui peut se traduire par l'examen de très nombreuses régions voisines.

Par ailleurs, éviter le chevauchement et préserver l'équilibre du K-D-B-Tree constituent une contrainte assez forte à respecter, contrainte qui entraîne souvent des restructurations importantes de la totalité de l'arbre dès qu'il y a fractionnement d'un nœud saturé. Ces restructurations entraînent notamment la création de nœuds ou de feuilles vides ou presque vides. Par conséquent, le K-D-B-Tree ne peut garantir un taux minimal d'utilisation de l'espace alloué. D'autre part, la procédure de fractionnement d'une page saturée ne dépend pas de la distribution des données : la dimension est choisie cycliquement et la position de fractionnement est fixée arbitrairement (le premier point qui génère deux pages non saturées est retenu).

1.5.2 Le LSD-Tree et le LSD^h-Tree

Le LSD-Tree [Henrich *et al.*, 1989] est une technique d'indexation très similaire au k-d-Tree [Bentley, 1979] mais qui utilise une structure d'index plus simple et organisée en deux niveaux. Le premier niveau réside en mémoire centrale, alors que le deuxième niveau est stocké sur disque. Le LSD-Tree est un arbre binaire dans lequel chaque nœud représente un partitionnement de l'espace en deux à l'aide d'un hyperplan. Ainsi, un partitionnement est représenté par le numéro de la dimension selon laquelle le partitionnement est effectué et une position sur l'axe associé à cette dimension. Ces deux informations sont stockées au niveau de chaque nœud. Au niveau le plus bas de l'arbre on retrouve les pages de données. Ces pages sont de taille limitée et fixée *a priori*.

La construction de l'arbre s'effectue par insertions successives : à chaque fois que l'ajout d'un vecteur entraîne la saturation d'une page, celle-ci est alors fractionnée en deux et un nœud d'index est créé pour matérialiser le partitionnement résultant. Le choix de la dimension et de la position de fractionnement sont pris localement, soit en fonction de la distribution des données, soit arbitrairement. La structure d'index ainsi créée est un arbre binaire non équilibré. La figure 1.8 montre un exemple de LSD-Tree et le découpage de l'espace qui lui est associé.

Lorsque la taille de la base de données à indexer est très grande et que le LSD-Tree ne peut être stocké totalement en mémoire centrale, une organisation de l'arbre en trois niveaux est réalisée. Le premier niveau est conservé en mémoire centrale alors que les deux autres sont stockés en mémoire secondaire. Le premier niveau correspond aux parties supérieures de l'arbre et ne contient que des nœuds. Le deuxième niveau contient lui aussi des nœuds mais qui sont mis en paquets et stockés sur disque. Chaque paquet correspond à un sous-arbre. Ces paquets sont formés de manière à ce que les sous-arbres qu'ils contiennent aient tous la même hauteur. Ceci permet de réduire les inconvénients dûs au déséquilibre du LSD-Tree. Cependant, le maintien de cette structure est très coûteux quand le contenu de la base évolue. Le troisième niveau contient les pages de données. Le découpage du LSD-Tree en trois niveaux est illustré dans la figure 1.9.

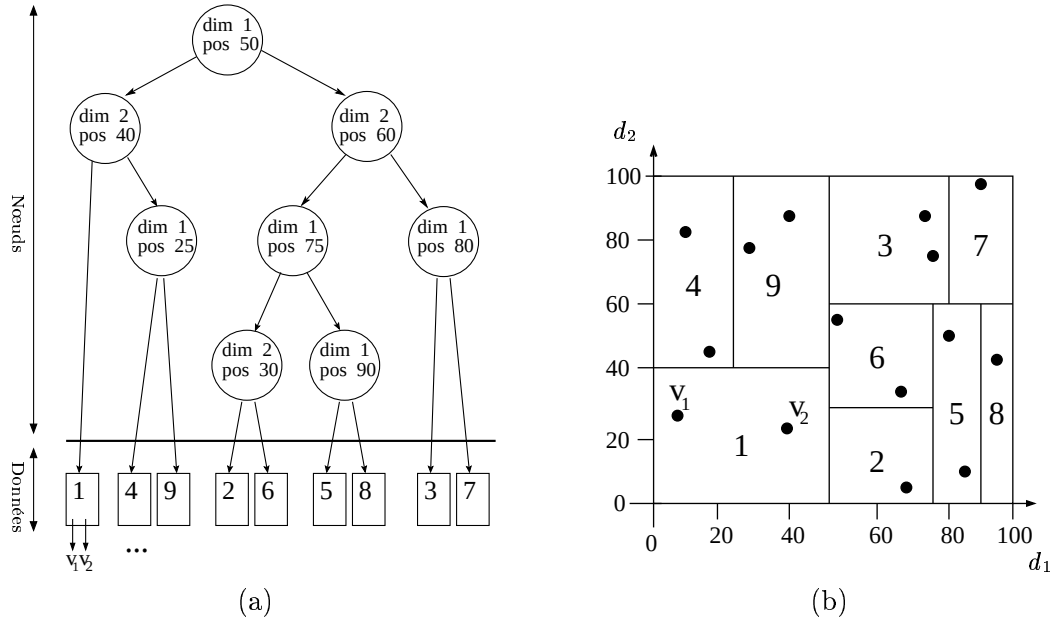


FIG. 1.8: Exemple d'un LSD-Tree associé au partitionnement d'un carré dans un espace de dimension 2.

La nature binaire de l'arbre simplifie énormément les procédures de recherche. Pour la recherche des k-plus proches voisins, une file contenant les régions candidates est gérée. Lors de la recherche, si l'un des nœuds fils est suivi, l'autre fils est mis dans la file si la distance minimale entre la région couverte par celui-ci et le point requête est inférieure à la distance entre le point requête et son k^e plus proche voisin courant. Dans le cas contraire, cela signifie que la région associée à ce fils ne peut contenir un des k-plus proches voisins du point requête. L'exploration du sous-arbre correspondant est alors inutile.

La structure du LSD-Tree est assez simple, sa construction ainsi que les algorithmes de recherche ne présentent aucune difficulté particulière. De plus, la taille des nœuds est indépendante de la dimension de l'espace. Cependant, le LSD-Tree possède quelques inconvénients. La structure non équilibrée ne permet d'avoir ni une estimation du temps de réponse ni une estimation de la taille du LSD-Tree. Sa complexité n'est donc pas calculable. D'autre part, la représentation d'une région en utilisant uniquement un plan de fractionnement de la région mère ne permet pas d'avoir une localisation précise des vecteurs au sein de la région. Cela réduit les performances des algorithmes de recherche et entraîne la gestion inutile de grandes zones d'espace vide. De plus, la détérioration des performances dues à cet inconvénient s'amplifie lorsque la dimension des données croît.

Pour remédier à ce problème, le LSD-Tree a été ensuite amélioré et étendu aux espaces de grande dimension par l'introduction du LSD^h-Tree [Henrich, 1998]. Pour diminuer la taille de la structure d'index et éviter de gérer les régions vides de l'espace, le LSD^h-Tree utilise la notion de « régions exactes codées ». Il s'agit de trouver au sein de chaque région la partie (une sous-région) où l'on y trouve effectivement les vecteurs, et de mémoriser cette sous-région plutôt que la région elle-même (qui est bien plus vaste que la sous-région identifiée). Le LSD^h-Tree englobe les vecteurs appartenant aux régions gérées par le LSD-

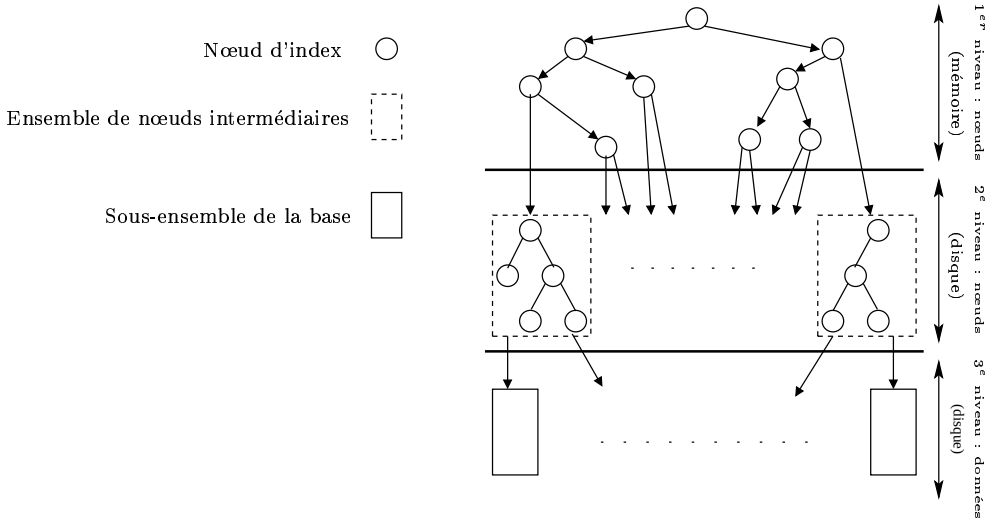


FIG. 1.9: Structure en trois niveaux du LSD-Tree.

Tree dans des hyper-rectangles puis les code à l'aide de chaînes binaires de petites tailles. Les régions codées sont ainsi des hyper-rectangles englobants mais non minimaux.

La figure 1.10 illustre le principe de ce codage dans un espace de dimension deux. Dans ce cas, chaque dimension est codée sur 3 bits.

De manière générale, étant donné que 2 valeurs selon chaque dimension sont nécessaires pour définir une région, $m \times 2 \times d$ bits sont utilisés pour coder chaque région dans un espace de dimension d où chaque dimension est codée sur m bits. Le codage est d'autant plus précis que le nombre de bits de codage par dimension est important. Ce nombre est un paramètre fixé par l'utilisateur.

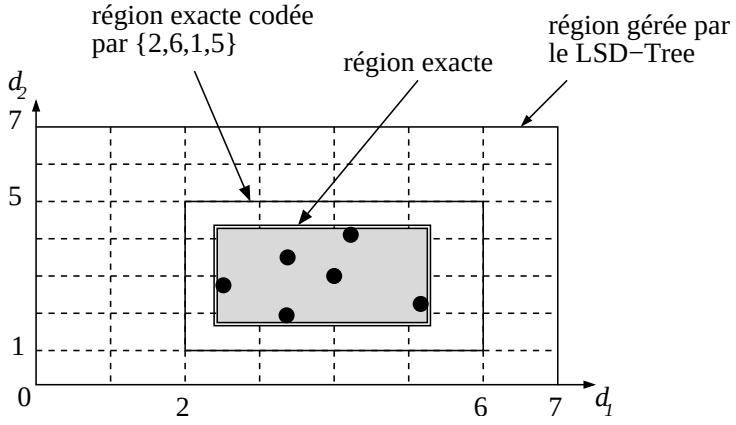


FIG. 1.10: Principe du codage des régions dans le LSD^{h}-Tree.

1.6 Synthèse

Cette section présente une synthèse des propriétés des techniques d'indexation traditionnelles pour la recherche de plus proches voisins présentées dans ce chapitre. Les tableaux 1.1 et 1.2 en résument les principales propriétés.

Plusieurs travaux ont montré que ces techniques effectuent des recherches de plus proches voisins de manière efficace dans les espaces de petite dimension ($<$16 d'après [Weber *et al.*, 1998]), leurs performances se dégradent cependant très rapidement lorsque la dimension des données croît. Seule la recherche séquentielle garde un coût linéaire lorsque l'on augmente la dimension, la complexité des techniques d'indexation devenant exponentielle avec la dimension. Malheureusement, le coût de la recherche séquentielle, bien que linéaire, est clairement prohibitif lorsque l'on gère de très grandes bases, ce qui rend ce mode de recherche non viable pour des applications réelles.

Dans le but de comprendre ce problème lié à la dimension, nous étudierons de manière détaillée les particularités des espaces de grande dimension dans le chapitre suivant.

Index	Forme des régions	Chevauchement	Complexité	Équilibre
R*-Tree	hyper-rectangle	oui	calculable	oui
X-Tree	hyper-rectangle	oui	non calculable	non
SS-Tree	hypersphère	oui	calculable	oui
SR-Tree	hyper-rectangle $\cap$ hypersphère	oui	calculable	oui
TV-Tree	L_p-sphère	oui	non calculable	non
M-Tree	M-sphère	oui	calculable	oui
K-D-B-Tree	hyper-rectangle	non	non calculable	oui
LSD^h-Tree	hyper-rectangle	non	non calculable	non

TAB. 1.1: Récapitulatif de quelques propriétés générales des index multidimensionnels cités.

Index	Points forts	Points faibles
R^*-Tree	formes englobantes permettant d'affiner les règles de filtrage (distance MINMAXDIST)	taille importante des entrées d'index ; fort taux de chevauchement ; éclatement problématique des nœuds
X-Tree	limite le chevauchement entre régions	paramètre de construction difficile à fixer : seuil max de chevauchement
SS-Tree	petite taille des entrées d'index (arbre compact)	chevauchement important
SR-Tree	formes englobantes mieux adaptées aux espaces de grande dimension	complexité des formes : opérations de recherche et m-à-j coûteuses ; taille importante des entrées d'index
TV-Tree	utilisation des techniques de réduction de la dimension pour réduire la taille de l'arbre	efficacité dépendante de la distribution et de la nature des données ; problème de gestion des BDs dynamiques
M-Tree	indépendance à la fonction de distance, réduction du coût CPU par l'utilisation de l'inégalité triangulaire	construction de l'arbre complexe
K-D-B-Tree	pas de chevauchement	fractionnements récursifs coûteux et arbitraires ; faible taux d'utilisation de l'espace alloué
LSD^h-Tree	codage binaire des régions de données $\Rightarrow$ réduire la taille de l'arbre et éviter de gérer les zones vides de l'espace	organisation mémoire/disque complexe

TAB. 1.2: Récapitulatif des points forts et des points faibles des index multidimensionnels cités.

Chapitre 2

Indexation et malédiction de la dimension

2.1 Introduction

Les espaces de grande dimension possèdent des propriétés mathématiques particulières qui affectent le comportement des méthodes manipulant des données dans ces espaces, méthodes comme la recherche de plus proches voisins ou l'indexation multidimensionnelle. Ce problème est connu sous le nom de « malédiction de la dimension » *(dimensionality curse)*. Il fait référence aux difficultés de la gestion et du traitement des données qui apparaissent dans les espaces de grande dimension. Dans le cas de l'indexation multidimensionnelle, cela se manifeste par une évolution exponentielle du temps de réponse des algorithmes de recherche de plus proches voisins lorsque la dimension augmente.

Dans ce chapitre, nous analyserons finement le problème de la malédiction de la dimension dans un cadre plus général que celui de la recherche de plus proches voisins. Nous donnerons l'origine de cette expression et son interprétation dans différents domaines d'application. Quelques propriétés surprenantes des espaces de grande dimension seront ensuite présentées, le but étant de montrer qu'il est très difficile d'étendre les intuitions que l'on a dans les espaces à deux ou trois dimensions, aux espaces de grande dimension. Les effets de certaines de ces propriétés sur les index multidimensionnels seront discutés. Nous approfondirons ensuite cette étude par une discussion autour de la notion de *dimension* des données et des méthodes de réduction de la dimension. Dans de nombreux travaux, celles-ci sont présentées comme l'ultime solution permettant de pallier les difficultés liées aux grandes dimensions. L'intérêt de ces méthodes est toutefois sujet à discussion.

2.2 Origine

Selon [Donoho, 2000], l'expression « malédiction de la dimension » a été utilisée pour la première fois par Richard Bellman dans [Bellman, 1961]. Bellman a utilisé ce terme en référence à la difficulté de faire de l'optimisation par des méthodes exhaustives dans les espaces de grande dimension[1]. Optimiser une fonction à plusieurs variables par une

[1] Il propose la programmation dynamique comme solution alternative.

méthode exhaustive consiste à partitionner le domaine de chacune des variables à intervalles réguliers, ce qui permet d'instaurer une grille cartésienne dans l'espace de recherche. Chaque point d'intersection de cette grille est un optimum possible. Il s'agit alors d'évaluer la fonction de coût en chacun de ces points et de choisir l'optimum global. Seulement, le nombre d'optimums possibles, et donc d'évaluations nécessaires, croît exponentiellement avec la dimension de l'espace de recherche (le nombre de variables). Dans un cube unitaire de dimension 10, si chaque dimension est partitionnée en 10 intervalles (i.e. en utilisant un pas de partitionnement de 0,10), 10^{10} évaluations de la fonction de coût sont nécessaires pour trouver l'optimum global. En dimension 20 et toujours dans les mêmes conditions, 10^{20} évaluations sont nécessaires.

Plus précisément, si l'on souhaite optimiser une fonction à d variables par une méthode exhaustive et si cette fonction est lipschitzienne[2] alors $(1/\varepsilon)^d$ évaluations sont nécessaires pour obtenir une approximation minimale avec une erreur ε sur les variables.

Ce phénomène se produit également pour l'approximation de fonctions et pour l'intégration numérique [Donoho, 2000] :

- l'approximation d'une fonction lipschitzienne à d variables nécessite $(1/\varepsilon)^d$ évaluations sur une grille de partitionnement (avec un pas de partitionnement de $1/\varepsilon$) afin d'obtenir un schéma d'approximation dont l'erreur d'approximation uniforme est ε ;
- l'intégration numérique d'une fonction lipschitzienne à d variables nécessite $(1/\varepsilon)^d$ évaluations sur une grille de partitionnement (avec un pas de partitionnement de $1/\varepsilon$) afin d'obtenir un schéma d'intégration dont l'erreur est ε.

L'expression « malédiction de la dimension » est aussi utilisée en statistique. Elle fait référence à la relation entre la taille de l'échantillon de données et la précision de l'estimation. Dans [Donoho, 2000], il a été montré que le nombre d'échantillons doit augmenter exponentiellement avec la dimension pour garder le même niveau de précision de l'estimation.

On retrouve également l'expression « malédiction de la dimension » dans le domaine de la recherche de similarité (recherche de plus proches voisins, indexation multidimensionnelle, etc.). Plusieurs travaux se sont intéressés à ce problème et ont montré que les performances des techniques de recherche de plus proches voisins se dégradent exponentiellement lorsque la dimension des données augmente. Au delà de 16 selon [Weber *et al.*, 1998], les performances de toutes les techniques traditionnelles pour l'indexation multidimensionnelle deviennent plus mauvaises que la recherche séquentielle et exhaustive.

Parmi les auteurs qui ont étudié ce problème, Beyer *et al.* présentent dans [Beyer *et al.*, 1999] l'un des résultats les plus intéressants. Ils montrent que sous certaines conditions sur la distribution des données, plus la dimension des données augmente, plus les vecteurs ont tendance à devenir équidistants. Cette équidistance des vecteurs rend difficile toute mise en cellules des données et engendre une forte instabilité des résultats (les plus proches voisins de deux vecteurs requêtes très proches peuvent être complètement différents). Le théorème de Beyer *et al.* est le suivant.

[2] Une application h est dite lipschitzienne s'il existe $K \in [0, +\infty[$ tel que $dist(h(x), h(x')) \leqslant K.dist(x, x')$.

Théorème 1 *Soient x_d et q_d deux variables aléatoires de dimension d et de densité de probabilité F_{X_d} et F_{Q_d} respectivement. Si*

$$\lim_{d \to +\infty} \frac{var(dist(x_d - q_d))}{E[dist(x_d - q_d)]^2} = 0 \tag{2.1}$$

alors, quel que soit $\varepsilon > 0$,

$$\lim_{d \to +\infty} P[\mathrm{DMIN}_d \leqslant (1+\varepsilon)\,\mathrm{DMAX}_d] = 1, \tag{2.2}$$

avec :

dist : fonction à deux variables à valeurs dans $\mathbb{R}_+$, une distance par exemple ;
$\mathrm{DMAX}_d = min\{dist(x_d - q_d)\}$;
$\mathrm{DMIN}_d = max\{dist(x_d - q_d)\}$.

Ce théorème montre que tous les vecteurs convergent vers la même distance par rapport au vecteur requête quand la dimension d tend vers l'infini si la distribution de distances inter-vecteurs satisfait la condition (2.1). Ce résultat soulève deux questions importantes : (i) pour quel type de distribution la condition (2.1) est-elle vérifiée ? et (ii), dans le cas où cette condition l'est, à partir de quelle dimension les distances entre les vecteurs de données et le vecteur requête ne sont plus significatives ?

Pour répondre à la première question, Beyer *et al.* ont étudié quelques distributions particulières et ont montré que la condition (2.1) est vérifiée notamment dans le cas d'une distribution de données uniforme et indépendante selon toutes les dimensions et une distribution des requêtes indépendante de celle des données.

Pour la deuxième question, étant donnée qu'une solution analytique est difficile à proposer, seule une étude expérimentale du rapport entre la distance au plus lointain voisin et la distance au plus proche voisin en fonction de la dimension est possible. D'après les expériences menées dans [Beyer *et al.*, 1999] sur des données uniformes, ce rapport vaut 10^7 pour des données de dimension 1, 6 pour des données de dimension 10 et 4 pour des données de dimension 20.

Il est ainsi primordial de s'assurer de l'existence d'une réelle similarité entre les vecteurs de la base avant d'envisager leur indexation. Dans le cas où la condition (2.1) est vérifiée, non seulement aucune stratégie de recherche ne peut être efficace mais, en plus, l'application qui exploite les données ne peut garantir la pertinence du résultat retourné. Toutefois, ce résultat n'est pas général dans le sens où les jeux de données réels sont généralement loin d'être conformes aux hypothèses de distribution nécessaires pour l'application du théorème.

2.3 Quelques propriétés surprenantes des espaces de grande dimension

Les propriétés des espaces de grande dimension ainsi que la distribution des données dans ces espaces sont très différentes de ce que l'on peut imaginer dans les espaces à une, deux ou trois dimensions. La principale difficulté heurtant la compréhension de ces

propriétés est notre incapacité à visualiser des données de dimension supérieure à trois[3]. Ces propriétés particulières sont résumées dans les points suivants. Nous donnerons pour certaines d'entre-elles leurs effets sur les index multidimensionnels.

Partitionnement exponentiel de l'espace. Le nombre de cellules résultant d'un partitionnement de l'espace croît exponentiellement avec le nombre de dimensions. Par exemple, dans le cas où chaque dimension d'un espace de dimension d est fractionnée en 2, le nombre de cellules formées dans cet espace est 2^d. Lorsque d est grand, le nombre de cellules devient tellement important qu'il y a de fortes chances pour qu'il dépasse de beaucoup le nombre de vecteurs de la base. L'énumération de toutes ces cellules (dont la vaste majorité ne contient aucun vecteur) pose de nombreux problèmes de gestion au niveau de l'index et ne permet pas d'avoir plusieurs vecteurs par cellule, ce qui remet en cause le principe fondamental des index.

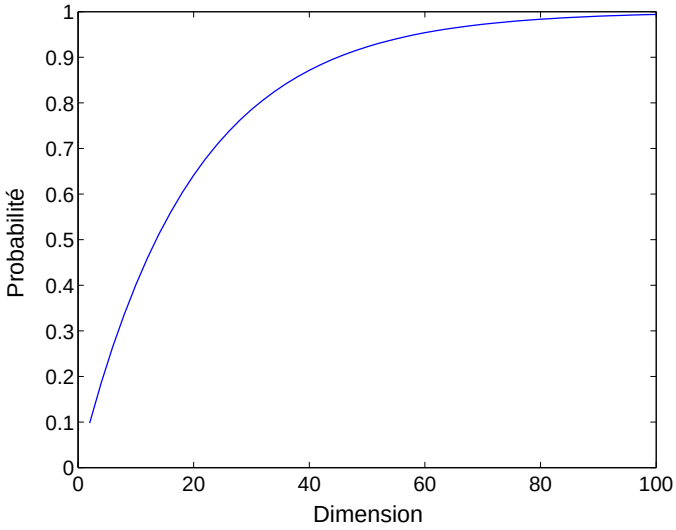


FIG. 2.1: La probabilité pour qu'un vecteur soit à une distance inférieure ou égale à $0,025$ d'une frontière entre cellules en fonction de la dimension de l'espace.

Frontières et recherches. À l'issue d'un partitionnement, et malgré le nombre important de cellules formées, chaque cellule possède un très grand nombre de cellules voisines. Cela pose de sérieux problèmes pour les procédures de recherche. En effet, dans le cas où un vecteur requête se trouve suffisamment près d'une frontière entre cellules, le processus de recherche de plus proches voisins doit nécessairement examiner la ou les cellules voisines. Or, en grande dimension, le nombre de cellules augmente exponentiellement et la probabilité d'être près d'une frontière augmente elle aussi. Si on considère un ensemble de vecteurs uniformément distribués dans l'espace $[0,1]^d$ dans lequel chaque dimension a été partitionnée en deux, alors la probabilité pour qu'un vecteur se trouve à une distance inférieure à δ d'une frontière est

[3]Même si certains travaux proposent des techniques de visualisation de données de dimension supérieure à 3 en utilisant la couleur [Veillon, 1984] ou par la technique dite des coordonnées parallèles [Inselberg, 1985, Inselberg & Dimsdale, 1990], les résultats obtenus sont peu convaincants et assez limités.

$1 - (1 - 2.\delta)^d$. On remarque que cette probabilité tend vers 1 quand la dimension tend vers l'infini. En particulier, pour un δ égale à $0,025$, la courbe de la figure 2.1 montre l'évolution de cette probabilité en fonction du nombre de dimensions. Au delà de 60 dimensions, presque tous les vecteurs de la base se trouvent près d'une frontière entre cellules. Ainsi, dans un espace de très grande dimension, même s'il est possible d'éliminer de très nombreuses cellules, la proximité de frontières entraîne la quasi-certitude de devoir accéder aux cellules voisines, qui, cumulée à l'existence de très nombreux voisins, tend à augmenter fortement le coût des recherches.

Phénomène de l'espace vide. Selon [Verleysen, 2003], ce phénomène a été décrit pour la première fois dans [Scott & Thompson, 1983]. Il fait référence au fait que plus la dimension d'un espace est grande, plus celui-ci *semble* vide. Pour illustrer ce phénomène, considérons une hypersphère de rayon r et placée dans un espace de dimension d. Son volume vaut :

$$V_d(r) = \frac{\pi^{d/2}\, r^d}{\Gamma(1 + d/2)}, \tag{2.3}$$

où

$$\Gamma(x) = \int_0^{+\infty} t^{x-1} e^t dt.$$

Cette fonction peut aussi être évaluée récursivement comme suit :

$$\begin{cases} \Gamma(0) & = 1, \\ \Gamma(1) & = 1, \\ \Gamma(\frac{1}{2}) & = \sqrt{\pi}, \\ \Gamma(x+1) & = x\Gamma(x). \end{cases}$$

Si l'on cherche à quantifier l'espace sphérique avec un pas de quantification de $r/10$ (c'est-à-dire partitionner chaque dimension à intervalles réguliers de longueur $r/10$ et compter le nombre de points d'intersection de la grille de partitionnement à l'intérieur de la sphère), alors le nombre N de points augmente en fonction de la dimension d comme suit[4] :

d	1	2	3	4	5	6
N	20	314	4188	49 348	562 379	$> 5\ 10^6$

Cet exemple montre que, pour garder le même rapport « nombre de vecteurs/volume de la sphère » en augmentant la dimension de l'espace, il faut augmenter de manière exponentielle le nombre de vecteurs. Dans le cas des index multidimensionnels, ce phénomène rend difficile la mise en paquets des vecteurs de la base car leur nombre n'est pas suffisamment élevé par rapport à la dimension (qui est souvent supérieure à 20 et peut prendre des valeurs allant jusqu'à plusieurs centaines). La distribution des données dans l'espace est par conséquent éparse, ce qui diminue les performances des index multidimensionnels.

[4]Exemple tiré de la thèse de Pierre Demartines [Demartines, 1994].

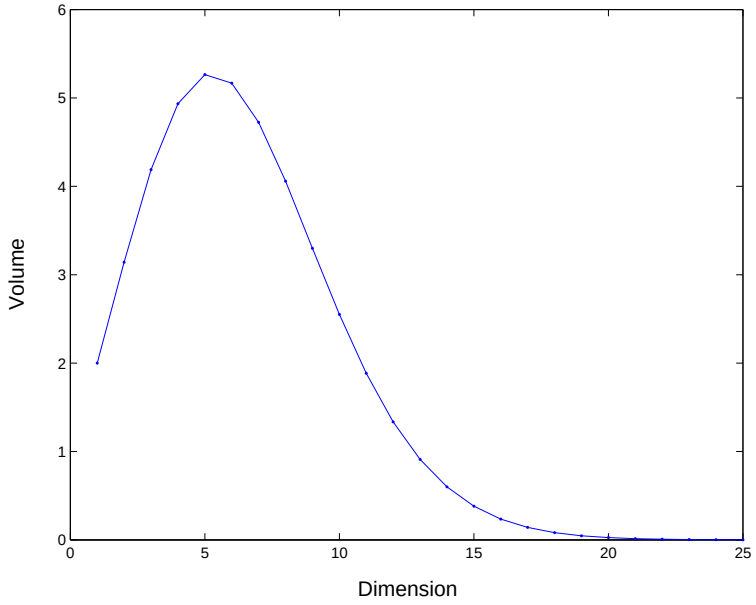


FIG. 2.2: Évolution du volume d'une hypersphère de rayon unité en fonction de la dimension de l'espace.

Paradoxalement, si l'on étudie l'évolution du volume de la sphère unité ($r = 1$) en fonction de la dimension de l'espace, il en résulte que le volume d'une sphère augmente jusqu'à la dimension 5 puis décroît pour tendre vers 0 quand la dimension tend vers l'infini (voir la courbe de la figure 2.2).

Normes des vecteurs aléatoires. Il a été prouvé dans [Demartines, 1994] que l'écart-type de la norme de vecteurs aléatoires converge vers une constante quand la dimension augmente. Plus précisément :

Théorème 2 *Soit x un vecteur de dimension d $[x_1, \ldots, x_d]^T$ dont les composantes sont aléatoires, indépendantes et de même loi*[5]*. On a :*

$$\mu_{||x||} = E(||x||) = \sqrt{ad - b} + O(1/d)$$
$$\sigma^2_{||x||} = var(||x||) = b + O(1/d)$$

où a et b sont des paramètres qui dépendent uniquement des moments centrés d'ordre 1, 2, 3 et 4 des x_i.

Ce théorème signifie que, à partir d'un certain nombre de composantes indépendantes, les vecteurs x semblent normalisés. Quelle que soit la distribution des x_i, l'écart-type de la norme $\sigma_{||x||}$ tend vers une constante lorsque l'on augmente d, tandis que la moyenne $\mu_{||x||}$ croît en $\sqrt{d}$.

En réalité, les implications de ce résultat rejoignent exactement les conclusions du théorème de Beyer *et al.* (Théorème 1) concernant le problème de l'équidistance des vecteurs en grande dimension. En effet, comme la distance euclidienne est la norme

[5] Et possédant un moment d'ordre 8 fini.

de la différence entre deux vecteurs et comme la différence entre deux vecteurs aléatoires est aussi un vecteur aléatoire, alors en appliquant le théorème 2 sur les distances euclidiennes, il résulte que leurs valeurs convergent vers une constante quand la dimension augmente. L'impact de ce résultat sur les index multidimensionnels et sur la recherche de similarité de manière générale est le même que celui évoqué pour le théorème de Beyer *et al.*

Formes géométriques usuelles en grande dimension. Les formes géométriques les plus utilisées par les index multidimensionnels pour englober les paquets de données sont les sphères, les cubes et les rectangles. Ces formes sont choisies pour leur simplicité. Seulement, en grande dimension, leurs volumes évoluent différemment de ce que l'on peut imaginer en deux ou trois dimensions. Nous avons vu dans la section précédente que le volume d'une hypersphère de rayon 1 s'approche de 0 quand la dimension augmente. Si l'on considère, par contre, un hypercube de longueur de côté égale à 1, son volume vaut 1 quelle que soit la dimension. Jusqu'à la dimension 5, le rapport du volume de l'hypersphère de rayon 1 par rapport à celui de l'hypercube de longueur de côté égale à 1 reste conforme à l'intuition que l'on a sur les cubes et les sphères. Au delà, il est extrêmement difficile de se faire une idée intuitive de cette évolution. Par conséquent, les motivations du choix d'une forme par rapport à une autre ne peuvent plus être fondées sur les propriétés géométriques connues en petite dimension.

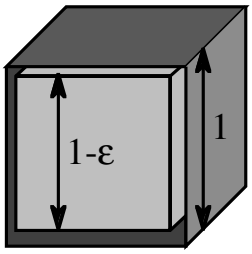


FIG. 2.3: La bordure d'un cube.

Un autre phénomène intéressant à étudier est celui des bordures. La bordure d'un hypercube de longueur de côté égale à 1 est la zone extérieure située entre un sous-hypercube de longueur de côté égale à $1-\varepsilon$ et l'hypercube lui-même (voir figure 2.3). Le volume de cette zone vaut $1-(1-\varepsilon)^d$. La figure 2.4 montre son évolution en fonction de la dimension pour deux valeurs de ε (0,02 et 0,05). Il apparaît clairement que le volume de cette zone est très petit par rapport au volume global de l'hypercube dans les espaces de petite dimension (pour $\varepsilon = 0,02$, ce volume vaut 0,06 en dimension 3). En revanche, en augmentant la dimension, ce volume devient de plus en plus grand. En dimension 60, il vaut 0,70 pour $\varepsilon = 0,02$ et 0,95 pour $\varepsilon = 0,05$. Il tend vers 1 quand la dimension tend vers l'infini. Cela signifie que la majeure partie du volume d'un hypercube en grande dimension se trouve dans la bordure de celui-ci. Si l'on considère ce phénomène sous un angle différent et si l'on étudie la croissance du volume d'un hypercube lorsque l'on augmente sa longueur de côté (même avec

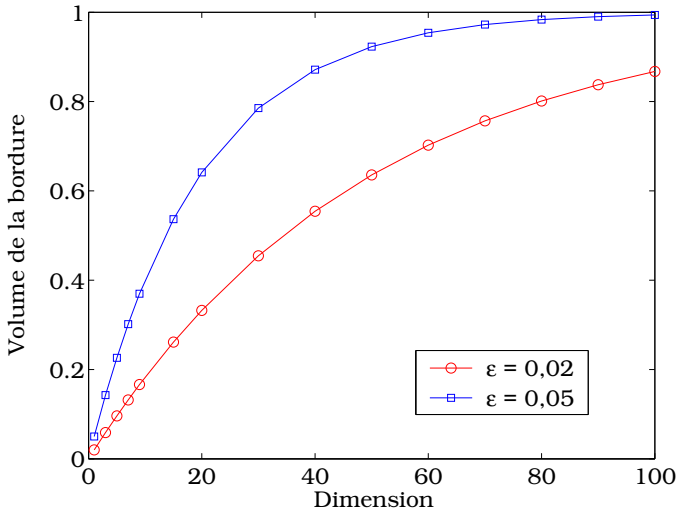


FIG. 2.4: Évolution du volume de la bordure d'un hypercube en fonction de la dimension de l'espace.

des valeurs très petites), nous remarquons que la croissance de son volume est exponentielle. La figure 2.5 le montre clairement.

Le même phénomène se produit dans le cas de l'hypersphère. Le rapport entre le volume de la bordure et le volume de l'hypersphère vaut $1 - (1 - \varepsilon)^d$. La figure 2.6 montre l'évolution de ce rapport en fonction de la dimension pour deux valeurs de ε (0,02 et 0,05).

L'impact de ce phénomène sur les index multidimensionnels concerne principalement la mise en paquets des données. L'affectation d'un vecteur à un paquet englobé dans une hypersphère (ou un hypercube) engendre un agrandissement considérable du volume de celle-ci même si le vecteur à ajouter semble *a priori* proche du paquet. Cette augmentation des volumes des hypersphères engendre un taux de chevauchement important entre cellules et rend les règles de filtrage utilisées par les algorithmes de recherche inefficaces.

2.4 La dimension intrinsèque des données

Jusqu'à présent, nous avons considéré que la dimension des données correspondait au nombre de composantes qui constituent les vecteurs. Cette dimension est appelée dimension *brute* par opposition à la dimension *intrinsèque* ou encore *nombre de degrés de liberté*. Si la dimension brute est simple à définir et facile à déterminer, la dimension intrinsèque l'est beaucoup moins car elle dépend de la distribution des données.

La dimension intrinsèque d'un ensemble de données est la dimension du plus petit sous-espace (espace de la plus petite dimension) qui permet de représenter l'ensemble de données tout en conservant la même quantité d'information que celle portée par les données dans leur espace original. La quantité d'information signifie dans ce contexte la proximité relative entre vecteurs, c'est-à-dire les distances inter-vecteurs.

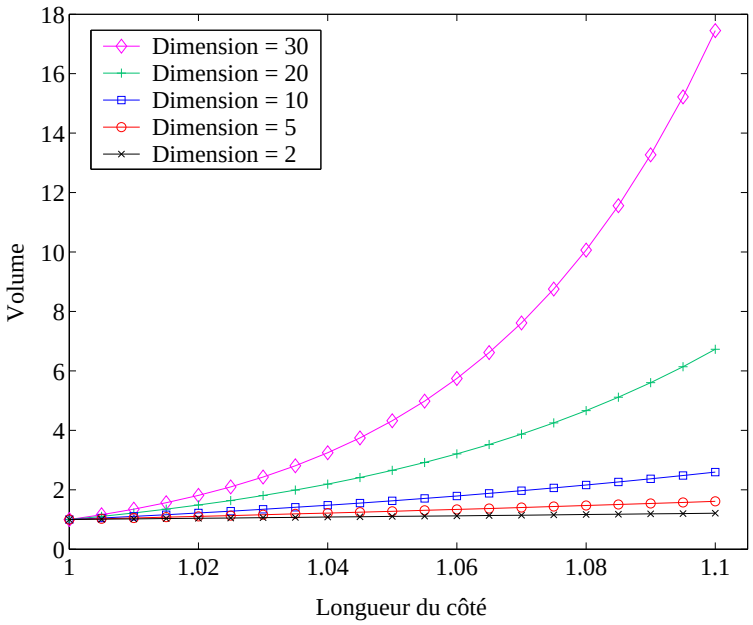


FIG. 2.5: Évolution du volume d'un hypercube en fonction de la longueur de son côté.

Les vecteurs de données représentés par leurs dimensions brutes correspondent aux données collectées sans aucun pré-traitement. Les composantes de ces vecteurs sont généralement corrélées. Il arrive souvent de représenter des données dans des espaces de dimensions plus grandes que leurs dimensions intrinsèques. Ceci est généralement dû à une connaissance insuffisante de la relation inter-attributs lors du calcul des vecteurs d'attributs ou lors de la collecte des données. On décide souvent de garder un nombre important d'attributs pour avoir le maximum d'information au risque d'avoir plusieurs composantes qui expriment le même type d'information.

À titre d'exemple, la dimension intrinsèque d'un ensemble de vecteurs tirés aléatoirement sur une droite dans un espace de dimension 2 est égale à 1. L'une des deux composantes des vecteurs peut être déterminée à partir de l'autre. Elle est donc superflue.

Partant de cette définition, deux questions importantes se posent.

1. Comment déterminer la valeur de la dimension intrinsèque d'un ensemble de données ?
2. Comment transformer les données dans l'espace correspondant à la dimension intrinsèque ?

Ces deux questions ne peuvent être traitées séparément. À la méthode qui permet de transformer les données dans un nouvel espace (celui correspondant à la dimension intrinsèque) est toujours associée une méthode qui permet de déterminer soit *a priori*, soit *a posteriori* la dimension intrinsèque.

La transformation des données dans le nouvel espace s'effectue par des méthodes de réduction de la dimension. Un panorama de ces méthodes est donné dans la section suivante. Elles sont classées en deux grandes classes. Pour chacune, la méthode qui permet de déterminer la dimension de l'espace transformé est donnée. Mais auparavant, il est im-

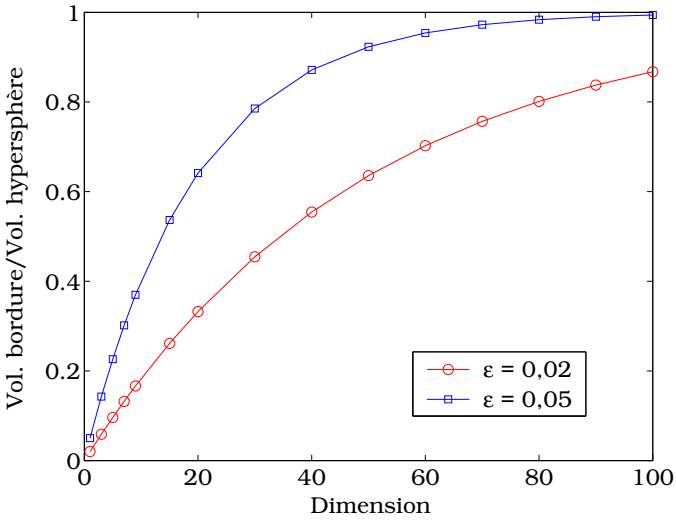


FIG. 2.6: Évolution du rapport entre le volume de la bordure d'une hypersphère de rayon 1 et son volume global en fonction de la dimension de l'espace.

portant de noter que plusieurs travaux en analyse de données se sont intéressés à la notion de dimension intrinsèque indépendamment de la question de la réduction de la dimension.

En algèbre matricielle, la notion de dimension correspond au rang de la matrice des échantillons (vecteurs de données représentés sous forme matricielle). Cette définition peut être utilisée dans l'exemple de la droite dans un espace de dimension 2 et permet de déduire la dimension intrinsèque (1 en l'occurrence). Cependant, cette définition ne peut être utilisée en pratique car les données sont souvent bruitées. Un léger décalage d'un point de la droite fait augmenter le rang de la matrice des échantillons de 1.

Une autre définition très répandue dans la littérature abordant la notion de dimension est celle de la *dimension fractale*. Cette notion est souvent présentée comme la vraie dimension des données car, outre la prise en compte des corrélations entre dimensions, elle prend aussi en compte l'*auto-similarité* des données : deux vues à des échelles de grossissement différentes sont similaires [Korn *et al.*, 2001].

Pour illustrer cette notion, considérons le procédé présenté dans [Demartines, 1994] qui se base sur les propriétés géométriques de la distribution des données pour déterminer leur dimension intrinsèque.

En reprenant la formule 2.3, le volume d'une hypersphère de rayon r est de la forme :

$$V_d = a_d \, r^d \tag{2.4}$$

En distribution uniforme (où le nombre de points est proportionnel au volume), le nombre de points N à l'intérieur d'une sphère de rayon r est proportionnel à r^d. Si la distribution des données n'a localement que p degrés de liberté, c'est à dire que la dimension intrinsèque est de p, le nombre N sera proportionnel à r^p. En effet, même dans un espace de très grande dimension, si la distribution se trouve dans un plan, les points que l'on compte sont situés à l'intersection entre l'hypersphère et le plan, c'est-à-dire sur un disque. Le nombre de points dénombrés est alors proportionnel à r^2.

Plus généralement, comme $N = a_p\ r^p$, on a :

$$log[N(r)] = log(a_p) + p\, log(r) \tag{2.5}$$

Il est donc facile de déduire la dimension intrinsèque p en effectuant une simple régression linéaire sur plusieurs valeurs de r et de $N(r)$.

Ce procédé permet de déduire localement autour d'un point (le centre de l'hypersphère) la dimension intrinsèque du nuage de données. Il est à noter, cependant, que les valeurs prises par cette dimension ne sont pas forcément entières.

Il existe plusieurs définitions de cette dimension fractale, qui se rapprochent plus ou moins de ce procédé empirique et qui sont liées entre elles [Hentschel & Procaccia, 1983]. Ces dimensions sont toutes basées sur l'étude de l'évolution, en fonction de l'échelle, d'une propriété mesurable d'un nuage de points. Nous présentons dans les quatre paragraphes suivants trois définitions différentes de la dimension fractale ainsi que la définition de la dimension fractale généralisée.

Prenons par exemple un pavage de l'espace $\mathbb{R}^d$ en cubes à d dimensions et de côté r. Si $N(r)$ est le nombre de cubes qui contiennent au moins un point, la *« dimension de similarité »* est alors définie par :

$$d_0 = \lim_{r \to 0} \frac{log\ N(r)}{log\ (1/r)}. \tag{2.6}$$

Une autre définition prend en compte l'évolution de la quantité d'information selon Shannon lorsqu'on quantifie l'espace par les cubes. C'est la *« dimension d'information »* :

$$d_1 = \lim_{r \to 0} \frac{I(r)}{log\ (1/r)}, \tag{2.7}$$

avec $I(r) = -\sum_{i=1}^{N(r)} p_i(r)\, log\, p_i(r)$ et $p_i(r)$ la probabilité du cube i (le nombre de points qu'il contient divisé par le nombre total de points).

Enfin, une troisième mesure est basée sur le nombre $C(r)$ de couples de points dont l'écart est inférieur à r. C'est la *« dimension de corrélation »* :

$$d_2 = \lim_{r \to 0} \frac{log\, C(r)}{log(r)}. \tag{2.8}$$

En fait, toutes ces mesures de dimensions sont des cas particuliers de la *« dimension fractale généralisée »* [Hentschel & Procaccia, 1983] :

$$d_q = \frac{1}{q-1} \lim_{r \to 0} \frac{log \sum_{i=1}^{N(r)} p_i(r)^q}{log(r)}. \tag{2.9}$$

En pratique, comme les distributions que l'on étudie possèdent un nombre fini d'échantillons, ces mesures tendent toutes vers 0 quand r tend vers 0. Il faut donc choisir convenablement la valeur de r. Celle-ci définit l'échelle d'estimation. Une trop petite valeur se concentre uniquement sur les points isolés (le bruit) tandis qu'une trop grande valeur « laisse passer » la structure locale de la distribution [Demartines, 1994].

Le problème est donc de déterminer les valeurs de r autour desquelles la dimension fractale correspond à la dimension intrinsèque. Ce problème limite l'efficacité de ces mesures pour l'estimation de la dimension intrinsèque. Il peut être illustré par l'exemple de Mandelbrot de la pelote de ficelle [Mandelbrot, 1984, Mandelbrot, 1982]. Vue de loin, la pelote ressemble à un point, sa dimension est donc 0. En se rapprochant, sa dimension passe à 3 puisqu'on commence à percevoir sa nature volumique. Lorsque l'on s'en approche de très près, on distingue alors un seul fil tordu et enroulé. La pelote est donc constituée d'une ligne tordue et est, dès lors, unidimensionnelle. En se rapprochant encore davantage, cette ligne se transforme en une colonne d'épaisseur finie et la ficelle devient tridimensionnelle. Plus près encore, on ne distingue plus que de fins fils individuels qui se tordent les uns autour des autres pour former la ficelle ; la pelote est redevenue unidimensionnelle.

En plus de ce problème de l'échelle d'estimation, il est important de noter qu'aucun lien mathématique n'a été établi entre la dimension fractale et la dimension intrinsèque d'un ensemble de données. Plusieurs travaux considèrent, tout de même, que la dimension fractale est une très bonne approximation de la dimension intrinsèque [Traina Jr. *et al.*, 2000, de Sousa *et al.*, 2002].

Nous terminons cette section par un exemple simple et illustratif de la procédure d'estimation de la dimension fractale[6]. Nous considérons une ligne courbe en dimension deux et nous estimons sa dimension fractale de similarité d_0 (équation 2.6). L'exemple est illustré par la figure 2.7. En considérant 4 valeurs différentes de r, nous avons pu tracer la courbe de variation de $log(N)$ en fonction de $log(1/r)$. Une régression linéaire sur l'ensemble des points résultants a permis de déduire la dimension fractale de similarité qui est, dans ce cas, égale à 1.

2.5 Techniques de réduction de la dimension

Plusieurs méthodes pour la réduction de la dimension ont été développées dans le domaine de la statistique et de l'analyse de données. Comme nous l'avons introduit précédemment, ces méthodes permettent de transformer les données dans un nouvel espace de dimension réduite en gardant le maximum d'information portée par les données dans leur espace original. Elles profitent pour cela de la corrélation et/ou de la dépendance entre les dimensions originales.

Mathématiquement, le problème que visent à résoudre ces méthodes peut se formuler de la manière suivante : étant donnée une variable aléatoire d-dimensionnelle $\mathbf{x} = (x_1, \ldots, x_d)^T$, il s'agit de trouver une autre représentation de dimension réduite $\mathbf{s} = (s_1, \ldots, s_p)^T$ avec $p < d$, qui exprime la même information que les données originales selon un certain critère. Les composantes de $\mathbf{s}$ sont appelées composantes cachées, variables latentes ou dimensions intrinsèques. Les composantes des vecteurs $\mathbf{x}$ (et aussi celles des vecteurs $\mathbf{s}$) sont appelées variables en statistique, attributs en informatique et en intelligence artificielle ou tout simplement dimensions. Dans ce chapitre, nous appelons espace original l'espace de dimension d de définition des données originales ($\mathbf{x}$), et,

[6] Exemple tiré de la thèse de Pierre Demartines [Demartines, 1994].

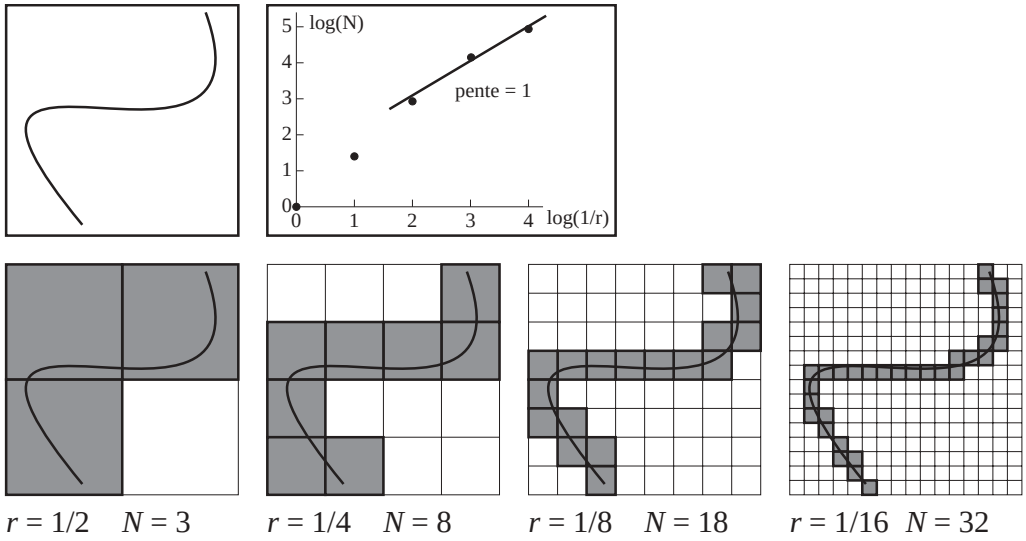


FIG. 2.7: Exemple de calcul de la dimension fractale de similarité d'une courbe représentée dans un espace de dimension (brute) 2.

espace transformé ou espace de projection l'espace de dimension réduite dans lequel seront projetés les $\mathbf{x}$.

Les méthodes de réduction de la dimension sont présentées par plusieurs travaux comme une solution palliative aux problèmes dûs aux espaces de grande dimension [Wu *et al.*, 2000, Chakrabarti & Mehrotra, 2000]. Ces travaux suggèrent d'effectuer d'abord un pré-traitement des données dans le but de réduire leur dimension initiale et d'éviter la malédiction de la dimension. Les données sont ensuite indexées par une technique traditionnelle dans l'espace transformé qui est de dimension réduite.

Dans cette section, nous présentons les méthodes de réduction de la dimension les plus citées dans le domaine de l'indexation multidimensionnelle et la recherche de plus proches voisins. Nous classons ces méthodes en deux grandes catégories : les méthodes linéaires et les méthodes non linéaires.

Les méthodes linéaires permettent de projeter linéairement les vecteurs dans un nouvel espace de même dimension que l'espace original. Elles associent ensuite à chaque axe du nouvel espace un indice sur la quantité d'information portée par la composante associée. La dimension est enfin réduite en éliminant les composantes qui portent *peu d'information*. La dimension du nouvel espace de représentation des données peut donc être fixée *a posteriori*.

Les méthodes non linéaires sont des méthodes itératives qui permettent de trouver un nouvel espace de représentation, soit en utilisant des projections non linéaires, soit en recherchant un nouvel espace de représentation en prenant pour seul critère la sauvegarde des distances inter-vecteurs dans le nouvel espace. Ces méthodes requièrent le choix de la valeur de la dimension réduite *a priori*.

Dans ce qui suit, nous considérons n vecteurs d'attributs de dimension d. Chacun est la réalisation d'une variable aléatoire $\mathbf{x} = (x_1, \ldots, x_d)^T$ de moyenne

$E(\mathbf{x}) = \mu = (\mu_1, \ldots, \mu_d)^T$ et de matrice de covariance $E(\mathbf{x} - \mu)(\mathbf{x} - \mu)^T = \Sigma_{d \times d}$. Les n vecteurs sont stockés dans la matrice $\mathbf{X} = \{x_{ij} : 1 \leqslant i \leqslant d, 1 \leqslant j \leqslant n\}$.

2.5.1 Techniques linéaires

Les techniques linéaires pour la réduction de la dimension permettent de transformer les données originales ($\mathbf{x}$) dans un espace de dimension plus réduite tel que la nouvelle représentation des données ($\mathbf{s}$) soit une combinaison linéaire des données originales, c'est à dire :

$$s_i = w_{1,i}x_1 + \cdots + w_{d,i}x_d, \quad \text{pour} \quad i = 1, \ldots, p, \tag{2.10}$$

$$\text{ou bien,} \quad \mathbf{s} = \mathbf{W}\mathbf{x}, \tag{2.11}$$

avec $\mathbf{W}$ est une matrice $p \times d$ de transformation linéaire. La transformée inverse est :

$$\mathbf{x} = \mathbf{A}\mathbf{s}, \tag{2.12}$$

avec $\mathbf{A}$ la matrice $d \times p$ de transformation inverse.

Toutes les techniques linéaires reposent sur ce même principe. Elles se distinguent par la manière dont sont déterminées les matrices de transformation ($\mathbf{W}$ et $\mathbf{A}$). Dans cette section, nous passerons en revue deux méthodes : l'analyse en composantes principales et l'analyse en composantes indépendantes.[7] Il existe de nombreuses autres méthodes que nous ne citons pas ici car elles sont dédiées à des domaines bien particuliers. L'analyse en fonctions communes et spécifiques *(factorial analysis)*, par exemple, repose sur le même principe que l'analyse en composantes principales sauf qu'elle suppose que tous les attributs dépendent d'un facteur commun inconnu et souvent non mesurable. Ce modèle a été introduit par des psychologues pour modéliser un facteur d'« intelligence » commun dans le cas où les attributs sont des scores de tests effectués sur des individus.

2.5.1.1 Analyse en composantes principales

L'analyse en composantes principales (ACP) est sans doute la méthode de réduction de la dimension la plus connue et la plus utilisée. Elle trouve ses origines dans les travaux de Hotelling, Karhunen et de Loève [Hotelling, 1933, Karhunen, 1947, Loève, 1948]. C'est une méthode de second ordre car elle se base uniquement sur l'étude de la matrice de covariance des variables (les moments d'ordre 2). Il existe plusieurs variantes de l'ACP. En fonction du domaine d'application, elle est connue sous le nom de décomposition en valeurs singulières (SVD), transformée de Karhunen-Loève (KLT), transformée de Hotelling ou bien encore méthode de fonction orthogonale empirique [Fodor, 2002]. Dans [Gerbrands, 1981], une analyse approfondie de la relation entre ACP, KLT et SVD est donnée.

L'ACP consiste à chercher un nouvel espace de représentation dont les axes sont orthogonaux et assurent une dispersion maximale des données selon chacun d'eux. Ces axes sont appelés axes principaux. La quantité d'information portée par chacun des axes est relative à la variance des données : plus la variance des données selon un axe est grande, plus l'information portée par celui-ci est importante. En fait, l'ACP consiste à effectuer

[7] Nous avons repris la description donnée dans [Fodor, 2002].

une translation suivie d'une rotation du repère de l'espace. La réduction de la dimension s'effectue en éliminant les axes qui portent peu d'information.

Un nouvel attribut est ainsi associé à chacun des axes principaux. Le vecteur de taille n constitué des valeurs résultantes de la projection de l'ensemble des vecteurs originaux sur un axe principal est appelé composante principale. Celle-ci est donc une combinaison linéaire des attributs originaux.

La méthode qui permet de déterminer le nouvel espace de représentation est la suivante. La première composante principale, s_1, est la combinaison linéaire des variables originales qui possède la plus grande variance. Nous avons $s_1 = \mathbf{x}^T\mathbf{w_1}$ avec $\mathbf{w_1} = (w_{1,1}, \ldots, w_{1,d})^T$ un vecteur de coefficients tel que :

$$\mathbf{w_1} = \arg \max_{||\mathbf{w}||=1} \text{var}\{\mathbf{x}^T\mathbf{w}\}. \tag{2.13}$$

La deuxième composante principale est la combinaison linéaire avec la deuxième plus grande variance et est orthogonale à la première composante, et ainsi de suite.

Comme la variance dépend de l'ordre de grandeur des attributs, les données doivent être centrées réduites. Ceci permet d'avoir des attributs comparables. Soit Σ la matrice de covariance. En utilisant le théorème de décomposition spectrale, Σ peut s'écrire sous la forme :

$$\Sigma = \mathbf{U}\Lambda\mathbf{U^T}, \tag{2.14}$$

où $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_d)$ est une matrice diagonale des valeurs propres avec $\lambda_1 \leqslant \cdots \leqslant \lambda_d$, et $\mathbf{U}$ est un matrice (orthogonale) $d \times d$ des vecteurs propres. Ces vecteurs propres sont les vecteurs directeurs des axes principaux. Les composantes principales sont les d lignes de la matrice $\mathbf{S}$ $(d \times n)$ donnée par l'équation :

$$\mathbf{S} = \mathbf{U}^T\mathbf{X}. \tag{2.15}$$

Il s'agit d'un résultat classique dont la preuve peut être trouvée dans [Saporta, 1990]. Ce même résultat peut aussi être obtenu en utilisant un réseau de neurones [Demartines, 1994].

Par ailleurs, la variance totale du nuage de point est égale à la somme des valeurs propres de la matrice de covariance :

$$\sum_{i=1}^{d} \text{var}(\mathbf{S_i}) = \sum_{i=1}^{d} \lambda_i = \text{trace}(\Sigma). \tag{2.16}$$

Par conséquent, la fraction

$$\frac{\sum_{i=1}^{p} \lambda_i}{\text{trace}(\Sigma)} \tag{2.17}$$

donne la proportion cumulée de la variance exprimée par les p premières composantes. Cette proportion est appelée pourcentage de l'inertie expliquée par les p premières composantes. En se basant sur ce pourcentage, l'utilisateur peut ainsi choisir le nombre de composantes à retenir *a posteriori* en fonction de la quantité d'information qu'il désire conserver.

2.5.1.2 Analyse en composantes indépendantes

L'analyse en composantes indépendantes (ACI) a été introduite initialement par Jutten et Hérault [Jutten, 1987]. C'est une méthode qui se base sur les moments d'ordre supérieur pour trouver une projection des données sur des axes pas forcément orthogonaux mais selon lesquels les vecteurs de données sont statistiquement aussi indépendants que possible. L'indépendance statistique est une contrainte plus forte que la non-corrélation.

Formellement, pour que les variables aléatoires $\mathbf{x} = \{x_1, \ldots, x_p\}$ soient non-corrélées, il suffit que $\forall i \neq j,\ 1 \leqslant i, j \leqslant p$:

$$\mathrm{cov}(x_i, x_j) = \mathrm{E}\{(x_i - \mu_i)(x_j - \mu_j)\} = \mathrm{E}(x_i x_j) - \mathrm{E}(x_i)\mathrm{E}(x_j) = 0 \tag{2.18}$$

Cependant, l'indépendance statistique nécessite que la densité de probabilité conjointe soit le produit des densités de probabilité marginales, c'est-à-dire :

$$f(x_1, \ldots, x_p) = f(x_1) \ldots f(x_p). \tag{2.19}$$

L'indépendance est une notion plus générale que la non-corrélation : deux variables aléatoires indépendantes sont forcément non-corrélées mais la réciproque n'est pas toujours vraie. Dans le cas particulier où la distribution conjointe des variables est gaussienne, ces deux notions sont équivalentes. Par conséquent, les composantes principales (issues à la suite d'une ACP) sont indépendantes et l'ACI n'apporte rien de plus par rapport à l'ACP.

En réalité, l'ACI n'est pas un outil de réduction de la dimension car pour certaines variantes de l'ACI, l'espace de projection peut être de dimension plus grande que la dimension de l'espace originale [Hyvärinen, 1999b]. En pratique, l'ACI est souvent utilisée conjointement avec une ACP. Dans un premier temps, l'ACP transforme les données dans un espace de dimension réduite et décorrèle les données. L'ACI, appliquée dans l'espace réduit, permet ensuite de transformer les données dans un autre espace dans lequel les dimensions sont indépendantes.

Dans cette section, nous ne visons pas à donner une description exhaustive de l'ACI mais seulement à en donner les principes. Le lecteur intéressé pourra se référer à [Jutten, 1987, Comon, 1994, Hyvärinen, 1999b, Hyvärinen & Oja, 2000, Hastie *et al.*, 2001] pour plus de détails.

Position du problème et formalisation. L'ACI consiste à chercher une décomposition d'un vecteur aléatoire en composantes statistiquement indépendantes. Elle est souvent assimilée au problème de la séparation aveugle de sources qui en est, en fait, une application. En traitement du signal, les vecteurs d'attributs originaux (le signal observé), $\mathbf{x}$, sont considérés comme un mélange inconnu de signaux dits sources, $\mathbf{s}$, supposés indépendants. Le cas le plus simple et le plus traité définit le mélange comme une combinaison linéaire décrite par une matrice $\mathbf{A}$ appelée matrice de « mélange ». L'objectif est d'estimer conjointement la matrice $\mathbf{A}$ et le vecteur $\mathbf{s}$, soit de « séparer » les sources en déterminant une matrice de « séparation » $\mathbf{W}$.

En statistique, le problème se formule comme suit :

l'analyse en composantes indépendantes d'un vecteur aléatoire d-dimensionnel $\mathbf{x} = (x_1, \ldots, x_d)^T$ *consiste à estimer le modèle :*

$$\mathbf{x} = \mathbf{A}\mathbf{s}, \tag{2.20}$$

où les variables s_i *du vecteur* $\mathbf{s} = (s_1, \ldots, s_p)^T$ *sont supposées indépendantes et où* $\mathbf{A}$ *est une matrice* $d \times p$ *de « mélange »*[8].

Le vecteur aléatoire des sources estimées (ou reconstruites) sera noté $\mathbf{y} = \hat{\mathbf{s}}$ avec $\mathbf{y} = \mathbf{W}\mathbf{x}$. Les composantes de y_i sont aussi indépendantes que possible au sens d'une certaine mesure d'indépendance $\phi(y_1, \ldots y_d)$.

Trois conditions sont requises pour l'estimation du modèle ACI : (1) au plus une composante peut être gaussienne, (2) la matrice de mélange doit être de rang plein, et, (3) le nombre de vecteurs d'attributs doit être supérieur au nombre de composantes indépendantes à estimer (sources). Le nombre de composantes indépendantes correspond, en général, à la dimension des vecteurs d'attributs originaux.

L'estimation du modèle ACI d'un ensemble de données nécessite d'abord la définition d'une fonction « objective » puis ensuite de procéder à sa minimisation ou sa maximisation. Cette fonction est appelée fonction de contraste.

L'analyse en composantes indépendantes peut s'exprimer par l'équation [Hyvärinen, 1999b] :

$$\text{ACI} = \text{Fonction de contraste} + \text{Algorithme d'optimisation.}$$

Les propriétés statistiques (e.g. consistance et robustesse) de l'ACI dépendent de la fonction de contraste choisie alors que les propriétés algorithmiques (e.g. vitesse de convergence, besoin en mémoire et stabilité numérique) dépendent de l'algorithme d'optimisation.

Fonctions de contraste. Plusieurs fonctions de contraste ont été proposées dans la littérature. Elles peuvent être classées en deux familles : celles qui se basent sur le maximum de vraisemblance et celles qui se basent sur la non-gaussianité.

Dans la première famille, on trouve la fonction de contraste suivante :

$$\phi^{IM}(y) = K(P_y | \prod_{i=1}^{d} p_i(y_i)), \tag{2.21}$$

où P_y est la densité de probabilité $\mathbf{y}$, $p_i(y_i)$ les densités de probabilité de y_i et K la divergence de Kullback-Leibler[9].

[8] Il existe également une variante de l'ACI permettant de prendre en compte le bruit : $\mathbf{x} = \mathbf{A}\mathbf{s} + \mathbf{b}$ où $\mathbf{b}$ est un vecteur aléatoire de bruit de dimension d. Cependant, l'estimation de ce modèle n'est pas résolue et est toujours un axe de recherche à explorer [Hyvärinen, 1999b].

[9] La divergence de Kullback-Leibler permet de mesurer l'écart entre deux densités de probabilité. Elle est définie par : $K(P|Q) = \int \log \frac{p(x)}{q(x)}\, p(x)\, dx$. Ce n'est pas une distance car elle n'est pas symétrique et ne vérifie pas l'inégalité triangulaire.

Cette divergence de Kullback-Leibler de la loi jointe à la loi produit est appelée « information mutuelle » et constitue une mesure de dépendance au sens de l'écart à l'indépendance.

L'autre classe de fonctions de contraste que nous présentons s'appuie sur une notion essentielle en ACI : la non-gaussianité. La non-gaussianité $\phi^G(\mathbf{y})$ du vecteur aléatoire $\mathbf{y}$ est l'écart à la gaussianité de ce vecteur. Elle est exprimée par la divergence de Kullback-Leibler de P_y à la loi normale de mêmes moments d'ordre 1 et 2 que $\mathbf{y}$:

$$\phi^G(\mathbf{y}) = K(P_y|\mathcal{N}(\mathrm{E}[\mathbf{y}], \mathrm{Cov}[\mathbf{y}])). \tag{2.22}$$

Cette quantité est désignée par le terme « néguentropie », également exprimée par :

$$J(\mathbf{y}) = H(\mathbf{y}_{gauss}) - H(\mathbf{y}), \tag{2.23}$$

où H désigne l'entropie, dite différentielle dans le cas continu, ou de Shannon. Pour un vecteur aléatoire $\mathbf{y}$ de densité $f(\mathbf{y})$, elle est donnée par :

$$H(\mathbf{y}) = -\int f(\mathbf{y})\ log\ f(\mathbf{y})\ d\mathbf{y}. \tag{2.24}$$

L'entropie exprime le degré d'information portée par une variable. Une variable gaussienne possède l'entropie la plus élevée parmi toutes les variables aléatoires de même variance. La néguentropie est donc une quantité positive. Elle est nulle si $\mathbf{y}$ suit une loi gaussienne.

La néguentropie semble être une mesure adéquate de la non-gaussianité, mais sa maximisation est difficile. C'est pourquoi elle n'est pas optimisée de façon directe mais approximée par des fonctions mettant en jeu des statistiques d'ordre supérieur. Ainsi, une mesure de non-gaussianité classiquement utilisée est le kurtosis ou cumulant d'ordre 4 :

$$kurt(\mathbf{y}) = \mathrm{E}[\mathbf{y}^4] - 3\mathrm{E}^2[\mathbf{y}^2]. \tag{2.25}$$

Algorithmes d'optimisation. Il existe de nombreux algorithmes d'estimation du modèle ACI. Parmi les travaux les plus cités, on trouve l'algorithme « JADE » [Cardoso, 1999] et l'algorithme « FastICA » [Hyvärinen, 1999a]. Nous présentons ici le principe de ce dernier algorithme. Le lecteur intéressé pourra se référer à [Hyvärinen, 1999b] pour un état de l'art complet de ces algorithmes.

En général, un pré-traitement est appliqué avant d'estimer les composantes indépendantes. Il s'agit de centrer et blanchir les données de telle sorte que leur moyenne soit nulle et leur covariance soit la matrice identité. Ceci est réalisé au moyen d'une ACP. Une première version de l'algorithme FastICA consiste à estimer les composantes indépendantes une à une. Chaque composante $y_i = \mathbf{w}^t\mathbf{x}$, où $\mathbf{w}^t$ est un vecteur ligne de la matrice de séparation $\mathbf{W}$ $(i = 1, \ldots, d)$, est obtenue en maximisant un critère de non-gaussianité. Le critère utilisé est la néguentropie, $J(\mathbf{w}^t\mathbf{x})$. La procédure d'optimisation s'appuie sur la méthode du point fixe et produit l'algorithme suivant.

1. Choisir un $\mathbf{w}$ initial de norme unité, aléatoirement par exemple.
2. $\mathbf{w} \leftarrow J(\mathbf{w}^t\mathbf{x})$.
3. $\mathbf{w} \leftarrow \mathbf{w}/||\mathbf{w}||$.
4. Reprendre en 2 jusqu'à convergence.

Cet algorithme est exécuté autant de fois que l'on veut estimer de composantes. Pour éviter d'estimer deux fois la même composante, les sorties $y_{i,i=1,\ldots,d}$ sont décorrélées à l'issue de chaque itération.

2.5.1.3 Limites des techniques linéaires

Dans cette section, nous discutons les limites des techniques linéaires pour la réduction de la dimension, d'abord de manière générale, puis spécifiquement dans le cadre de la recherche de plus proches voisins. Dans cette dernière partie, nous nous focaliserons particulièrement sur l'ACP.

Comme nous l'avons expliqué plus haut, les techniques linéaires pour la réduction de la dimension permettent de trouver une projection linéaire des données dans un nouvel espace de dimension plus réduite tout en contrôlant l'information perdue. Malheureusement, la projection étant linéaire, seules les dépendances linéaires entre les attributs peuvent être révélées. Pour certaines distributions de données, les projections linéaires ne suffisent pas.

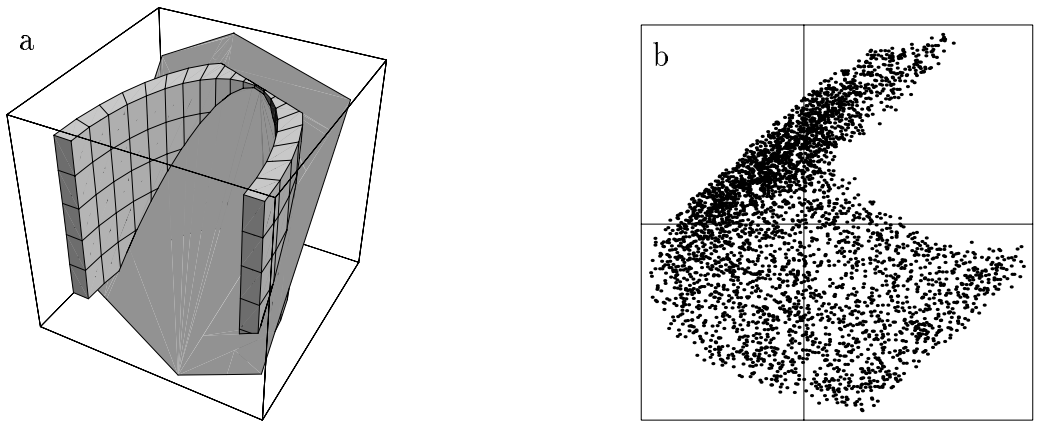


FIG. 2.8: (a) Distribution de données en fer à cheval et le plan principal de projection calculé par ACP. (b) Projection des données sur ce plan. (Figures tirées de la thèse de P. Demartines [Demartines, 1994].)

Prenons à titre d'exemple, la distribution particulière dite en fer à cheval (*cf.* figure 2.8.(a)). Effectuer une ACP sur ces données ne permet de déduire aucune dépendance entre attributs. Les trois axes principaux portent la même quantité d'information (1/3 de la variance). Cela ne veut pas dire qu'il n'y a aucune dépendance entre les 3 attributs, mais que celle-ci n'est pas linéaire et ne peut être révélée par une ACP.

Si l'on s'obstine quand même à tenter une projection sur deux axes, le plan obtenu, par exemple celui de la figure 2.8.(a), supporte 2/3 de la variance mais ne révèle aucune information sur la distribution des données. Plus grave encore, la proximité des données est

sauvegardée plus ou moins bien en fonction de la partie où l'on se positionne dans l'espace. La figure 2.8.(b) montre la projection des données sur *un* plan choisi par ACP. Nous remarquons que dans certaines parties de l'espace (la partie basse du plan), les distances inter-vecteurs sont très proches de leurs vraies valeurs calculées dans l'espace original. Par contre, dans la partie supérieure du plan de projection, il y a eu un « écrasement des données », et, par conséquent, les distances inter-vecteurs sont très mal sauvegardées.

Dans le contexte de la recherche de similarité, ce problème ne permet pas d'avoir un raisonnement global sur l'erreur introduite par la réduction de la dimension. Cette erreur dépend plutôt de la région où l'on se positionne. Ce raisonnement, purement géométrique, se base uniquement sur des interprétations graphiques, et ne peut malheureusement pas être exploité lorsque la projection des données est effectuée dans un espace de dimension supérieure à 3.

En recherche de plus proches voisins, l'ACP est souvent présentée comme l'ultime solution au problème de la malédiction de la dimension. Le principe consiste à réaliser une ACP sur les données de grande dimension avant de les indexer. Ceci permet de réduire leur dimension et de faciliter leur indexation puisque cette solution mise sur une réduction importante de la dimension qui permettrait de s'affranchir complètement de la malédiction de la dimension. Malheureusement, cette solution n'est pas aussi évidente et aussi simple qu'elle ne l'est souvent présentée. Elle possède plusieurs inconvénients majeurs qui limitent son utilisation dans des applications pratiques. Nous les résumons dans les quatre points suivants.

Transformation statique L'ACP est une transformée qui s'applique à un ensemble de données dans le but d'étudier sa distribution et de réduire sa dimension à un moment donné. L'évolution du contenu de la base de données nécessite une mise à jour de l'analyse, c'est-à-dire une mise à jour de la matrice de covariance, la mise à jour des axes principaux et enfin la projection de l'ensemble des vecteurs de la base dans le nouvel espace. Si, de plus, un index est utilisé en amont pour accélérer les recherches au sein de l'espace de dimension réduite, cet index doit être reconstruit. Cette opération de mise à jour est extrêmement coûteuse lorsque le contenu de la base de données évolue régulièrement.

Réduction de la dimension souvent non suffisante La dimension brute des vecteurs d'attributs est souvent très élevée et peut prendre des valeurs allant jusqu'à 512 dans le cas des histogrammes de couleurs par exemple. Sachant que les problèmes liés à la malédiction de la dimension se manifestent selon [Weber *et al.*, 1998] à partir d'une dimension 16 pour des données réelles, la réduction de la dimension de tels ensembles est souvent non suffisante pour s'affranchir de la malédiction de la dimension. Si l'on veut quand même réduire la dimension en dessous de ce seuil, la perte d'information risque d'être telle que le résultat ne sera plus exploitable.

Sensibilité aux bruits L'ACP est une méthode de second ordre car elle se base uniquement sur l'étude de la matrice de covariance (les moments d'ordre 2). Elle est, par conséquent, très sensible aux vecteurs aberrants. Pour étudier ce problème, nous avons mené une expérience sur un ensemble de données réelles composé de 413 412 vecteurs de dimension 24. Ces vecteurs sont des descripteurs locaux différentiels d'images couleurs. Nous

avons réalisé une ACP sur cet ensemble de données. Les résultats de l'analyse montrent qu'une seule composante principale porte à elle seule l'essentiel de l'information avec un pourcentage de l'inertie expliquée égale à 96,04 % (la valeur propre associée à cette composante est égale à 838,74 alors que la somme de toutes les valeurs propres associées aux 24 composantes principales est égale à 873,33).
Nous avons ensuite filtré manuellement 1004 vecteurs sur les 413 412 vecteurs de la base originale et nous avons refait une ACP. Ces 1004 vecteurs possèdent des composantes aberrantes obtenues suites à des divisions proches de 0. Cette fois-ci, il s'avère que pour avoir un pourcentage de l'inertie expliquée égale à 96,04 %, il faut retenir 19 composantes principales.
Cette expérience montre clairement la sensibilité de l'ACP aux points aberrants. Seuls 1004 vecteurs sur un ensemble de 413 412 vecteurs, soit moins de 0,24 % de la totalité de la base, ont pu fausser complètement l'analyse.

Approximation introduite par la perte d'information Réduire la dimension des données entraîne une perte d'information. Cette perte se manifeste par des approximations des distances inter-vecteurs dans l'espace réduit. Dans le cas particulier de la recherche de plus proches voisins, ceci entraîne des imprécisions au niveau de la recherche. Même si l'ACP permet de contrôler la réduction de la dimension en fonction de la proportion de l'inertie exprimée, aucune relation directe ne peut être établie entre cette proportion et la qualité des plus proches voisins fournis en résultats. D'autant plus que l'ACP vise à préserver globalement les distances inter-vecteurs (inertie globale) alors que les plus proches voisins dépendent du voisinage local, et donc, de la qualité d'approximation des distances inter-vecteurs localement. Par ailleurs, le problème de la non-uniformité des approximations – évoqué plus haut – diminue davantage l'intérêt de ce contrôle.
Pour mieux illustrer ce problème, nous présentons ici deux expériences sur deux ensembles de données réelles. Le premier, noté E_1, est composé de 7074 descripteurs de séquences de musique de dimension brute égale à 73. Le deuxième, noté E_2, contient 30 021 histogrammes d'images couleurs de dimension brute égale à 512. Nous avons effectué une ACP sur chacun des deux ensembles puis nous avons créé des ensembles de différentes dimensions. Les deux premières lignes des tableaux 2.1 donnent pour chacun des deux ensembles, le nombre de composantes principales retenues (NCPR) avec le pourcentage de l'inertie expliquée associé (PIE). Nous rappelons que ce pourcentage correspond à la proportion de la variance cumulée par les composantes retenues. Il est calculé par la formule 2.17 (page 47).
Dans chacun des deux ensembles, nous avons tiré aléatoirement 200 vecteurs requêtes puis nous les avons transformés par ACP et nous avons tronqués leurs dimensions de la même manière que les vecteurs de données. Nous avons ensuite effectué des recherches de k-plus proches voisins pour trois valeurs de k, 1, 10 et 20. Notons que les résultats de la recherche obtenus pour les ensembles de dimensions réduites sont approximatifs. Nous avons alors calculé la précision de ces résultats par rapport aux plus proches voisins(exacts) retrouvés en considérant la totalité des dimensions. Pour un vecteur requête, la précision des k-plus proches voisins approximatifs obtenus dans un espace réduit est égale à la proportion des plus proches voisins exacts qui se retrouvent dans l'ensemble des résultats approximatifs.
Les résultats sont résumés dans les tableaux 2.1 pour les deux ensembles de données E_1 et E_2. Les valeurs de précision reportées dans les tableaux 2.1 sont les moyennes des précisions obtenues sur les 200 vecteurs requêtes.

Ensemble de données E_1

NCPR	60	50	40	35	25	20	15	10
PIE (%)	99,98	99,85	99,34	98,79	96,29	94,07	90,92	86,53
Précision 1-PPV	0,98	0,97	0,92	0,91	0,80	0,69	0,55	0,42
Précision 10-PPV	0,99	0,98	0,95	0,94	0,83	0,76	0,65	0,50
Précision 20-PPV	0,99	0,98	0,96	0,94	0,85	0,78	0,68	0,56

Ensemble de données E_2

NCPR	350	250	200	150	125	100	75	50	25	10
PIE (%)	99,63	97,04	93,66	87,79	83,37	77,49	69,29	57,64	41,17	25,63
Précision 1-PPV	1	0,99	0,98	0,95	0,90	0,85	0,79	0,70	0,59	0,41
Précision 10-PPV	1	0,99	0,99	0,97	0,95	0,92	0,86	0,78	0,66	0,49
Précision 20-PPV	1	0,99	0,99	0,97	0,95	0,92	0,87	0,81	0,67	0,49

TAB. 2.1: Impact de la réduction de la dimension par ACP sur la précision de la recherche de plus proches voisins. PIEs et précisions moyennes des recherches de plus proches voisins pour différentes dimensions réduites par ACP.

Ces tableaux montrent clairement qu'une relation entre la proportion de l'inertie exprimée (utilisée pour choisir la dimension réduite) et la précision de la recherche de plus proches voisins est difficile à établir. Pour l'ensemble E_1, même lorsque ce critère indique que la quantité d'information sauvegardée en termes d'inertie est de 90,92 % pour une dimension réduite égale à 15, la précision moyenne de la recherche du plus proche voisin sur les 200 vecteurs requêtes n'est que de 0,55, c'est-à-dire, pour 45 % des vecteurs requêtes, le plus proche voisin retrouvé n'est pas le même que celui qui aurait pu être retrouvé si la recherche avait été effectuée dans l'espace original.
Cette relation est complètement différente pour le deuxième ensemble E_2. Pour des PIEs faibles, la précision moyenne des plus proches voisins est relativement élevée. Ces deux exemples montrent que cette relation dépend plutôt de la nature et de la distribution des données. Par conséquent, il est extrêmement difficile de choisir la dimension réduite dans le contexte de la recherche de plus proches voisins en utilisant uniquement la proportion de l'inertie expliquée.

2.5.2 Techniques non linéaires

Les méthodes non linéaires pour la réduction de la dimension permettent de trouver un nouvel espace de représentation des données de dimension réduite. Ces méthodes sont pour une grande partie itératives et visent à minimiser une fonction de coût liée aux distances inter-vecteurs. Le but est de garder les mêmes distances entre les vecteurs dans l'espace original et leurs correspondances dans l'espace transformé.

Idéalement, l'objectif de ces méthodes est de transformer n vecteurs d'attributs x_i initialement de dimension d dans un nouvel espace de dimension p avec $p < d$ tel que :

$$\forall\ 1 \leqslant i, j \leqslant n, \quad d(x_i, x_j) = \delta(\hat{x}_i, \hat{x}_j), \tag{2.26}$$

où d est la mesure de similarité entre les vecteurs dans l'espace original, δ est la fonction de distance entre vecteurs dans l'espace transformé et $\hat{x}_i$ et $\hat{x}_j$ sont respectivement les transformées de x_i et x_j dans l'espace transformé.

En fonction de la méthode, d et δ peuvent être différents, d n'est pas forcément une distance et δ est une distance mais pas nécessairement euclidienne. Il est évident que dans la grande majorité des cas, il est impossible de réaliser cet objectif. Les méthodes existantes permettent de trouver un espace dans lequel les distances entre vecteurs sont les plus proches possibles des distances entre les vecteurs dans l'espace original. Elles guident leurs processus de recherche ou de construction de l'espace transformé par une fonction de coût liée aux distances inter-vecteurs.

Contrairement aux méthodes linéaires, le choix de la dimension de l'espace réduit doit être effectué *a priori*. Ceci constitue le principal inconvénient de ces méthodes. En effet, il n'existe aucune méthode permettant de trouver la valeur optimale de la dimension de l'espace réduit. La meilleure solution consiste à exécuter l'algorithme de réduction de la dimension avec l'ensemble des valeurs que peut prendre la dimension de l'espace réduit, et à retenir la transformation qui réalise le meilleur compromis entre le taux de réduction de la dimension et la minimisation de la fonction de coût. Malheureusement, le coût de cette solution est prohibitif.

En pratique, les méthodes existantes laissent le choix de la dimension de l'espace réduit à l'utilisateur. En général, elle est fixée arbitrairement. Certains travaux suggèrent de choisir la valeur qui correspond à la dimension intrinsèque des données, mais cette solution demeure théorique car il n'existe pas de méthode universelle permettant de déduire la dimension intrinsèque d'un ensemble de données (*cf.* section 2.4). Plusieurs travaux approchent cette dimension par la dimension fractale de données [Traina Jr. *et al.*, 2000, de Sousa *et al.*, 2002], mais cette approximation n'a aucun fondement théorique.

Dans cette section, nous présentons trois méthodes non linéaires pour la réduction de la dimension : deux méthodes issues de la statistique (le *Multidimensional Scaling* et l'analyse en composantes curvilinéaires) et une méthode non-statistique (*FastMap*) développée spécifiquement pour des applications d'indexation.

2.5.2.1 Multidimensional Scaling

C'est une méthode qui permet de représenter un ensemble de n vecteurs dans un espace de dimension p uniquement à partir d'une matrice $n \times n$ des distances inter-vecteurs dans l'espace original. Dans le cas où les distances inter-vecteurs sont euclidiennes, il s'agit de la version la plus simple : le *Multidimensional Scaling* (MDS) classique (travaux de Young et Householder (1938)).

Dans ce qui suit, nous présentons le MDS classique et nous donnons le principe des deux autres variantes : le MDS par les moindres carrés et le MDS non métrique. Pour une description détaillée de toutes les variantes du MDS, le lecteur intéressé pourra se référer à [Everitt & Rabe-Hesketh, 1997, Borg & Groenen, 1997, Cox & Cox, 2001].

Soit $\mathbf{D}$ la matrice des distances inter-vecteurs dans l'espace original. C'est une matrice symétrique $n \times n$. L'objectif est de trouver une matrice $\mathbf{X}$ $(n \times p)$ telle que $d(x_i, x_j)$ soit le plus proche possible de $\mathbf{D}_{ij}$. Dans le cas où les vecteurs d'attributs originaux qui ont servi à calculer la matrice $\mathbf{D}$ sont disponibles, le MDS permet de réduire leur dimension en les projetant dans un espace de dimension réduite. Dans le cas contraire où seule la matrice $\mathbf{D}$ est disponible, le MDS permet simplement de créer un espace de représentation des données.

Il est important de noter qu'il n'existe pas de solution unique à ce problème puisque toute translation ou rotation des lignes de $\mathbf{X}$ est aussi une solution. Pour forcer l'unicité de la solution, des contraintes supplémentaires sont ajoutées. Il s'agit par exemple de placer la moyenne des vecteurs à l'origine et de choisir une orientation qui correspond aux axes principaux du nuage de points. Pour pouvoir déterminer $\mathbf{X}$ à partir de $\mathbf{D}$, nous introduisons tout d'abord des données élémentaires pour établir un lien entre les deux matrices.

Soit $\mathbf{B}$ la matrice $n \times n$ définie par :

$$\mathbf{B} = \mathbf{X}\mathbf{X}^T. \tag{2.27}$$

Les éléments de $\mathbf{B}$ s'écrivent alors :

$$b_{ij} = \sum_{k=1}^{p} x_{ik} x_{jk}, \tag{2.28}$$

où x_{ik} est l'élément de la i^e ligne et de la k^e colonne de $\mathbf{X}$. Il est facile de montrer que les carrés des distances euclidiennes entre vecteurs x_i (lignes de la matrice $\mathbf{X}$) s'expriment en fonction de b_{ij} comme suit :

$$d_{ij}^2 = b_{ii} + b_{jj} - 2b_{ij}. \tag{2.29}$$

$\mathbf{B}$ est donc une matrice intermédiaire entre $\mathbf{X}$ et $\mathbf{D}$. Le problème se décompose alors en deux sous-problèmes : (1) déterminer $\mathbf{B}$ à partir de $\mathbf{D}$ puis (2) $\mathbf{X}$ à partir de $\mathbf{B}$.

Calculer les b_{ij} à partir de d_{ij} nécessite d'inverser l'équation (2.29). Comme il existe une infinité de solutions à ce problème, une contrainte locale est introduite. Il s'agit de considérer que le centre de gravité des vecteurs correspond à l'origine de l'espace, c'est-à-dire que $\sum_{i=1}^{k} x_{ik} = 0,\ \forall k$. Cette contrainte ainsi que l'équation (2.28) impliquent que la somme de chacune des lignes de la matrice $\mathbf{B}$ vaut 0. Ainsi :

$$\sum_{i=1}^{n} d_{ij}^2 = T + n\, b_{jj}, \tag{2.30}$$

$$\sum_{j=1}^{n} d_{ij}^2 = n\, b_{ii} + T, \tag{2.31}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij}^2 = 2\, n\, T, \tag{2.32}$$

où $T = \sum_{i=1}^{n} b_{ii}$ est la trace de la matrice $\mathbf{B}$.

T peut être calculée à partir de l'équation (2.32), les b_{ii} à partir de l'équation (2.31), et les b_{ij} peuvent donc être déduits à partir de l'équation (2.29). De cette façon, $\mathbf{B}$ peut être calculée à partir de $\mathbf{D}$. Pour résoudre le 2^e sous-problème qui consiste à calculer $\mathbf{X}$ à partir de $\mathbf{B}$, il suffit d'effectuer une décomposition spectrale de $\mathbf{B}$:

$$\mathbf{B} = \mathbf{V}\Lambda\mathbf{V}^T, \tag{2.33}$$

où Λ est la matrice diagonale ($\Lambda = [\lambda_1, \ldots, \lambda_n]$) des valeurs propres de $\mathbf{B}$ et $\mathbf{V}$ la matrice des vecteurs propres normalisés. Les valeurs propres sont ordonnées comme suit : $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_n$.

Si la dimension de l'espace transformé a été fixée à p, la matrice $\mathbf{X}$ des coordonnées des vecteurs dans l'espace transformé est :

$$\mathbf{X} = \mathbf{V}_p \, \Lambda_p^{1/2}, \tag{2.34}$$

où $\Lambda_p = [\lambda_1, \ldots, \lambda_p]$ et $\mathbf{V}_p = [v_1, \ldots, v_p]$.

Dans cette version de base du MDS, la dimension des vecteurs peut être choisie *a posteriori*. En effet, le nombre de dimensions retenues pour la représentation des données définit la qualité de la représentation. Cette qualité peut être mesurée de la même manière que pour l'ACP en exploitant les valeurs propres :

$$C_p = \frac{\sum_{i=1}^{p} \lambda_i}{\sum_{i=1}^{n} \lambda_i}. \tag{2.35}$$

Malheureusement, cette version n'est utilisable que lorsque la matrice de distance $\mathbf{D}$ correspond à des distances euclidiennes. Dans le cas contraire, certaines valeurs propres peuvent être négatives et l'analyse présentée plus haut n'est plus valable. Par ailleurs, cette version de MDS vise à minimiser le critère idéal, c'est-à-dire à se rapprocher le plus des distances fournies en entrée quelle que soit la grandeur de ces valeurs. Ce qui donne un critère très contraignant qui complexifie beaucoup la tâche [Everitt & Rabe-Hesketh, 1997].

Des variantes plus adaptatives ont été proposées. Il s'agit de MDS par les moindres carrés. Le critère utilisé est celui des moindres carrés. Il est plus global puisqu'il considère la somme des différences entre les distances fournies et celles calculées dans l'espace transformé. Ce critère peut se formuler de la façon suivante :

$$S = \frac{\sum_{i<j} (d_{ij} - \delta_{ij})^2}{\sum_{i<j} d_{ij}^2}, \tag{2.36}$$

où d_{ij} est la distance fournie entre x_i et x_j et δ_{ij} représente la distance dans l'espace de représentation à déterminer (l'espace transformé). Le calcul de la matrice $\mathbf{X}$ de représentation des vecteurs s'effectue par un algorithme d'optimisation traditionnel.

Deux autres variantes du MDS existent. Il s'agit du MDS non métrique et du MDS non euclidien. Le premier permet de trouver une représentation pour des matrices de dissimilarité qui n'ont pas été produites par des fonctions de distance. Quant au deuxième, il permet de trouver une représentation des données dans un espace muni d'une fonction de distance non euclidienne. Nous ne présentons pas ici ces deux variantes car elles reposent sur le même principe que le MDS par moindres carrés. Le lecteur intéressé pourra se référer à [Cox & Cox, 2001, Everitt & Rabe-Hesketh, 1997] pour plus de détails.

2.5.2.2 Analyse en composantes curvilinéaires

L'analyse en composantes curvilinéaires (ACC) est une méthode non linéaire pour la réduction de la dimension. Elle a été développée par [Demartines & Hérault, 1997]. Comme

le MDS, l'ACC vise à transformer les vecteurs de façon à ce que les distances inter-vecteurs dans le nouvel espace soient les plus proches possibles de celles calculées dans l'espace original. Elle utilise pour cela un réseau de neurones dont le rôle est d'apprendre la transformation entre l'espace original et l'espace transformé. Pour mesurer l'erreur de transformation, le réseau de neurones utilise la fonction de coût suivante :

$$E = \frac{1}{2} \sum_{i} \sum_{i \neq j} (\text{dist}(x_i, x_j) - \text{dist}(\hat{x}_i, \hat{x}_j))^2 F(\text{dist}(\hat{x}_i, \hat{x}_j), \lambda_{\hat{x}}), \tag{2.37}$$

où x_i et x_j sont des vecteurs originaux de dimension d, $\hat{x}_i$ et $\hat{x}_j$ leurs transformations respectives dans l'espace réduit de dimension p, dist est la distance euclidienne et $F(\text{dist}(\hat{x}_i, \hat{x}_j), \lambda_{\hat{x}})$ est une fonction de pondération permettant de donner plus d'importance aux petites distances de façon à favoriser la conservation de la topologie locale des données. Une telle fonction pourrait être aussi simple que la fonction *seuil* : $F(\text{dist}(\hat{x}_i, \hat{x}_j), \lambda_{\hat{x}}) = \begin{cases} 0 & \text{si} \quad \text{dist}(\hat{x}_i, \hat{x}_j) \leqslant \lambda_{\hat{x}} \\ 1 & \text{si} \quad \text{dist}(\hat{x}_i, \hat{x}_j) > \lambda_{\hat{x}} \end{cases}$, ou bien une fonction plus complexe telle que la fonction sigmoïde ($F(y) = \frac{1}{1+e^{-y}}$).

Cette fonction de coût nécessite le calcul de toutes les distances inter-vecteurs dans l'espace transformé après chaque époque d'apprentissage, Demartines *et al.* proposent alors d'effectuer l'apprentissage sur les données quantifiées. Cela réduit le coût de calcul et permet d'accélérer la procédure d'apprentissage garantissant ainsi une convergence plus rapide. Cependant, toute projection d'un vecteur dans l'espace transformé nécessite une interpolation.

Hormis le problème du choix *a priori* de la dimension de l'espace de projection, le choix de certains paramètres internes à la procédure d'apprentissage utilisée constitue la principale difficulté de son utilisation [Verleysen, 2003].

2.5.2.3 FastMap

FastMap est une méthode de réduction de la dimension qui se positionne dans les mêmes conditions que le MDS, c'est-à-dire qu'elle suppose que seule une matrice de dissimilarité entre vecteurs est disponible. Dans le cas où des vecteurs d'attributs sont déjà disponibles mais que leur dimension est grande, FastMap permet de réduire leur dimension. Dans le cas où seule une matrice de dissimilarité est disponible, FastMap, tout comme le MDS, permet simplement de créer un espace de représentation pour les données.

Cette méthode a été développée par [Faloutsos & Lin, 1995, Faloutsos, 1996]. Elle a été proposée pour pallier les insuffisances du MDS dans le domaine de l'indexation. En effet, la complexité du MDS est quadratique et constitue une difficulté majeure quant à son utilisation dans le contexte de très grandes bases de données. D'autre part, le MDS est une transformation statique qui permet de trouver un espace de représentation pour une matrice de dissimilarité et ne permet de prendre en compte aucun vecteur supplémentaire. Un tel ajout nécessite de refaire complètement l'analyse. Or, en indexation et plus particulièrement en recherche par similarité, toute nouvelle recherche nécessite la projection de la requête dans l'espace de définition des données de la base.

Dans ce qui suit, nous désignerons par le terme « objets » les vecteurs dans leur espace original. Nous avons choisi cette terminologie puisque les vecteurs originaux ne sont pas définis, seule une distance pour mesurer la (dis-)similarité entre eux est disponible.

Les données en entrée se résument donc à une matrice symétrique $(n \times n)$ des distances entre objets. FastMap suppose que ces objets sont des points d'un espace multidimensionnel inconnu de dimension d, et son but est de les projeter dans un espace euclidien de dimension p.

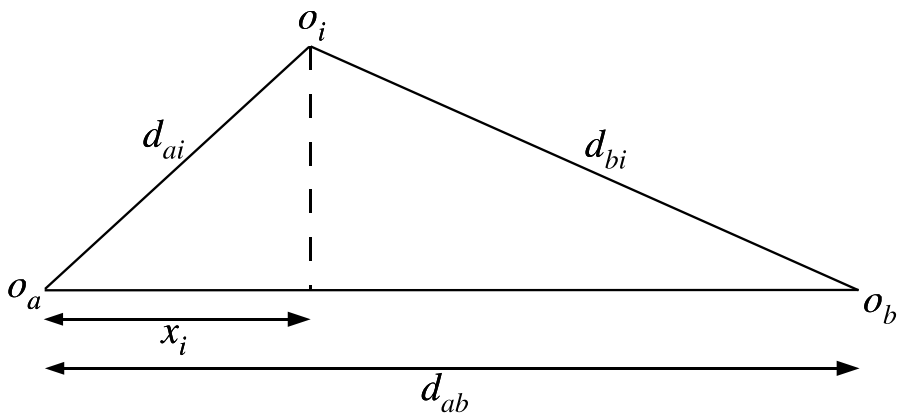


FIG. 2.9: Exemple de la projection d'un objet o_i sur une droite définie par les deux objets pivots o_a et o_b.

Dans le cas particulier où $p = 1$, FastMap procède par une projection des objets sur une droite. La droite est définie par deux objets o_a et o_b, appelés objets pivots. La projection des objets de la base sur cette droite s'effectue selon la procédure décrite dans la figure 2.9. Pour la projection d'un objet o_i, les distances entre cet objet et les deux objets pivots sont utilisées. x_i, la coordonnée de o_i sur la droite par rapport à o_a, est donnée par :

$$x_i = \frac{{d_{a,i}}^2 + {d_{a,b}}^2 - {d_{b,i}}^2}{2\, d_{a,b}} \tag{2.38}$$

Grâce à cette équation, les objets peuvent être projetés sur une droite tout en préservant quelque peu les distances entre les objets (si o_i est plus proche de o_a que de o_b, sa projection x_i sera plus proche de o_a que de o_b).

L'extension de cette solution dans le cas où $p > 1$ est simple. Une projection des objets sur une droite selon la procédure décrite précédemment est d'abord effectuée. Ceci permet de déterminer la première composante des vecteurs d'attributs dans l'espace transformé. Les objets sont ensuite projetés dans un hyperplan perpendiculaire à la première droite de projection. Les distances entre objets sont calculées dans ce nouvel espace, et, de nouveau, deux objets pivots sont choisis. Une droite passant par ces deux objets est déterminée et les objets sont projetés. La deuxième composante est ainsi calculée. Cette procédure est ensuite appliquée récursivement pour déterminer l'ensemble des composantes restantes.

La figure 2.10 illustre cette procédure pour $p = 3$. Les objets o_a et o_b ont été choisis comme objets pivots. Les objets o_i et o_j ont d'abord été projetés sur la droite $(o_a o_b)$. Si

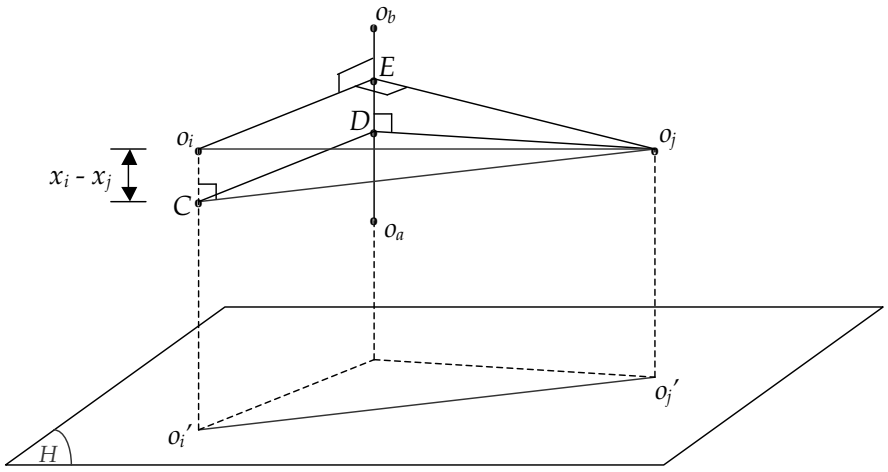


FIG. 2.10: Application de FastMap dans un espace de dimension 3.

l'on suppose que l'espace original des objets est de dimension m, l'ensemble des objets est ensuite projeté sur un hyperplan de dimension $m-1$ et perpendiculaire à la droite $(o_a o_b)$ (le plan H dans la figure 2.10). Dans ce nouveau plan, les distances entre objets sont calculées en utilisant des propriétés géométriques de base et, de nouveau, deux objets pivots sont choisis et les objets sont projetés.

L'heuristique proposée pour le choix des objets pivots consiste à choisir les deux objets les plus éloignés afin d'avoir les projections les plus dispersées possibles sur la droite de projection. Ceci nécessite malheureusement le calcul de toutes les distances entre les objets (complexité quadratique). Les auteurs proposent un algorithme linéaire mais qui ne garantit pas de trouver les deux objets les plus éloignés.

Outre la complexité linéaire de FastMap ($O(N\,p)$), cet algorithme permet d'inclure facilement de nouveaux objets dans la base. En effet, en sauvegardant pour chaque dimension les deux objets pivots associés, l'insertion d'un nouvel objet consiste simplement à le projeter sur les différentes dimensions de l'espace selon la procédure décrite dans la figure 2.9. Toutefois, il est à noter qu'une ré-exécution de l'algorithme sur l'ensemble de la base peut s'avérer nécessaire pour garder une bonne approximation des distances si le contenu de la base change énormément.

Par ailleurs, outre le problème du choix *a priori* de la dimension de l'espace de projection, cette méthode ne repose pas sur des principes théoriques solides. Elle est purement heuristique. Nous l'avons citée ici car c'est la seule méthode qui a été développée par des chercheurs spécialisés en indexation multidimensionnelle et dans le but d'être utilisée dans ce domaine.

2.6 Synthèse et discussions

Nous avons étudié dans ce chapitre le problème de la malédiction de la dimension. Nous avons présenté quelques-unes des propriétés mathématiques des espaces de grande dimension afin de montrer qu'il est difficile d'étendre les intuitions que l'on a dans les espaces à une, deux ou trois dimensions aux espaces de dimensions supérieures. Nous avons montré comment certaines de ces propriétés amplifient les problèmes liés aux index multi-dimensionnels jusqu'à rendre leurs performances plus mauvaises que celles de la recherche séquentielle et exhaustive [Weber *et al.*, 1998, Beyer *et al.*, 1999, Amsaleg & Gros, 2001].

L'étude menée sur la notion de dimension a permis de lever le voile sur cette notion souvent mal comprise. Nous avons fait la distinction entre dimension brute et dimension intrinsèque et nous avons dressé un panorama des méthodes de réduction de la dimension brute. L'objectif de ces méthodes est de réduire la dimension brute afin d'approcher le plus possible la dimension intrinsèque. Nous avons donné les inconvénients de ces méthodes lorsqu'elles sont utilisées dans le but de contourner le problème de la malédiction de la dimension dans le domaine de l'indexation. Les méthodes linéaires sont limitées car les dépendances entre attributs sont rarement linéaires. De plus, comme nous l'avons montré expérimentalement, elles sont très sensibles au bruit et à la distribution des données. Les méthodes non linéaires ne sont pas envisageables en pratique car elles nécessitent de fixer la dimension réduite *a priori*. Cependant, nous tenons à préciser que ce panorama n'est pas exhaustif. Nous nous sommes limités aux méthodes les plus citées et les plus utilisées dans le domaine. Nous n'avons pas présenté par exemple les courbes de Hilbert qui peuvent être considérées comme des techniques de réduction de la dimension. Le principe de ces courbes est de paver l'espace avec une courbe qui passe par l'ensemble des points de la base. À chaque point est ensuite attribué un nombre de sorte que les points adjacents sur la courbe auront des nombres successifs. L'objectif est de trouver des courbes qui préservent la proximité des données multidimensionnelles. Ces courbes sont ainsi des techniques de transformation des données multidimensionnelles dans un espace de dimension 1. Elles ont été utilisées par plusieurs travaux pour accélérer les recherches de plus proches voisins [Kamel & Faloutsos, 1994, Lawder & King, 2001].

Il reste néanmoins une question que nous n'avons pas abordée dans ce chapitre et qui concerne la limite entre petites et grandes dimensions. En d'autres termes, à partir de quel nombre de dimensions se considère t-on dans un espace de grande dimension ?

La réponse à cette question n'est pas évidente. En fait, cette question possède deux réponses différentes selon que l'on considère la dimension brute ou la dimension intrinsèque. Dans le cas de cette dernière, le seul travail à notre connaissance qui propose des éléments de réponse est celui de Verleysen [Verleysen, 2003]. Celui-ci définit la frontière entre les deux par la dimension à partir de laquelle l'intuition de l'être humain ne peut plus expliquer les phénomènes qui apparaissent. L'exemple de l'évolution du volume de l'hypersphère (section 2.3) peut ainsi être utilisé pour déterminer cette limite. La figure 2.2 montre qu'à partir de la dimension 5, le volume de la sphère commence à décroître et s'approche de 0 lorsque la dimension augmente. En se basant sur la proposition de Verleysen, cet exemple suggère que la limite entre petite et grande dimension est proche

de 5. Une autre proposition de Verleysen met en balance la dimension et le nombre de vecteurs dont on dispose. Sachant que le nombre de vecteurs peuplant un espace doit croître exponentiellement avec la dimension pour garder le même niveau de peuplement, Verleysen qualifie chaque ensemble de données dont la taille ne croît pas exponentiellement (avec la dimension) de *petite*, ou bien, de *grande* la dimension des données. Cette dernière définition est malheureusement liée à une certaine évolution des données difficile à saisir en pratique.

D'autres travaux se sont intéressés directement aux données avec leur dimension brute pour essayer de trouver cette limite. Ces travaux se basent uniquement sur des expérimentations. Leurs résultats sont, par conséquent, restreints au domaine d'application des expérimentations. En indexation multidimensionnelle, le travail de référence est celui de Weber *et al.* [Weber *et al.*, 1998]. Les expérimentations menées ont permis de montrer que les performances de toutes les techniques d'indexation traditionnelle pour la recherche de plus proches voisins se dégradent exponentiellement lorsque la dimension augmente. Elles deviennent plus mauvaises que la recherche séquentielle au delà de 16 dimensions. Ainsi, en indexation multidimensionnelle, la limite citée par la majorité des travaux est égale à 16. Néanmoins, il faut être conscient que cette limite est un résultat expérimental ; elle dépend, par conséquent, de la distribution des données utilisées.

Cette discussion ainsi que l'ensemble de ce chapitre montrent bien la difficulté de maîtriser toutes les questions relatives à la malédiction de la dimension. Néanmoins, ils suggèrent :

- d'éviter de présenter les méthodes de réduction de la dimension comme la solution miracle et d'affronter directement le problème de la malédiction de la dimension,
- de comparer les performances des techniques d'indexation et de recherche de plus proches voisins à la recherche séquentielle et exhaustive, celle-ci étant la technique la plus rapide en grande dimension. Sa complexité est linéaire quels que soit la dimension et le nombre de données alors que les performances des techniques traditionnelles de recherche de plus proches voisins se dégradent exponentiellement lorsque la dimension augmente.

Chapitre 3

Nouvelles approches pour la recherche de plus proches voisins

3.1 Introduction

Ce chapitre présente les nouvelles techniques pour la recherche de plus proches voisins. Contrairement aux index multidimensionnels traditionnels, ces techniques ont été proposées en prenant en compte explicitement les problèmes liés aux grandes dimensions. Ainsi, toutes ces techniques proposent des idées originales leur permettant de mieux résister face à la malédiction de la dimension. Nous présenterons le Pyramid-Tree qui se base sur le principe du partitionnement de l'espace mais qui utilise une stratégie de partitionnement permettant de créer un nombre de cellules dont le nombre croît linéairement et non pas exponentiellement lorsque la dimension augmente. Nous présenterons également le VA-File qui est une méthode basée sur la compression des données mais qui, au final, peut être considérée comme une amélioration de la recherche séquentielle. Enfin, nous présenterons toute une classe de méthodes qui permettent d'effectuer des recherches approximatives de plus proches voisins. Ces méthodes se basent sur le principe qui consiste à accélérer les recherches de plus proches voisins en introduisant de l'imprécision dans la recherche. En d'autres termes, il s'agit de négocier une réduction du temps de réponse contre l'introduction de quelques imprécisions au niveau des résultats.

3.2 Le Pyramid-Tree

Contrairement à tous les index multidimensionnels basés sur le partitionnement de l'espace, le Pyramid-Tree [Berchtold *et al.*, 1998b] permet de partitionner l'espace en cellules dont le nombre croît *linéairement* et non pas exponentiellement avec la dimension. Le Pyramid-Tree n'est toutefois pas destiné à faire des recherches de plus proches voisins, mais peut uniquement répondre à des requêtes par intervalle. Cependant, son principe et son excellent comportement dans les espaces de grande dimension en font une technique très intéressante. C'est pourquoi nous l'intégrons à ce chapitre.

Si l'on considère que les données sont représentées dans un hyper-rectangle, le principe du Pyramid-Tree consiste à partitionner l'espace en pyramides telles que le sommet

de chacune d'entre elles corresponde au point central de l'hyper-rectangle et sa base corresponde à un des côtés de l'hyper-rectangle. La figure 3.1.(a) montre un exemple de ce partitionnement dans un espace de dimension 2.

Dans un espace $[0,1]^d$, le nombre de pyramides résultant est $2\,d$. Chaque pyramide est ensuite partitionnée horizontalement en tranches, à l'aide d'hyperplans parallèles à sa base (figure 3.1.(b)).

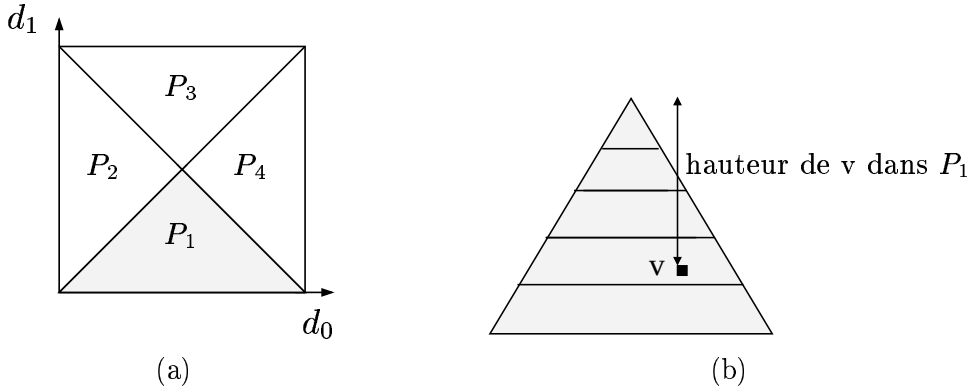


FIG. 3.1: Partitionnement de l'espace en pyramides.

À chaque pyramide est associé un identifiant. Un point de l'espace peut être identifié par le numéro de la pyramide à laquelle il appartient et par sa distance par rapport au sommet de celle-ci. Ces deux valeurs sont ensuite codées par un seul nombre réel. Ce codage est en fait une transformation d'un espace multidimensionnel dans un espace unidimensionnel[1]. Les valeurs transformées peuvent être gérées par une technique d'indexation unidimensionnelle, par le B^+-Tree [Beyer & McCreight, 1972] par exemple. Pour une insertion, un point d-dimensionnel est d'abord transformé en un point défini dans un espace à une seule dimension puis inséré dans le B^+-Tree.

Pour la recherche exacte d'un vecteur particulier, le vecteur requête est transformé dans l'espace unidimensionnel et la valeur transformée est utilisée pour interroger le B^+-Tree. Le résultat de cette recherche est un ensemble de vecteurs multidimensionnels qui possèdent la même valeur transformée que le vecteur requête. Il suffit ensuite de comparer le vecteur requête à chacun de ces vecteurs.

La recherche par intervalle procède d'abord par une transformation de l'intervalle multidimensionnel en un intervalle unidimensionnel avec lequel le B^+-Tree est interrogé. Cette transformation n'est cependant pas triviale et est très coûteuse. Il est possible d'en trouver une description détaillée dans [Berchtold *et al.*, 1998b]. Les vecteurs retournés par le B^+-Tree sont ensuite examinés un à un pour éliminer les vecteurs n'appartenant pas à l'intervalle requête.

La phase supplémentaire de filtrage des résultats est due au fait que l'application qui permet de transformer un vecteur multidimensionnel en une valeur réelle n'est pas injective. En effet, plusieurs vecteurs multidimensionnels peuvent correspondre à une même valeur transformée.

[1] Même s'il n'y a pas de notion de sauvegarde des proximités entre vecteurs dans l'espace transformé, cette technique est parfois classée avec les techniques basée sur la réduction de la dimension.

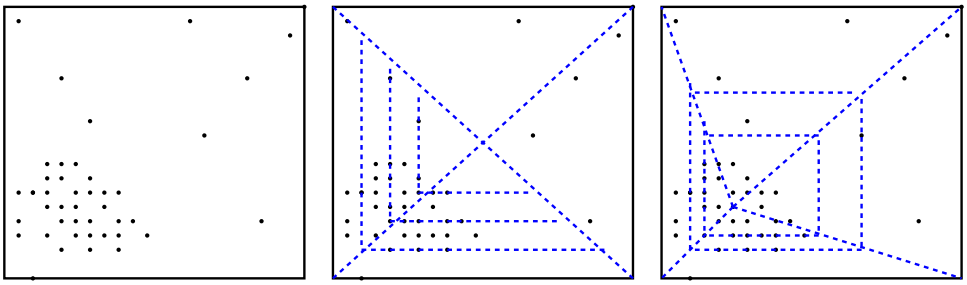


FIG. 3.2: Le Pyramid-Tree et les données de distribution non uniforme.

Bien que cette technique de partitionnement soit originale et simple à mettre en œuvre, elle ne peut être efficace que si les données sont uniformément distribuées. En effet, si les données sont regroupées, le partitionnement résultant n'est pas équilibré, ce qui rend la structure du Pyramid-Tree non calculable et non équilibrée.

Pour cela, dans l'*Extended Pyramid-Tree* [Berchtold *et al.*, 1998b], les sommets des pyramides ne correspondent plus au centre de l'espace mais plutôt au centre de gravité des données. La figure 3.2 illustre la différence entre le partitionnement d'un Pyramid-Tree et celui d'un *Extended* Pyramid-Tree. Cette amélioration ne permet de régler le problème que partiellement. En effet, dans le cas de données regroupées en plusieurs *clusters*, le problème d'équilibre entre les différentes partitions se pose de nouveau.

Par ailleurs, le problème de la bordure dans les espaces de grande dimension décrit dans la section 2.3 (page 39), fait que, quelle que soit la distribution des vecteurs, seul le dernier niveau de la pyramide (près de la bordure) risque d'être peuplé. En effet, en grande dimension, la majeure partie du volume d'un hypercube se trouve dans sa bordure. La probabilité des vecteurs de se trouver à la bordure est donc très importante.

3.3 Amélioration de la recherche séquentielle

Dans cette section, nous présentons deux méthodes basées sur la compression des données et qui peuvent être considérées comme des améliorations de la recherche séquentielle. Il s'agit du VA-File et du LPC-File. L'idée de base de ces approches est d'effectuer dans un premier temps une recherche séquentielle, non pas sur les vecteurs eux-mêmes, mais sur une représentation fortement compressée de ceux-ci. Cette étape, non coûteuse car exécutée sur un ensemble de vecteurs compressés de taille beaucoup plus réduite que l'ensemble des vecteurs originaux, permet d'éliminer une grande partie des vecteurs n'ayant aucune chance d'être parmi les k-plus proches voisins recherchés. Les identifiants des vecteurs retenus permettent ensuite à la recherche d'accéder aux vecteurs eux-mêmes et de construire l'ensemble résultat.

3.3.1 VA-File

Le principe de base du VA-File [Weber *et al.*, 1998] repose sur la gestion de 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. La taille du fichier d'approximation est plus petite que celle du fichier de données. Lors d'une interrogation, un premier parcours séquentiel du fichier d'approximation permet de sélectionner les vecteurs qui ont le plus de chances d'appartenir à l'ensemble résultat. Ensuite, l'accès au fichier de données est effectué sur la base des résultats issus de la première phase. La recherche séquentielle s'effectue sur le (petit) fichier d'approximation, celle-ci est donc très rapide et n'entraîne ensuite l'accès à la base que pour un sous-ensemble réduit de vecteurs (ceux retenus). Ce procédé diminue donc le nombre d'opérations d'E/S et le coût de calcul CPU par rapport à la recherche séquentielle qui, elle, analyse la *totalité* de la base.

Pour calculer les approximations géométriques, chaque dimension d_i est partitionnée en 2^{b_i} intervalles où chaque intervalle est codé sur b_i bits. Les intervalles sont calculés de façon à contenir le même nombre de vecteurs. De cette façon, à chaque cellule, ainsi qu'aux vecteurs qu'elle contient, est attribué un code binaire de longueur $\sum_{i=1}^{d} b_i$. La figure 3.3 illustre un exemple de codage de vecteurs définis dans un espace de dimension deux.

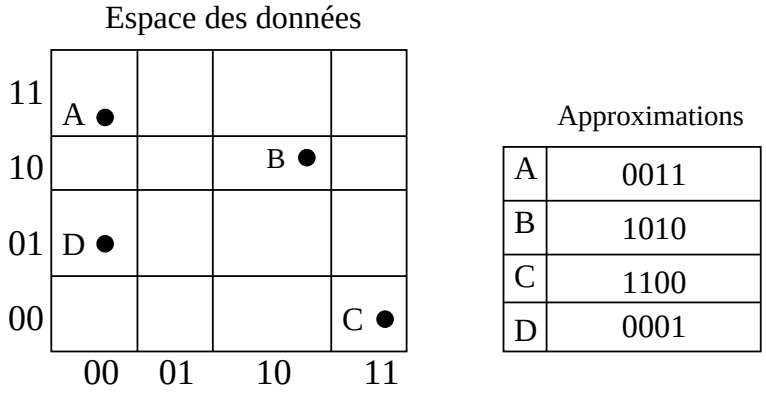


FIG. 3.3: Technique de codage du VA-File.

Lors de la recherche, l'approximation du vecteur requête est calculée. S'il s'agit d'une recherche ponctuelle (recherche exacte d'un vecteur particulier), il suffit de lire tous les vecteurs de même code que le vecteur requête. Il en va de même pour la recherche par intervalle où il faut lire tous les vecteurs appartenant aux cellules avec lesquelles l'intervalle requête possède une intersection non vide.

La recherche de plus proches voisins s'effectue en deux étapes : d'abord le fichier des approximations est parcouru pour éliminer les cellules non pertinentes, c'est la phase de filtrage. Cette phase utilise les mêmes règles de filtrage détaillées dans la section 1.3 du chapitre 1, c'est-à-dire qu'elle se base sur les distances dmin et dmax entre le vecteur requête et les différentes cellules. Ces distances sont facilement calculables puisque le découpage des axes est mémorisé, ce qui permet de déduire les coordonnées des coins de chaque cellule en utilisant son code.

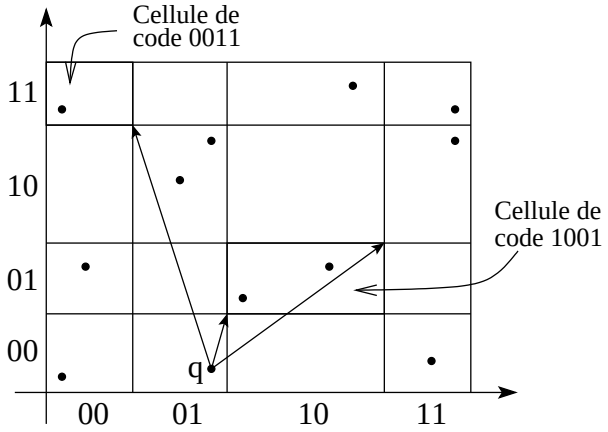


FIG. 3.4: Mesures associées aux cellules gérées par le VA-File.

La figure 3.4 donne un exemple de recherche du plus proche voisin d'un vecteur requête q. En considérant les cellules de codes 0011 et 1001, et en se basant sur les distances dmin et dmax entre celles-ci et q, il est évident que la cellule 0011 n'a aucune chance de contenir de meilleurs plus proches voisins de q que les vecteurs de la cellule 1001.

Les cellules sélectionnées sont ensuite ordonnées selon leur distance minimale, et des accès directs sont ensuite effectués au fichier de données. La recherche s'arrête dès que la distance au plus proche voisin courant est inférieure à la distance minimale de la prochaine cellule candidate. Pour la recherche des k-plus proches voisins, la même procédure est utilisée, sauf qu'il faut considérer les k meilleures solutions.

Les performances du VA-File dépendent essentiellement de la taille du fichier des approximations et du taux de filtrage. Concernant la taille du fichier des approximations, si celui-ci ne peut plus tenir en mémoire centrale, les performances du VA-File se dégradent et deviennent à leur tour plus mauvaises que celles obtenues par la recherche séquentielle [Weber *et al.*, 1998]. Ceci a été vérifié expérimentalement dans [Amsaleg & Gros, 2001]. Le taux de filtrage est quant à lui lié à l'efficacité des règles de filtrage. En réalité, ces deux facteurs sont corrélés puisqu'ils dépendent du même paramètre : le nombre de bits utilisés pour le codage d'un vecteur. Attribuer une grande valeur à ce nombre permet d'avoir des approximations précises (une grille de codage plus fine) et donc améliore le taux de filtrage, mais en contrepartie, génère un fichier d'approximation de taille importante.

Dans [Weber & Böhm, 2000], une étude pour quantifier la relation entre le nombre de bits de codage et la qualité des approximations obtenus a été menée. Il a été montré notamment qu'il faut que la croissance du nombre de bits de codage soit logarithmique par rapport à la croissance de la dimension des vecteurs pour avoir la même qualité d'approximation. Outre l'hypothèse de distribution identique et indépendante des vecteurs selon toutes les dimensions qui a été faite, cette étude ne donne aucune indication sur le choix du nombre de bits dans l'absolu.

Par ailleurs, [Ferhatosmanoglu *et al.*, 2000] remarque que la technique de codage du VA-File est particulièrement adaptée aux données de distribution uniforme mais peut être améliorée dans le cas de données corrélées et/ou regroupées en *clusters*. La technique proposée, le VA$^+$-File, procède d'abord par une transformation des vecteurs vers un nouvel espace à l'aide de la transformée de Karhunen-Loève (une analyse en composantes principales) afin d'éliminer les éventuelles corrélations entre les dimensions des vecteurs puis applique une technique de codage similaire à celle utilisée par le VA-File. La répartition des bits de codage sur les différentes dimensions n'est cependant plus uniforme mais dépend de « l'importance » de chaque dimension. Cette dernière notion est liée à la quantité d'information portée par chaque dimension. Par ailleurs, une nouvelle procédure de partitionnement des axes a été proposée. Il ne s'agit plus de découper les axes de sorte que chaque intervalle contienne le même nombre de vecteurs, mais une procédure plus complexe qui utilise l'algorithme de *clustering K-Means* a été utilisée pour avoir une meilleure quantification.

3.3.2 LPC-File

[Cha *et al.*, 2002] remarque que les performances du VA-File ne peuvent être améliorées que par une augmentation du nombre de bits utilisés lors du codage. Or, une telle augmentation peut entraîner un surcoût non négligeable lors du stockage et du traitement du fichier des approximations. Plutôt que d'augmenter simplement le nombre de bits de codage, il propose alors d'enrichir les information 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. Le codage des vecteurs ainsi que l'algorithme de recherche s'en trouvent ainsi modifiés comme suit.

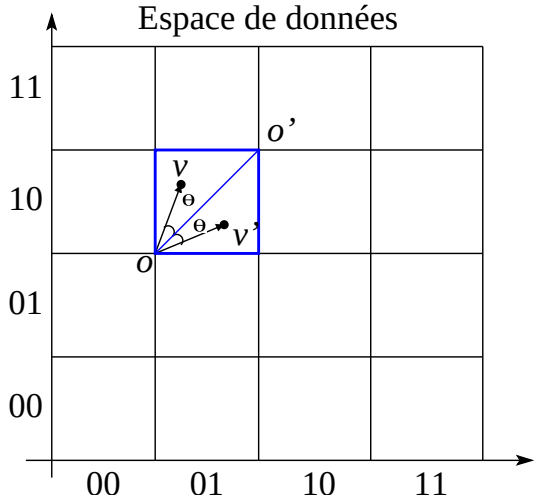


FIG. 3.5: Codage des vecteurs du LPC-File.

Codage des vecteurs. L'espace est tout d'abord partitionné en cases de taille identique et à chaque cellule est attribué un code binaire de la même manière que dans le VA-File. À

chaque vecteur sont associées aussi ses coordonnées polaires (r et θ). Ces coordonnées sont calculées par rapport au coin inférieur gauche de la case et la diagonale reliant ce dernier et le coin supérieur droit (figure 3.5). Le vecteur v de la figure 3.5 reçoit le code $< c, r, \theta >$ où c est le code de la case contenant v, 0110 en l'occurrence, r la distance euclidienne entre v et o et θ l'angle entre la diagonale oo' et le vecteur ov. Il est à noter que plusieurs vecteurs peuvent avoir le même code (v et v' dans la figure 3.5 par exemple). Dans un espace de dimension 3, les vecteurs ayant le même code forment un cercle. Ce cas est illustré dans la figure 3.6. Dans un espace de dimension supérieure à trois, les vecteurs ayant le même code forment une hypersphère.

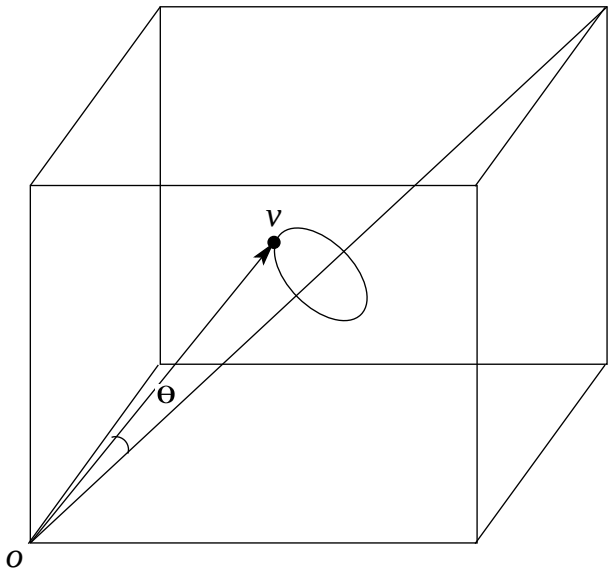


FIG. 3.6: L'ensemble de vecteurs ayant le même code forme un cercle en dimension 3.

Algorithme de recherche. Tout comme le VA-File, en présence d'un vecteur requête, le fichier des approximations est d'abord parcouru pour effectuer une première phase de filtrage. Entre chaque approximation et le vecteur requête, une borne supérieure dmax et une borne inférieure dmin de la distance entre le vecteur de donnée et le vecteur requête sont calculées. Un vecteur est filtré (non considéré par la recherche) car ne pouvant être le plus proche voisin, si la borne inférieure de sa distance vers le vecteur requête est supérieure à la borne supérieure d'un autre vecteur.

Dans la deuxième phase, les vecteurs retenus sont lus dans l'ordre de leur dmin. La distance entre chaque vecteur lu et le vecteur requête est calculée et l'ensemble résultat est mis à jour au fur et à mesure. La recherche s'arrête dès que la distance dmin du prochain vecteur à lire est supérieure à la distance au plus proche voisin courant.

Cette procédure est identique à celle utilisée par le VA-File. La nouveauté de cette technique réside dans le calcul des bornes dmin et dmax qui fait intervenir les coordonnées polaires. De manière générale, plus le calcul des bornes est précis, plus grand est le nombre de vecteurs filtrés et plus rapide est l'algorithme de recherche.

En considérant un vecteur requête q et une approximation a d'un vecteur de donnée s (*cf.* figure 3.7), calculer les distances dmin et dmax entre un vecteur requête q et a revient

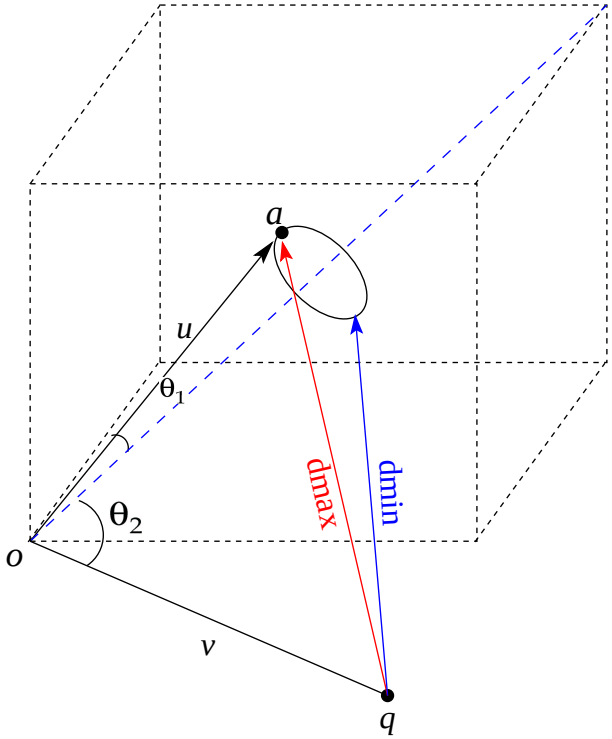


FIG. 3.7: Calcul de la distance minimale et de la distance maximale entre un vecteur requête et l'ensemble de vecteurs ayant la même approximation.

à calculer la distance la plus proche et la distance la plus lointaine entre le vecteur requête q et l'hypersphère ayant comme surface les points de code a. Si on note u le vecteur reliant o et a, et v le vecteur reliant o et q, alors dmin et dmax se calculent comme suit :

$$d_{min} = \|u\|^2 + \|v\|^2 - 2\|u\| \ \|v\| \ cos|\theta_1 - \theta_2| \tag{3.1}$$

$$d_{max} = \|u\|^2 + \|v\|^2 - 2\|u\| \ \|v\| \ cos(\theta_1 + \theta_2) \tag{3.2}$$

où θ_1 et θ_2 sont les angles entre la diagonale de la case et les vecteurs u et v respectivement.

Il est évident que le taux de filtrage de l'algorithme de recherche du LPC-File est meilleur que celui du VA-File car les approximations du LPC-File sont plus précises que celles du VA-File. Néanmoins, il est important de souligner que les calculs de dmin et dmax sont nettement plus coûteux dans le cas du LPC-File. Par ailleurs, le choix du nombre de bits de codage demeure problématique et les codes sont bien plus longs que ceux du VA-File.

3.4 La recherche approximative

Cette approche a été proposée après qu'il a été constaté que les performances de toutes les techniques de recherche des plus proches voisins *exacts* se dégradent inévitablement lorsque la dimension des données augmente. L'idée de la recherche approximative est de négocier une forte réduction du temps de recherche contre quelques imprécisions au niveau

des résultats (les plus proches voisins recherchés). Il ne s'agit plus alors de retrouver forcément les véritables k vecteurs les plus proches mais plutôt k vecteurs parmi les voisins proches du vecteur requête. Dans la suite, nous appelons plus proches voisins exacts, les véritables vecteurs les plus proches au vecteur requête, et, plus proches voisins approximatifs, les voisins retournés par une recherche approximative. La notion d'imprécision est définie en fonction de la méthode. Elle peut être fonction de la proportion des plus proches voisins exacts dans le résultat approximatif ou bien fonction de la distance entre le vecteur requête et les plus proches voisins approximatifs retournés.

Dans cette section, nous présentons un panorama des techniques qui ont été proposées. Nous les classons en deux grandes catégories selon qu'elles offrent ou non la possibilité de contrôler la précision du résultat obtenu.

3.4.1 Approximations sans contrôle de précision

Ces techniques permettent d'effectuer des recherches approximatives de plus proches voisins sans donner la possibilité à l'utilisateur de contrôler leur précision ni d'ailleurs de lui fournir la moindre indication sur le degré de précision des résultats retournés.

3.4.1.1 Approche par réduction de la dimension

Ces méthodes utilisent les techniques de réduction de la dimension afin de transformer les vecteurs de la base dans un espace de dimension plus petite que l'espace original puis indexent les données dans l'espace transformé. Elles misent sur une réduction de la dimension qui soit suffisante pour éviter les problèmes liés à la « malédiction de la dimension ».

Le principe ainsi que les limites de ces méthodes ont été déjà discutés dans les sections 2.5 et 2.6 du chapitre 2. Nous les citons de nouveau ici uniquement pour montrer la catégorie où elles peuvent être classées.

3.4.1.2 Arrêts prématurés

De façon générale, il est assez facile de concevoir un algorithme de recherche approximative basé sur le principe de l'arrêt prématuré. Ce type d'algorithme repère d'abord par divers moyens (filtrage, arborescences, etc.) les cellules susceptibles de contenir des vecteurs pouvant appartenir au résultat, ordonne ensuite ces cellules (par distance croissante au point requête par exemple), les accède les unes après les autres et arrête la recherche après la lecture d'un nombre de cellules arbitrairement fixé.

Le résultat est approximatif car certains vecteurs qui appartiendraient au résultat exact peuvent appartenir aux cellules ignorées. Un tel algorithme est efficace en temps de réponse (car peu de données sont lues) mais ne permet en aucune manière de contrôler la précision du résultat approximatif ni de renvoyer vers l'utilisateur la moindre indication du degré de précision atteint par la recherche accomplie.

Ce principe est repris notamment dans les travaux de Weber et Böhm à travers la version approximative du VA-File [Weber & Böhm, 2000], dans les travaux de Li *et al.* à travers CLINDEX [Li *et al.*, 2002], et dans les travaux de Ferhatosmanoglu *et al.* [Ferhatosmanoglu *et al.*, 2001]. Dans ce dernier travail, l'approche proposée est une hybridation du principe de l'arrêt prématuré et de l'approche par réduction de la dimension. Dans la phase de structuration des données, les vecteurs sont transformés par KLT

puis regroupés à l'aide d'un algorithme de *clustering*. Lors de la recherche, les *clusters* sont lus dans l'ordre de leur proximité par rapport au vecteur requête et le nombre de dimensions prises en compte dans les calculs de distance augmente progressivement au cours de la recherche. Les résultats sont alors améliorés soit par la prise en compte de plus de dimensions dans les calculs, soit par la lecture de plus de *clusters*. Cette technique cumule malheureusement à la fois les inconvénients cités dans cette section et ceux de l'approche par réduction de la dimension.

3.4.1.3 Hachage multidimensionnel

Le hachage multidimensionnel a été introduit par Indyk et Motwani dans [Indyk & Motwani, 1998] puis amélioré par Gionis *et al.* dans [Gionis *et al.*, 1999].

L'idée de base consiste à transformer les vecteurs en utilisant plusieurs fonctions de hachage. Chaque transformation vise à maximiser la probabilité de *collision* entre les vecteurs proches dans l'espace. Le but de chaque transformation est ainsi d'attribuer aux vecteurs similaires (donc proches) une même valeur. Comme plusieurs transformations sont appliquées à chaque vecteur, la probabilité d'avoir au moins une valeur transformée identique pour deux vecteurs très proches est renforcée. Cette propriété est exploitée pour effectuer des recherches de plus proches voisins approximatives car seuls les vecteurs de la base ayant au moins une valeur transformée commune avec le vecteur requête sont considérés lors d'une recherche.

La mise en œuvre proposée dans [Gionis *et al.*, 1999] procède tout d'abord par une transformation des vecteurs de la base vers un cube de Hamming. La distance de Hamming est ensuite utilisée comme mesure de similarité. Dans l'espace original, cette distance correspond à la distance L_1. Les fonctions de hachage utilisées sont de simples projections orthogonales sur les axes de l'espace de représentation. Le nombre de fonctions de hachage ainsi que le nombre de composantes utilisées par chaque projection sont des paramètres internes de l'algorithme. Plus le nombre de fonctions est important, plus la probabilité de collision est renforcée, et plus la recherche est précise.

Bien que les idées présentées dans ce travail soient originales et très simples, la technique telle qu'elle a été mise en œuvre possède beaucoup d'inconvénients et de limitations. Tout d'abord, seuls les vecteurs représentés dans un espace Euclidien et muni de la distance L_1 peuvent être gérés. Cette limitation en restreint sévèrement les domaines d'applications. De plus, le nombre de fonctions de hachage ainsi que le nombre de composantes pour les projections sont des paramètres difficiles à fixer et critiques, puisque la précision de la recherche en dépend. Même si une analyse théorique a été proposée pour régler automatiquement ces deux paramètres, aucune expérimentation n'a été réalisée pour sa validation. Par conséquent, aucun contrôle effectif et direct de la précision de la recherche ne peut être réalisé. Seule une bonne connaissance du fonctionnement de la technique et de la distribution des données permettent de contrôler la précision au travers des paramètres internes de l'algorithme.

3.4.2 Approximations avec contrôle de précision

Les techniques classées dans cette deuxième catégorie permettent d'effectuer des recherches approximatives de plus proches voisins avec contrôle de la précision des résultats.

3.4.2.1 Approximations géométriques

La « malédiction de la dimension » a notamment pour conséquence de diminuer fortement l'efficacité des règles de filtrage énoncées dans la section 1.3 du chapitre 1. L'efficacité de ces règles, c'est-à-dire leur capacité à éliminer beaucoup de cellules, est directement liée à la dimension de l'espace où elles sont appliquées. Elles sont efficaces en petite dimension ($\leqslant 16$ d'après [Weber *et al.*, 1998]) mais ne le sont plus au-delà. Cette absence d'efficacité est due aux particularités des espaces de grande dimension décrites dans la section 2.3.

Afin de restaurer l'efficacité de ces règles, certaines méthodes considèrent des approximations des tailles des cellules au lieu de leurs tailles exactes. Ces méthodes utilisent typiquement un paramètre supplémentaire ε lors du calcul des distances dmin et dmax. Considérer que les cellules sont « plus petites » qu'elles ne le sont en réalité permet notamment de diminuer le chevauchement entre cellules, renforce la sévérité des règles de filtrage et élimine donc plus de cellules comparativement à la recherche exacte.

Il se peut, néanmoins, que des cellules contenant des vecteurs intéressants soient éliminées. Les deux règles de filtrage 1.1 et 1.2, présentées dans la section 1.3.2 (page 16 et page 17), deviennent dans le cas des recherches approximatives basées sur des approches géométriques :

$$\begin{aligned} \text{si } \mathrm{dmin}(q, C_i) + \varepsilon &\geqslant \mathrm{dmax}(q, C_j) - \varepsilon \text{ alors éliminer } C_i \\ \text{si } \mathrm{dmin}(q, C_i) + \varepsilon &\geqslant d(q, nn_k) \text{ alors arrêter la recherche} \end{aligned}$$

Bien que ces méthodes reposent sur un principe très simple, le choix de la valeur de ε reste un problème ouvert. Un petit ε préserve la précision de la recherche mais ne garantit pas forcément une réduction significative du temps de réponse. À l'inverse, une grande valeur de ε permet de réduire fortement le temps de réponse mais ne garantit pas une bonne précision. De plus, les notions de grandes et petites valeurs sont très relatives. Elles dépendent de la distribution des vecteurs, de leur dimension et des ordres de grandeur de leurs composantes.

Weber et Böhm reprennent ce principe d'approximation pour améliorer les performance du VA-File [Weber & Böhm, 2000]. Il choisissent ε empiriquement en étudiant la distribution des distances inter-vecteurs de la base. Ils montrent ensuite expérimentalement que la valeur choisie est suffisamment élevée pour améliorer, dans la majorité des cas, les performances de la recherche tout en préservant un niveau de précision acceptable. Les principaux inconvénients de cette approche viennent de l'unicité du ε estimé et de la difficulté de son estimation. En effet, il est peu probable que tous les vecteurs soient distribués de manière analogue dans toutes les cellules, et appliquer le même ε pour toutes les cellules peut fortement biaiser le niveau de précision du résultat.

Sur le même principe, Ciaccia et Patella [Ciaccia & Patella, 2000] proposent une technique de recherche appelée AC-NN pour les structures arborescentes du type M-Tree. Dans AC-NN, la valeur de ε représente l'erreur relative maximale entre la distance du plus proche voisin exact du vecteur requête et la distance du plus proche voisin approximatif du vecteur requête. De nouveau, appliquer la même valeur de ε à l'ensemble des cellules est problématique. Par ailleurs, leurs expérimentations montrent que l'erreur relative constatée *a posteriori* est le plus souvent très inférieure à ε, ce qui remet en cause le principe même de réglage de ce paramètre.

Pour pallier ce dernier problème, et rendre le processus de recherche plus fiable, Ciaccia et Patella proposent également dans [Ciaccia & Patella, 2000] une technique appelée PAC-NN. Cette technique se base sur la même erreur relative ε, mais permet aussi de tolérer des erreurs relatives supérieures avec une certaine probabilité δ fixée elle aussi par l'utilisateur. À partir de ε et δ, la distance entre le vecteur requête et son plus proche voisin est estimée. La recherche commence ensuite puis s'arrête dès que la distance entre le vecteur requête et le plus proche voisin courant est supérieure à la distance estimée. Le principal problème de cette technique concerne l'estimation de la distance au plus proche voisin qui présuppose connue la distribution des données *autour du vecteur requête*. De plus, cette technique est intrinsèquement limitée à la recherche du plus proche voisin et ne peut être appliquée au cas de la recherche de k-plus proches voisins.

3.4.2.2 Approximations probabilistes

Ces approches utilisent des critères probabilistes pour contrôler la précision de la recherche. Il en existe principalement deux méthodes, DBIN et P-Sphere. Ces deux méthodes sont cependant limitées car elles permettent de rechercher uniquement le plus proche voisin. Leur extension à la recherche des k-plus proches voisins n'a pas été proposée.

Density-Based Indexing (DBIN) [Bennett *et al.*, 1999]. DBIN regroupe les vecteurs de la base en *clusters* à l'aide de l'algorithme EM *(Expectation Maximization)*. C'est un algorithme de *clustering* basé sur l'estimation paramétrique de la densité des données. Les données sont supposées être distribuées selon un modèle de mélange de lois gaussiennes où chaque gaussienne est un *cluster*. Cet algorithme sera détaillé dans la section 4.3.4.1, page 93.

Lors de la recherche, la probabilité d'appartenance du vecteur requête par rapport à chacun des *clusters* est calculée. Les *clusters* sont ordonnés en fonction de cette mesure. Après la lecture de chaque *cluster*, les distances entre le vecteur requête et les vecteurs du *cluster* sont calculées et l'ensemble des résultats est éventuellement mis à jour. DBIN utilise ensuite une analyse probabiliste pour estimer la probabilité de retrouver une meilleure solution dans les *clusters* non encore lus. La recherche s'arrête dès que cette probabilité devient inférieure à un seuil fixé. Celui-ci est le paramètre de contrôle de la précision.

Étant donné un vecteur requête q, son plus proche voisin courant p_c ($d_c = dist(q, p_c)$) et l'ensemble des *clusters* non encore lus $\{C_1, \ldots, C_n\}$, chaque *cluster* C_i est modélisé par une gaussienne de moyenne μ_i et de matrice de covariance Σ_i ($G(\mu_i, \Sigma_i)$). Si l'on note $B(q, d_c)$ l'hypersphère centrée en q et de rayon d_c, alors la probabilité de ne pas retrouver un vecteur plus proche du vecteur requête que le plus proche voisin courant p_c est la probabilité pour que la boule $B(q, d_c)$ soit vide ($P(B(q, d_c)\ est\ vide)$). La recherche de plus proches voisins s'arrête dès que cette probabilité est inférieure à un seuil fixé. Tout le processus de recherche repose donc sur le calcul de cette probabilité qui est égale à :

$$P(B(q, d_c)\ est\ vide) = \prod_{i=1}^{n} \left[1 - P(x \in B(q, d_c) \mid x \in C_i)\right]. \tag{3.3}$$

La probabilité $P\big(x \in B(q, d_c) \mid x \in C_i\big)$ est nulle si le *cluster* C_i a déjà été lu. Elle peut être non nulle pour les autres *clusters*.

En considérant le modèle gaussien estimé et en supposant que les vecteurs de chaque *cluster* ont été générés par le modèle associé, il est possible d'effectuer l'approximation suivante :

$$P\big(x \in B(q,d_c) \mid x \in C_i\big) \;\approx\; P\big(x \in B(q,d_c) \mid x \text{ a été généré par } G(\mu_i,\Sigma_i)\big). \tag{3.4}$$

Dans ce cas là :

$$P\big(x \in B(q,d_c) \mid x \in C_i\big) \approx P\big((x-q)^T D(x-q) \leqslant d_c^2 \mid x \text{ a été généré par } G(\mu_i,\Sigma_i)\big), \tag{3.5}$$

où D est une matrice semi-définie positive définissant la métrique associée aux vecteurs. Suivant cette matrice (D) et suivant la position du vecteur requête q, la densité de probabilité de la variable aléatoire $(x-q)^T D(x-q)$ est définie comme suit :

- une loi de χ^2 quand D est la matrice identité et $q = \mu_i$;
- une loi de χ^2 non centrée quand D est la matrice identité et $q \neq \mu_i$;
- une somme de lois de χ^2 non centrées dans le cas général.

Dans [Bennett *et al.*, 1999], les auteurs présentent uniquement les détails de calculs de cette probabilité dans le cas où la matrice de covariance Σ_i est une matrice diagonale et dans le cas où la métrique utilisée est euclidienne, c'est-à-dire $D = I$. Dans ce cas

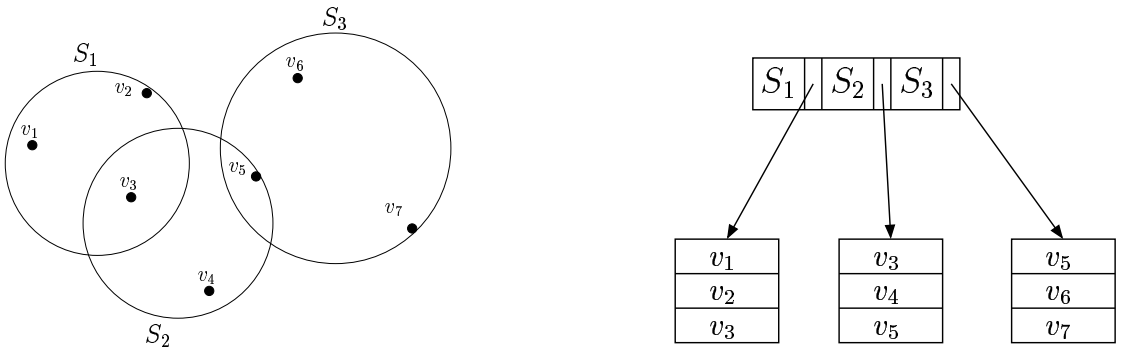
$$P\big((x-q)^T D(x-q) \leqslant d_c^2\big) = P\left[\sum_{j=1}^{n} \sigma_j^2 \chi^2(1,\delta_j^2) \leqslant d_c^2\right], \tag{3.6}$$

où $\delta_j = \frac{(q_j,\mu_{ij})}{\sigma_{ij}}$, μ_{ij} est le j^e élément de μ_i, σ_{ij} est le j^e élément de la diagonale de Σ_i et $\chi^2(1,\delta_j^2)$ est une loi de χ^2 à un degré de liberté et de paramètre de non-centralité δ_j^2. Pour être évaluée, cette fonction est exprimée sous la forme d'une série infinie.

Cette technique hérite des inconvénients de l'algorithme EM, notamment pour ce qui est du nombre maximum de gaussiennes (et donc de *clusters*) pouvant être correctement estimées à partir de l'ensemble de données. En pratique, ce nombre est relativement petit, rendant cette méthode inopérante dans le cas où l'on doit gérer de très larges volumes de vecteurs. De plus, les calculs nécessaires à l'estimation de la probabilité qui sert à interrompre la recherche reposent sur de nombreuses hypothèses pouvant remettre en cause la fiabilité de cette mesure si elles s'avèrent non vérifiées (distribution gaussienne des vecteurs, notamment).

P-Sphere Tree [Goldstein & Ramakrishnan, 2000]. Le P-Sphere Tree permet aussi d'effectuer des recherches approximatives de plus proches voisins avec contrôle probabiliste de la précision. Il offre, en plus, dans sa version déterministe, la possibilité de savoir si le plus proche voisin approximatif retourné est effectivement le plus proche voisin exact ou non. Dans le cas affirmatif, le plus proche voisin trouvé est dit « plus proche voisin prouvable ».

Le P-Sphere Tree est une structure arborescente à deux niveaux dont les régions gérées sont des hypersphères. Les centres et les rayons des hypersphères sont stockés dans la racine alors que les vecteurs sont stockés dans les feuilles. Chaque feuille est associée à une hypersphère et contient tous les vecteurs qui lui appartiennent (géométriquement). Un



(a) (b)

FIG. 3.8: Représentation géométrique et structure du P-Sphere Tree – v_3 et v_5 sont dupliqués.

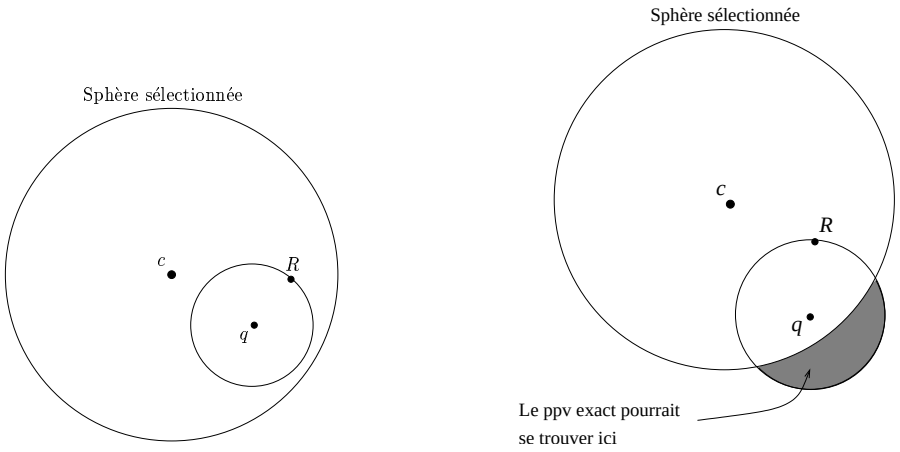
vecteur se trouvant dans la zone d'intersection de deux hypersphères est ainsi dupliqué dans les deux feuilles correspondantes. La figure 3.8 illustre la représentation géométrique d'un P-Sphere Tree et la structure de données associée.

La construction de l'arbre s'effectue en sélectionnant aléatoirement un ensemble de centres d'hypersphères dont le nombre est fixé par l'utilisateur. Les vecteurs sont ensuite répartis dans les feuilles selon la règle du plus proche centre. Les feuilles sont de taille identique mais les hypersphères associées peuvent être de rayons différents.

En présence d'un vecteur requête, la recherche se limite à explorer uniquement la feuille associée à l'hypersphère dont le centre est le plus proche du vecteur requête. Le résultat est alors approximatif. Cependant, dans certains cas où le plus proche voisin trouvé est le plus proche voisin exact, l'algorithme de recherche peut l'affirmer. Pour cela, il se base sur l'étude de la distance entre le vecteur requête et le centre de la sphère sélectionnée, et la distance entre le vecteur requête et le plus proche voisin trouvé. Si la somme de ces deux distances est inférieure au rayon de la sphère sélectionnée (cas de la figure 3.9.(a) où R est le plus proche voisin trouvé), alors il n'est pas possible de trouver un meilleur plus proche voisin car si celui-ci existait, il aurait appartenu à la sphère examinée et donc aurait été retrouvé lors de la recherche. Par contre, dans le cas où cette somme est supérieure au rayon de la sphère sélectionnée (cas de la figure 3.9.(b)), un meilleur plus proche voisin que R pourrait être retrouvé car d'éventuels vecteurs se trouvant à l'extérieur de la sphère sélectionnée et dont la distance au vecteur requête est inférieure à la distance au plus proche voisin trouvé n'auront pas été examinés. La zone de l'espace où de tels vecteurs pourraient être retrouvés est représentée dans la figure 3.9.(b) en gris.

Le nombre de vecteurs stockés dans chaque feuille est donc le seul élément qui détermine à la fois la précision de la recherche, sa rapidité et le coût de stockage du P-Sphere Tree. Plus ce nombre est important et plus il y a de chances de trouver le plus proche voisin exact lors de la recherche, cependant, la recherche est plus lente et la taille de l'arbre est plus grande.

Pour fixer la taille des feuilles, le P-Sphere Tree utilise un échantillon de vecteurs requêtes. Pour une valeur donnée de ce paramètre, cet échantillon sert à calculer la proportion des vecteurs requêtes pour lesquels il est possible de trouver le plus proche voisin



(a) (b)

FIG. 3.9: Plus proche voisin prouvable. (a) Le plus proche voisin trouvé R est le plus proche voisin exact. (b) Impossible de certifier que le plus proche voisin trouvé R est le plus proche voisin exact.

exact prouvable. La valeur retenue pour la taille des feuilles est celle qui garantit que la probabilité estimée en utilisant l'échantillon est au moins égale à la probabilité exigée par l'utilisateur. L'inconvénient majeur de cette technique est sa sensibilité au choix de l'échantillon des vecteurs requêtes. Comme le processus de contrôle de la précision repose entièrement sur cet échantillon, alors si celui-ci n'est pas assez représentatif, la fiabilité du contrôle de la précision sera remise en cause.

Par ailleurs, la duplication des vecteurs se trouvant dans des zones d'intersection d'hypersphères peut entraîner une forte croissance de la taille de l'arbre. Les expériences rapportées dans [Goldstein & Ramakrishnan, 2000] montrent que le facteur de croissance de la taille de l'arbre par rapport à la taille originale de la base peut atteindre des valeurs très élevées (une de leurs expériences génère une base 98 fois plus grande que la base originale).

D'autres variantes de cette approche ont été proposées. Dans la version « non déterministe » du P-Sphere Tree, le calcul de la taille des feuilles ne différencie pas les plus proches voisins exacts prouvables de ceux qui ne le sont pas. En d'autres termes, l'option qui permettait d'affirmer l'exactitude du plus proche voisin n'est plus prise en compte, ce qui rend la construction du P-Sphere Tree moins contrainte et donc permet de réduire le facteur de croissance de la taille de l'arbre. Dans une autre version de cette approche, le Pk-Sphere Tree, les k sphères les plus proches au vecteur requête sont considérées. Ce qui permet une nouvelle fois de réduire le facteur de croissance de la taille de l'arbre mais rend sa construction plus complexe.

3.5 Synthèse

Dans ce chapitre, nous avons présenté les nouvelles approches pour la recherche de plus proches voisins. Ces approches ont été proposées pour combler l'insuffisance des techniques traditionnelles et leur inefficacité dans les espaces de grande dimension. En particulier,

nous avons présenté les méthodes de recherche approximative. Celles-ci semblent être les plus prometteuses pour faire face au problème de la malédiction de la dimension. Leur principe de base est d'échanger une petite imprécision lors de la recherche contre une forte réduction du temps de réponse. Nous avons classé ces techniques en deux grandes familles selon qu'elles permettent de contrôler la précision de la recherche ou non. La propriété de contrôle de la précision est une propriété extrêmement importante pour les applications qui utilisent ces méthodes. En effet, un résultat approximatif sans aucune indication sur sa précision est difficilement exploitable en pratique. Les méthodes permettant un contrôle de la précision ont été classées en deux catégories en fonction de la nature de contrôle proposés : géométrique ou probabiliste. Dans les méthodes géométriques, c'est la distance entre les plus proches voisins retournés et le vecteur requête qui est contrôlée, alors que dans les méthodes probabilistes, c'est la probabilité de ne pas retrouver les plus proches voisins recherchés qui est contrôlée. Le passage en revue de ces méthodes a permis de montrer leurs limites qui sont principalement relatives à la procédure et aux paramètres de contrôle. Dans le cas des méthodes géométriques, le choix d'un paramètre de contrôle basé sur les distances inter-vecteurs nécessite une bonne connaissance de la distribution des vecteurs. Les méthodes probabilistes sont quant à elles plus intuitives mais ne permettent de retrouver que *le* plus proche voisin du vecteur requête. Leur extension à la recherche de k-plus proches voisins n'a pas été proposée.

Deuxième partie

Méthode proposée pour la recherche de plus proches voisins

Introduction

L'étude menée dans les trois chapitres de la première partie a montré l'insuffisance des techniques existantes pour la recherche de plus proches voisins : les performances des techniques traditionnelles pour la recherche exacte se dégradent exponentiellement lorsque la dimension augmente ; les techniques de recherche approximative sont difficilement utilisables dans un système réel. Pour certaines d'entres elles, aucun contrôle de la précision de la recherche n'est proposé, et, pour les autres, le contrôle est mis en œuvre au travers de nombreux paramètres compliqués ou très difficiles à fixer. La recherche approximative reste néanmoins l'approche la plus prometteuse pour faire face à la malédiction de la dimension.

Dans cette partie, nous présentons une nouvelle méthode pour la recherche approximative de plus proches voisins. Elle permet de contrôler la précision de la recherche de façon probabiliste au travers d'un seul paramètre α appelé niveau d'imprécision. α est « la probabilité maximale de ne pas retrouver un plus proche voisin dans le résultat approximatif ». Ce paramètre a été choisi de façon à permettre à n'importe quel utilisateur de contrôler la précision de la recherche même s'il n'a aucune connaissance sur l'algorithme et sur la distribution et la nature des données.

Comme toute méthode de recherche de plus proches voisins, cette méthode se compose de deux phases : une phase hors ligne de structuration des données et une phase en ligne d'interrogation de la base. La phase hors ligne a pour objectif de regrouper les données en petits paquets les plus compacts et les plus séparés possibles. Ces deux propriétés (la compacité et la séparabilité) sont essentielles afin que les règles de filtrage utilisées par les algorithmes de recherche puissent être efficaces (*cf.* section 1.3.2, page 16). Pour cela, nous utilisons un algorithme de *clustering*. L'algorithme utilisé est inspiré de l'algorithme BIRCH. Il est décrit dans le chapitre suivant.

Les paquets sont ensuite englobés dans des hypersphères de rayons sous-estimés. L'approximation envisagée est ainsi d'ordre géométrique et concerne les rayons des hypersphères englobantes. Cette réduction des rayons des hypersphères permet d'accroître l'efficacité des règles de filtrage lors de la recherche. Par contre, elle introduit de l'imprécision dans l'algorithme de recherche car certains paquets contenant des plus proches voisins recherchés risquent de ne pas être sélectionnés lors de la recherche. Le calcul des rayons approximatifs s'effectue cependant en fonction du niveau d'imprécision. Il garantit que l'imprécision du résultat approximatif retourné est inférieure au seuil d'imprécision toléré par l'utilisateur et formulé au travers du paramètre α.

Plan de présentation de la méthode Le chapitre suivant présente l'algorithme de formation de paquet utilisé. Nous commencerons par la présentation des propriétés importantes que devra satisfaire cet algorithme. Nous donnerons ensuite un état de l'art

des algorithmes de *clustering* classés en quatre grandes catégories. Enfin, nous présenterons l'algorithme que nous proposons et qui est inspiré de l'algorithme BIRCH.

Dans le chapitre 5, seront présentés l'algorithme détaillé pour la recherche de plus proches voisins ainsi que la stratégie de calcul des rayons approximatifs des *clusters*.

Enfin, les résultats des expérimentations effectuées seront présentés dans le chapitre 6. Outre des études de temps de réponse et de précision de la recherche, nous présenterons des expériences qui permettent de corroborer les choix et les différentes analyses effectuées lors de la conception de la méthode.

Chapitre 4

Regroupement des données hors-ligne

Nous présentons dans ce chapitre l'algorithme de regroupement en paquets des vecteurs de la base. Comme nous l'avons expliqué précédemment, nous utilisons pour cela un algorithme de clustering. Tout d'abord, nous présentons les contraintes spécifiques à l'indexation, c'est-à-dire les conditions que doit satisfaire l'algorithme de partitionnement afin d'avoir des paquets de données adaptés à la recherche de plus proches voisins. Nous donnons également la différence par rapport aux objectifs initiaux du clustering et nous dressons ensuite un panorama des méthodes de clustering que nous classons en quatre grandes catégories. Enfin, en nous inspirant de l'algorithme de clustering BIRCH, nous proposons un algorithme pour le regroupement des données hors ligne. Cet algorithme est détaillé dans la section 4.4.

4.1 *Clustering :* principe et applications

Le *clustering* permet de partitionner un ensemble de vecteurs en utilisant uniquement la proximité des vecteurs dans l'espace, l'objectif étant d'effectuer un partitionnement dans lequel les vecteurs proches dans l'espace se retrouvent dans le même groupe. Ces groupes sont appelés *clusters* et l'ensemble des *clusters* forme une partition. Certains vecteurs isolés *(outliers)* qui ne peuvent être associés à aucun *cluster* sont considérés comme du bruit. Ils sont isolés et traités séparément. Cependant, il est à noter que tous les algorithmes de *clustering* ne sont pas capables d'identifier et d'isoler ces vecteurs.

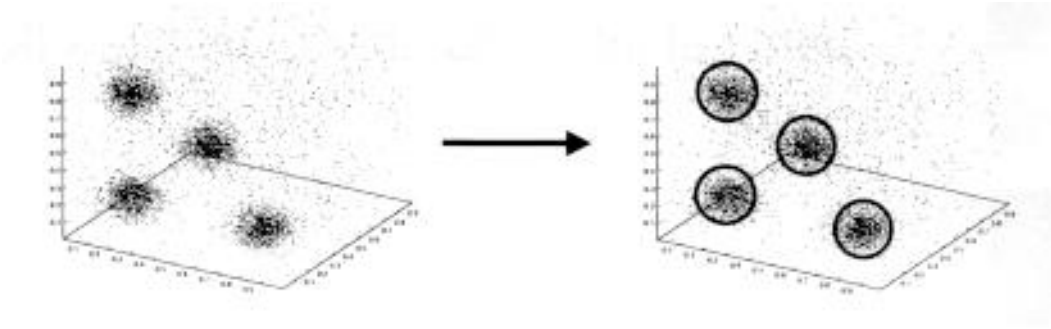


FIG. 4.1: Exemple d'une distribution de données en dimension 3 et une partition associée.

La figure 4.1 montre un exemple d'une distribution de données de dimension 3 que l'on peut partitionner en quatre *clusters* auxquels se rajoutent un ensemble de vecteurs isolés (ici, distribués uniformément dans l'espace).

De manière générale, l'objectif des algorithmes de *clustering* se limite au partitionnement de l'ensemble des vecteurs. L'analyse et l'exploitation des *clusters* est une tâche qui incombe à l'application qui fait appel au *clustering*.

En *data mining* [Grabmeier & Rudolph, 2002], le *clustering* sert notamment à former des groupes de vecteurs d'attributs similaires qui permettent ensuite de déduire des dépendances ou des relations « causes / effets » entre les attributs. En traitement d'images, une des applications du *clustering* est la segmentation d'images [Hermes *et al.*, 2002], c'est-à-dire le partitionnement de l'image en régions homogènes. Pour cela, un vecteur d'attributs est calculé pour décrire chaque pixel (sa couleur, sa position, son voisinage, etc.) et une mesure de similarité est définie entre les vecteurs d'attributs. Le *clustering* permet alors de partitionner les pixels en groupes similaires où chaque groupe est considéré comme une région homogène de l'image.

4.2 Application à la l'indexation

Le *clustering* produit habituellement un nombre peu élevé de *clusters*. Il est toutefois possible de s'en servir pour la formation de paquets dans le domaine de l'indexation et de la recherche de plus proches voisins car le principe de base reste le même. Cependant, l'objectif étant différent, le cahier des charges s'en trouve lui aussi modifié. Ainsi, en entrée, les données à traiter sont de grande dimension et en nombre très important. En sortie, un nombre relativement élevé de petits *clusters* et éventuellement un ensemble de vecteurs isolés sont attendus.

De notre point de vue, les propriétés requises ou du moins souhaitables pour qu'un algorithme de *clustering* puisse être utilisé dans ce contexte sont résumées dans les points suivants.

L'efficacité. C'est une des conditions les plus importantes. En effet, les bases d'images sont de grande taille et, le plus souvent, croissante. Même si cette phase de regroupement est exécutée hors-ligne, il est très difficile, par exemple, d'envisager l'utilisation d'un algorithme de complexité quadratique en nombre de vecteurs à partitionner.

Compacité et équilibre des *clusters*. Il est important d'avoir des *clusters* très compacts qui se chevauchent le moins possible afin d'accroître l'efficacité des règles de filtrage et de confiner ainsi la recherche à un sous-ensemble de paquets le plus petit possible. Il est également souhaitable d'avoir des *clusters* équilibrés en nombre de vecteurs, c'est-à-dire qui contiennent à peu près le même nombre de vecteurs. Une grande différence entre les nombres de vecteurs des *clusters* peut entraîner la lecture d'un nombre important de vecteurs de la base même si le taux de filtrage des *clusters* est élevé. Si par exemple un *cluster* compte à lui seul 30 % de la totalité des vecteurs de la base, alors il suffit que ce *cluster* soit sélectionné pour lire *au moins* 30 % de la totalité de la base.

Comportement dans les espaces de grande dimension. Les données manipulées dans le domaine de l'indexation sont des vecteurs de grande dimension. Par conséquent, il est important que l'algorithme de *clustering* prenne explicitement en compte ce facteur « dimension ». En effet, à l'instar de toutes les méthodes destinées à manipuler des données de grande dimension, les algorithmes de *clustering* sont également affectés par la « malédiction de la dimension ».

Identification du bruit. L'algorithme de *clustering* doit pouvoir identifier les vecteurs isolés. Autrement, ces vecteurs seront associés à des *clusters*. Auquel cas les *clusters* sont contraints de s'agrandir considérablement pour les absorber. Dans le cadre des applications de recherche de plus proches voisins, ceci entraînera généralement un fort taux de chevauchement entre les cellules associées aux *clusters* surtout en grande dimension, et rendra ainsi les règles de filtrage complètement inefficaces.

Après la présentation d'un état de l'art des méthodes de *clustering* dans la section suivante, nous utilisons les critères précédemment cités pour discuter le choix de l'algorithme à retenir pour être adapté et utilisé pour la mise en paquets des données.

4.3 État de l'art des méthodes de *clustering*

Nous présentons dans cette section un état de l'art des algorithmes de *clustering*. Notons cependant que cet état de l'art n'est pas exhaustif. Nous nous sommes limités aux algorithmes les plus cités qui permettent de gérer des données numériques définies dans des espaces métriques. Les algorithmes qui permettent de traiter des données symboliques, qualitatives ou bien des données définies dans des espaces non métriques ne sont pas présentés car ils se situent en dehors de notre contexte applicatif. Pour un état de l'art plus complet, le lecteur intéressé pourra se référer à [Jain & Murty, 1999, Grabmeier & Rudolph, 2002].

4.3.1 Classification des techniques existantes

Les algorithmes de *clustering* présentés ici sont classés en quatre catégories :

1. approches basées sur le partitionnement des données ;
2. approches hiérarchiques ;
3. approches basées sur l'estimation de la densité ;
4. approches basées sur le partitionnement de l'espace.

Cependant, il est à noter que cette classification n'est pas stricte car certaines méthodes utilisent plusieurs principes à la fois et peuvent être considérées comme des méthodes hybrides. La figure 4.2 donne un résumé de cette classification et énumère les algorithmes que nous citons dans la suite.

4.3.2 Approches basées sur le partitionnement des données

Les méthodes basées sur le partitionnement des données permettent de partitionner l'ensemble des vecteurs sur k *clusters*. Le nombre de *clusters* k est fixé *a priori* par l'utilisateur. Ces méthodes procèdent généralement par un choix d'une partition initiale

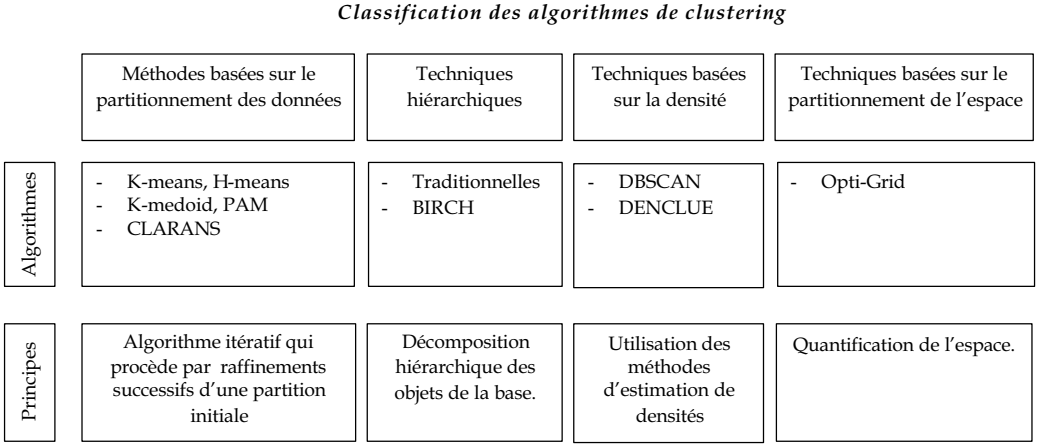


FIG. 4.2: Classification et principes des algorithmes de *clustering* présentés.

qui est ensuite améliorée grâce à un algorithme itératif. Un représentant est choisi pour chaque *cluster*. Il s'agit généralement du centre de gravité ou du médoïde[1].

L'objectif de ces méthodes est de minimiser l'inertie inter-*clusters* définie par :

$$I = \sum_{j=1}^{k} \sum_{i=1}^{n_j} sim(x_{ij}, c_j), \tag{4.1}$$

où k est le nombre de *clusters*, c_j le représentant du *cluster* j (centroïde, médoïde ou tout simplement un vecteur du même espace que les vecteurs de données), n_j le nombre de vecteurs appartenant au *cluster* j, x_{ij} les vecteurs du *cluster* j et sim la mesure de similarité associée aux vecteurs.

Parmi les algorithmes basés sur le partitionnement des données les plus connus et aussi les plus utilisés, on retrouve le *K-Means*. Cet algorithme est également connu sous le nom de « nuées dynamiques » ou encore sous le nom de « partitionnement autour de centres mobiles ». Il permet de partitionner des données dans un espace muni de la distance euclidienne. Le critère d'optimisation est donc :

$$I = \sum_{j=1}^{k} \sum_{i=1}^{n_j} D_2(x_{ij}, c_j), \tag{4.2}$$

où D_2 est la distance euclidienne.

La minimisation de ce critère est en fait un problème d'optimisation dont le but est d'explorer l'espace de définition des centres et de trouver ceux qui minimisent l'inertie inter-*clusters*. Le *K-Means* utilise pour cela une méthode itérative qui utilise une optimisation par descente de gradient.

[1] Le médoïde d'un *cluster* est un vecteur qui appartient au *cluster* et qui se trouve au centre de celui-ci. Une définition plus détaillée du médoïde est donnée plus loin (à la fin de la section 4.3.2, page 88).

Si on suppose qu'une partition initiale existe et si on considère dans un premier temps que les vecteurs sont de dimension 1, le problème se formule comme suit : soient k *clusters* $P_1, P_2, \ldots, P_k$ où chaque *cluster* P_j contient n_j vecteurs $x_{.j}$, calculer les c_j de la partition suivante qui minimisent le critère d'optimalité 4.2. Il s'agit alors de déterminer les c_j qui annulent les dérivées partielles de 4.2. En d'autres termes, il s'agit de résoudre le système d'équations suivant :

$$\text{pour } j = 1, \ldots, k \ : \qquad \frac{\partial I}{\partial c_j} = 0, \tag{4.3}$$

$$\frac{\partial}{\partial c_j} \sum_{j=1}^{k} \sum_{i=1}^{n_j} (x_{ij} - c_j)^2 = 0. \tag{4.4}$$

Après quelques étapes de calcul simples, la solution du système est :

$$\text{pour } j = 1, \ldots, k \ : c_j = \frac{1}{n_j} \sum_{i=1}^{n_j} x_{ij}. \tag{4.5}$$

On remarque que les c_j sont les centres de gravité des partitions. L'extension de ce résultat lorsque la dimension est supérieure à 1 est évidente.

Ainsi, à partir d'une partition initiale, il suffit de calculer les centres de gravité des *clusters* pour avoir une partition meilleure, c'est-à-dire dont l'inertie inter-*clusters* est plus faible. Comme c'est une descente de gradient, une solution *localement optimale* est atteinte lorsque deux partitions successives sont identiques.

L'algorithme *K-Means* se résume ainsi aux étapes suivantes :

1. choisir k partitions initiales $P_1, P_2, \ldots, P_k$;
2. calculer les centres de gravité des partitions ;
3. répartir les vecteurs sur les centres de gravité selon la règle du plus proche centre, c'est-à-dire chaque vecteur est attribué à la partition dont le centre lui est le plus proche ;
4. si la nouvelle partition est identique à la précédente alors l'algorithme s'arrête et renvoie comme résultat final la partition courante, sinon reprendre l'algorithme en 2.

Les inconvénients majeurs du *K-Means* sont ceux hérités de la procédure d'optimisation qu'il utilise, c'est-à-dire la descente de gradient. Il est bien connu que celle-ci est très sensible aux conditions initiales et souffre des problèmes des minima locaux. Par conséquent, le *K-Means* est très sensible au choix de la partition initiale et la partition finale n'est qu'une solution parmi d'autres et ne minimise le critère 4.2 que localement. Le *K-Means* est donc un algorithme non déterministe puisque deux partitions initiales différentes donneront deux partitions finales différentes.

Par ailleurs, le *K-Means* ne permet pas d'identifier les vecteurs isolés et nécessite $m.k.N$ opérations de calculs de distance (m est le nombre d'itérations et N le nombre de vecteurs de la base).

Notons enfin que plusieurs améliorations possibles du *K-Means* ont été proposées. Il s'agit du principe des « formes fortes » et du principe de « transfert ». Le premier consiste à exécuter plusieurs *K-Means* en faisant varier les conditions initiales et à détecter les

clusters qui sont formés à chaque fois indépendamment des conditions initiales. Ceux-ci sont appelés formes fortes. Le principe de « transfert » quant à lui permet d'effectuer des migrations aléatoires de vecteurs entre *clusters* en cours de formation dans le but de modifier le parcours de la recherche de la partition optimale, c'est-à-dire le chemin de la descente du gradient. Ceci permet de mieux explorer l'espace de recherche.

Sur le même principe que le K-Means, le *K-Medoid* permet de partitionner un ensemble de vecteurs définis dans un espace pas forcément métrique en k *clusters* et utilise les médoïdes comme représentants des *clusters*. Le médoïde d'un *cluster* est un vecteur qui appartient au *cluster* et qui se trouve au centre de celui-ci, au sens où c'est le vecteur qui minimise la somme des distances entre lui et tous les autres vecteurs du *cluster*. Il est utilisé comme représentant du *cluster* notamment dans les espaces non métriques. Dans ces espaces, le centroïde n'est pas défini. Il est également utilisé dans certaines applications où le représentant d'un *cluster* doit nécessairement être un vecteur du *cluster*. Malheureusement, l'algorithme qui permet de déterminer le médoïde d'un *cluster* est de complexité quadratique. Ceci constitue l'un des principaux problèmes de cet algorithme et de tous les algorithmes qui utilisent les médoïdes. Nous ne détaillons pas davantage cet algorithme, ni d'ailleurs d'autres algorithmes comme PAM et CLARANS [Ester *et al.*, 1995] qui sont tous destinés à partitionner des vecteurs définis dans des espaces non métriques.

4.3.3 Approches hiérarchiques

Les méthodes hiérarchiques se basent sur la construction d'une hiérarchie dont chaque niveau représente une partition possible des données et où les partitions associées aux niveaux inférieurs sont plus « fines » que celles associées aux niveaux supérieurs. Dans cette catégorie d'algorithmes, on trouve les méthodes traditionnelles qui utilisent des arbres binaires indicés (les dendrogrammes) et l'algorithme BIRCH.

4.3.3.1 Méthodes traditionnelles

Le principe de ces méthodes consiste à construire un arbre binaire indicé appelé dendrogramme. Le niveau le plus bas de l'arbre correspond à la partition la plus « fine » possible où chaque vecteur forme un *cluster*. Le niveau le plus haut correspond à la partition constituée d'un seul *cluster* englobant l'ensemble des vecteurs de la base. Entre les deux, plusieurs partitions intermédiaires peuvent être déduites en coupant l'arbre horizontalement. Chaque nœud correspond à un *cluster* et englobe l'ensemble des vecteurs contenus dans le sous-arbre correspondant. Sa hauteur dans l'arbre est égale à son indice de dispersion qui correspond à une mesure de proximité entre les points qu'il contient. La figure 4.3 présente un exemple d'un dendrogramme construit à partir de 9 vecteurs.

La construction de l'arbre nécessite la définition d'une mesure d'écart entre groupes de vecteurs. C'est cette mesure qui détermine la hauteur du nœud qui résulte de la fusion de deux groupes de vecteurs. Elle caractérise également le niveau de dispersion des vecteurs au sein d'un même groupe. Il existe de nombreuses manières de calculer cette distance entre groupes. Les plus simples considèrent la distance entre centres de gravité, la distance minimale ou la distance maximale entre les vecteurs des deux groupes. Les plus complexes utilisent la variance des vecteurs [Lebart *et al.*, 1984]. La construction de l'arbre peut être initiée par une approche ascendante *(bottum-up)* ou descendante *(top-down)*.

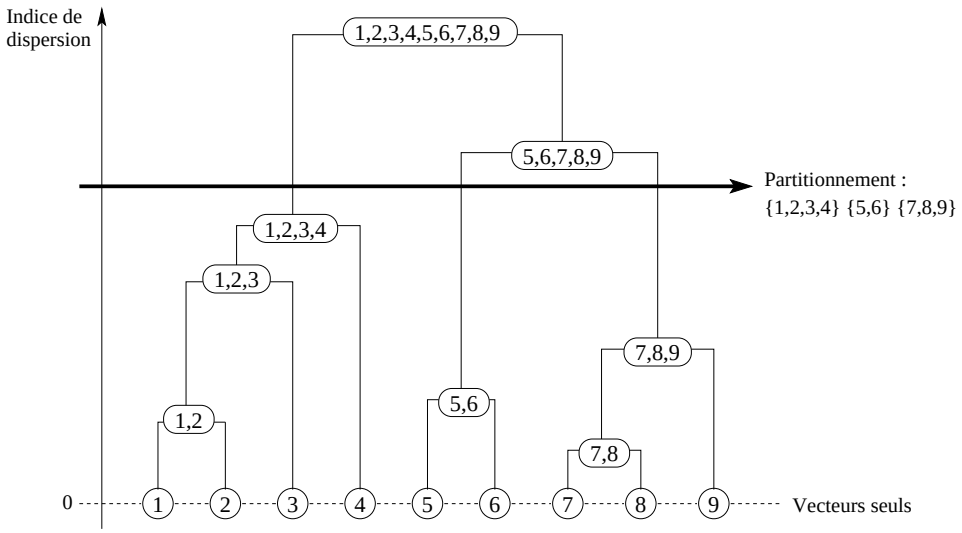


FIG. 4.3: Exemple d'un arbre binaire indicé construit à partir d'un ensemble de neuf vecteurs. Chaque nœud de l'arbre correspond à un *cluster*. Sa hauteur dans l'arbre est définie par son indice de dispersion. Une partition est construite en coupant l'arbre horizontalement.

Approche ascendante [Lance & Williams, 1967]. Cette approche procède par des regroupements successifs de groupes de vecteurs. Elle part d'une partition initiale où chaque vecteur constitue un groupe. À chaque itération, les deux groupes les plus proches au sens de la mesure d'écart choisie sont regroupés. Chaque regroupement donne lieu à la création d'un nœud dans l'arbre dont la hauteur correspond à l'écart entre les deux groupes qui ont servi à sa création. Ainsi, à chaque itération, le nombre de groupes de vecteurs diminue de 1. Par conséquent, si le nombre de vecteurs de la base est de N, le nombre total d'itération est de $N-1$. Comme à chaque itération, l'algorithme permettant de déterminer les deux groupes les plus proches est de complexité quadratique, la complexité globale de l'approche ascendante est $O(N^3)$.

Approche descendante [Hubert, 1973]. À l'inverse de l'approche ascendante, l'approche descendante part d'un groupe unique contenant l'ensemble des vecteurs de la base et applique récursivement un algorithme de scission de groupe de vecteurs en deux. Pour effectuer cette scission, toutes les distances deux à deux entre les vecteurs d'un groupe sont calculées et triées par ordre décroissant. Les deux vecteurs les plus éloignés sont placés chacun dans un nouveau groupe et la liste des couples de vecteurs triés en fonction de leurs distances est ensuite parcourue. Pour chaque couple de vecteurs, trois possibilités sont à considérer :

1. **si** les deux vecteurs sont déjà placés dans les deux nouveaux groupes **alors** passer au couple suivant ;
2. **si** seulement l'un des vecteurs est déjà placé dans un des deux nouveaux groupes en cours de création **alors** le second vecteur est placé dans l'autre nouveau groupe ;

3. **si** aucun des deux vecteurs n'est déjà placé **alors** placer chacun des deux vecteurs dans un nouveau groupe.

Tout comme l'approche ascendante, la complexité de l'approche descendante est $O(N^3)$.

Une des propriétés intéressantes de ces deux algorithmes hiérarchiques (ascendant et descendant) est la stabilité de leurs résultats. En effet, ce sont les deux seuls algorithmes de *clustering* qui n'ont aucun paramètre interne et qui ne requièrent pas le choix d'une configuration initiale (e.g. un partitionnement initial). Ils ont également l'avantage d'être indépendants de l'ordre d'insertion des vecteurs. Malheureusement, leur complexité exorbitante ($O(N^3)$) les rend difficilement utilisables dans des applications réelles traitant de très gros ensembles de données.

4.3.3.2 BIRCH

BIRCH [Zhang *et al.*, 1996] *(Balanced Iterative Reducing and Clustering using Hierarchies)* est un algorithme de *clustering* qui s'exécute en quatre étapes. Il a été conçu pour partitionner des ensembles de données de très grande taille qui ne tiennent pas forcément en mémoire centrale.

La première étape de BIRCH construit une hiérarchie appelée CF-Tree *(Clustering Feature Tree)* dont la taille ne dépasse pas la taille mémoire allouée à l'exécution de l'algorithme. Cette première étape permet également d'isoler les vecteurs éloignés (les *outliers*). Les feuilles de cette hiérarchie contiennent des vecteurs appelés CF-vecteurs. Chacun est associé à un micro-*cluster* (un petit paquet de vecteurs) et décrit les vecteurs qu'il contient. Il est par exemple possible de calculer le centre de gravité d'un micro-*cluster* à partir de son CF-vecteur. Les CF-vecteurs sont considérés comme un résumé de l'ensemble des vecteurs de la base et sont utilisés lors des deuxième et troisième étapes pour créer les *clusters*.

La deuxième étape est optionnelle. Elle permet de réduire le nombre de CF-vecteurs si celui-ci est jugé trop élevé car, lors de la troisième étape, un algorithme de *clustering* hiérarchique traditionnel (ascendant) est utilisé pour partitionner les centres de gravité des micro-*clusters*. Cette façon de travailler sur des « représentants » des vecteurs de la base au lieu des vecteurs eux-mêmes permet à BIRCH de s'attaquer à des bases de vecteurs de très grande taille et constitue sa principale originalité.

Enfin, la quatrième étape, optionnelle également, permet de répartir l'ensemble des vecteurs de la base sur les centres de *clusters* issus de la troisième étape. Elle permet d'affiner la partition qui a été réalisée en utilisant uniquement les représentants des vecteurs. Cette dernière étape peut être exécutée plusieurs fois pour améliorer la qualité de la partition des données. En fait, cette étape revient finalement à effectuer un *K-Means*.

Dans cette section, nous détaillons uniquement la première étape. En effet, les troisième et quatrième étapes correspondent à des algorithmes déjà présentés dans les sections 4.3.3.1 et 4.3.2, et la deuxième étape est simplement une extension de la première.

CF-vecteurs et CF-Tree. Le concept de CF-vecteur *(Clustering Feature Vector)* est à la base de la première phase de BIRCH. C'est un triplet qui résume l'essentiel de l'information sur un *cluster* de vecteurs $(\vec{X}_1, \ldots, \vec{X}_N)$. Il est défini par : $CF = (N, \vec{LS}, SS)$, où N est le nombre de vecteurs du *cluster*, $\vec{LS}$ est la somme des vecteurs de *cluster* $(\sum_{i=1}^{N} \vec{X}_i)$, et SS est la somme des carrés des vecteurs $(\sum_{i=1}^{N} \vec{X}_i^2)$.

À partir du CF-vecteur d'un *cluster*, il est notamment possible de calculer le centre de gravité $\vec{X}_0$ du *cluster*, son rayon moyen R (distance moyenne d'un vecteur au centre du *cluster*) et son diamètre moyen D (distance moyenne entre deux vecteurs du *cluster*). Ces trois entités sont données par les formules suivantes :

$$\vec{X}_0 = \frac{\sum_{i=1}^{N} \vec{X}_i}{N} \tag{4.6}$$

$$R = \left(\frac{\sum_{i=1}^{N} (\vec{X}_i - \vec{X}_0)^2}{N} \right)^{1/2} \tag{4.7}$$

$$D = \left(\frac{\sum_{j=1}^{N} \sum_{i=1}^{N} (\vec{X}_i - \vec{X}_j)^2}{N(N-1)} \right)^{1/2} \tag{4.8}$$

Leur calcul à partir du CF-vecteur du *cluster* est simple à établir.

Il est également possible de déduire plusieurs distances inter-*clusters* à partir des CF-vecteurs. Il s'agit notamment de la distance euclidienne entre centres de gravité (notée D_0), la distance L_1 entre centres de gravité (notée D_1), la distance inter-*clusters* moyenne (notée D_2) et la distance intra-*clusters* moyenne (notée D_3). Nous ne donnons pas les formules de ces mesures, le lecteur intéressé pourra se référer à [Zhang *et al.*, 1996].

Par ailleurs, les CF-vecteurs possèdent une propriété d'additivité qui permet de déduire facilement le CF-vecteur d'un micro-*cluster* résultant de la fusion de deux micro-*clusters* à partir de leurs CF-vecteurs. En d'autres termes, si deux micro-*clusters* de CF-vecteurs CF_1 et CF_2 fusionnent, alors le CF-vecteur du micro-*cluster* résultant est :

$$\mathrm{CF}_1 + \mathrm{CF}_2 = (N_1 + N_2, \vec{LS}_1 + \vec{LS}_2, SS_1 + SS_2).$$

Création du CF-Tree. La création du CF-Tree s'effectue par insertions successives de la même manière que dans les B-Tree et les R-Tree. Cette construction se base sur un paramètre clé appelé Ft qui est le diamètre (ou le rayon) maximal des micro-*clusters*. Le premier vecteur inséré forme le premier micro-*cluster*. Le deuxième est soit absorbé par le premier micro-*cluster* si le diamètre (ou le rayon) du micro-*cluster* résultant ne dépasse pas le seuil Ft ou bien entraîne la création d'un nouveau micro-*cluster*. Et ainsi de suite pour tous les vecteurs à insérer. Chaque nœud ou feuille de l'arbre est de taille limitée et fixée *a priori* (typiquement de même taille qu'une page du disque). Ainsi, lorsqu'une page est saturée, elle est partitionnée en deux et un nœud père est créé.

Il est évident que le paramètre Ft contrôle la taille de l'arbre. Une grande valeur de ce paramètre entraîne la création d'un CF-Tree de petite taille car un nombre important de vecteurs seront absorbés par le même micro-*cluster*. À l'inverse, une petite valeur de Ft permet de créer un nombre très important de micro-*clusters*, et par conséquent, fait

accroître la taille globale de l'arbre. L'objectif étant de construire un CF-Tree qui puisse tenir dans un espace mémoire de taille fixée *a priori*, le problème est donc de trouver la valeur appropriée de Ft. Pour cela, la construction du CF-Tree s'effectue par construction/reconstruction. Au début du processus, Ft est fixé à une valeur initiale relativement petite. Les vecteurs de la base sont ensuite insérés jusqu'à ce que la taille du CF-Tree dépasse l'espace qui lui est alloué ou bien jusqu'à l'insertion du dernier vecteur de la base. Si le CF-Tree atteint la taille maximale autorisée avant l'insertion de la totalité des vecteurs de la base, le Ft est incrémenté et le CF-Tree est reconstruit. Lors de la reconstruction, les vecteurs déjà insérés ne sont pas réinsérés car ils sont déjà représentés par des CF-vecteurs. Ce sont ces derniers qui sont directement insérés dans le nouvel arbre.

En plus de la réduction du nombre de CF-vecteurs due à l'incrémentation de Ft, avant toute reconstruction du CF-Tree, les micro-*clusters* dont le nombre de vecteurs est jugé « trop faible » par rapport à la moyenne sont mis à part et les vecteurs qu'ils contiennent sont considérés comme des vecteurs isolés *(outliers)*. Ces CF-vecteurs sont réinsérés après chaque incrémentation du Ft. À la fin de la construction du CF-Tree, les micro-*clusters* considérés comme bruités sont définitivement isolés. Optionnellement, ils peuvent être réintégrés lors de la redistribution des vecteurs dans la quatrième étape.

Il est clair que Ft est un paramètre crucial pour cette première étape. Sa valeur initiale ainsi que la procédure d'incrémentation ont un impact direct sur le nombre et la qualité des micro-*clusters* construits à l'issue de cette première phase. En l'absence de connaissances *a priori* sur la distribution des données, [Zhang *et al.*, 1996] propose d'utiliser un Ft initial nul. Il propose également des heuristiques pour la procédure d'incrémentation. Nous en résumons deux dans les points suivants.

1. Utilisation du volume des *clusters*. Cette heuristique repose sur l'hypothèse qui consiste à considérer que le volume des *clusters* croît linéairement avec le nombre de vecteurs insérés. Deux mesures approximatives du volume sont utilisées : le volume des hypersphères englobantes des *clusters* de la racine du CF-Tree et le volume du plus grand micro-*cluster* multiplié par le nombre total de micro-*clusters*. L'étude de l'évolution en fonction du nombre de vecteurs insérés de ces deux entités permet de prédire une nouvelle valeur du Ft.
2. Étude de la proximité des micro-*clusters*. Il s'agit de rechercher dans la feuille qui contient le plus de CF-vecteurs, les deux micro-*clusters* les plus proches et d'adapter la valeur du Ft de telle façon que ces deux micro-*clusters* puissent fusionner lors de la reconstruction suivante. Ceci permet de garantir une réduction du nombre de micro-*clusters* lors de la reconstruction suivante.

À l'issue de cette première étape, si le nombre de micro-*clusters* est jugé trop élevé car la troisième étape risque alors d'être trop coûteuse, alors des fusions de micro-*clusters* sont opérées et davantage de micro-*clusters* sont isolés et considérés comme bruités. Ceci est le but de la deuxième étape optionnelle de BIRCH.

4.3.4 Approches basées sur l'estimation de la densité

Les méthodes basées sur l'estimation de la densité partitionnent un ensemble de vecteurs en utilisant leur densité locale. Celle-ci est estimée à l'aide de méthodes statistiques d'estimation de la densité. Chaque groupe de vecteurs localement dense est considéré

comme un *cluster*. Les données uniformément distribuées sont considérées comme des données isolées *(outliers)*.

Il existe deux types de méthodes d'estimation de la densité : les méthodes paramétriques et les méthodes non paramétriques. Les premières supposent que les données sont distribuées suivant un modèle connu. L'objectif de ces méthodes est donc d'estimer les paramètres du modèle. Les méthodes non paramétriques n'émettent aucune hypothèse sur la distribution des données et permettent d'estimer une fonction de densité qui peut avoir une forme quelconque.

Dans cette section, nous présentons un algorithme de *clustering* paramétrique qui permet d'estimer les paramètres d'un mélange de distributions normales. Il s'agit de l'algorithme EM. Nous présentons également deux algorithmes, DBSCAN [Ester *et al.*, 1996] et DENCLUE [Hinneburg & Keim, 1998], qui utilisent des méthodes d'estimation non paramétriques, à savoir, les plus proches voisins pour le premier et la méthode des noyaux pour le deuxième.

4.3.4.1 Approches paramétriques

Le but de ces approches est d'estimer les paramètres d'un modèle statistique de la distribution des vecteurs. Le modèle est supposé connu *a priori* et est un mélange de distributions de probabilité de forme paramétrique. Plus formellement, cela revient à supposer que les vecteurs de données $\mathbf{x}_1, \ldots, \mathbf{x}_N$ de dimension d sont issus d'un vecteur aléatoire de densité

$$f(\mathbf{x}; \theta) = \sum_{k=1}^{p} \pi_k \varphi_k(\mathbf{x}; \alpha_k), \tag{4.9}$$

où les φ_k forment une famille de distributions dans $\mathbb{R}^d$ et $\theta = (\Pi, \alpha)$ sont les paramètres du modèle. $\Pi = (\pi_1, \ldots, \pi_p)$ est la proportion du mélange et $\alpha = (\alpha_1, \ldots, \alpha_p)$ les paramètres de la densité des *clusters*.

Une méthode largement utilisée pour estimer les paramètres d'un tel modèle est celle du maximum de vraisemblance. Sous l'hypothèse d'indépendance des vecteurs de données $(\mathbf{x}_i)$, cette méthode vise à estimer les paramètres qui maximisent la fonction de vraisemblance :

$$V_\theta = \prod_{i=1}^{N} f(\mathbf{x}_i; \theta) = \prod_{i=1}^{N} \sum_{k=1}^{p} \pi_k \varphi_k(\mathbf{x}_i; \alpha_k). \tag{4.10}$$

En pratique, c'est généralement le logarithme népérien du maximum de vraisemblance qui est maximisé. Il s'agit de la log-vraisemblance qui est donnée par :

$$L_\theta = \sum_{i=1}^{N} \ln \left(\sum_{k=1}^{p} \pi_k \varphi_k(\mathbf{x}_i; \alpha_k) \right) \tag{4.11}$$

En d'autres termes, cette méthode permet d'estimer les paramètres du modèle qui maximise la probabilité d'appartenance des vecteurs de données $(\mathbf{x}_i)$ au modèle.

En pratique, le seul cas traité analytiquement est le cas gaussien, c'est-à-dire le cas où la distribution de probabilité de chaque *cluster* est supposée normale. Dans ce cas, l'algorithme EM *(Expectation Maximization)* permet d'estimer les paramètres optimaux

du mélange gaussien (les moyennes et les matrices de covariances des *clusters*) par une procédure itérative pour un nombre fixé de *clusters* p.

Si on pose :

$$h_j(\mathbf{x}_i) = \frac{\pi_j \varphi_j(\mathbf{x}_i|\alpha_j)}{\sum_{k=1}^{p} \pi_k \varphi_k(\mathbf{x}_i|\alpha_k)} \tag{4.12}$$

$$\pi_j = \frac{1}{N} \sum_{k=1}^{N} h_j(\mathbf{x}_i) \tag{4.13}$$

$$\mu_j = \frac{\sum_{i=1}^{N} h_j(\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^{N} h_j(\mathbf{x}_i)} \tag{4.14}$$

$$\sigma_j = \frac{\sum_{i=1}^{N} h_j(\mathbf{x}_i)(\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T}{\sum_{i=1}^{N} h_j(\mathbf{x}_i)} \tag{4.15}$$

où :

$h_j(\mathbf{x}_i)$ est la probabilité d'appartenance du vecteur $\mathbf{x}_i$ au *cluster* j en considérant les paramètres du modèle, c'est-à-dire les π_j et les α_j ;

π_j est la probabilité du *cluster* j ;

μ_j et σ_j sont les paramètres de la distribution du *cluster* j, respectivement la moyenne et la matrice de covariance. Selon les notations introduites précédemment $\alpha_j = (\mu_j, \sigma_j)$;

alors l'algorithme EM s'exécute selon les quatre étapes suivantes.

1. Initialiser les paramètres du modèle (θ).
2. Pour chaque vecteur $\mathbf{x}_i$ de la base, calculer $h_j(\mathbf{x}_i)$ en utilisant l'équation 4.12. Cette entité est la probabilité d'appartenance du vecteur $\mathbf{x}_i$ au *cluster* j. Cette étape est appelée *Expectation* ou bien *E-step*.
3. Mettre à jour le paramètres du modèle $\theta = (pi_1, \ldots, \pi_p, \mu_1, \ldots, \mu_p, \sigma_1, \ldots, \sigma_p)$ à l'aide des équations 4.13, 4.14 et 4.15. Cette étape est appelée *Maximization* ou bien *M-step*.
4. Répéter les étapes (2) et (3) jusqu'à la stabilisation des paramètres, c'est-à-dire jusqu'à ce que les variations des paramètres d'une étape à une autre deviennent négligeables.

Une fois les paramètres des lois normales du mélange estimés, chaque composante du mélange génère un *cluster*. La probabilité d'appartenance d'un vecteur $\mathbf{x}_i$ à un *cluster* C_j est donnée par :

$$P(\mathbf{x}_i|C_j) = \frac{\pi_j \varphi_j(\mathbf{x}_i|\alpha_j)}{\sum_{k=1}^{p} \pi_k \varphi_k(\mathbf{x}_i|\alpha_k)}. \tag{4.16}$$

Il suffit alors d'affecter chaque vecteur $\mathbf{x}_i$ au *cluster* qui maximise $P(\mathbf{x}_i|C_j)$, pour $j = 1, \ldots, p$.

L'algorithme EM est un algorithme très simple à mettre en œuvre. De plus, à l'aide d'heuristiques très simples, il permet d'identifier les vecteurs isolés (par exemple, les vecteurs dont la probabilité d'appartenance aux *clusters* est inférieure à un seuil fixé). Ses inconvénients majeurs concernent principalement le choix du modèle et la complexité de la procédure d'estimation. Il est évident que la qualité de l'estimation dépend fortement de l'adéquation du modèle choisi par rapport à la distribution réelle des données. Si le modèle choisi ne correspond pas à la distribution réelle des vecteurs, les *clusters* obtenus peuvent être très mauvais. Or, en pratique, il est très difficile voire impossible d'avoir des informations *a priori* sur la distribution des données en grande dimension. Concernant sa complexité, c'est un algorithme très coûteux en temps de calcul car la moyenne et la matrice de covariance de chaque *cluster* sont recalculées à chaque itération.

4.3.4.2 Approches non paramétriques

DBSCAN [Ester *et al.*, 1996] est un algorithme de *clustering* inspiré de la méthode d'estimation de la densité par les plus proches voisins. L'idée de base de cet algorithme est d'étudier la densité locale autour de chaque vecteur pour déduire les sous-ensembles de vecteurs localement denses. Ces sous-ensembles sont considérés comme des *clusters*. Les vecteurs dont les voisinages sont relativement épars sont quant à eux considérés comme des vecteurs isolés *(outliers)*.

DBSCAN utilise principalement deux paramètres ε et $MinPts$. ε définit un voisinage autour d'un vecteur et $MinPts$ est le nombre minimal de vecteurs devant se trouver dans le voisinage d'un vecteur pour que celui-ci soit considéré dans une zone localement dense. Un vecteur dont le voisinage contient au moins $MinPts$ vecteurs est appelé vecteur *noyau*.

Pour explorer la densité locale des vecteurs, trois relations entre couples de vecteurs (p et q) sont définies :

p **est *directement* atteignable par densité à partir de** q si p appartient au voisinage de q (c'est-à-dire, la distance entre p et q est inférieure à ε), et si q est un vecteur noyau. Notons que cette relation n'est pas symétrique ;

p **est atteignable par densité à partir de** q s'il existe une suite de vecteurs entre p et q telle que chaque vecteur soit directement atteignable par densité à partir du vecteur suivant. Un exemple illustratif de cette relation est donné dans la figure 4.4. Notons que cette relation n'est pas symétrique ;

p est connecté à q s'il existe un vecteur o tel que p et q soient tous les deux atteignables par densité à partir de o. Un exemple illustratif de cette relation est donné dans la figure 4.5. Notons que cette relation est symétrique.

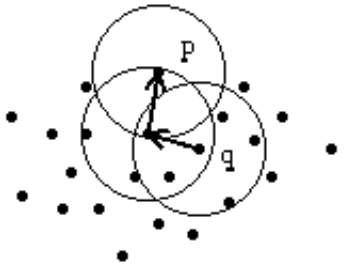


FIG. 4.4: Exemple d'un vecteur p atteignable par densité à partir de q. Cependant, q n'est pas atteignable par densité à partir de p car p n'est pas un vecteur noyau.

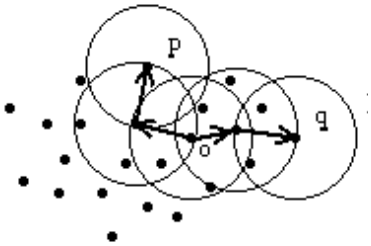


FIG. 4.5: Exemple de deux vecteurs p et q connectés par densité.

En se basant sur ces relations, un *cluster* C est défini comme un ensemble de vecteurs qui satisfont les deux conditions suivantes :

1. $\forall p, q$: si $p \in C$ et q est atteignable par densité à partir de p alors $q \in C$;
2. $\forall p, q \in C$: p est connecté à q.

L'ensemble des vecteurs isolés est l'ensemble des vecteurs n'appartenant à aucun *cluster*.

La formation des *clusters* s'effectue ainsi par un algorithme très simple mais extrêmement coûteux. Cet algorithme part d'un vecteur de la base et recherche tous les vecteurs atteignables à partir de celui-ci. Si le vecteur de départ est un vecteur noyau, cette recherche entraîne la création d'un *cluster* contenant le vecteur de départ et tous les vecteurs retrouvés. Sinon, aucun *cluster* n'est créé et cette procédure est ré-executée pour le vecteur suivant de la base.

Les *clusters* ainsi créés peuvent être de formes arbitraires. La figure 4.6 montre des exemples de *clusters* de formes arbitraires qui peuvent être formés par DBSCAN. Ces *clusters* n'ont pas de représentant unique (par exemple un centroïde). Chaque *cluster* peut être, cependant, représenté par un sous-ensemble de ses vecteurs noyaux.

Une procédure d'estimation des deux paramètres ε et $MinPts$ a également été proposée. Ces deux paramètres sont estimés tels que le plus petit *cluster* puisse être détecté.

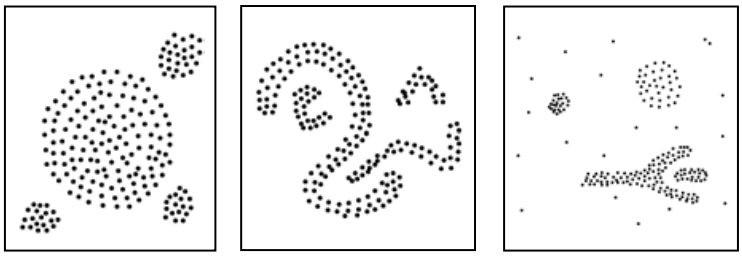


FIG. 4.6: Exemples de *clusters* pouvant être créés par DBSCAN.

Cependant, le coût de cette procédure est exorbitant car elle nécessite le calcul de la distance entre chaque vecteur de la base et son k^e plus proche voisin. En grande dimension, un tel calcul n'est pas envisageable.

DENCLUE [Hinneburg & Keim, 1998] est un algorithme de *clustering* qui utilise la méthode d'estimation de la densité par les noyaux [Silverman, 1986, Comaniciu & Meer, 2002]. Dans cette méthode, il est considéré que chaque vecteur *influence* l'ensemble des autres vecteurs de la base. Cette influence est modélisée par une fonction mathématique quelconque qui peut être une simple fonction de distance, une fonction en onde carrée ou bien une gaussienne. Cette fonction est appelée noyau et son but est de quantifier l'influence d'un vecteur sur un autre en fonction de leur proximité relative. La somme des influences de tous les vecteurs en un point donné est assimilée à la densité de ce point. Formellement, la fonction d'influence et la fonction de densité se définissent comme suit.

Définition 4 *La fonction d'influence d'un vecteur $y \in \mathbb{R}^d$ est la fonction $f^y : \mathbb{R}^d \to \mathbb{R}_+$ qui permet de mesurer l'influence du vecteur y sur n'importe quel autre vecteur de l'espace. Cette fonction doit être réflexive et symétrique. Les fonctions les plus utilisées sont la fonction en onde carrée f_{carre} et la fonction gaussienne f_{gauss} :*

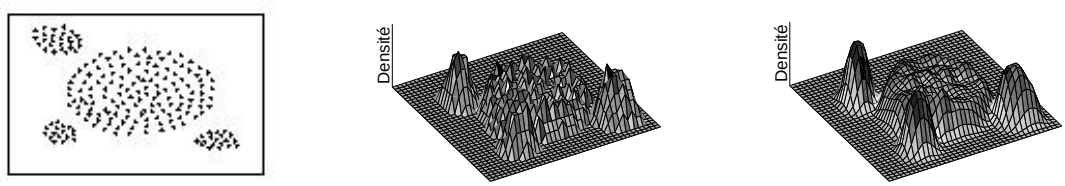
$$f^y_{carre}(x) = f^x_{carre}(y) = f_{carre}(x,y) = \begin{cases} 0 & si\ d(x,y) > \sigma \\ 1 & sinon \end{cases}, \tag{4.17}$$

$$f^y_{gauss}(x) = f^x_{gauss}(y) = f_{gauss}(x,y) = e^{-\frac{d(x,y)^2}{2\sigma^2}}, \tag{4.18}$$

où d la fonction de distance associée aux vecteurs et σ un paramètre de la fonction d'influence, largeur du support de la fonction f_{carre} et l'écart-type de la gaussienne dans le cas de f_{gauss}.

Définition 5 *La fonction de densité est la somme de la fonction d'influence de tous les vecteurs de la base en un point donné. Si l'on dispose de N vecteurs $x_1, \ldots, x_N \in \mathbb{R}^d$, la fonction de densité en un point x de l'espace $\mathbb{R}^d$ est définie par :*

$$f(x) = \sum_{i=1}^{N} f^{x_i}(x). \tag{4.19}$$



(a) (b)

FIG. 4.7: Exemples d'une distribution de vecteurs en dimension 2 et les deux fonctions de densité associées. (a) Fonction de densité obtenue par un noyau carré. (b) Fonction de densité obtenue par un noyau gaussien.

La figure 4.7 donne un exemple d'une distribution en dimension deux et les deux fonctions de densité estimées par la méthode des noyaux. La figure 4.7.(a) montre la fonction obtenue en utilisant un noyau carré et la figure 4.7.(b) la fonction obtenue en utilisant un noyau gaussien.

À partir de la fonction de densité, il est ensuite facile de déduire les maxima locaux à l'aide d'une procédure d'ascension de collines basée sur l'utilisation du gradient. Chaque maximum local est appelé « attracteur ». En suivant la direction du gradient de la fonction de densité, chaque vecteur de la base est ensuite associé à un maximum local.

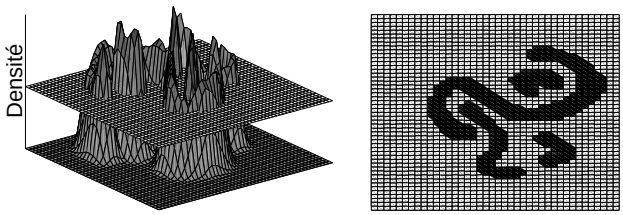


FIG. 4.8: Procédure de création de *clusters* de formes arbitraires à partir de la fonction de densité.

Si l'on désire créer des *clusters* définis autour d'un centre, il suffit de considérer chaque vecteur attracteur comme le centre d'un *cluster*. Les vecteurs associés au vecteur attracteur constituent les autres membres du *cluster*. Si par contre, aucune contrainte n'est imposée sur la forme des *clusters*, c'est-à-dire qu'ils peuvent être de formes quelconques, une manière de les créer est de couper la fonction de densité à l'aide d'un hyperplan de dimension $d-1$ à la façon illustrée dans la figure 4.8. La hauteur du plan de coupe détermine le nombre et la forme des *clusters*. C'est aussi le seuil de bruit car tous les vecteurs dont la densité est inférieure à cette valeur sont considérés comme des *outliers*.

Le principe de cette méthode est très simple et très intéressant. Cependant, sa mise en œuvre est extrêmement coûteuse notamment en ce qui concerne le calcul de la fonction de densité et de ses maximum locaux. Hinneburg et Keim proposent alors une méthode approximative pour estimer la fonction de densité de manière efficace. Elle procède en deux étapes. La première est une phase de pré-*clustering* qui permet de sélectionner grossièrement les vecteurs de la base qui appartiennent à des zones denses et d'identifier les vecteurs isolés. Les vecteurs sélectionnés sont ensuite utilisés pour estimer la fonction de densité.

Cette estimation est effectuée selon la même procédure que la méthode des noyaux sauf que la densité en chaque point ne prend en compte que l'influence des vecteurs se trouvant dans un voisinage local. Ceci permet de réduire considérablement le coût global de l'estimation de la fonction de densité. Néanmoins, ce coût reste très élevé à cause de la nécessité de calculer les voisinages de tous les points où la densité sera estimée.

4.3.5 Approches basées sur le partitionnement de l'espace

Le principe de ces approches est de partitionner l'espace en cellules plus ou moins régulières où les vecteurs de chaque cellule forment un *cluster*. Dans cette section, nous présentons une méthode basée sur le partitionnement de l'espace appelé OptiGrid.

OptiGrid [Hinneburg & Keim, 1999] est un algorithme de *clustering* qui permet de partitionner l'espace à l'aide d'une grille en prenant en compte la distribution des données. Une cellule suffisamment peuplée entraîne la création d'un *cluster*. Les vecteurs appartenant à des cellules faiblement peuplées sont considérés comme des vecteurs isolés.

Pour déterminer une grille de partitionnement pour un ensemble de vecteurs de dimension d, OptiGrid procède en quatre étapes.

1. Un ensemble d'opérateurs de projection est déterminé puis les vecteurs sont projetés en utilisant chacun des opérateurs.
2. Sur chaque projection, des hyperplans de séparation possibles sont déterminés. Ces hyperplans sont calculés de façon à partitionner les vecteurs à des endroits de faible densité. Pour cela, une estimation de la densité des vecteurs projetés est effectuée. Les q meilleurs hyperplans de séparation sont ensuite déterminés et seulement ceux qui permettent de partitionner les vecteurs en des endroits où la densité est inférieure à un seuil fixé sont retenus.
3. Si aucun hyperplan de séparation n'a été retenu, alors l'ensemble des vecteurs forme un seul *cluster*. Sinon, la grille de partitionnement est construite et les vecteurs de la base sont placés dans les cellules créées.
4. L'algorithme est ensuite appelé récursivement pour les cellules résultantes.

Dans l'implémentation proposée par les auteurs, les projections utilisées sont tout simplement des projections orthogonales sur les axes de l'espace. Par conséquent, les hyperplans de la grille sont parallèles aux axes de l'espace.

Un exemple des douze meilleurs plans de séparation obtenus sur des données de dimension deux est illustré dans la figure 4.9[2]. Dans ce cas, le paramètre q a été fixé à 1, c'est-à-dire qu'à chaque itération, seul le meilleur hyperplan de séparation est retenu.

Dans le cas d'ensembles de données contenant des *clusters* de taille et de distribution très différentes, plusieurs passes de partitionnement sont souvent nécessaires pour trouver l'ensemble des *clusters*. Une première passe permet de retrouver les *clusters* très denses mais identifie comme vecteurs isolés l'ensemble des *clusters* peu peuplés ou de faible densité. Les passes suivantes permettent d'affiner le partitionnement en considérant uniquement les vecteurs isolés.

[2]Exemple tiré de [Hinneburg & Keim, 1999].

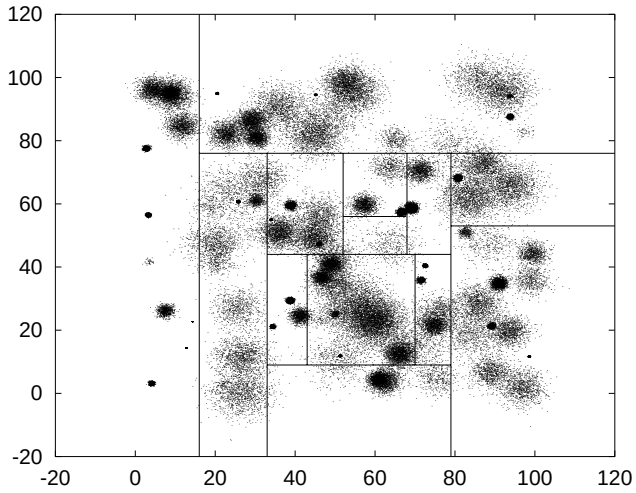


FIG. 4.9: Exemple des 12 premiers plans de séparation obtenus par OptiGrid sur des données de dimension deux.

Le principe de l'algorithme OptiGrid est très intéressant et son partitionnement est indépendant de l'ordre d'insertion des vecteurs. Malheureusement, sans connaissance approfondie de la distribution des données, les paramètres internes de l'algorithme sont extrêmement difficiles à fixer. Ceci constitue le principal inconvénient de cet algorithme d'autant plus que la qualité du partitionnement dépend fortement de ces paramètres.

4.3.6 Synthèse

Le passage en revue des principaux algorithmes de *clustering* montre clairement la diversité de leurs propriétés. Cependant, en se basant sur les critères énumérés dans la section 4.2, il est facile de montrer qu'aucun des algorithmes cités ne peut être utilisé pour la mise en paquets des données dans le domaine de l'indexation. Le *K-Means* ainsi que les techniques hiérarchiques traditionnelles sont de complexités $O(N^2)$ et $O(N^3)$ respectivement ; BIRCH et OptiGrid sont des algorithmes très complexes de par le grand nombre de paramètres internes requis ; et DBSCAN et DENCLUE génèrent des *clusters* généralement de formes arbitraires (figures 4.6 et 4.8) qui ne sont donc pas compacts et qui entraînent un fort taux de chevauchement entre les cellules englobantes des *clusters* lorsque ceux-ci sont utilisés pour la recherche de plus proche voisin. De plus, aucun de ces algorithmes ne permet de contrôler la taille des *clusters* et/ou leur population.

La première phase de l'algorithme BIRCH est néanmoins intéressante car elle permet de former des micro-*clusters* dont la taille en terme de rayon ou de diamètre moyen est contrôlée. Par conséquent, nous nous y sommes intéressés et nous l'avons retenue pour l'adapter et l'utiliser pour la mise en paquets des vecteurs de la base.

4.4 Algorithme proposé

L'algorithme proposé pour la formation de paquets est une adaptation et une amélioration de la première phase de BIRCH (Section 4.3.3.2, page 90). Dans cette section, nous revenons sur cette phase et nous donnons ses points forts et ses points faibles par rapport à la tâche envisagée. Nous présentons ensuite les améliorations et les modifications que nous apportons à cette phase. Enfin, nous décrivons de façon détaillée notre algorithme pour la formation de paquets. Nous ne donnons cependant aucun résultat des expériences effectuées en utilisant cet algorithme dans ce chapitre. L'ensemble des résultats expérimentaux est reporté au chapitre 6.

4.4.1 Retour sur la première phase de BIRCH

La première phase de BIRCH est une opération de micro-*clustering* dont le but est de regrouper les vecteurs de la base en petits paquets et d'identifier les vecteurs isolés. Le but de cette opération est de résumer la base de données à l'aide des centres des micro-*clusters*. Le nombre de micro-*clusters* est contrôlé par la taille mémoire allouée à la construction du CF-Tree.

Sa mise en œuvre comporte, cependant, les problèmes et les limitations résumés dans les points suivants.

Insertions des CF-vecteurs. Insérer un CF-vecteur consiste à rechercher le micro-*cluster* dont le centre est le plus proche au centre du CF-vecteur à insérer. L'objectif est de fusionner le CF-vecteur avec le micro-*cluster* qui lui est le plus proche de façon à minimiser la taille du micro-*cluster* résultant. Cette opération est extrêmement coûteuse car elle revient à effectuer une recherche de plus proches voisins au sein de l'ensemble des micro-*clusters* .

Pour réduire le coût des insertions, BIRCH utilise une structure hiérarchique (le CF-Tree) permettant des insertions plus rapides. Lors d'une insertion, la branche dont le CF-vecteur est le plus proche au CF-vecteur à insérer est suivie. Cette approche est cependant approximative car le micro-*cluster* sélectionné pourrait ne pas être le plus proche au CF-vecteur à insérer, c'est-à-dire, il existerait un micro-*cluster* dans une autre feuille qui peut absorber le CF-vecteur à inséré mieux que celui qui a été sélectionné au travers de la hiérarchie[3]. Ceci est dû au fort taux de chevauchement entre les *clusters* dans les niveaux supérieurs du CF-Tree.

Une des expériences que nous avons effectuées a montré que, pour un ensemble de 411 409 vecteurs de dimension 24, plus de 40 % des CF-vecteurs insérés au travers de la hiérarchie du CF-Tree sont insérés dans des feuilles qui ne contiennent pas le micro-*cluster* le plus proche au CF-vecteur inséré.

Pour retrouver le micro-*cluster* le plus proche de façon certaine, il aurait fallu utiliser un algorithme de recherche de plus proche voisin similaire à ceux utilisés par les index multidimensionnels (par exemple HS ou RKV (*cf.* section 1.3.2, page 16)). Malheureusement, ces algorithmes sont inefficaces en grande dimension.

[3]Nous rappelons qu'un micro-*cluster* M_1 absorbe un CF-vecteur v_1 mieux qu'un autre micro-*cluster* M_2 si le rayon du micro-*cluster* résultant de la fusion de M_1 et v_1 est inférieur au rayon du micro-*cluster* résultant de la fusion de M_2 et v_1.

Contrôle de la taille des micro-*clusters*. Le paramètre Ft utilisé par BIRCH pour contrôler la taille du CF-Tree, et de manière indirecte le nombre et la taille des micro-*clusters*, correspond au rayon (ou au diamètre) moyen des micro-*clusters*. Le rayon moyen permet de donner une information sur la dispersion des vecteurs autour du centre du micro-*cluster*. Cependant, il peut être très différent du rayon effectif du micro-*cluster* (la distance entre le centre et le plus lointain voisin parmi les vecteurs du micro-*cluster*). Un micro-*cluster* qui contient un grand nombre de vecteurs autour de son centre, peut facilement absorber un vecteur très éloigné sans que son rayon (ou son diamètre) moyen ne soit significativement augmenté. Or en indexation, les cellules englobantes sont calculées en utilisant les rayons effectifs. Par conséquent, le contrôle de la taille des micro-*clusters* par leur rayon moyen n'a aucune utilité dans ce cas.

Schéma d'exécution de l'algorithme. La construction du CF-Tree est effectuée telle que la taille du CF-Tree ne dépasse pas une taille mémoire fixée *a priori*. Lors de l'insertion des vecteurs, à chaque fois que la taille du CF-Tree dépasse la taille mémoire allouée, la construction est interrompue et le CF-Tree est reconstruit. Cette procédure est extrêmement sensible à l'ordre d'insertion des vecteurs. Si les premiers vecteurs insérés sont très dispersés, alors le CF-Tree atteint rapidement sa taille maximale autorisée et plusieurs reconstructions sont nécessaires dès le début du processus de construction. Ceci peut entraîner une augmentation rapide du paramètre Ft dès le début de la construction ce qui pourrait avoir un impact négatif sur la qualité des micro-*clusters* formés lors de cette étape.

4.4.2 Améliorations et modifications apportées

Les principales modifications que nous avons apportées à la première phase de l'algorithme BIRCH concernent les trois problèmes cités précédemment. Elles sont résumées dans les trois points suivants.

- Aucune structure hiérarchique accélératrice n'est utilisée lors de l'insertion des CF-vecteurs. Les CF-vecteurs des micro-*clusters* en cours de création sont stockés dans une liste simplement chaînée. Lors de l'insertion d'un nouveau CF-vecteur, une recherche séquentielle est exécutée pour déterminer le micro-*cluster* qui peut absorber le mieux le CF-vecteur à insérer. Cette organisation est plus coûteuse qu'une organisation hiérarchique. Par contre, elle permet d'éviter les erreurs d'insertion causées par le chevauchement dans les niveaux supérieurs de la hiérarchie. Ceci permet d'insérer les CF-vecteurs dans les bons micro-*clusters*, et donc, d'améliorer la qualité des micro-*clusters* formés à l'issue de cette phase. Cette amélioration justifie pleinement l'augmentation du coût de la procédure d'insertion due à l'utilisation d'une organisation séquentielle, d'autant plus que cette phase est exécutée hors ligne.
- Pour contrôler la taille des micro-*clusters*, nous avons choisi d'utiliser le rayon exact au lieu du rayon moyen. Le paramètre Ft correspond ainsi à un rayon qui définit une hypersphère englobante mais pas forcément minimale. Cette propriété résulte de la méthode utilisée pour le calcul du rayon (*cf.* section suivante). Cette méthode surestime le rayon des micro-*clusters* mais permet d'éviter le calcul des distances entre tous les vecteurs du *cluster* et son centre de gravité. Par ailleurs, comme l'objectif est d'avoir des *clusters* compacts et les plus petits possibles en terme de volume, cette

surestimation des rayons ne pose aucun problème car les rayons des micro-*clusters* obtenus sont effectivement encore plus petits que la valeur maximale autorisée lors du *clustering* ($\mathrm{Ft}_{\max}$).

- Le schéma d'exécution de l'algorithme ainsi que sa condition de terminaison ont été modifiés. Au lieu de considérer une taille mémoire fixée *a priori*, nous utilisons le nombre de *clusters* comme critère d'arrêt. Dans le nouveau schéma d'exécution, le paramètre Ft est incrémenté uniquement lorsque la totalité des vecteurs de la base est insérée et que le nombre de micro-*clusters* construits est supérieur au nombre fixé. Dans ce cas, le paramètres Ft est incrémenté et les CF-vecteurs des micro-*clusters* construits lors de la première passe sont réinsérés. Ceci permet de réduire le nombre de micro-*clusters* d'une passe à une autre. L'algorithme s'arrête dès que le nombre de micro-*clusters* est inférieur à un nombre fixé *a priori*. Cette façon d'incrémenter le Ft seulement lorsque la totalité des vecteurs de la base ont été insérés, permet de diminuer la sensibilité de l'algorithme à l'ordre d'insertion des vecteurs. Notons cependant que ce nouveau schéma ne permet pas de créer exactement un nombre de *clusters* égal au nombre fixé. Il crée un nombre de *clusters* inférieur mais le plus proche possible du nombre demandé. Ceci est dû au schéma d'exécution de l'algorithme qui procède par des réductions successives du nombre de *clusters*. Cette réduction dépend de l'incrémentation du Ft d'une passe à une autre : plus grande est cette incrémentation, plus grande est la probabilité de fusion des *clusters*, et, par conséquent, plus importante est la diminution du nombre de *clusters* à la passe suivante. Cette propriété ne pose cependant aucun problème car l'algorithme de recherche offre une grande souplesse lors du choix de ce paramètre. L'expérience présentée dans la section 6.5, page 135, et qui étudie l'influence du nombre de *clusters* sur les performances de la recherche de plus proches voisins, montre que le nombre optimal de *clusters*, c'est-à-dire celui qui minimise le temps de recherche, s'étale sur un intervalle assez large (entre 1 et 3 fois la racine carrée du nombre de vecteurs de la base). À titre d'exemple, ce paramètre peut être fixé à n'importe quelle valeur entre 684 et 1298 pour un ensemble contenant 411 409 vecteurs de dimension 24. Toute partition dont le nombre *cluster* est compris dans cet intervalle induit un temps de recherche de plus proches voisins pratiquement identique.

4.4.3 Détails de l'algorithme proposé

Pour prendre en compte le rayon effectif d'un micro-*cluster* au lieu de son rayon moyen, nous avons modifié la composition du CF-vecteur comme suit : le CF-vecteur d'un micro-*cluster* contenant les vecteurs $\vec{X}_1, \ldots, \vec{X}_N$ est défini par CF$= (N, \vec{LS}, r)$ où r est le rayon d'une hypersphère englobante de l'ensemble des vecteurs mais qui n'est pas forcément minimale. Initialement, lorsque chaque vecteur forme un *cluster* à part entière, r vaut 0. Lorsque deux micro-*clusters* de CF-vecteurs $\mathrm{CF}_1 = (N_1, \vec{LS}_1, r_1)$ et $\mathrm{CF}_2 = (N_2, \vec{LS}_2, r_2)$ fusionnent, le CF-vecteur du micro-*cluster* résultant est :

$$\mathrm{CF}_1 + \mathrm{CF}_2 = (N_1 + N_2, \vec{LS}_1 + \vec{LS}_2, max(dist(c, c_1) + r_1, dist(c, c_2) + r_2))),$$

où $c = \frac{1}{N_1+N_2}(\vec{LS}_1 + \vec{LS}_2)$ est le centre de gravité du micro-*cluster* résultant, $c_1 = \frac{1}{N_1}(\vec{LS}_1)$ et $c_2 = \frac{1}{N_2}(\vec{LS}_2)$ sont respectivement les centres de gravité du premier et du deuxième micro-*clusters*.

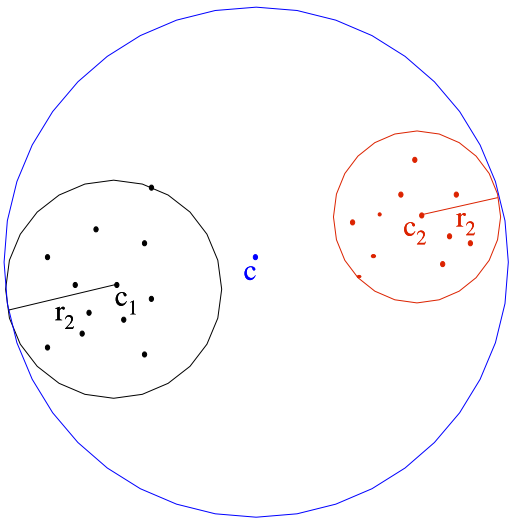


FIG. 4.10: Fusion de deux micro-*clusters*.

La figure 4.10 illustre la fusion de deux micro-*clusters*. Elle montre notamment que l'hypersphère englobante définie par le rayon du micro-*cluster* résultant de la fusion n'est pas minimale.

Le fait que les hypersphères englobantes ne soient pas minimales ne pose cependant aucun problème. La taille réelle des micro-*cluster* sera plus petite que la taille exprimée par le rayon r de leurs CF-vecteurs. Sachant que plus les *clusters* sont petits et compacts, meilleure sera l'efficacité des règles de filtrage lors de la recherche, cette surestimation des rayons des micro-*clusters* ne pose aucun problème.

Le choix de la valeur initiale de Ft ainsi que son incrémentation sont effectués par les méthodes proposées par BIRCH. Nous avons également gardé la même heuristique que BIRCH pour déterminer les vecteurs isolés : un micro-*cluster* contenant moins que $\beta\,\%$ vecteurs que la moyenne des populations de tous les micro-*clusters* est considéré comme bruité et les vecteurs qu'il contient sont considérés comme vecteurs isolés. Le paramètre β est appelé « taux de bruit ».

Avant de présenter l'algorithme de *clustering* en détail, introduisons les notations et les procédures suivantes :

$|L|$ est un opérateur permettant de renvoyer le nombre d'éléments dans une liste simplement chaînée L ;

$L[i]$ est un opérateur permettant de renvoyer le i^e élément d'une liste simplement chaînée L ;

insérer(elem, L**)** est une procédure permettant d'insérer un CF-vecteur (elem) dans une liste L ;

supprimer(i, L**)** est une procédure permettant de supprimer le i^e élément d'une liste L ;

vider(L**)** est une procédure permettant de supprimer tous les éléments d'une liste L.

L'algorithme de *clustering* se compose de trois parties. Elles sont détaillées dans les algorithmes 1, 2 et 3. Il s'agit d'un algorithme itératif. Initialement, chaque vecteur forme

un micro-*cluster* et l'ensemble des micro-*clusters* bruités est vide. Chaque itération se compose principalement de trois parties :

1. l'insertion des CF-vecteurs des micro-*clusters* courants ;
2. l'absorption des CF-vecteurs correspondant aux micro-*clusters* bruités isolés lors de la passe précédente (algorithme 2) ;
3. le calcul du nouveau seuil de bruit et la détection des CF-vecteurs correspondant aux micro-*clusters* bruités (algorithme 3).

4.5 Évaluation de la méthode

L'utilisation que nous faisons du *clustering* est différente de son utilisation traditionnelle. Les critères traditionnels d'évaluation ne peuvent donc être utilisés [Halkidi *et al.*, 2001]. Par exemple, dans les applications traditionnelles de data mining, l'objectif est de former un nombre relativement petit de *clusters*. Ces *clusters* sont ensuite analysés pour en extraire des relations de causes à effets ou bien pour en déduire des groupes homogènes d'individus et de les affecter à des catégories données. Souvent dans ces applications, une notion sémantique est associée à chaque *cluster*. Dans ce cas, l'évaluation des méthodes de *clustering* s'effectue sur des ensembles bien étudiés, souvent visualisables en dimension deux ou trois. Ces ensembles contiennent des groupes de vecteurs que l'algorithme de *clustering* est censé retrouver automatiquement. Son évaluation s'effectue donc par rapport à sa capacité de les retrouver.

Dans notre cas, le rôle du *clustering* est complètement différent. Il s'agit seulement de créer des petits paquets de données les plus compacts possibles. Le but est de permettre à l'algorithme de recherche d'en sélectionner le plus petit nombre possible lors d'une recherche de plus proches voisins à l'aide des règles de filtrage géométrique. Aucune sémantique particulière n'est associée aux *clusters*. L'évaluation s'effectue donc uniquement par rapport à l'algorithme de recherche de plus proches voisins qui exploitera les *clusters* pour confiner la recherche. De même pour le choix du nombre de *clusters*, la seule contrainte qui guide le choix de ce paramètre est la réduction du temps de recherche (*cf.* section 6.5, page 135).

C'est pour ces raisons que nous ne présentons aucune évaluation de performance dans ce chapitre. L'ensemble des expériences réalisées sont présentées dans le chapitre 6. Le seul critère d'évaluation retenu est le temps de réponse de l'algorithme de recherche de plus proches voisins.

Algorithme 1 *Clustering*

Entrées :

V : liste de vecteurs ;
$nbMaxClust$: nombre maximum de *clusters* à créer ;
β : taux de bruit ;
Ft_{init} : la valeur initiale du paramètre Ft ;

Sorties :

$listeμClust$: liste des CF-vecteurs des micro-*clusters* ;
$listeμClustBruit$: liste des CF-vecteurs des micro-*clusters* bruités ;
Ft_{max} : le rayon maximal des micro-*clusters* créés ;

Début
Initialiser ($listeTempo$, V);
$Ft \leftarrow Ft_{init}$;
tant que $|listeTempo| < nbMaxClust$ **faire**
vider($listeμClust$);
Pour $i = 1$ à $|listeTempo|$ **faire**
si $|listeμClust| = 0$ **alors**
insérer($listeTempo[i]$, $listeμClust$)
sinon
retrouver p, l'indice du micro-*cluster* de $listeμClust$ le plus proche à $listeTempo[i]$;
calculer le rayon du micro-*cluster* résultant de la fusion de $listeTempo[i]$ et de $listeμClust[p]$, soit r_{fus} ;
si $r_{fus} > Ft$ **alors**
insérer($listeTempo[i]$, $listeμClust$)
sinon
$p \leftarrow p + listeTempo[i]$
fin si
fin si
fin Pour
absorberBruit($listeμClust$, $listeμClustBruit$, Ft)
isolerBruit($listeμClust$, $listeμClustBruit$, β)
$Ft_{max} \leftarrow Ft$;
incrémenter Ft;
vider($listeTempo$);
$listeTempo \leftarrow listeμClust$;
fin tant que
Fin

Algorithme 2 absorberBruit

Entrées :

listeμClust : liste des CF-vecteurs des micro-*clusters* ;
listeμClustBruit : liste des CF-vecteurs des micro-*clusters* bruités ;
Ft : rayon maximum des micro-*clusters* ;

Sorties :

listeμClust : liste des CF-vecteurs des micro-*clusters* mise à jour ;
listeμClustBruit : liste des CF-vecteurs des micro-*clusters* bruités mise à jour ;

Début
  Pour $i = 1$ à $|listeμClustBruit|$ **faire**
    retrouver dans *listeμClust* le micro-*cluster* le plus proche à $listeμClustBruit[i]$, soit p ;
    calculer le rayon du micro-*cluster* résultant de la fusion de $listeμClustBruit[i]$ et de p, soit r_{fus} ;
    si $r_{fus} < Ft$ **alors**
      $p \leftarrow p + listeμClustBruit[i]$
      $supprimer(i, listeμclustBruit)$
    fin si
  fin Pour
Fin

Algorithme 3 isolerBruit

Entrées :

listeμClust : liste des CF-vecteurs des micro-*clusters* ;
listeμClustBruit : liste des CF-vecteurs des micro-*clusters* bruités ;
β : taux de bruit ;

Sorties :

listeμClustBruit : liste des CF-vecteurs des micro-*clusters* bruités modifiée ;

Début
  calculer la moyenne des populations des micro-*clusters* de *listeμClust*, soit moy_{pop};
  Pour $i = 1$ à $|listeμClust|$ **faire**
    si $listeμClust[i].N < \beta.moy_{pop}$ **alors**
      insérer($listeμClust[i]$, $listeμClustBruit$)
      supprimer($i, listeμClust$)
    fin si
  fin Pour
Fin

4.6 Synthèse

Après avoir introduit le *clustering* de manière générale, nous avons montré sa similitude avec le problème de formation des paquets pour la recherche de plus proches voisins. Nous avons ensuite présenté les conditions que doit satisfaire un algorithme de *clustering* pour être utilisé dans ce contexte. Nous avons décrit les algorithmes les plus usités. Ces algorithmes se sont cependant tous avérés inadaptés car ils ne satisfont pas les contraintes liées à la recherche de plus proches voisins. Pour cela, nous avons repris et adapté la première phase de l'algorithme BIRCH. L'algorithme proposé est un algorithme itératif qui ne nécessite qu'une seule passe sur les données. Il permet également de contrôler les tailles exactes des *clusters* et d'identifier les vecteurs isolés.

Le chapitre suivant présente l'algorithme de recherche approximative de plus proches voisins qui exploite les *clusters* formés hors ligne à l'aide de l'algorithme de *clustering* proposé dans ce chapitre.

Chapitre 5

Recherche approximative de plus proches voisins - Calculs des rayons approximatifs

Dans ce chapitre seront présentés l'algorithme de recherche de plus proches voisins ainsi que la méthode de calcul des rayons approximatifs. Nous considérons que les vecteurs de la base ont préalablement été regroupés en paquets et que les vecteurs isolés ont été identifiés à l'aide de l'algorithme présenté dans le chapitre précédent.

5.1 Introduction

De notre point de vue, pour que la recherche approximative de plus proches voisins soit utilisable dans un système réel, elle doit vérifier un certain nombre de critères dont les plus importants sont ceux relatifs à la procédure de contrôle de la précision.

Paramètres de contrôle intuitifs. Pour qu'une méthode de recherche approximative soit utilisable dans un système de recherche, il est important de pouvoir contrôler la précision de la recherche au moyen de paramètres intuitifs et simples à manipuler par un utilisateur non spécialisé.

Fiabilité. La procédure de contrôle doit être fiable. L'imprécision effective des résultats doit être au moins égale sinon inférieure mais aussi le plus proche possible de l'imprécision tolérée par l'utilisateur et exprimée au travers des paramètres de contrôle.

Efficacité. La méthode de recherche doit être efficace, c'est-à-dire que le gain en temps de réponse doit être élevé pour justifier la perte de précision occasionnée.

Elle doit être également capable d'effectuer des recherches de k-plus proches voisins. Plusieurs des méthodes existantes se limitent à la recherche *du* plus proche voisin. L'état de l'art des techniques de recherche approximative, présenté dans le chapitre 3, a montré qu'aucune technique existante ne satisfait l'ensemble de ces critères.

Dans ce chapitre, nous détaillons une nouvelle méthode de recherche approximative de plus proches voisins au sein de vecteurs définis dans un espace muni de la distance eucli-

dienne[1]. Elle satisfait l'ensemble de ces critères. Elle permet de contrôler la précision de la recherche de façon probabiliste au travers d'un seul paramètre appelé niveau d'imprécision et noté α. Il s'agit de « la probabilité maximale de ne pas retrouver un des plus proches voisins dans le résultat approximatif ». En d'autres termes, si les k plus proches voisins d'un vecteur requête q retrouvés par un algorithme de recherche exact sont $\{p_1, \ldots, p_k\}$ alors notre méthode de recherche approximative garantit que la probabilité de ne pas retrouver un des p_i par une recherche approximative est inférieure à α. Ce paramètre de contrôle a été choisi de façon à permettre à n'importe quel utilisateur non spécialisé de la méthode de contrôler la précision de la recherche.

Retourner un résultat à l'imprécision bornée constitue le cœur de la méthode présentée dans ce chapitre. Cependant, comme nous le verrons dans la section 5.4, aux principes de base de cette méthode, peuvent s'ajouter deux extensions offrant deux autres modes de recherches : la recherche incrémentielle de base et la recherche incrémentielle avec indication sur la possibilité d'amélioration des résultats. Ce sont deux modes de recherche intéractive. Les résultats de la recherche sont présentés à l'utilisateur régulièrement, celui-ci peut alors décider d'interrompre la recherche à tout moment. Dans le premier mode, cette décision est précise en considérant uniquement les résultats courants. Dans le deuxième, en plus des résultats courants, la probabilité d'amélioration des résultats courants est présentée, ce qui donne à l'utilisateur un élément d'information supplémentaire sur la qualité du résultat courant.

5.2 Présentation de la méthode de recherche approximative

Comme toute méthode de recherche de plus proches voisins, la méthode que nous proposons se compose de deux phases : une phase hors ligne de structuration des données et une phase en ligne d'interrogation de la base. La phase hors ligne a pour objectif de regrouper les données en petits paquets les plus compacts et les plus séparés possibles. L'algorithme utilisé pour cette tâche a été présenté dans le chapitre précédent. Les paquets sont englobés dans des cellules de formes hypersphériques. Cette forme a été choisie pour sa simplicité mais aussi pour sa propriété d'isotropie. Nous verrons l'importance de cette propriété en détails dans la section 5.3.2.

L'approximation envisagée est d'ordre géométrique. Elle concerne les rayons des hypersphères englobantes. Nous considérons des rayons approximatifs sous-estimés qui définissent des hypersphères approximatives. Sachant que même une bonne technique de création de paquets ne peut échapper au problème de chevauchement en grande dimension, l'idée est de diminuer les rayons des hypersphères englobantes afin de réduire le chevauchement entre cellules en profitant des propriétés des bordures décrites dans la section 2.3. Puisque, en grande dimension, la majeure partie du volume d'une hypersphère se trouve dans la bordure de celle-ci, des réductions de rayons permettent de réduire suffisamment le volume des hypersphères englobantes tout en englobant l'essentiel des vecteurs appartenant aux paquets. Cette réduction permet donc d'accroître le taux de filtrage des algorithmes de recherche mais peut entraîner également des imprécisions dans les résultats de la recherche.

[1] Notons toutefois, que cette méthode s'applique également à tout ensemble de vecteurs muni d'une métrique qui se réduit à une distance euclidienne par un prétraitement des vecteurs (e.g. la distance de Mahalanobis).

Certains des plus proches voisins recherchés risquent de ne pas être retrouvés s'ils appartiennent à des paquets filtrés à tort. Le problème est ainsi de mettre au point une stratégie de réduction des rayons des hypersphères englobantes en fonction du paramètre de contrôle de la précision. En d'autres termes, cette stratégie doit calculer des rayons approximatifs de façon à ce que, lors de la recherche, l'imprécision des résultats approximatifs retournés soit en dessous du seuil d'imprécision fixé par l'utilisateur.

Pour un niveau d'imprécision donné, notre méthode a donc besoin de calculer pour chaque *cluster*, un rayon approximatif. Pour considérer plusieurs niveaux d'imprécision différents, il suffit de calculer plusieurs rayons approximatifs pour chaque *cluster*, chacun correspondant à un niveau d'imprécision donné. Il est donc nécessaire de définir *a priori* un ensemble de niveaux d'imprécision. Toutefois, si l'utilisateur a besoin de rajouter un niveau d'imprécision supplémentaire, il suffit de relancer la procédure de calculs des rayons approximatifs en fonction du nouveau niveau d'imprécision.

Notre méthode est présentée selon le plan suivant. Tout d'abord, nous décrivons les règles de filtrage utilisées par l'algorithme de recherche en ligne. Ces règles permettent de sélectionner les *clusters* pertinents par rapport à un vecteur requête donné. Ensuite, nous présentons la méthode permettant de calculer, pour chaque *cluster*, un rayon approximatif correspondant à un niveau d'imprécision donné. Pour cela, nous traduisons la contrainte globale de contrôle de la précision en contraintes locales exprimées au niveau des *clusters*. Le but est d'exprimer des contraintes locales qui, lorsqu'elles sont vérifiées, garantissent que la contrainte globale l'est également. Nous procédons ainsi car la réduction des rayons s'effectue localement au niveau des *clusters*. Il est donc plus simple d'utiliser des contraintes définies localement au niveau des *clusters* pour calculer leurs rayons approximatifs. Nous étudions ensuite la relation entre la contrainte locale et le rayon approximatif d'un *cluster* et nous détaillons la méthode de calcul des rayons approximatifs. Dans un premier temps, nous faisons une hypothèse sur la distribution des vecteurs du *clusters*. Nous relachons ensuite cette hypothèse et nous proposons une solution plus générale. Celle-ci utilise un paramètre interne. Nous présentons alors en dernier lieu une méthode permettant de l'estimer automatiquement.

5.2.1 La recherche en ligne à l'aide des règles de filtrage

En présence d'un vecteur requête (en ligne), en fonction du niveau d'imprécision choisi par l'utilisateur, un rayon approximatif est sélectionné pour chaque *cluster*. L'algorithme de recherche utilisé est identique aux algorithmes de recherche associés aux techniques traditionnelles d'indexation multidimensionnelle sauf qu'il utilise les rayons approximatifs des *clusters* au lieu de leurs rayons exacts.

Dans la première phase de la recherche, les *clusters* non pertinents pour le vecteur requête sont filtrés en utilisant la règle suivante :

$$\text{si } \widehat{\text{dmin}}(q, C_i) \geqslant \widehat{\text{dmax}}(q, C_j) \text{ alors éliminer } C_i. \tag{5.1}$$

où C_i et C_j sont deux *clusters* , q est le vecteur requête et $\widehat{\text{dmin}}$ et $\widehat{\text{dmax}}$ sont respectivement les distances minimales et maximales entre le vecteur requête et les hypersphères approximatives des *clusters* C_i et C_j. Cette règle est illustrée dans la figure 5.1. Dans cette figure, les cercles en traits continus sont les rayons approximatifs des *clusters*. Les rayons

exacts sont représentés en traits discontinus. Pour le *cluster* C_i, nous remarquons que trois de ces vecteurs se trouvent à l'extérieur de sa cellule (l'hypersphère définie par son rayon approximatif).

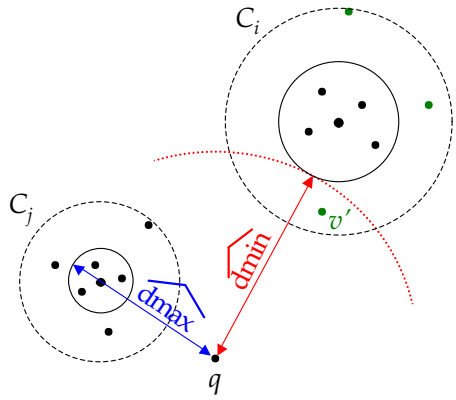


FIG. 5.1: Algorithme de recherche de plus proches voisins. Première règle de filtrage.

Cette première règle est appliquée selon les trois étapes suivantes :

- Calculer la distance $\widehat{\text{dmax}}_i$ pour chaque *cluster* C_i contenant au moins k vecteurs à l'intérieur de son hypersphère approximative. Les *clusters* qui ne contiennent pas k vecteurs à l'intérieur de leurs hypersphères approximatives ne sont pas considérés ;
- La plus petite $\widehat{\text{dmax}}$ est ensuite retrouvée. Considérons qu'il s'agit de $\widehat{\text{dmax}}_m$, c'est-à-dire la distance qui a été calculée entre le vecteur requête q et le *cluster* C_m. On peut alors affirmer que la distance entre q et son k^e plus proche voisin est inférieure à $\widehat{\text{dmax}}_m$. Au moins k vecteurs du *cluster* C_m sont à des distances inférieures à $\widehat{\text{dmax}}_m$;
- Les distances minimales $\widehat{\text{dmin}}$ sont ensuite calculées entre le vecteur requête q et tous les *clusters*. En appliquant la règle 5.1, un *cluster* de $\widehat{\text{dmin}}$ supérieure à $\widehat{\text{dmax}}_m$ est filtré car tous ses vecteurs sont à une distance supérieure à $\widehat{\text{dmax}}_m$. Ils se trouvent, par conséquent, à une distance du vecteur requête supérieure à la distance entre celui-ci et son k^e plus proche voisin.

Les *clusters* sélectionnés lors de la première phase sont ensuite ordonnés selon leurs distances minimales ($\widehat{\text{dmin}}$) ou bien leurs distances par rapport aux centres des *clusters*. L'idée est de commencer à lire les *clusters* qui ont le plus de chance de contenir les plus proches voisins recherchés. *A priori*, ce sont ceux qui sont les plus proches de q. Si ces plus proches voisins sont retrouvés dès le début du processus de recherche, celui-ci a plus de chance de s'arrêter très vite car le taux de filtrage des *clusters* sera meilleur. Ce point sera détaillé davantage après la présentation de la deuxième règle de filtrage. En pratique, le nombre de *clusters* lus en moyenne est quasiment le même si les *clusters* sont ordonnés en fonction de la distance minimale ou bien la distance au centre. Dans la suite, nous considérons que les *clusters* sont toujours ordonnés selon leurs distances minimales par rapport au vecteur requête.

Les *clusters* sont mis dans une liste appelée liste des *clusters* candidats puis traités dans l'ordre considéré. Tous les vecteurs du *cluster* à traiter, y compris ceux se trouvant à l'extérieur de l'hypersphère approximative, sont chargés en mémoire et leurs distances au vecteur requête sont calculées. Ceci permet de construire une liste de k plus proches voisins qui se met à jour au fur et à mesure de l'avancement de ces calculs.

Dans cette phase, tous les *clusters* dans la liste des *clusters* candidats ne sont cependant pas lus. Dès que la distance minimale entre le vecteur requête et le prochain *cluster* à traiter est supérieure à la distance au k^e plus proche voisin courant, la recherche s'arrête. Tous les *clusters* restant sont à des distances minimales supérieures à la distance au k^e plus proche voisin courant. Ils ne peuvent donc contenir des vecteurs plus proches au vecteur requête que son k^e plus proche voisin courant. Ceci est la deuxième règle de filtrage. Elle est illustrée dans la figure 5.2 et est donnée par :

$$\text{si } \widehat{\text{dmin}}(q, C_i) \geqslant d(q, kppv_c) \text{ alors arrêter la recherche} \tag{5.2}$$

où $d(q, kppv_c)$ est la distance entre q et son k^e plus proche voisin courant.

L'utilisation des rayons approximatifs peut introduire de l'imprécision lors de la recherche. Les vecteurs des *clusters* se trouvant à l'extérieur de l'hypersphère approximative ne sont pas pris en compte lors du calcul des distances minimales. Par exemple, dans le *cluster* C_i (figure 5.2), $\widehat{\text{dmin}}$ ne prend pas en compte le vecteur v'. Si ce vecteur est l'un des plus proches voisins du vecteur requête, il pourrait ne pas être retrouvé car le *cluster* auquel il appartient peut être filtré à tort par la deuxième règle de filtrage. Ce même raisonnement s'applique également à la première règle de filtrage (le vecteur v' du *cluster* C_i de la figure 5.1 risque d'être ignoré). Cependant, comme les rayons approximatifs sont calculés par rapport à un niveau d'imprécision donné, la probabilité pour que cela arrive est contrôlée. Le calcul des rayons approximatifs en fonction d'un niveau d'imprécision fait l'objet de la section suivante.

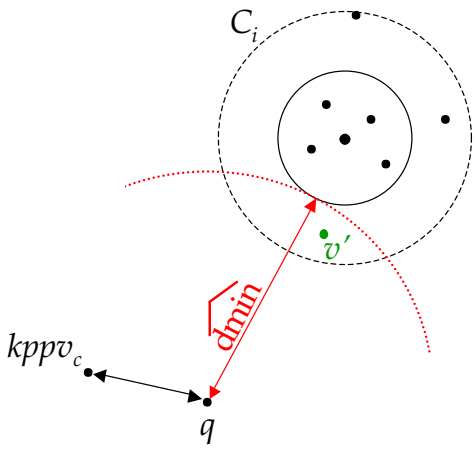


FIG. 5.2: Algorithme de recherche de plus proches voisins. Deuxième règle de filtrage.

Le fichier des vecteurs isolés est parcouru séquentiellement. Ceci peut être effectué avant ou après la recherche dans les *clusters*. Le considérer avant permet de construire un

ensemble de résultats temporaires et de disposer d'une distance au k^e plus proche voisin courant avant de commencer la recherche au sein des *clusters*. Si cette distance est inférieure à $\widehat{\text{dmax}}_m$, elle peut être utilisée au lieu de $\widehat{\text{dmax}}_m$ lors de l'application de la première règle de filtrage. Ceci permet de filtrer plus de *clusters*.

5.3 Calcul des rayons approximatifs et contrôle de la précision

Afin d'assurer un contrôle de l'imprécision de la recherche de plus proches voisins, il est nécessaire d'établir une relation entre l'approximation géométrique (réductions des rayons) et le critère probabiliste global de contrôle de la précision (α). Cette relation permet ensuite de calculer le rayon approximatif d'un *cluster* en fonction de α.

Comme le critère de contrôle de l'imprécision est global, alors que les réductions de rayons s'effectuent localement au niveau de chaque *cluster*, il est nécessaire dans un premier temps d'exprimer la contrainte globale de la précision localement au niveau de chaque *cluster*. Les contraintes locales doivent être définies de telle façon que, lorsqu'elles sont vérifiées, la contrainte globale l'est également. Chaque contrainte locale est ensuite utilisée au niveau d'un *cluster* pour calculer son rayon approximatif.

La présentation de la méthode de calcul des rayons approximatifs s'effectue donc en deux étapes. Dans la section suivante, la méthode de déductions des contraintes locales en fonction du paramètre global de contrôle α est présentée. Dans la section 5.3.2, l'algorithme de calcul du rayon approximatif est décrit. Dans cette section, nous nous appuierons dans un premier temps sur une hypothèse qui est ensuite relâchée et étudiée de manière approfondie dans les sections 5.3.3 et 5.3.4.

5.3.1 De l'imprécision globale à l'imprécision locale

Le critère global pour le contrôle de la précision doit garantir que :

$$\forall q, \quad P(\text{miss}(\text{PPV}(q))) \leqslant \alpha, \tag{5.3}$$

où $\text{PPV}(q)$ désigne un des plus proches voisins du vecteur requête q et $\text{miss}(v)$ désigne l'événement de ne pas retrouver le vecteur v dans le résultat de la recherche.

Exprimer cette contrainte localement au niveau d'un *cluster* C_i revient à trouver une borne maximale de la probabilité de ne pas retrouver un des plus proches voisins du vecteur requête q si l'on sait que ce plus proche voisin appartient au *cluster* C_i. Soit v un des plus proches voisins recherchés du vecteur q. Il s'agit alors de trouver un α_i (spécifique au *cluster* C_i) tel que :

$$\forall q, \quad P(\text{miss}(v)|v \in C_i) \leqslant \alpha_i. \tag{5.4}$$

Pour une base de vecteurs partitionnée en un ensemble de *clusters* $\{C_1, C_2, \ldots, C_n\}$, le problème est donc de trouver un ensemble de valeurs $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ qui vérifient :

$$P(\text{miss}(v)|v \in C_i) \leqslant \alpha_i, \text{ pour } i = 1, \ldots, n \quad \Rightarrow \quad P(\text{miss}(v)) \leqslant \alpha. \tag{5.5}$$

Sachant que les *clusters* C_i sont disjoints, c'est-à-dire, chaque vecteur de la base appartient à un seul *cluster*, le théorème des probabilités totales nous permet d'écrire $P(\text{miss}(v))$ comme suit :

$$P(\text{miss}(v)) = \sum_{i=1}^{n} P(\text{miss}(v)|v \in C_i)P(v \in C_i) \leqslant \alpha. \tag{5.6}$$

Par ailleurs, on a également :

$$\sum_{i=1}^{n} P(v \in C_i) = 1.$$

À partir de ces deux équations, nous pouvons affirmer qu'il suffit que

$$P(\text{miss}(v)|v \in C_i) \leqslant \alpha, \forall i \tag{5.7}$$

pour que

$$P(\text{miss}(v)) \leqslant \alpha,$$

c'est-à-dire :

$$P(\text{miss}(v)|v \in C_i) \leqslant \alpha, \quad \forall\, i \quad \Rightarrow \quad P(\text{miss}(v)) \leqslant \alpha. \tag{5.8}$$

La démonstration de cette implication est un résultat immédiat de l'équation 5.6.

En résumé, calculer un rayon approximatif pour chaque *cluster* C_i garantissant que $P(\text{miss}(v)|v \in C_i) \leqslant \alpha$ permet d'effectuer des recherches approximatives telles que $P(\text{miss}(v)) \leqslant \alpha$.

5.3.2 Calcul des rayons approximatifs

Pour calculer les rayons approximatifs des *clusters* qui correspondent à un niveau d'imprécision α, nous utilisons les contraintes locales $P(\text{miss}(v)|v \in C_i) \leqslant \alpha$ déduites dans la section précédente.

Tout d'abord, nous commençons par localiser la partie d'un *cluster* qui est ignorée lors de la recherche si le rayon approximatif est pris en compte au lieu du rayon exact pour un vecteur requête q donné. Seuls les vecteurs se trouvant dans cette zone et appartenant à l'ensemble des plus proches voisins de q risquent de ne pas figurer dans l'ensemble des résultats. Nous rappelons que la réduction des rayons s'effectue hors ligne indépendamment de tout vecteur requête. Ici, le vecteur q est utilisé uniquement dans un but illustratif. Il permet de montrer pourquoi certains vecteurs sont ignorés durant la recherche et de dégager ainsi les conditions selon lesquelles un résultat est approximatif. Ces conditions peuvent être artificiellement reproduites hors ligne, sans aucun vecteur requête, et, permettre ainsi les calculs des rayons approximatifs.

Influence de l'approximation des rayons sur la recherche. Si l'on reprend les deux règles de filtrage utilisées lors de la recherche, on remarque que, lorsque les rayons approximatifs sont utilisés au lieu des rayons exacts, seules les distances minimales et maximales entre le vecteur requête et les *clusters* sont modifiées. La distance maximale intervient uniquement dans la première règle de filtrage. Son objectif est de majorer la distance entre le vecteur requête et son k^e plus proche voisin, c'est-à-dire calculer $\widehat{\text{dmax}}_m$. Comme cette distance est calculée uniquement pour les *clusters* contenant au moins k vecteurs à l'intérieur de leur hypersphère approximative, cette distance n'a aucun effet sur la précision de la recherche.

Par contre, lorsque la distance minimale est calculée en utilisant le rayon approximatif, tous les vecteurs du *cluster* sont censés être à une distance $\widehat{\text{dmin}}$ du vecteur requête. Or, certains vecteurs se trouvant en dehors de l'hypersphère approximative peuvent être à une distance inférieure à $\widehat{\text{dmin}}$ du vecteur requête. De tels vecteurs ne sont pas pris en compte par les deux règles de filtrage. Si ces vecteurs sont des plus proches voisins d'un vecteur requête et, si leurs *clusters* sont filtrés par les règles de filtrage, alors le résultat de la recherche est approximatif car ces plus proches voisins là n'y figureront pas.

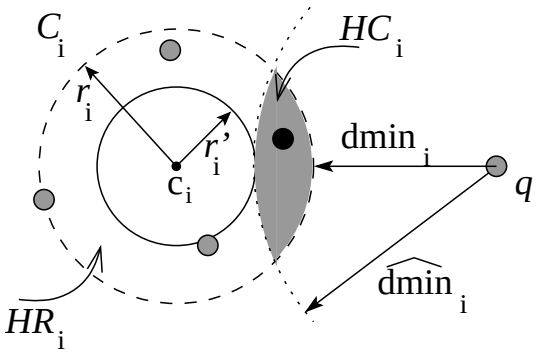


FIG. 5.3: Un *cluster* avec rayon exact (r_i) et rayon approximatif r'_i. q est un vecteur requête. HC_i est la partie de C_i ignorée lors de la recherche si le rayon approximatif est pris en compte au lieu du rayon exact.

Pour bien illustrer cette situation et localiser de manière précise la zone où de tels vecteurs se situent à l'intérieur d'un *cluster*, on considère un *cluster* C_i, de centre c_i et de rayon exact r_i (voir figure 5.3). Ce rayon définit une sphère englobante minimale exacte MHS_i *(Minimal bounding Hyper-Sphere)*, représentée en traits discontinus sur la figure 5.3. On suppose qu'un rayon approximatif r'_i a été calculé pour ce *cluster*. La sphère approximative correspondante est appelée AHS_i *(Approximate Hyper-Sphere)*. Elle est représentée par le cercle en traits continus sur la figure 5.3. On suppose aussi que parmi les vecteurs de C_i (représentés par des points sur la figure 5.3), un vecteur particulier v (point noir) fait partie de l'ensemble des plus proches voisins exacts du vecteur requête q. En considérant que le rayon de C_i est r'_i, et en appliquant les règles de filtrage, le *cluster* peut ne pas être considéré pertinent par rapport à q car tous ses vecteurs sont censés être à une distance supérieure à $\widehat{\text{dmin}}_i$, alors que v est plus proche de q que cela. Il est donc ignoré lors de la recherche et le vecteur v n'apparaît alors pas dans le résultat de la recherche.

La règle de filtrage est donc correcte pour tous les vecteurs du *cluster* dont la distance au vecteur requête est supérieure à $\widehat{\text{dmin}}_i$. Elle ne prend cependant pas en compte les vecteurs qui se trouvent, comme v, à une distance de q inférieure à $\widehat{\text{dmin}}_i$. Par conséquent, la partie du *cluster* C_i qui n'est pas prise en compte lors de la recherche si le rayon approximatif est utilisé par les règles de filtrage au lieu du rayon exact, est l'intersection entre MHS_i et l'hypersphère de centre q et de rayon égal à $\widehat{\text{dmin}}_i$. Cette zone est une (hyper) calotte sphérique. Elle est notée HC_i et est représentée dans la figure 5.3 en gris.

Calcul du nombre de vecteurs ignorés. La probabilité d'ignorer l'un des plus proches voisins, v, dans le *cluster* C_i correspond donc à la probabilité d'appartenance de v à la calotte sphérique HC_i. Cette probabilité est égale au rapport entre le nombre de vecteurs de C_i appartenant à la calotte sphérique, noté N_{HC_i}, et le nombre total des vecteurs de C_i, noté N_i :

$$P(\text{miss}(v)) = \frac{N_{HC_i}}{N_i}. \tag{5.9}$$

On remarque que N_{HC_i} dépend à la fois de la distance du vecteur par rapport au *cluster* et de sa localisation autour de celui-ci. Or, nous rappelons que le calcul des rayons approximatifs s'effectue hors ligne et doit donc être indépendant des vecteurs requêtes.

Pour lever ces deux dépendances, nous proposons d'effectuer deux approximations.

1. On remarque que le volume de HC_i dépend de la distance entre q et le *cluster*: plus q est éloigné de C_i, plus grand est le volume de HC_i, puisque la courbure de HC_i, côté du centre du *cluster* tend à s'aplatir. Ce volume est maximal lorsque la distance entre q et le *cluster* tend vers l'infini. Dans ce cas, HC_i est défini par l'intersection entre MHS_i et le sous-espace contenant q et limité par l'hyperplan orthogonal à (q, c_i) situé à une distance $\widehat{\text{dmin}}_i$ de q. La figure 5.4 illustre ce cas particulier dans un espace de dimension deux. HC_i est noté $\widehat{HC}_i$ dans ce cas là. Par conséquent, pour lever la première dépendance, une solution possible est de surestimer HC_i en considérant le cas le plus défavorable, c'est-à-dire lorsque la distance entre q et le *cluster* est infinie. Ceci permet de faire une estimation « conservative » du nombre de points dans la calotte.
L'inégalité 5.7 devient ainsi :

$$\frac{N_{\widehat{HC}_i}}{N_i} \leqslant \alpha, \tag{5.10}$$

où $N_{\widehat{HC}_i}$ est le nombre de vecteurs se trouvant dans $\widehat{HC}_i$.

2. L'autre dépendance concerne la position du vecteur requête autour du *cluster*. Le nombre de vecteurs dans $\widehat{HC}_i$ dépend du nombre de vecteurs dans la couronne HR_i et de leur distribution. Une façon d'éviter cette dépendance est de supposer que les vecteurs de C_i sont distribués de manière isotrope[2] à l'intérieur de HR_i, c'est-à-dire que $N_{\widehat{HC}_i}$ calculé

[2]Les distributions des vecteurs dans les couronnes HR_i des *clusters* ne sont pas forcément identiques. Elles partagent uniquement la propriété d'isotropie.

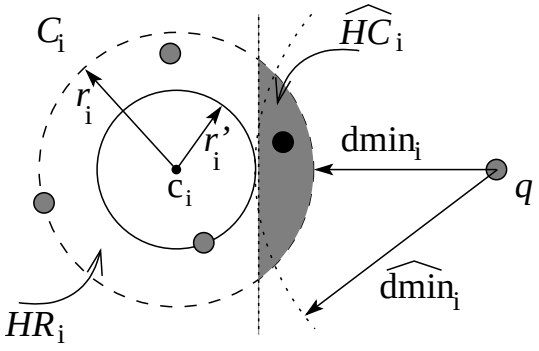


FIG. 5.4: Approximation de HC_i. Cette approximation est nécessaire pour s'affranchir de la dépendance entre le volume de HC_i et la distance de q par rapport au *cluster* C_i.

pour différentes positions de q autour du *cluster* est constant. À titre d'exemple, si les vecteurs de C_i suivent une distribution gaussienne, cette hypothèse revient à supposer que la matrice de covariance est égale à aI où I est la matrice identité et a un nombre réel positif. Cette hypothèse d'isotropie de la distribution des vecteurs dans la couronne, associée à la propriété d'isotropie de la forme hypersphérique, nous permet d'exprimer le nombre de vecteurs contenus dans $N_{\widehat{HC}_i}$ à l'aide des volumes de $\widehat{HC}_i$ et de $\widehat{HR}_i$ de la manière suivante :

$$N_{\widehat{HC}_i} = N_{HR_i} \frac{\text{vol}(\widehat{HC}_i)}{\text{vol}(MHS_i) - \text{vol}(AHS_i)} \tag{5.11}$$

vol étant une fonction de calcul du volume.
La propriété d'isotropie des hypersphères garantit que le volume de $\widehat{HC}_i$ est constant quel que soit la position du vecteur requête autour du *cluster*. Cette propriété est l'une des motivations principales du choix de la forme hypersphérique comme forme englobante des *clusters*. L'hypothèse d'isotropie de la distribution des vecteurs assure quant à elle que le nombre de vecteurs dans $\widehat{HC}_i$ est constant.

Calcul du rayon approximatif. En remplaçant la valeur de $N_{\widehat{HC}_i}$ dans l'inégalité 5.10, on obtient :

$$\frac{\text{vol}(\widehat{HC}_i)}{\text{vol}(MHS_i) - \text{vol}(AHS_i)} \frac{N_{HR_i}}{N_{C_i}} \leqslant \alpha \tag{5.12}$$

Dans cette inégalité, le seul paramètre inconnu est r'_i. Les entités $\text{vol}(\widehat{HC}_i)$, $\text{vol}(AHS_i)$ et N_{HR_i} en dépendent. Par conséquent, pour calculer r'_i qui vérifie la contrainte locale de précision, il suffit de résoudre l'inégalité 5.12. Notons que toute valeur de r_i qui satisfait cette inégalité est une solution possible. Cependant, comme l'objectif est de calculer un rayon approximatif qui garantit un niveau d'imprécision inférieur, *mais aussi le plus proche possible* du niveau d'imprécision α, il faut donc trouver le plus petit rayon approximatif compris dans l'intervalle $[0, r_i]$ qui vérifie l'inégalité 5.12.

Cette inégalité ne peut se résoudre analytiquement car le nombre de vecteurs dans la calotte HR_i ne peut s'exprimer analytiquement. Elle se résout donc numériquement dans

l'intervalle $[0, r_i]$. Nous remarquons que la partie gauche de l'inégalité 5.12 est une fonction décroissante en r'_i, c'est-à-dire que le nombre $N_{\widehat{HC}_i}$ de vecteurs dans la calotte augmente lorsque le rayon approximatif diminue. Elle est nulle lorsque le rayon approximatif est égal à r_i, et elle prend sa valeur maximale lorsque le rayon approximatif est égal à 0. Notons au passage que les valeurs possibles de α sont comprises entre 0,5 et 1 car r'_i doit forcément être dans l'intervalle $[0, r_i]$. Lorsque le rayon approximatif est nul, la calotte $\widehat{HC}_i$ représente en volume la moitié du *cluster*. La probabilité pour un vecteur v de s'y trouver et donc d'être ignoré est égale à 0,5. Lorsque $r'_i = r_i$, la recherche est exacte et, par conséquent, la probabilité d'ignorer v est nulle.

Algorithme 4 Calcul du rayon approximatif

Entrées :

C_i : un *cluster* de rayon exact r_i ;
N_i : le nombre de vecteurs du *cluster* C_i ;
d : dimension des vecteurs ;
$tabDist_i$: tableau contenant les distances entre les vecteurs de C_i et son centre ;
α : niveau d'imprécision ;
E : une constante. Erreur absolue dans le calcul du rayon approximatif.

Sortie :

r'_i : le rayon approximatif du *cluster* C_i qui garantit que la contrainte locale de précision relative à α est vérifiée.

Début
sup $\leftarrow r_i$;
inf $\leftarrow 0$;
$r'_i \leftarrow \frac{\text{sup+inf}}{2}$;
$p \leftarrow$ `eval_proba`(r'_i, r_i, N_i, d, $tabDist_i$);
tant que ($p \neq \alpha$) et (sup - inf > E) **faire**
  si $p > \alpha$ **alors**
    inf $\leftarrow r'_i$;
  fin si
  si $p < \alpha$ **alors**
    sup $\leftarrow r'_i$;
  fin si
  $r'_i \leftarrow \frac{\text{sup+inf}}{2}$;
  $p \leftarrow$ `eval_proba`(r'_i, r_i, N_i, d, $tabDist_i$);
fin tant que
Fin

En profitant de la propriété de monotonie de la partie gauche de l'inégalité 5.12, nous proposons de résoudre cette inégalité en utilisant une recherche dichotomique dans l'intervalle $[0, r_i]$. Initialement, l'intervalle considéré est $[0, r_i]$. La partie gauche de l'inégalité 5.12 est évaluée pour un rayon approximatif égal à $r_i/2$. Si le résultat est supérieur à α, alors la recherche est relancée dans l'intervalle $[r_i/2, r_i]$, sinon, le nouvel intervalle de recherche est $[0, r_i/2]$. Cette recherche se poursuit jusqu'à ce que l'évaluation de la partie gauche de l'inégalité 5.12 soit égale à α ou bien que la largeur de l'intervalle de recherche devienne

Algorithme 5 Fonction `eval_proba`

Entrées :

r'_i : un rayon approximatif du *cluster* C_i ;
r_i : le rayon exact du *cluster* C_i ;
N_i : le nombre de vecteurs *cluster* C_i ;
d : dimension des vecteurs ;
$tabDist_i$: tableau contenant les distances entre les vecteurs de C_i et son centre ;

Sortie :

p : la probabilité d'appartenance des vecteurs du *cluster* C_i à la calotte $\widehat{HC_i}$.

Début
$nbVectHR \leftarrow 0$; // Nombre de vecteurs dans HR_i.
Pour $j = 1$ à N_i **faire**
si $tabDist_i[j] \leqslant r'_i$ **alors**
$nbVectHR \leftarrow nbVectHR + 1$;
fin si
fin Pour
$VolHC \leftarrow$ `vol_calotte`(d, r_i, r'_i) ; // calcul du vol. de la calotte hypersphérique
$VolMHS \leftarrow$ `vol_hypersphere`(d, r_i) ; // calcul du vol. de l'hypersphère exacte de c_i
$VolAHS \leftarrow$ `vol_hypersphere`(d, r'_i) ; // calcul du vol. de l'hypersphère approx. de c_i
$p \leftarrow \frac{VolHC}{VolMHS - VolAHS} \frac{nbVectHR}{N_i}$;
Fin

plus petite qu'un seuil prédéfini. Ce seuil correspond à l'erreur absolue de calcul de r'_i. Dans le premier cas, le rayon approximatif recherché, r'_i, est égal à la valeur qui a servi à l'évaluation de l'inégalité 5.12. Dans le deuxième cas, r'_i est fixé à une valeur arbitraire dans l'intervalle de recherche final.

Pour évaluer la partie gauche de l'inégalité 5.12 pour un rayon approximatif donné, seul le terme N_{HR_i} dépend de la distribution des vecteurs. Pour évaluer N_{HR_i}, nous calculons une seule fois les distances entre tous les vecteurs du *cluster* et le centre de celui-ci. Ces distances sont stockées dans un tableau. Elles permettent de déduire, pour toute valeur de r'_i, le nombre de vecteurs dans la couronne HR_i en effectuant un simple parcours du tableau de distance et en comptant le nombre de vecteurs se trouvant à une distance inférieure à r'_i. Ainsi, il n'est pas nécessaire de recalculer toutes les distances entre les vecteurs et le centre du *cluster* à chaque fois que l'on désire calculer N_{HR_i} pour un rayon approximatif particulier.

La procédure de calcul du rayon approximatif est détaillée dans l'algorithme 4. Dans cet algorithme, [inf,sup] est l'intervalle de recherche courant. La fonction `eval_proba` permet d'évaluer la partie gauche de l'inégalité 5.12. Elle est présentée dans l'algorithme 5. Cette fonction repose sur le calcul des volumes des hypersphères exacte et approximative, ainsi que sur le calcul du volume de l'hypercalotte sphérique. Les formules mathématiques pour le calcul de ces volumes sont les suivantes.

Volumes des hypersphères. Le volume d'une hypersphère de rayon r en dimension d est donné par :

$$V_d(r) = \frac{\pi^{d/2}\, r^d}{\Gamma(1 + d/2)}. \tag{5.13}$$

Dans cette formule, intervient la fonction Γ donnée dans le cas général par :

$$\Gamma(x) = \int_0^{+\infty} t^{x-1} e^t dt. \tag{5.14}$$

Dans notre cas, nous utilisons sa version récursive :

$$\begin{cases} \Gamma(0) & = 1, \\ \Gamma(1) & = 1, \\ \Gamma(\frac{1}{2}) & = \sqrt{\pi}, \\ \Gamma(x+1) & = x\Gamma(x). \end{cases}$$

Le volume $V_d(r)$ s'exprime donc comme suit :

$$V_d(r) = \begin{cases} \frac{1}{(d/2)!}\pi^{(d/2)} r^d & \text{si } d \text{ est pair,} \\ \frac{2^{(\frac{d-1}{2})}}{d!!}\pi^{(\frac{d-1}{2})} r^d & \text{si } d \text{ est impair,} \end{cases}$$

où $d!! = d(d-2)(d-4)\ldots$

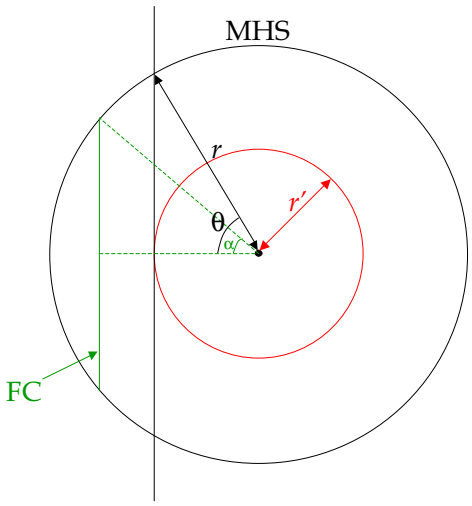


FIG. 5.5: Calcul du volume de l'hypercalotte sphérique.

Volumes des hypercalottes sphériques. Pour définir une hypercalotte sphérique, on a besoin d'un rayon exact r et d'un rayon approximatif r'. Pour expliquer la façon dont

est calculée ce volume, nous considérons le cas de dimension 2 (*cf.* figure 5.5). Dans cette figure, FC est une fine couche de la calotte. Dans le cas où l'hypersphère *MHS* est en dimension d, FC est une hypersphère de dimension $d-1$. Le volume de l'hypercalotte *HC* se calcule donc en sommant les volumes de FC pour α variant entre 0 et θ :

$$V_{HC}(r, r') = \int_0^{\theta} V_{d-1}(r\ sin(\alpha))\ r\ sin(\alpha)\ d\alpha. \tag{5.15}$$

$$= \frac{\pi^{\frac{d-1}{2}}\ r^d}{\Gamma(\frac{d+1}{2})} \int_0^{\theta} sin^d(\alpha)\ d\alpha. \tag{5.16}$$

où $\theta = argcos(\frac{r'}{r})$.

Dans cette formule, $\int_0^{\theta} sin^d(\alpha)\ d\alpha$ dépend également de la parité de d. Cette entité s'exprime comme suit [Jeffrey, 1995] :

$$\begin{cases} \text{si d=2n,} & \int_0^{\theta} sin^{2n}(\alpha)\ d\alpha = \frac{1}{2^{2n}}\binom{n}{2n}\theta + \frac{(-1)^n}{2^{2n-1}}\sum_{j=0}^{n-1}(-1)^j\binom{j}{2n}\frac{sin((2n-2j)\theta)}{2n-2j} \\ \text{si d=2n+1,} & \int_0^{\theta} sin^{2n+1}(\alpha)\ d\alpha = \frac{(-1)^{n+1}}{2^{2n}}\left[\sum_{j=0}^{n}(-1)^j\binom{j}{2n+1}\frac{cos((2n+1-2j)\theta)-1}{2n+1-2j}\right] \end{cases}$$

5.3.3 Retour sur l'hypothèse d'isotropie

La méthode pour le calcul du rayon approximatif présentée dans la section précédente, repose sur une approximation du nombre de vecteurs qui se trouvent dans la calotte $N_{\widehat{HC}_i}$. Pour un *cluster* C_i, le rayon approximatif r'_i calculé pour un niveau d'imprécision α garantit que :

$$P(\text{miss}(PPV(q))|PPV(q) \in C_i) \leqslant \alpha_i. \tag{5.17}$$

($PPV(q)$ désigne un des plus proches voisins du vecteur requête q).

En fait, à cause de l'approximation faite lors du calcul de $N_{\widehat{HC}_i}$, cette contrainte n'est vérifiée que pour les vecteurs requêtes qui se situent selon une direction du *cluster* tel que :

$$N_{\widehat{HC}_i} \leqslant N_{HR_i}\frac{\text{vol}(\widehat{HC}_i)}{\text{vol}(HR_i)}. \tag{5.18}$$

Dans le cas contraire, la probabilité de ne pas retrouver l'un des plus proches voisins de q, sachant que ce plus proche voisin appartient à C_i, est supérieure à α.

La fiabilité du processus de contrôle de la précision dépend donc de la validité de cette approximation qui, elle, est proportionnelle au nombre de vecteurs requêtes pour lesquels l'inégalité 5.18 est vérifiée.

Cette approximation est tout à fait valide si la distribution des vecteurs au sein du *cluster* est isotrope. Cependant, même pour un *cluster* dont la distribution des vecteurs n'est pas isotrope, la proportion des vecteurs requêtes autour du *cluster* pour lesquels l'inégalité 5.18 est vérifiée est très élevée. Prenons l'exemple de la distribution de la figure 5.6. La distribution des vecteurs de ce *cluster* n'est clairement pas isotrope. Cependant, l'inégalité 5.18 n'est pas vérifiée seulement lorsque les vecteurs se trouvent dans la direction du nuage de vecteurs (le long ou proche de la droite discontinue bleue). Pour toutes les autres

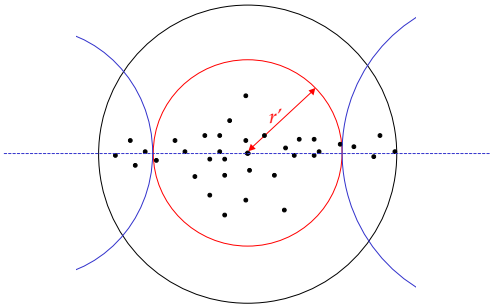


FIG. 5.6: Exemple d'un *cluster* dont la distribution des vecteurs n'est pas isotrope mais dont le $P(H)$ est élevé.

positions possibles du vecteur requête, l'inégalité 5.18 est vérifiée. Cette proportion est encore plus élevée en grande dimension en raison du grand nombre de directions possibles du vecteur requête autour du *cluster*.

Pour réduire l'impact de l'approximation de $N_{\widehat{HC}_i}$ sur la fiabilité du processus de contrôle de la précision de la recherche, il est nécessaire de prendre explicitement en compte le cas des vecteurs pour lesquels l'inégalité 5.18 n'est pas vérifiée. Ceci revient à diviser l'ensemble des vecteurs requêtes possibles en deux sous-ensembles selon que l'inégalité 5.18 est vérifiée ou non. Pour le sous-ensemble pour lequel l'inégalité 5.18 est vérifiée, $N_{\widehat{HC}_i}$ est approximé par :

$$N_{HR_i} \frac{\text{vol}(\widehat{HC}_i)}{\text{vol}(HR_i)}.$$

Pour l'autre sous-ensemble de vecteurs, une façon de faire une approximation « conservative » de $N_{\widehat{HC}_i}$ est de considérer que tous les vecteurs de la couronne HR_i se trouvent dans la calotte sphérique HC_i et, d'approximer ainsi $N_{\widehat{HC}_i}$ par N_{HR_i}.

Cela nous permet de proposer une solution combinant les deux approximations de $N_{\widehat{HC}_i}$ en fonction de la proportion des vecteurs requêtes pour lesquels l'inégalité est vérifiée, et d'exprimer la probabilité $P(\text{miss}(v))$ sous la forme d'une somme pondérée de deux probabilités comme suit :

$$P(\text{miss}(v)) = P(\text{miss}(v)|H_i)P(H_i) + P(\text{miss}(v)|\overline{H}_i)P(\overline{H}_i) \tag{5.19}$$

$$P(\text{miss}(v)) = \left(\frac{\text{vol}(\widehat{HC}_i)}{\text{vol}(HR_i)} \frac{N_{HR_i}}{N_i}\right) P(H_i) + \left(\frac{N_{HR_i}}{N_i}\right) P(\overline{H}_i) \tag{5.20}$$

où $P(H_i)$ est la probabilité pour que l'inégalité 5.18 soit vérifiée. En d'autres termes, c'est la proportion des vecteurs requêtes (parmi l'ensemble des vecteurs requêtes possibles) qui sont situés dans une direction autour du *cluster* selon laquelle le nombre de vecteurs dans la calotte vérifie l'inégalité 5.18.

Pour calculer le rayon approximatif tel que la contrainte globale de la précision $P(\text{miss}(v)) \leqslant \alpha$ soit vérifiée, il suffit donc de résoudre :

$$\left(\frac{\text{vol}(\widehat{HC}_i)}{\text{vol}(HR_i)}\frac{N_{HR_i}}{N_i}\right) P(H_i) + \left(\frac{N_{HR_i}}{N_i}\right) P(\overline{H}_i) \leqslant \alpha, \tag{5.21}$$

en utilisant la méthode par recherche dichotomique décrite dans la section précédente.

Dans cette inégalité, $P(H_i)$ est le seul paramètre inconnu. Nous avons expliqué intuitivement plus haut que la valeur de ce paramètre est très élevée même lorsque la distribution des vecteurs au sein du *cluster* n'est pas isotrope. Néanmoins, pour effectuer une estimation plus rigoureuse de ce paramètre, nous utilisons une méthode basée sur des techniques de sondage aléatoire simple [Barnett, 2002]. Cette méthode est détaillée dans la section suivante.

5.3.4 Estimation de $P(H_i)$

Pour estimer $P(H_i)$ pour un *cluster* C_i et un niveau d'imprécision α, il suffit de considérer un échantillon de vecteurs requêtes (tirés aléatoirement autour du *cluster*) et d'estimer la proportion des cas où l'inégalité 5.18 est vérifiée.

La procédure d'estimation se résume dans les points suivants :
- calculer le rayon approximatif en supposant que les vecteurs de C_i sont distribués de manière isotrope ;
- estimer à l'aide de l'équation 5.11 le nombre de vecteurs appartenant à la calotte sphérique $\widehat{HC}_i$ sous l'hypothèse d'isotropie, soit $N_{\widehat{HC}_i|H}$;
- pour chaque vecteur de l'échantillon des vecteurs requêtes, calculer empiriquement le nombre de vecteurs de C_i qui appartiennent effectivement à HC_i, soit $N_{\widehat{HC}_i|\,\text{eff}}$;
- déduire $P(H_i)$ comme étant la proportion des vecteurs requêtes pour lesquels $N_{\widehat{HC}_i|H}$ est inférieur à $N_{\widehat{HC}_i|\,\text{eff}}$.

Cette procédure permet d'effectuer une estimation empirique de $P(H_i)$. Dans cette section, nous montrons tout d'abord que cet estimateur converge en probabilité vers la vraie valeur de $P(H_i)$. Nous étudions ensuite la relation entre la précision de l'estimation et la taille de l'échantillon de vecteurs requêtes. Enfin, nous décrivons la méthode que nous avons retenue pour la génération de l'échantillon des vecteurs requêtes.

Convergence de l'estimateur. L'estimation empirique de $P(H_i)$ revient à effectuer m tests ponctuels où m est la taille de l'échantillon des vecteurs requêtes. La conclusion d'un test est binaire. Cette série de tests permet donc de créer une suite de variables aléatoires $(X_j)_j$ de Bernoulli. X_j vaut 1 si $N_{\widehat{HC}_i|H}$ est inférieur à $N_{\widehat{HC}_i|\,\text{eff}}$ (i.e l'inégalité 5.18 est vérifiée), 0 sinon.

Ainsi, on a $P(X_j = 1) = P(H_i) = p$ et $P(X_j = 0) = 1 - P(H_i) = 1 - p$. On introduit alors la fréquence empirique :

$$\hat{p}_m = \frac{1}{m}\sum_{i=1}^{m} X_i = \bar{X}_m, \tag{5.22}$$

avec $E(\hat{p}_m) = p$ et $var(\hat{p}_m) = \frac{p(1-p)}{m}$. En appliquant l'inégalité de Bienaymé-Tchébycheff, on a :

$$\forall \varepsilon > 0, P(|p_m - p| > \varepsilon) \leqslant \frac{p(1-p)}{\varepsilon^2}. \tag{5.23}$$

Il en résulte que

$$\hat{p}_m \underset{p}{\rightarrow} p$$

lorsque m tend vers l'infini.

Ce résultat, connu sous le nom de théorème de Bernoulli, montre bien que la fréquence empirique de l'événement « l'inégalité 5.18 est vérifiée » converge en probabilité vers la probabilité de réalisation de cet événement lorsque le nombre de tests ponctuels augmente indéfiniment. Cela n'étant pas possible en pratique, nous présentons dans la section suivante une étude de la précision de l'estimation en fonction du nombre d'échantillons.

Précision de l'estimateur *vs.* taille de l'échantillon. Étant donné que la taille de l'échantillon des vecteurs requêtes utilisés pour l'estimation de $P(H_i)$ est finie, il nous faut trouver un moyen de juger de la qualité de l'estimation en fonction de la taille de l'échantillon. Pour cela, nous utilisons le théorème central-limite :

$$\sqrt{m}\,\frac{\hat{p}_m - p}{\sqrt{p(1-p)}} \underset{loi}{\longrightarrow} \mathcal{N}(0,1). \tag{5.24}$$

qui nous permet d'écrire :

$$P\left(-\mu_\theta < \sqrt{m}\frac{\hat{p}_m - p}{\sqrt{p(1-p)}} < \mu_\theta\right) = 1 - \theta, \tag{5.25}$$

où μ_θ peut être facilement déterminé à partir de θ car $\sqrt{m}\frac{\hat{p}_m - p}{\sqrt{p(1-p)}}$ suit une loi normale centrée réduite.

Pour déterminer un intervalle de confiance de p à partir de l'équation 5.25, nous approchons l'écart-type $\sqrt{p(1-p)}$ par $\sqrt{\hat{p}_m(1-\hat{p}_m)}$. Nous avons ainsi avec une probabilité de $1-\theta$:

$$\hat{p}_m - \mu_\theta\sqrt{\frac{\hat{p}_m(1-\hat{p}_m)}{m}} < p < \hat{p}_m + \mu_\theta\sqrt{\frac{\hat{p}_m(1-\hat{p}_m)}{m}}, \tag{5.26}$$

avec $\mu_\theta = \Phi^{-1}(1 - \frac{\theta}{2})$ où Φ est la fonction de densité d'une distribution normale centrée réduite.
Si on pose :

$$d = \mu_\theta\sqrt{\frac{\hat{p}_m(1-\hat{p}_m)}{m}}, \tag{5.27}$$

alors d correspond à la moitié de la largeur de l'intervalle de confiance. Le nombre m de vecteurs requêtes est donc donné par :

$$m = \frac{\hat{p}_m(1-\hat{p}_m)\mu_\theta^2}{d^2}. \tag{5.28}$$

Cette équation fournit ainsi deux critères pour fixer le nombre de vecteurs requêtes: la largeur de l'intervalle de confiance d, et le niveau de confiance θ exprimé au travers du paramètre μ_θ. θ correspond à la probabilité pour que $\hat{p}_m$ soit dans l'intervalle de confiance. Quant à $\hat{p}_m(1-\hat{p}_m)$, en l'absence de connaissance *a priori*, on utilise une valeur de $\hat{p}_m$ qui maximise m. L'étude de la fonction $f(p) = p(1-p)$ montre que celle-ci atteint son maximum lorsque $\hat{p}_m = 0{,}5$. Nous fixons par conséquent $\hat{p}_m$ à 0,5.

À titre d'exemple, si on fixe d à 0,01 et θ à 0,95, le nombre de vecteurs requêtes à générer est égal à :

$$m = \frac{0{,}5(1-0{,}5)1{,}96^2}{0{,}01^2} = 9604 \text{ vecteurs.}$$

Ce nombre d'échantillons garantit que la valeur estimée de $P(H_i)$ se situe dans un intervalle de $\pm$ 0,01 autour de la vraie valeur avec un niveau de confiance de l'ordre de 95 %.

Génération de l'échantillon de vecteurs requêtes. La génération des vecteurs requêtes s'effectue par un tirage aléatoire uniforme sur une hypersphère de même centre que le *cluster*. L'idée est de générer un échantillon qui permet de couvrir l'ensemble des directions possibles où peut se situer le vecteur requête autour du *cluster*. Le rayon de l'hypersphère sur laquelle sont tirés les vecteurs requêtes doit être cependant très grand afin de maximiser le nombre de vecteurs du *cluster* qui risquent d'être ignorés (les vecteurs qui appartiennent à la calotte HR_i. Le volume de celle-ci augmente avec la distance entre le vecteur requête et le *cluster*). Ceci permet de prendre en compte le pire cas et d'éviter de générer plusieurs vecteurs requêtes à des distances différentes du *cluster* selon la même direction.

Pour effectuer ce tirage aléatoire, nous utilisons le théorème proposé à l'origine par G.W. Brown [Brown, 1956] et présenté dans [Knuth, 1997].

Théorème 3 *Soit $Y_1, Y_2, \ldots, Y_d$ d variables aléatoires normales centrées réduites ($Y_k \sim \mathcal{N}(0,1)$). Soit de plus $\rho = \sqrt{\sum_{k=1}^{d} Y_k^2}$,*

alors le vecteur normé $U = \frac{1}{\rho}(Y_1, Y_2, \ldots, Y_d)$ suit une distribution uniforme sur une hypersphère de centre (0,...,0) et de rayon 1.

Ce théorème décrit une méthode permettant de générer aléatoirement des vecteurs distribués uniformément sur la surface de l'hypersphère unité à partir de nombres distribués selon une loi normale centrée réduite. Le passage à des vecteurs requêtes distribués sur une hypersphère autour du *cluster* est direct. Il suffit de translater les vecteurs au centre du *cluster* puis d'appliquer un facteur multiplicatif pour augmenter le rayon de l'hypersphère sur laquelle les vecteurs sont distribués.

Expérimentations. Les expérimentations effectuées pour l'estimation du paramètre $P(H)$ pour chaque *cluster* sont présentées dans la section 6.4 (page 134). Ces expérimentations montrent que $P(H)$ est très proche de 1 (en général $\approx$ 0,99) pour différents ensembles de données. Le considérer égal à 1 est donc une hypothèse tout à fait acceptable et confirme bien l'explication intuitive donnée précédemment dans la section 5.3.3.

5.4 Extension de la méthode de base : deux autres modes de recherche possibles

Dans les sections précédentes, nous avons présenté une méthode de recherche approximative de plus proches voisins qui permet de contrôler la précision de la recherche. Il est toutefois possible d'effectuer d'autres modes de recherche en utilisant les mêmes structures que celles construites par la méthode de recherche approximative.

Après le regroupement des vecteurs de la base en paquets à l'aide de l'algorithme de *clustering* présenté dans le chapitre 4, l'implémentation d'une recherche incrémentielle de plus proches voisins est immédiate. Lors de la recherche, il suffit d'ordonner les *clusters* en fonction de leurs distances au vecteur requête puis de les traiter dans l'ordre. Après la lecture de chaque *cluster*, les résultats courants sont affichés à l'utilisateur. Celui-ci a la possibilité d'interrompre la recherche à n'importe quel moment. Ce mode de recherche est très pratique dans un système de recherche d'images par le contenu[3]. Dans ce cas, les images appartenant au résultat courant d'une requête donnée sont affichées et mises à jour au fur et à mesure de l'avancement de la recherche. L'utilisateur peut décider d'interrompre la recherche dès que les images dans le résultat courant sont jugées satisfaisantes. Cette décision est prise en considérant les résultats courants. L'utilisateur n'a cependant aucune idée sur l'amélioration possible des résultats s'il n'interrompt pas la recherche. Il serait donc intéressant de fournir à l'utilisateur, en plus du résultat courant, une indication sur l'amélioration des résultats si la recherche se poursuit. Un tel indicateur pourrait être par exemple, la probabilité de retrouver des vecteurs plus proches que les vecteurs courants. Dans le cas de la recherche d'images, cette probabilité serait la probabilité de trouver des images plus similaires que les images appartenant au résultat courant.

Pour calculer cette probabilité, nous nous basons sur l'analyse effectuée dans la section précédente. On considère un ensemble de données partitionné en N *clusters*. Pour chacun, nous supposons qu'un rayon exact a été calculé. Ce rayon définit une hypersphère englobante minimale exacte. Pour effectuer une recherche incrémentielle d'un vecteur requête q, il suffit d'ordonner les *clusters* selon leurs distances au vecteur requête (par exemple en utilisant la distance entre le vecteur requête et les centres des *clusters*). La liste des *clusters* ordonnés en fonction de cette distance est notée $LC(q) = C_1, C_2, \ldots, C_N$. Ces *clusters* sont ensuite traités dans l'ordre. Le traitement d'un *cluster* revient à charger ses vecteurs en mémoire et à calculer leurs distances par rapport au vecteur requête. Ces distances sont ensuite utilisées pour mettre à jour l'ensemble des k-plus proches voisins courants. Ces résultats courants sont ensuite affichés à l'utilisateur. La probabilité de retrouver des vecteurs plus proches que le k^e plus proche voisin courant dépend de la distance entre q et son k^e plus proche voisin courant. Cette distance définit un champ de recherche autour du *cluster* sous la forme d'une hypersphère centrée en q. Cette hypersphère dont le rayon est égal à la distance entre q et son k^e plus proche voisin courant est appelée hypersphère requête. Tout *cluster* dont l'hypersphère englobante possède une intersection non vide avec l'hypersphère requête est susceptible de contenir des vecteurs plus proches de q que le k^e plus proche voisin courant. Ceci est illustré dans la figure 5.7. Dans cette figure, $kppv_c$ est le k^e plus proche voisin courant de q. Sa distance au vecteur requête est D_{kppvc}. Les *clusters*

[3]Nous supposons que chaque vecteur décrit le contenu d'une image, et que la recherche des images les plus similaires à une image requête revient à effectuer une recherche de plus proches voisins en utilisant comme requête le vecteur décrivant l'image requête.

C_1, C_2, C_3 et C_4 appartiennent à la liste des *clusters* à traiter. Cependant, seuls les *clusters* C_1 et C_3 peuvent contenir des vecteurs plus proches de q que son k^e plus proche voisin courant. De tels vecteurs, s'ils existent, se trouveront forcément dans l'intersection entre l'hypersphère requête (de centre q et de rayon D_{kppvc}) et les hypersphères englobantes des *clusters* C_1 et C_3 (zones en gris dans la figure 5.7). Ces deux zones sont notées HC_1 et HC_3 respectivement pour C_1 et C_3.

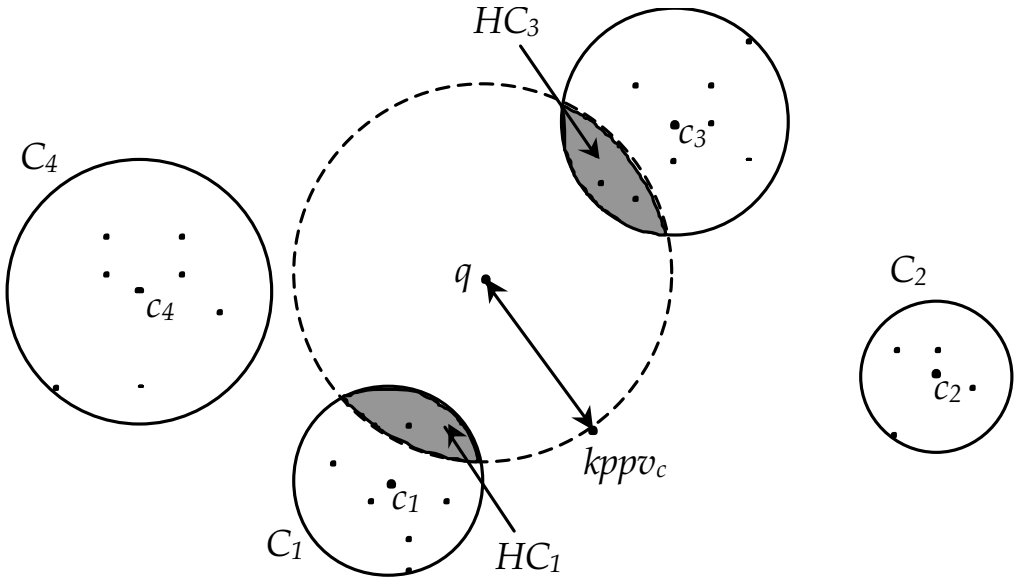


FIG. 5.7: Recherche incrémentielle. Exemple d'un vecteur requête et de 4 *clusters* candidats. Seuls les deux *clusters* C_1 et C_3 peuvent contenir des vecteurs plus proches de q que son k^e plus proche voisin courant.

La probabilité de retrouver des vecteurs dans C_1 plus proches de q que son k^e plus proche voisin courant est donc égale à la probabilité de trouver des vecteurs de C_1 dans HC_1. Le même raisonnement s'applique également à C_3. Notons ces probabilités P_1 et P_3 respectivement pour C_1 et C_3. Ainsi, la probabilité de retrouver des vecteurs plus proches de q que son k^e plus proche voisin courant est égale à : $P_1 + P_3$.

Dans le cas général, la probabilité de retrouver des vecteurs plus proches du vecteur requête que son k^e plus proche voisin dans une liste de *clusters* $C_1, C_2, \ldots, C_N$ non encore lus est donc égale à :

$$P = \sum_{i=1}^{N} P_i, \tag{5.29}$$

où P_i est la probabilité de trouver des vecteurs de C_i dans l'intersection de l'hypersphère requête et l'hypersphère du *cluster* C_i.

Pour estimer cette probabilité pour un *cluster* donné, considérons un vecteur requête q, son k^e plus proche voisin courant $kppv_c$ et un *cluster* C_i de la liste des *clusters* non encore

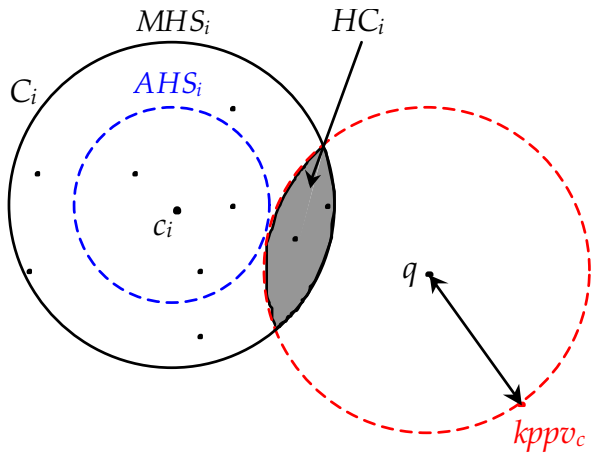


FIG. 5.8: Recherche incrémentielle. HC_i est la seule partie du *cluster* C_i qui peut contenir des vecteurs plus proches du vecteur requête que son k^e plus proche voisin courant.

lus (voir figure 5.8). Notons le nombre de vecteurs de ce *cluster* N_i, son centre c_i et son rayon (exact) r_i. Ce rayon définit une hypersphère englobante minimale du *cluster* C_i. Elle est notée MHS_i. L'intersection entre l'hypersphère requête (cercle en traits discontinus rouges dans la figure 5.8) et MHS_i définit une hypercalotte sphérique notée HC_i. L'hypersphère centrée en c_i et tangente à l'hypersphère requête est notée AHS_i.

Dans le *cluster* C_i, seuls les vecteurs appartenant à HC_i, sont plus proches de q que $kppv_c$. Nous avons donc :

$$P_i = \frac{N_{HC_i}}{N_i}, \tag{5.30}$$

où N_{HC_i} est le nombre de vecteurs de C_i appartenant à HC_i. Pour calculer cette probabilité, il suffit donc d'estimer N_{HC_i}. Si on utilise l'approximation basée sur l'hypothèse d'isotropie de la distribution des vecteurs du *cluster*, nous pouvons exprimer N_{HC_i} comme suit :

$$N_{HC_i} = \frac{vol(HC_i)}{vol(MHS_i) - vol(AHS_i)} \frac{N_{HR_i}}{N_i}. \tag{5.31}$$

D'où :

$$P_i = \frac{vol(HC_i)}{vol(MHS_i) - vol(AHS_i)} N_{HR_i}. \tag{5.32}$$

Ainsi, lors de la recherche en ligne, une probabilité d'amélioration des résultats courants peut être calculée par les équations 5.29 et 5.32.

Dans cette section, nous ne présentons pas l'analyse complète du cas où les vecteurs ne sont pas distribués de façon isotrope. Elle est identique à celle présentée dans la section 5.3.3.

5.5 Synthèse

Dans ce chapitre, nous avons présenté notre méthode pour la recherche approximative de k-plus proches voisins. Cette méthode permet de contrôler la précision de la recherche au travers d'un seul paramètre noté α et appelé niveau d'imprécision. α est la probabilité maximale de ne pas retrouver un des plus proches voisins dans le résultat approximatif. Hors ligne, elle utilise l'algorithme de *clustering* proposé dans le chapitre 4 pour regrouper les données en petits paquets. Pour chaque paquet, elle calcule plusieurs rayons approximatifs, un rayon par niveau d'imprécision prédéfini.

En ligne, lors de la recherche, le niveau d'imprécision choisi détermine pour chaque *cluster* , le rayon à prendre en compte. Ce rayon définit une cellule hypersphérique associée au *cluster*. Des règles de filtrage géométrique sont alors utilisées pour sélectionner les *clusters* qui ont le plus de chance de contenir les plus proches voisins recherchés. Ces règles utilisent uniquement les propriétés géométriques des cellules des *clusters*. Comme ces cellules sont approximatives car elles n'englobent pas la totalité des vecteurs du *cluster*, certains des plus proches voisins recherchés risquent de ne pas être retrouvés. Notre méthode permet cependant de borner la probabilité de cet événement.

Nous avons également proposé deux autres modes de recherches possibles. La recherche incrémentielle dont l'implémentation est immédiate et la recherche incrémentielle avec un indicateur permettant de renseigner l'utilisateur sur les chances d'amélioration des résultats courants. Ces deux modes de recherche sont des modes de recherche interactive car l'utilisateur peut interrompre la recherche à n'importe quel moment. Pour le premier, l'utilisateur prend la décision d'interrompre la recherche en se basant uniquement sur les résultats courants. Cependant, pour le deuxième, en plus des résultats courants, le système fournit à l'utilisateur une mesure qui est la probabilité pour que les résultats courants s'améliorent. L'utilisateur peut ainsi interrompre la recherche même si les résultats courants ne sont pas très satisfaisants si cette probabilité est très faible, car de toute façon l'amélioration des résultats est peu probable. Il peut aussi décider de ne pas interrompre la recherche même si les résultats courants sont bons, si cette probabilité est très forte. Dans ce cas, en poursuivant la recherche, il a de fortes chances de récupérer des résultats meilleurs. Ces deux modes de recherche n'ont cependant pas été expérimentés. Aucune évaluation ne sera donc présentée dans le chapitre 6. Ceci fait partie de nos travaux futurs.

Chapitre 6

Résultats expérimentaux

6.1 Introduction

Ce chapitre décrit les évaluations de performance de la technique de recherche approximative proposée. Ces évaluations concernent à la fois l'efficacité, la fiabilité et la qualité des réponses. Pour ce qui concerne le temps de réponse, nous comparons notre méthode à la recherche séquentielle car celle-ci est la seule technique de recherche exacte de plus proches voisins qui garde un coût linéaire quelle que soit la dimension des vecteurs. Les performances de toutes les autres techniques pour la recherche exacte se dégradent exponentiellement lorsque la dimension augmente [Weber *et al.*, 1998, Amsaleg & Gros, 2001]. Nous présentons également des expérimentations permettant de corroborer certaines des analyses effectuées lors de la conception de la méthode de recherche, en particulier, en ce qui concerne le paramètre $P(H)$ et la réduction des rayons des hypersphères.

Ce chapitre est organisé comme suit : après la présentation du contexte matériel des expérimentations et l'examen de la composition des bases de données utilisées, nous présentons deux expériences relatives aux paramètres internes de la méthode. La première étude concerne l'estimation du paramètre $P(H)$. Ce paramètre intervient dans le calcul des rayons approximatifs. La deuxième expérience étudie les variations du temps de réponse de la recherche en fonction du nombre de *clusters*. L'objectif est de déduire le nombre optimal de *clusters* en fonction de la taille de la base et de la dimension des vecteurs. À la fin de la section présentant cette étude, nous donnons, pour chaque ensemble de données utilisé dans les expériences suivantes, les caractéristiques de sa partition.

Ces deux paramètres étant fixés, une étude étendue des performances de notre méthode est alors menée. Les recherches effectuées dans toutes ces expériences sont des recherches de 20 plus proches voisins. Une étude des performances en fonction du nombre de plus proches voisins k est présentée à la fin de ce chapitre. Cette étude montre que, hormis quelques cas particuliers, les performances de notre méthode ne dépendent pas de ce paramètre.

Dans cette étude des performances, la qualité des résultats de la recherche approximative est étudiée en premier. L'objectif est d'évaluer la fiabilité du contrôle de la précision en comparant la précision effective des résultats obtenus par rapport à la précision fixée *a priori* au travers du paramètre α. Nous présentons ensuite des expérimentations pour étudier l'évolution du temps de réponse de la méthode lorsque la dimension et/ou la taille de la base augmentent. Nous étudions également la sensibilité des performances de la mé-

thode par rapport à l'algorithme utilisé pour le regroupement des données hors ligne. Pour cela, nous utilisons un autre algorithme pour le partitionnement des données. Il s'agit du SS-Tree. Une étude comparative de notre méthode par rapport à Clindex [Li *et al.*, 2002] est ensuite présentée. C'est la seule méthode de recherche approximative dont nous avons pu récupérer l'implémentation auprès des auteurs.

Enfin, nous concluons ces expérimentations par une étude des performances en fonction du nombre de plus proches voisins k.

6.2 Contexte des expérimentations

Nous avons implémenté l'algorithme de recherche ainsi que l'algorithme de *clustering* en C++. Ces algorithmes ont été exécutés sur une station de travail Sun Blade 100 sous SunOS 5.7, disposant d'un processeur UltraSPARC-IIe cadencé à 502 MHz, de 763 Mo de mémoire centrale et d'un disque local de 73 Go. Les temps d'exécution des algorithmes reportés dans la suite sont obtenus grâce à la fonction `getrusage()`.

Par ailleurs, afin de réduire le coût de calcul des distances, nous avons intégré l'option « distance partielle » présenté dans la section 1.4.6, page 24 à notre algorithme de recherche. Nous rappelons que cette option permet d'interrompre un calcul de distance dès que la distance partielle (calculée en considérant uniquement un sous-ensemble des composantes des vecteurs) est supérieure à la distance au k^e plus proche voisin courant. Dans ce cas, il n'est pas nécessaire d'aller jusqu'au bout dans le calcul de la distance car le vecteur considéré n'a aucune chance d'être plus proche que les k-plus proches voisins courants. Ainsi, plusieurs opérations d'addition et de multiplication sont sauvegardées au prix de quelques opérations de comparaison.

6.3 Nature des données

Pour conduire nos expérimentations, nous avons utilisé plusieurs bases de données réelles. La première est composée de 5 017 298 vecteurs de dimension 24. Ces vecteurs sont des descripteurs locaux d'images couleurs calculés à partir de 52 273 images. 610 images sont issues de la base MOVI[1], alors que les autres sont des images non contigues extraites à partir de séquences vidéo d'émissions de télévision. Nous avons eu recours à ce procédé car au début de cette thèse nous ne disposions pas d'un ensemble d'images fixes de cette taille. Notons toutefois que cela n'a aucun impact sur les temps de réponse de notre méthode de recherche. Nous avons vérifié expérimentalement que, pour deux ensembles de vecteurs de même taille, l'un composé de descripteurs calculés sur des images de cette base et l'autre composé de descripteurs calculés sur des images fixes, les performances de notre méthode sont identiques.

Cette base occupe 478 Mo. Les descripteurs utilisés sont les descripteurs locaux développés par [Florack *et al.*, 1994, Schmid & Mohr, 1997]. Plus de détails sur le calcul et les propriétés de ces descripteurs sont donnés dans la section 7.2.1, page 163. En moyenne, 96 descripteurs sont calculés par image. Pour certaines expériences, nous avons utilisé une sous-base contenant 1 816 images (soit 411 409 descripteurs). Cette sous-base a déjà été utilisée dans des travaux antérieurs conduits par Amsaleg *et al.* [Amsaleg & Gros, 2001].

[1] Disponible en ligne en suivant l'URL : `http://www.irisa.fr/texmex/bases_images/index.html`

La deuxième base est en réalité composée de cinq bases, chacune contenant 30 021 histogrammes de couleurs calculés à partir du même ensemble d'images fixes mais en faisant varier la dimension des histogrammes. Les dimensions originales des histogrammes sont 64, 125, 216, 343 et 512. Une analyse en composantes principales a été ensuite effectuée sur chaque ensemble pour en réduire la dimension. Les dimensions réduites obtenues sont respectivement 25, 45, 70, 100 et 150. Les proportions de l'inertie expliquée par les composantes retenues se situent entre 86 % et 88 %. Notons que pour effectuer cette réduction nous n'avons pas utilisé uniquement le critère basé sur la proportion de l'inertie expliquée. Comme nous l'avons expliqué dans la section 2.5.1.3, ce critère est clairement insuffisant dans le contexte de la recherche de plus proches voisins. Pour cela, nous nous sommes également appuyés sur l'expérience présentée dans la section 2.5.1.3. Cette expérience a montré que la précision des plus proches voisins est de 0,97 pour une proportion de l'inertie expliquée de l'ordre de 87 %. Nous remarquons enfin que cette réduction n'a pas été effectuée dans le but de s'affranchir de la malédiction de la dimension mais simplement pour réduire la corrélation des données.

Dans la suite de ce chapitre, nous utilisons les notations suivantes :

BD_1 la base des 5 017 298 de vecteurs de dimension 24 ;

BD_2 la base des 411 409 de vecteurs de dimension 24 ;

$BD_3 = \{BD_3^{25},\ BD_3^{45},\ BD_3^{70},\ BD_3^{100},\ BD_3^{150}\}$ l'ensemble des bases composées de 30 021 histogrammes de couleurs de dimension variable. La notation BD_3^* sera utilisée pour faire référence à ces cinq ensembles.

BD_1 est utilisée dans l'expérience qui étudie l'évolution du temps de réponse lorsque le nombre de vecteurs augmente ; BD_3 est utilisé notamment dans l'expérience qui étudie l'évolution du temps de réponse lorsque la dimension augmente ; et BD_2 et BD_3 sont utilisées dans les autres expériences. L'idée est d'utiliser dans chaque expérience deux distributions de données différentes.

Dans toutes les expériences qui suivent, les vecteurs requêtes utilisés sont tirés aléatoirement à partir des bases de données dans lesquelles la recherche est effectuée. Dans des expériences non reportées ici, nous avons vérifié que les performances de la méthode sont pratiquement identiques quelle que soit la provenance des vecteurs requêtes, c'est-à-dire qu'ils soient des vecteurs de la base, des vecteurs d'une autre base, ou tout simplement des vecteurs tirés aléatoirement dans le même espace que les vecteurs de la base.

Nous avons considéré uniquement des données réelles car nous considérons inutile l'étude des performances sur des données générées aléatoirement selon des distributions particulières, notamment uniformes. Ce type de données n'est géré par aucune application réelle. De plus, plusieurs travaux théoriques ont montré que toutes les méthodes de recherche sont inefficaces face à de tels ensembles de données en grande dimension [Beyer *et al.*, 1999].

Notons enfin que les expériences reportées dans ce chapitre ne mesurent en aucune manière la capacité de reconnaissance des descripteurs. Elles étudient seulement les performances de notre méthode de recherche en termes de temps de réponse et la précision des plus proches voisins retrouvés.

6.4 Expérience 1 : étude du paramètre $P(H)$

Dans cette section, nous présentons une étude relative à l'hypothèse d'isotropie des *clusters*. Le but de cette étude est d'estimer le paramètre $P(H)$. Nous rappelons que $P(H)$ est un paramètre qui intervient dans le calcul des rayons approximatifs. Pour un *cluster* C_i, $P(H_i)$ est la probabilité pour que l'inégalité :

$$N_{\widehat{HC}_i} \leqslant N_{HR_i} \frac{\text{vol}(\widehat{HC}_i)}{\text{vol}(HR_i)} \tag{6.1}$$

soit vérifiée. $N_{\widehat{HC}_i}$ est le nombre de vecteurs qui appartiennent effectivement à la calotte $\widehat{HC}_i$ et, $N_{HR_i} \frac{\text{vol}(\widehat{HC}_i)}{\text{vol}(HR_i)}$ est le nombre de vecteurs qui appartiendraient à $\widehat{HC}_i$ si les vecteurs du *cluster* C_i étaient distribués de manière isotrope. $P(H)$ est donc la proportion des vecteurs requêtes pour lesquels l'inégalité 6.1 est vérifiée.

Dans cette section, nous présentons les résultats de l'estimation de $P(H)$ pour tous les *clusters* des ensembles de données BD_2, BD_3^{25}, BD_3^{45}, BD_3^{70}, BD_3^{100} et BD_3^{150}. La procédure d'estimation est décrite dans la section 5.3.4, page 124. La description des partitions générées pour ces ensembles est donnée à la fin de la section suivante, page 141.[2]

Pour chaque *cluster*, nous avons estimé $P(H)$ en considérant quatre rayons approximatifs. Chacun a été calculé pour un niveau d'imprécision donné en considérant que $P(H) = 1$ pour tous les *clusters*. Les quatre niveaux d'imprécision considérés sont 0,01, 0,10, 0,20 et 0,40. Nous avons également généré 9604 vecteurs requêtes pour chaque *cluster* selon la procédure décrite dans la section 5.3.4, page 126. Nous rappelons que ce nombre garantit que la valeur estimée de $P(H)$ se situe dans un intervalle de $\pm 0,01$ autour de la vraie valeur avec un niveau de confiance de l'ordre de 95 %. Nous avons fixé la distance entre les vecteurs requêtes et le centre du *cluster* à 1000 fois le rayon de celui-ci. Nous rappelons que nous plaçons les vecteurs requêtes à une distance très grande du *cluster* pour augmenter le nombre de vecteurs du *cluster* qui risquent d'être ignorés. Ceci permet de considérer le cas le plus défavorable.

Ens. de données	Dim.	Nbre de *clusters*	Estimation de $P(H)$							
			$\alpha = 0,01$		$\alpha = 0,10$		$\alpha = 0,20$		$\alpha = 0,40$	
			Moy.	Écart-t.	Moy.	Écart-t.	Moy.	Écart-t.	Moy.	Écart-t.
BD_2	24	1868	0,995	0,005	0.991	0,014	0,990	0,018	0,959	0,033
BD_3^{25}	25	300	0,993	0,010	0,999	0,015	0,989	0,019	0,954	0,039
BD_3^{45}	45	298	0,996	0,007	0,992	0,014	0,991	0,016	0,964	0,031
BD_3^{70}	70	299	0,997	0,005	0,992	0,014	0,992	0,014	0,970	0,027
BD_3^{100}	100	295	0,997	0,004	0,993	0,012	0,992	0,014	0,976	0,024
BD_3^{150}	150	298	0,998	0,003	0,993	0,012	0,992	0,017	0,982	0,023

TAB. 6.1: Exp. 1 : estimation de $P(H)$. Moyennes et écart-types des valeurs estimées pour différents ensembles de données et différents rayons approximatifs.

Les résultats de l'estimation de $P(H)$ pour les *clusters* des ensembles de données utilisés dans cette expérience sont présentés dans le tableau 6.1. Ces résultats montrent que

[2]Nous ne pouvions pas intervertir l'ordre de présentation de ces deux expériences, car l'expérience suivante dépend de $P(H)$.

$P(H)$ est très proche de 1 pour l'ensemble des *clusters* considérés. En d'autres termes, l'inégalité 6.1 est vérifiée pour la quasi totalité des *clusters* et des vecteurs requêtes, c'est-à-dire, le nombre de vecteurs du *cluster* qui appartiennent effectivement à la calotte HR_i est généralement plus petit que celui estimé sous l'hypothèse d'isotropie. À titre d'exemple, 94 % des *clusters* de l'ensemble BD_2 possèdent un $P(H)$ estimé supérieur à 0,99 pour un rayon approximatif correspondant à $\alpha = 0,01$.

Sur la base de ces résultats, nous fixons $P(H)$ à 1 dans toutes les expériences qui suivent. Les résultats présentés dans la section 6.6, montrent que cette estimation est tout à fait acceptable. L'imprécision effective des résultats est toujours inférieure à l'imprécision fixée au travers du paramètre α, sauf pour deux cas particuliers présentés dans la section 6.11.

Toutefois, notons que cette valeur est une estimation basée sur un tirage aléatoire de vecteurs requêtes. Elle peut donc être sur-estimée ou bien sous-estimée pour d'éventuels cas très particuliers. Dans le cas où la valeur de $P(H)$ est sous-estimée, l'imprécision effective des résultats est inférieure à l'imprécision fixée. Cela ne pose donc aucun problème. Dans le cas contraire, c'est-à-dire si la valeur de $P(H)$ est sur-estimée, l'imprécision effective des résultats *risque* d'être supérieure à l'imprécision fixée car les rayons des *clusters* sont réduits plus qu'il n'en faut.

6.5 Expérience 2 : impact du nombre de *clusters*

Le but de cette deuxième expérience est d'étudier l'impact du nombre de *clusters* sur les performances de la technique de recherche. De manière générale, il n'existe pas de critères universels permettant de choisir le nombre de *clusters*. Ce choix dépend de l'application. Ici, dans le cas de la recherche de plus proches voisins, l'objectif est de trouver le nombre de *clusters* qui minimise le temps de recherche. Malheureusement, la relation entre le nombre de *clusters* et le temps de recherche ne peut être exprimée analytiquement car elle dépend fortement de la distribution des données manipulées et aussi de la distribution des requêtes.

Le but de cette expérience est donc de rechercher le nombre optimal de *clusters* empiriquement. Ce nombre se situe entre deux nombres particuliers correspondant à deux cas triviaux. Le premier est celui où tous les vecteurs de la base sont mis dans un seul et unique *cluster*. Dans ce cas, la recherche de plus proches voisins est équivalente à une recherche séquentielle car quel que soit le vecteur requête, tous les vecteurs de la base seront considérés. Le second cas est celui où chaque vecteur forme un *cluster*, c'est-à-dire que l'on a autant de *clusters* que de vecteurs. Là, le taux de filtrage des *clusters* est maximum car les rayons des *clusters* sont tous nuls et les distances dmin et dmax correspondent aux distances exactes entre le vecteur requête et les vecteurs de la base. Cependant, la phase de filtrage devient extrêmement coûteuse du fait du nombre très élevé des *clusters*.

Pour effectuer cette étude, nous avons utilisé l'ensemble BD_2 contenant 411 409 vecteurs de dimension 24 et nous avons réalisé plusieurs partitions qui diffèrent par le nombre de *clusters*. Précisons toutefois que l'algorithme de *clustering* utilisé ne permet pas de générer exactement un nombre de *clusters* fixé *a priori*. Il permet uniquement de spécifier un nombre maximum de *clusters* à générer.

Les différentes exécutions ont généré respectivement 98, 245, 482, 684, 742, 800, 868, 922, 1064, 1155, 1298, 1435, 1868, 2747, 3998, 5831 et 8506 *clusters*. Le taux de bruit a été fixé à 15%, c'est-à-dire qu'un *cluster* contenant moins de 15 % de vecteurs que la moyenne

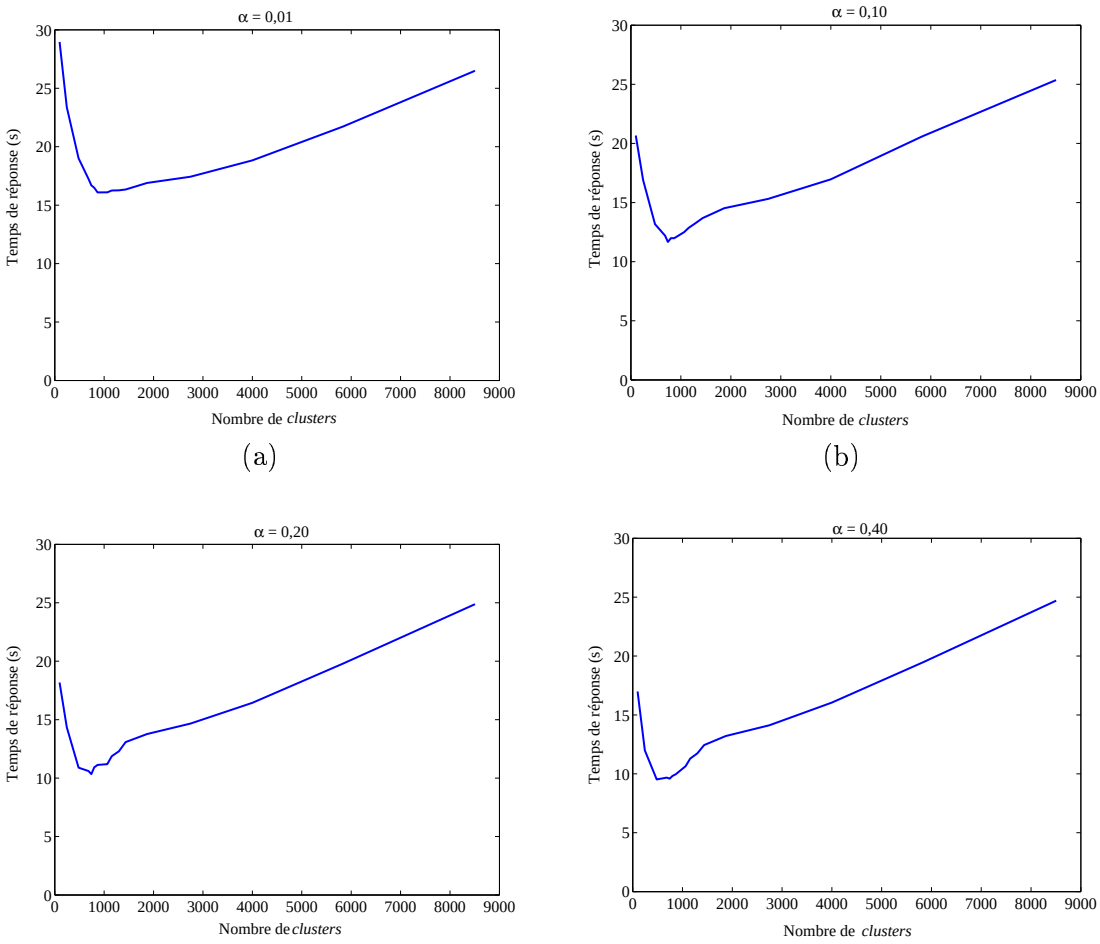


FIG. 6.1: Exp. 2 : ensemble de données BD_2, dimension 24, 411 409 vecteurs, 200 vecteurs requêtes, 20 plus proches voisins, temps de réponse cumulé en fonction du nombre de *clusters* pour différentes valeurs de α.

des populations de tous les *clusters* est considéré comme bruité et les vecteurs qu'il contient sont isolés. Nous avons ensuite tiré aléatoirement 200 vecteurs requêtes parmi les vecteurs de la base. Pour chacun, nous avons recherché les 20 plus proches voisins en considérant plusieurs niveaux d'imprécision (α) 0,01, 0,10, 0,20 et 0,40. Les temps de réponse cumulés (pour les 200 vecteurs requêtes) en fonction du nombre de *clusters* sont reportés dans les figures 6.1.a, 6.1.b, 6.1.c et 6.1.d respectivement pour chaque niveau d'imprécision.

Ces courbes montrent bien que, entre les deux cas extrêmes cités précédemment, il existe bien un optimum. Cet optimum dépend légèrement de α. Notons cependant qu'il n'existe pas de valeur optimale unique, mais que le nombre de *clusters* peut être fixé dans un intervalle assez large, ce qui donne beaucoup de souplesse lors du choix de cette valeur. Le tableau 6.2 donne l'intervalle optimal du nombre de *clusters* et la variation du temps de réponse dans cet intervalle pour différentes valeurs de α. Ce tableau montre par exemple que, lorsque $\alpha = 0,01$, un temps de réponse minimal est obtenu pour un nombre de *clusters* variant entre 684 et 2747. Dans cet intervalle, le temps de réponse est relativement constant.

α	Intervalle optimal du nombre de *clusters*	Temps de réponse (secondes)
0,01	[684, 2747]	16,76 ± 0,67
0,10	[482, 1298]	12,47 ± 0,80
0,20	[482, 1298]	11,31 ± 0,97
0,40	[482, 1155]	10,40 ± 0,88

TAB. 6.2: Exp. 2 : ensemble de données BD_2, dimension 24, 411 409 vecteurs, 200 vecteurs requêtes, 20 plus proches voisins. Nombres optimaux de *clusters* et les temps de réponse associés pour différentes valeurs de α.

Analyse de la complexité. Pour mieux comprendre la relation entre le nombre de *clusters* et le temps de réponse, nous avons analysé l'impact du nombre de *clusters* sur les temps passés dans les trois phases de l'algorithme de recherche :

1. 1^{re} phase : l'application de la première règle de filtrage ;
2. 2^{e} phase : l'application de la deuxième règle de filtrage ;
3. 3^{e} phase : la recherche séquentielle dans l'ensemble des vecteurs isolés.

Lors de la première phase, la première règle de filtrage est appliquée pour filtrer les *clusters* n'ayant aucune chance de contenir des plus proches voisins du vecteur requête. Il s'agit alors de calculer les distances dmin et dmax entre le vecteur requête et l'ensemble des *clusters* et d'effectuer des opérations de comparaison. Le coût de cette première phase de filtrage croît linéairement avec le nombre de *clusters*. Il peut donc être parfaitement calculé. Plus le nombre de *clusters* est grand, plus le taux de filtrage des *clusters* non pertinents est élevé. Dans ce cas, les *clusters* contiennent un petit nombre de vecteurs. Ils sont donc compacts et le taux de chevauchement est réduit.

Lors de la deuxième phase, les *clusters* retenus après l'application de la première règle de filtrage sont ordonnés en fonction de leurs distances dmin au vecteur requête puis lus dans l'ordre. Après la lecture de chaque *cluster*, l'ensemble courant des plus proches voisins est mis à jour. La recherche s'arrête dès que la distance minimale du prochain *cluster* à lire est plus grande que la distance entre le vecteur requête et son k^e plus proche voisin courant. Le coût de cette phase dépend essentiellement du taux de filtrage de la première phase. Par conséquent, plus le nombre de *clusters* est grand, moins élevé est le coût de cette phase. Malheureusement, le taux de filtrage dépend principalement de la distribution des données et de la « qualité » des *clusters* . Par conséquent, le coût de cette phase n'est pas calculable.

La troisième et dernière phase est une recherche séquentielle dans l'ensemble des vecteurs isolés. Elle est donc de complexité linéaire. Cependant, le nombre de vecteurs isolés dépend de la distribution des données. Par conséquent, tout comme la deuxième phase, le coût de cette phase n'est pas calculable.

L'analyse des trois phases de l'algorithme de recherche montre clairement qu'une relation entre le nombre de *clusters* et le temps de recherche ne peut être établie. Pour cela, nous nous contentons d'une étude expérimentale des variations du coût de chaque phase en fonction du nombre de *clusters* sur des ensembles de données réelles. Nous avons réexécuté les 200 requêtes de 20 plus proches voisins dans l'ensemble BD_2 mais cette fois-ci

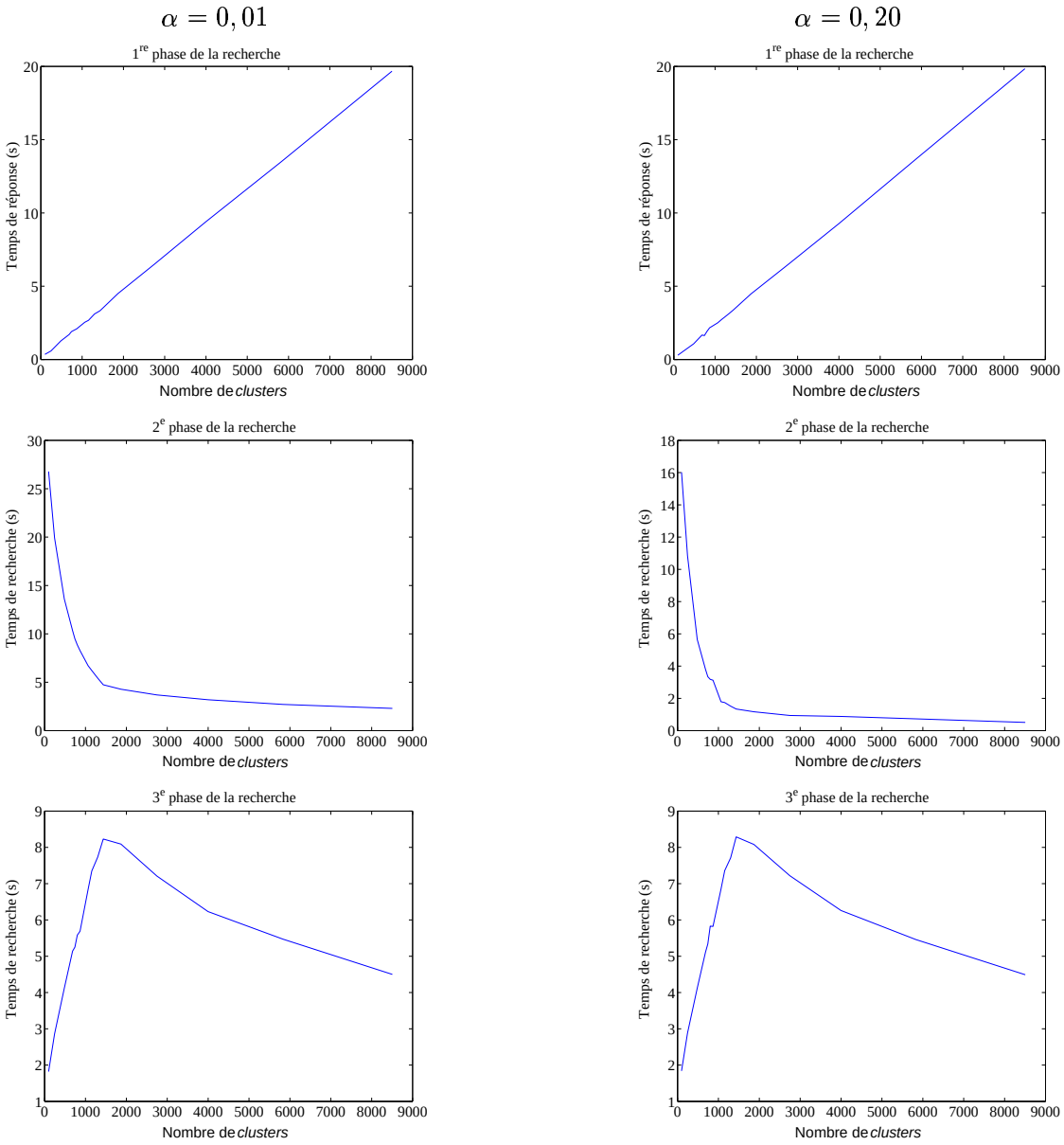


FIG. 6.2: Exp. 2 : ensemble de données BD_2, dimension 24, 411 409 vecteurs, 200 vecteurs requêtes, 20 plus proches voisins, temps de réponse pour chaque phase de l'algorithme de recherche en fonction du nombre de *clusters* pour deux valeurs de α, 0,01 et 0,20.

en comptabilisant le temps passé dans chacune des trois phases. Les graphes de la figure 6.2 montrent les variations du temps de recherche cumulé pour les 200 vecteurs requêtes en fonction du nombre de *clusters* pour les trois phases de l'algorithme de recherche. Dans cette expérimentation, quatre valeurs différentes de α ont été utilisées (0,01, 0,10, 0,20 et 0,40). Dans la figure 6.2, nous présentons uniquement les résultats pour α égale à 0,01 et 0,20.

Ces graphes confirment bien l'analyse du coût des trois phases de l'algorithme de recherche en fonction du nombre de vecteurs. L'évolution du temps de recherche de la première phase est linéaire ; le temps de recherche de la deuxième phase décroît avec le nombre de *clusters* ; et le coût de la troisième phase dépend du nombre de vecteurs isolés. Notons d'ailleurs que le coût de cette dernière phase est le même quelle que soit la valeur de α. La forme de la courbe de variation du temps de réponse en fonction du nombre de *clusters* est identique à celle du nombre de vecteurs isolés en fonction du nombre de *clusters*. Cette dernière courbe est donnée dans la figure 6.3.

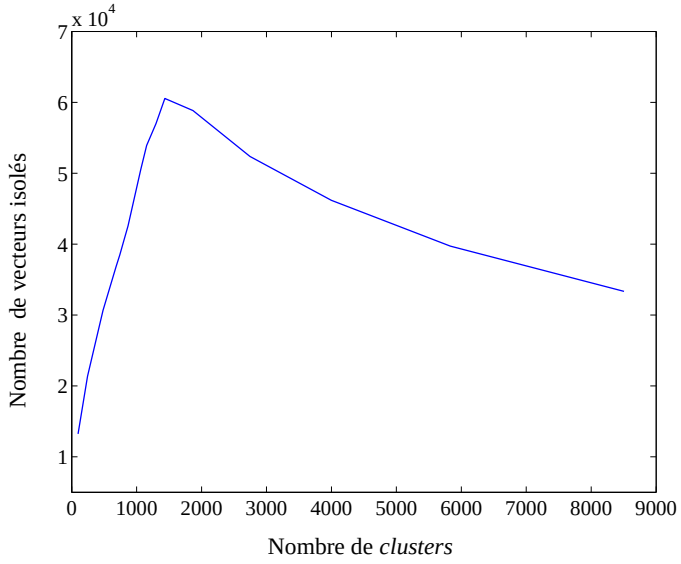


FIG. 6.3: Exp. 2 : ensemble de données BD_2. Le nombre de vecteurs isolés par l'algorithme de *clustering* en fonction du nombre de *clusters* formés.

En faisant cette même expérience sur d'autres ensembles de données de différentes tailles, nous avons remarqué que l'intervalle des nombres de *clusters* qui minimisent le temps de réponse s'étale souvent entre 1 et 3 fois la racine carrée du nombre de vecteurs de la base. Nous utiliserons donc cette heuristique pour le choix du nombre de *clusters*.

Autres résultats. Pour étudier la validité de ce résultat lorsque la dimension augmente et pour d'autres distributions de données, nous avons effectué ces mêmes expérimentations sur les ensembles BD_3^{25}, BD_3^{45}, BD_3^{70}, BD_3^{100}, BD_3^{150}.

Les conclusions de cette étude sont identiques à celles faites sur l'ensemble BD_2. Nous ne reprenons pas la totalité des résultats obtenus. Nous présentons uniquement quelques graphes de variations du temps de réponse en fonction du nombre de *clusters* (figures 6.4).

Sur la base de cette étude, nous avons généré pour chacun de nos ensembles de données (BD_1, BD_2 et BD_3^*), une partition dont le nombre de *clusters* est choisi selon l'heuristique déduite précédemment. Les caractéristiques de ces partitions sont résumées dans le tableau 6.3. Ces partitions sont utilisées dans toutes les expériences présentées dans ce chapitre. Ce tableau montre que le taux de vecteurs isolés *(outliers)* dépend bien de la distribution. Il vaut 14,30% et 10,88% pour les ensembles de données BD_1 et BD_2 respec-

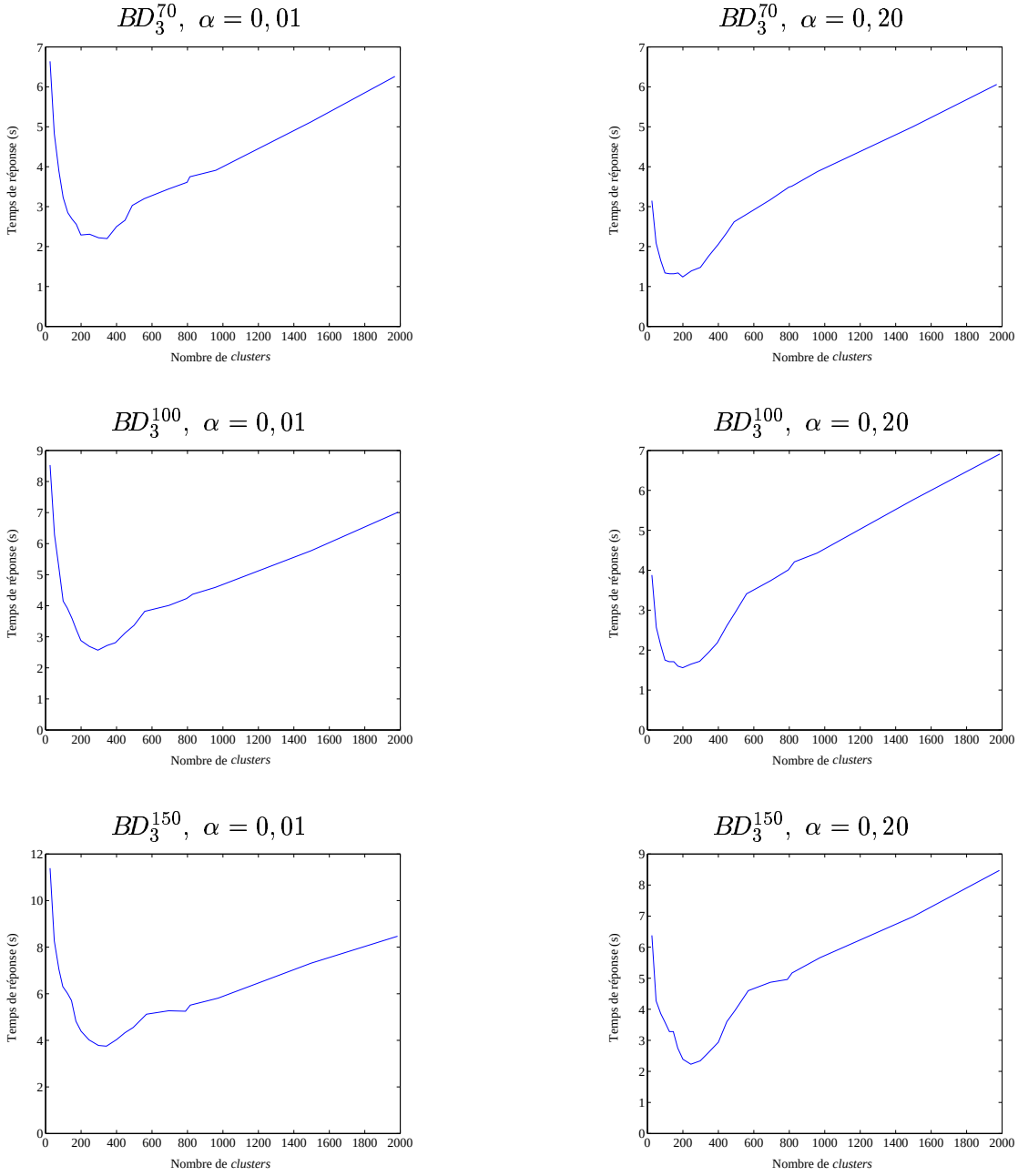


FIG. 6.4: Exp. 2 : Temps de recherche cumulé des 20 plus proches voisins de 200 vecteurs requêtes en fonction du nombre de *clusters* pour deux valeurs de α, 0,01 et 0,20. Ensembles de données : $BD_3^{70}, BD_3^{100}, BD_3^{150}$.

tivement, alors qu'il est inférieur à 2,45% pour les ensembles BD_3^*. Ceci montre bien que ce taux dépend de la nature et de la distribution des données. Nous rappelons que BD_1 et BD_2 contiennent des vecteurs de dimension 24 qui sont des invariants locaux différentiels d'images couleurs tandis que les ensembles BD_3^* contiennent des vecteurs qui sont des histo-

grammes de couleurs. Ce tableau montre également que les temps de calcul de l'algorithme de *clustering* sont relativement élevés bien que cet algorithme ne soit pas de complexité quadratique et ne nécessite qu'une seule passe sur les données. Ce point est rediscuté dans la synthèse de ce chapitre. Il est également repris dans les perspectives générales de ce travail (page 195).

Ens. de don.	Dim.	# de vect.	# de *clust.*	# *de vect. isolés*	Tps de calcul
BD_1	24	5 017 298	4720	545 766 (soit 10,88%)	$>$ 3 jours
BD_2	24	411 409	1868	58 836 (soit 14,30%)	$\approx$ 8 heures
BD_3^{25}	25	30 021	300	736 (soit 2,45%)	28 minutes
BD_3^{45}	45	30 021	298	621 (soit 2,07%)	41 minutes
BD_3^{70}	70	30 021	299	495 (soit 1,65%)	30 minutes
BD_3^{100}	100	30 021	295	300 (soit 1,00%)	37 minutes
BD_3^{150}	150	30 021	298	413 (soit 1,38%)	40 minutes

TAB. 6.3: Caractéristiques des partitions retenues pour les ensembles de données BD_1, BD_2 et BD_3 – Temps de calcul des algorithmes de *clustering*.

6.6 Expérience 3 : précision de la recherche

Le but de cette expérience est d'étudier la fiabilité de la méthode utilisée pour le contrôle de la précision de la recherche. Il s'agit d'évaluer et de comparer l'imprécision des résultats obtenus par rapport à ce que l'utilisateur a spécifié au travers du paramètre α. Pour cette expérience, nous avons utilisé l'ensemble de données BD_2 (411 409 vecteurs de dimension 24). Pour interroger la base, nous avons utilisé 200 vecteurs requêtes.

Tout d'abord, nous avons recherché les 20 plus proches voisins exacts de chaque vecteur requête. Nous avons ensuite effectué des recherches approximatives des 20 plus proches voisins en faisant varier α entre 0,01 et 0,5. Ces résultats approximatifs ont été ensuite comparés aux résultats exacts afin d'estimer empiriquement l'imprécision de la recherche. Pour chaque vecteur requête, la proportion des plus proches voisins exacts non retrouvés dans le résultat approximatif est calculée. La moyenne de ces proportions, calculée sur la totalité des vecteurs requêtes, est une estimation empirique de la probabilité de ne pas retrouver un des plus proches voisins exacts dans le résultat approximatif. Cette estimation doit être plus petite mais aussi la plus proche possible de la valeur de α choisie par l'utilisateur.

La figure 6.5 présente les résultats obtenus pour différentes valeurs de α. Cette figure montre que l'estimation de l'imprécision des résultats approximatifs est très proche mais toujours inférieure à α (la courbe ne passe pas au dessus de la diagonale). Cela montre bien la fiabilité de la procédure de contrôle de la précision.

Notons cependant que cette estimation est une moyenne sur 200 vecteurs requêtes. Il est donc possible d'avoir une imprécision supérieure à α pour certains vecteurs requêtes. Par exemple, lorsque α est fixé à 0,01, la moyenne de l'imprécision des résultats sur les 200 vecteurs requêtes est de 0,01 et l'écart-type est de 0,046. Globalement, l'imprécision effective est inférieure à l'imprécision autorisée. Elle est cependant supérieure à 0,01 pour 18 vecteurs sur l'ensemble des 200 vecteurs requêtes. Pour ces 18 vecteurs, elle vaut en moyenne 0,125.

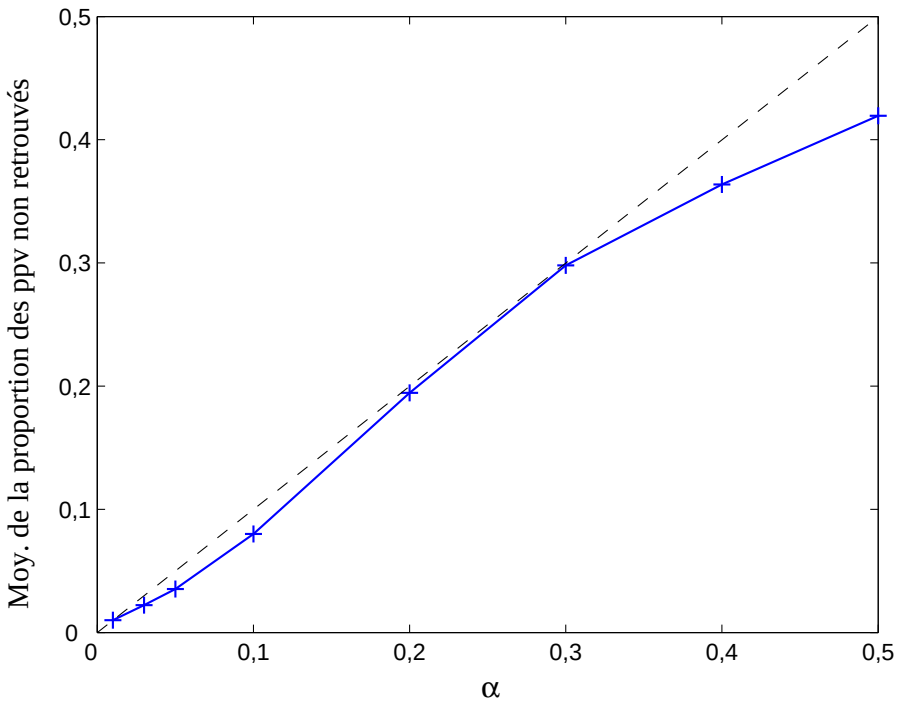


FIG. 6.5: Exp. 3 : BD_2, 411 409 vecteurs de dimension 24, 200 vecteurs requêtes, 20 ppv, $P(H) = 1$. Estimation de la probabilité de ne pas retrouver un des plus proches voisins exacts dans le résultat approximatif pour différentes valeurs de α.

Cependant, cela ne signifie en aucune manière que le processus de contrôle de la précision n'est pas fiable. Ce contrôle est probabiliste. Sa fiabilité doit donc être évaluée sur un ensemble de vecteurs requêtes et non pas en considérant certains cas particuliers.

6.7 Expérience 4 : influence de la dimension des données

Cette expérience étudie l'influence de la dimension des descripteurs sur les performances de la recherche. Nous avons utilisé pour cela les ensembles BD_3^*. Nous rappelons qu'il s'agit de 5 bases de 30 021 vecteurs. La dimension de ces ensembles varie entre 25 et 150. Pour chaque ensemble, nous avons tiré aléatoirement 200 vecteurs requêtes de la base. Nous avons ensuite recherché les 20 plus proches voisins de chaque vecteur requête en faisant varier le niveau d'imprécision α. Cinq valeurs différentes de α ont été considérées (0, 0,01, 0,10, 0,20, 0,40). Nous avons également effectué les mêmes recherches par un algorithme de recherche séquentielle, celle-ci étant la seule technique de recherche qui garde un coût linéaire lorsque la dimension augmente (*cf.* section 2.2, page 33 et section 2.6, page 61). Nous avons utilisé deux versions de la recherche séquentielle : la recherche séquentielle de base et la recherche séquentielle munie de l'option « distance partielle ». La figure 6.6 résume les résultats obtenus. Cette figure montre que l'évolution du temps de réponse de notre technique est linéaire mais avec une pente moins importante que celle de la recherche

séquentielle avec ou sans distance partielle. Elle montre également que notre méthode est plus rapide que la recherche séquentielle même lorsque les résultats obtenus sont exacts ($\alpha = 0$).

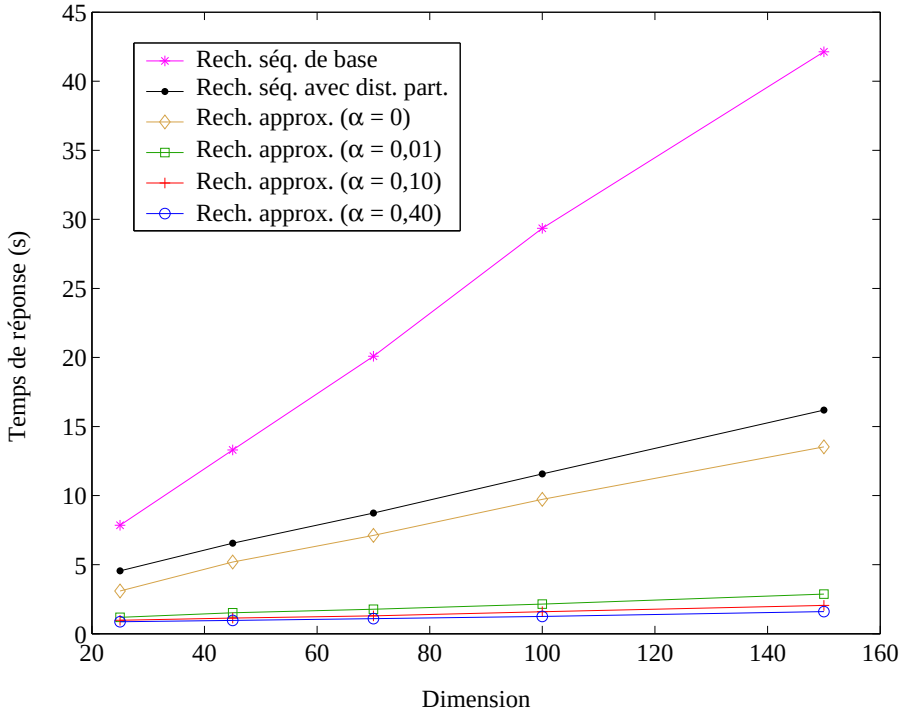


FIG. 6.6: Exp. 4 : BD_3^*, 30 021 vecteurs, 200 vecteurs req., 20 ppv, variation du temps de réponse en fonction de la dimension des vecteurs.

À titre d'exemple, lorsque la dimension est égale à 150, le temps de réponse de la recherche séquentielle de base est de 42,13 secondes alors que notre technique *retourne exactement les mêmes résultats* en 13,53 secondes. Elle retourne un résultat approximatif en 2,87 secondes lorsque $\alpha = 0,01$, soit dans un temps 15 fois plus rapide que la recherche séquentielle de base. Dans ce cas, en moyenne, seulement 8 *clusters* ont été lus pour chaque vecteur requête, ce qui correspond à un taux de filtrage des *clusters* de 97,23 %. En termes de vecteurs, 2287 vecteurs ont été lus en moyenne pour chaque vecteur requête, ce qui correspond à un taux de filtrage moyen des vecteurs de 92,28 %. Ces informations sont résumées pour l'ensemble des dimensions dans le tableau 6.4. Nous remarquons dans ce tableau que le taux de filtrage des vecteurs est légèrement inférieur au taux de filtrage des *clusters*. Ceci est dû au fait que les *clusters* ne contiennent pas tous le même nombre de vecteurs.

Il est cependant important de remarquer que les améliorations du temps de recherche ne sont pas très significatives lorsqu'on augmente α. En fait, les différences relatives entre rayons approximatifs consécutifs deviennent de plus en plus petites. Dès les premières valeurs de α, il y a une forte réduction du rayon étant donné qu'il y a peu de vecteurs vers les bords extérieurs des *clusters*. Augmenter α ne permet pas d'obtenir d'autres rayons

Dimension	# total de *clusters*	*Clusters* filtrés		Taux de filtrage moy. des Vecteurs
		Nbre moy.	Taux moy.	
25	300	289,40	96,47 %	94,14 %
45	298	286,88	96,27 %	92,57 %
70	299	289,17	96,71 %	93,18 %
100	295	285,93	96,93 %	93,18 %
150	298	289,76	97,23 %	92,28 %

TAB. 6.4: Exp. 4 : BD_3, expérience d'étude de l'influence de la dimension sur l'algorithme de recherche. Statiatiques sur le nombre moyen de *clusters* et de vecteurs lus lors de la recherche de 20 plus proches voisins de 200 vecteurs requête à l'aide de la recherche approximative lorsque $\alpha = 0,01$.

significativement plus petits. La distribution radiale non uniforme des vecteurs à l'intérieur des *clusters* fait que, plus on s'approche de leurs centres, plus la distribution des vecteurs est dense. Dans le tableau 6.5, nous donnons les rayons approximatifs moyens des *clusters* pour différentes valeurs de α. Ce tableau montre par exemple que la moyenne des rayons approximatifs calculés pour les *clusters* de l'ensemble BD_3^{150} lorsque $\alpha = 0,01$ vaut 7268, ce qui est équivalent à un taux de réduction moyen des rayons (par rapport aux rayons exacts) de 84,48 %. Ce tableau montre aussi que des réductions supplémentaires sont obtenues pour des valeurs plus grandes de α. Cependant, ces réductions ne sont pas très significatives comparativement à la réduction obtenue pour $\alpha = 0,01$. La différence entre les rayons moyens correspondant à $\alpha = 0,01$ et ceux correspondant à $\alpha = 0,20$ est de 4600 ce qui équivaut seulement à 11 % du rayon exact.

Dim.	Moy. des ray. exacts	Rayons approximatifs									
		$\alpha = 0,01$		$\alpha = 0,10$		$\alpha = 0,20$		$\alpha = 0,30$		$\alpha = 0,40$	
		Moy.	T.R.	Moy.	T.R.	Moy.	T.R.	Moy.	T.R.	Moy.	T.R.
25	35 858	14 192	60,42	8491	76,32	5778	83,89	3670	89,76	1791	95,00
45	39 123	12 406	68,29	7198	81,60	4820	87,68	3032	92,25	1469	96,25
70	39 267	10 346	73,65	5889	85,00	3906	90,05	2445	93,77	1184	96,98
100	40 257	8990	77,67	5075	87,39	2104	94,77	3367	91,63	1017	97,47
150	39 088	7268	84,48	4043	89,66	2668	93,17	1668,5	95,73	805	97,94

*T.R. : Taux de réduction moyen par rapport au rayon exact (%).

TAB. 6.5: Exp. 4 : BD_3, statistiques sur la réduction des rayons des *clusters* par rapport à leurs rayons exacts en fonction de α.

Distribution radiale des vecteurs au sein des *clusters*. Pour mieux comprendre ce phénomène, nous avons étudié la distribution des vecteurs par rapport au centre de leurs *clusters*. Pour cela, nous avons sélectionné quelques *clusters* et nous avons calculé, pour chacun, les distances entre ses vecteurs et leur centre de gravité (le centre du *cluster*). Nous avons ensuite calculé la densité de ces distances. L'étude de cette densité permet d'avoir une idée très précise de la dispersion des vecteurs et de leur distribution radiale.

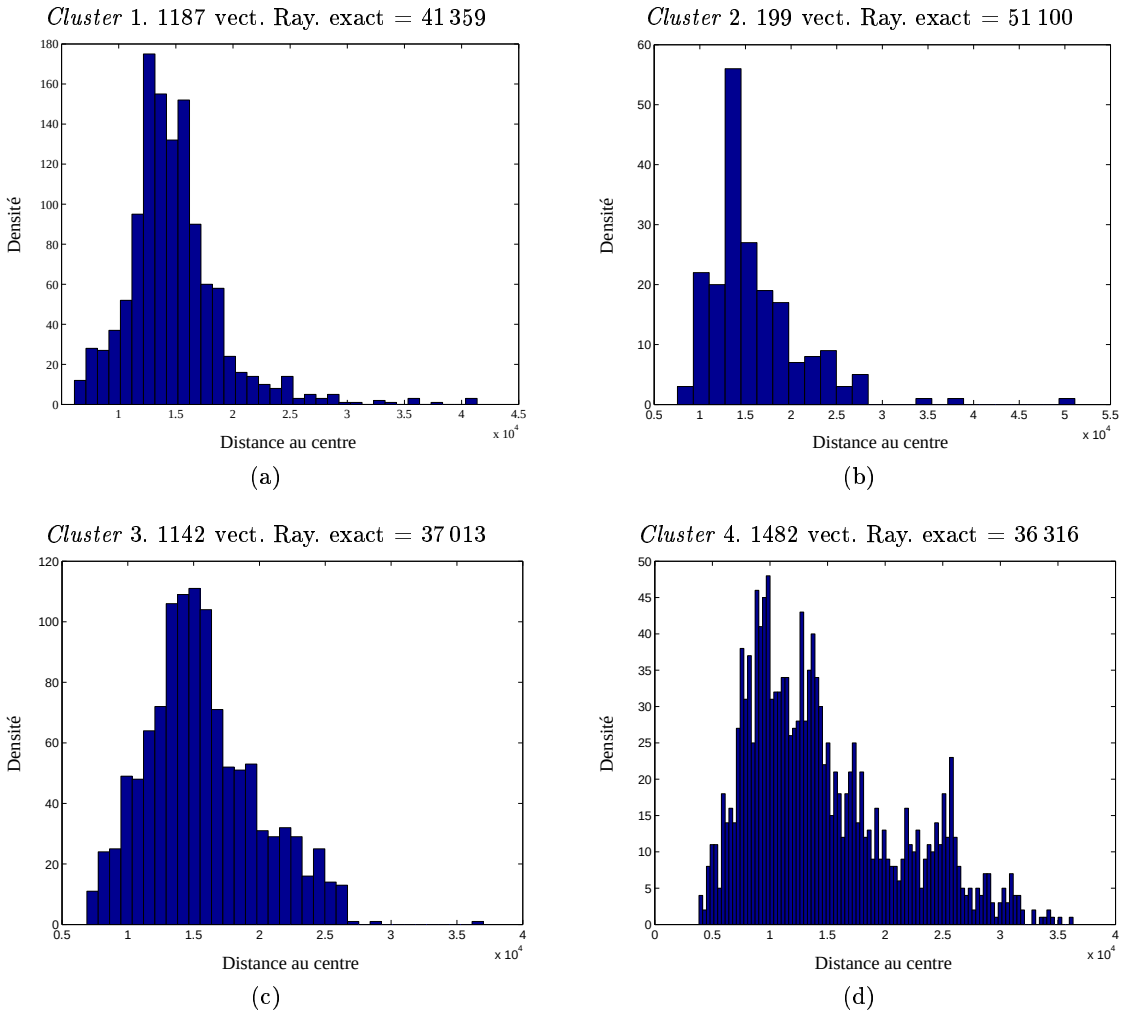


FIG. 6.7: Exp. 4 : la distribution des distances entre les vecteurs d'un *cluster* et son centre. Étude menée sur 4 *clusters* de la partition associée à l'ensemble BD_3^{150} (30 021 vecteurs de dimension 150).

Dans la figure 6.7, nous présentons les densités obtenues pour quatre *clusters* choisis au hasard. Globalement, ces graphes montrent clairement que la densité des vecteurs est très importante localement autour du centre. Elle diminue cependant fortement lorsqu'on s'en éloigne. À titre d'exemple, l'histogramme 6.7.a montre la densité d'un *cluster* contenant 1187 vecteurs et de rayon exact égal à 41 359. Dans ce *cluster*, plus de 95 % des vecteurs se trouvent à une distance du centre inférieure à la moitié du rayon exact. Dans ce cas, le rayon approximatif calculé pour $\alpha = 0,01$ vaut 7747,67 (soit une réduction de plus de 81 %). Les autres rayons calculés pour des α supérieurs à 0,01 sont plus petits mais les taux de réduction du rayon sont relativement plus faibles.

À travers cette étude, nous avons montré qu'il n'existe pas de relation linéaire entre l'augmentation de α et la réduction des rayons approximatifs. Par conséquent, il n'existe pas de relation linéaire entre l'augmentation de α et le gain en temps de réponse.

6.8 Expérience 5 : influence du nombre de vecteurs

Cette quatrième expérience a pour but d'étudier l'évolution du temps de réponse en fonction du nombre de vecteurs dans la base. Pour cela, nous avons utilisé notre plus grand ensemble de données BD_1. Cet ensemble contient 5 017 298 vecteurs de dimension 24. Nous avons ensuite généré plusieurs bases de différentes tailles en ne retenant que certains vecteurs (100 000, 411 409, 1 000 000, 1 490 733 et 3 208 771). Chaque base a été partitionnée par l'algorithme de *clustering* séparément. Les propriétés des partitions obtenues sont résumées dans le tableau 6.6. Le taux de bruit (β) a été fixé à 0,15. À partir de chaque base, 200 vecteurs requêtes ont été tirés aléatoirement. Nous avons ensuite recherché les 20 plus proches voisins de chaque vecteur requête à l'aide de notre technique de recherche avec 4 valeurs différentes de α (0,01, 0,10, 0,20 et 0,40). Comme dans l'expérience précédente, nous avons effectué les mêmes recherches à l'aide d'un algorithme de recherche séquentielle de base et un algorithme de recherche séquentielle avec distance partielle. Les temps de réponse obtenus sont résumés dans la figure 6.8.

Nombre de vect.	Nombre de *clusters*	Nombre d'*outliers*
100 000	451	15 381
411 409	1868	58 836
1 000 000	2763	147 039
1 490 733	3294	213 006
3 208 771	3468	371 486
5 017 298	4720	545 766

TAB. 6.6: Exp. 5 : description des partitions des ensembles de données utilisés dans l'expérience 5. (Parties des BD_1).

Cette figure montre que l'évolution du temps de réponse de notre approche est linéaire et sa pente est moins importante que celle de la recherche séquentielle, même lorsque les résultats obtenus sont exacts ($\alpha = 0$). À titre d'exemple, pour la base la plus grande (5 017 298 vecteurs), le temps de réponse obtenu par notre technique lorsque α est égal à 0 représente 42,62 % du temps de réponse de la recherche séquentielle avec l'option distance partielle et seulement 25,01 % du temps de réponse de la recherche séquentielle de base (soit un facteur d'accélération de 4).

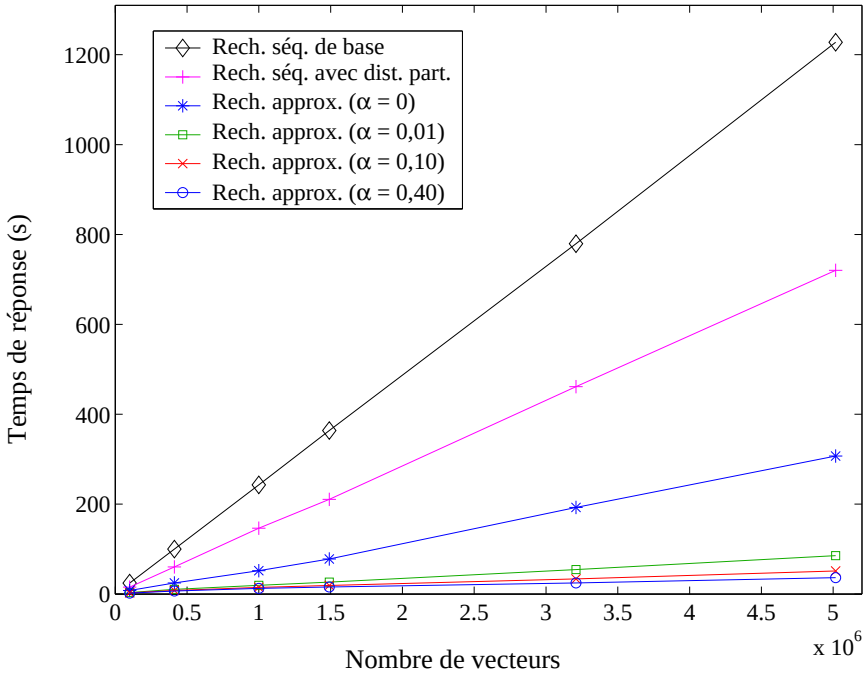


FIG. 6.8: Exp. 5 : BD_1, dimension 24, 200 vecteurs requêtes, 20 ppv. Variation du temps de réponse en fonction du nombre de vecteurs de la base.

Avec un α égal à 0,01, notre technique est 14,39 fois plus rapide que la recherche séquentielle de base. Le temps de recherche est de 85,28 secondes. Ce temps se décompose en deux : 67,18 secondes de recherche au sein des *clusters* et 18,10 secondes de recherche séquentielle dans l'ensemble des vecteurs isolés. En moyenne, 99,08% des *clusters* sont filtrés pour chaque vecteur requête, ce qui correspond à 91,62% des vecteurs appartenant aux *clusters*. Dans le tableau 6.7, sont données des statistiques sur le nombre de *clusters* et de vecteurs filtrés lors de la recherche pour $\alpha = 0,01$.

Par ailleurs, ces résultats montrent à nouveau que seules des améliorations minimes du temps de réponse sont obtenues lorsque α augmente au-delà de 0,01.

Nombre de vect.	# total de *clusters*	*Clusters* filtrés		Taux de filtrage moy. des vecteurs
		Nbre moy.	Taux moy.	
100 000	451	437	96,83 %	88,25 %
411 409	1868	1844	98,71 %	93,50 %
1 000 000	2763	2734	98,96 %	94,44 %
1 490 733	3294	3256	98,87 %	94,29 %
3 208 771	3468	3428	98,85 %	92,34 %
5 017 298	4720	4676	99,08 %	91,62 %

TAB. 6.7: Exp. 5 : BD_1, 200 vecteurs requêtes, 20 ppv, $\alpha = 0,01$. Étude de l'influence du nombre de vecteurs de la base sur l'algorithme de recherche. Statistiques sur le nombre moyen de *clusters* et de vecteurs lus lors de la recherche.

6.9 Expérience 6 : sensibilité à l'algorithme de *clustering*

Théoriquement, notre méthode de recherche de plus proches voisins est indépendante de l'algorithme du *clustering*. N'importe quel algorithme de *clustering* peut être utilisé hors ligne pour partitionner les données. Il est cependant évident que les performances de la méthode de recherche dépendent de la qualité des *clusters* (compacité, séparabilité...).

Pour mieux comprendre l'impact de l'algorithme de *clustering* sur les performances de la méthode de recherche de plus proches voisins, nous présentons ici une expérience dans laquelle nous avons comparé les performances de la méthode de recherche lorsque celle-ci est appliquée sur deux partitions différentes du même ensemble de données. La première est générée par l'algorithme de *clustering* proposé dans le chapitre 4 et utilisé dans toutes les expériences précédentes. La deuxième est extraite à partir d'un SS-Tree [White & Jain, 1996] construit à partir de l'ensemble de données. Nous avons présenté le SS-Tree dans la section 1.4.3, page 23. Il ne s'agit pas d'un « vrai » algorithme de *clustering* mais plutôt d'un index multidimensionnel. Il permet néanmoins de regrouper les données en fonction de leur proximité en petits paquets et de les englober dans des hypersphères.

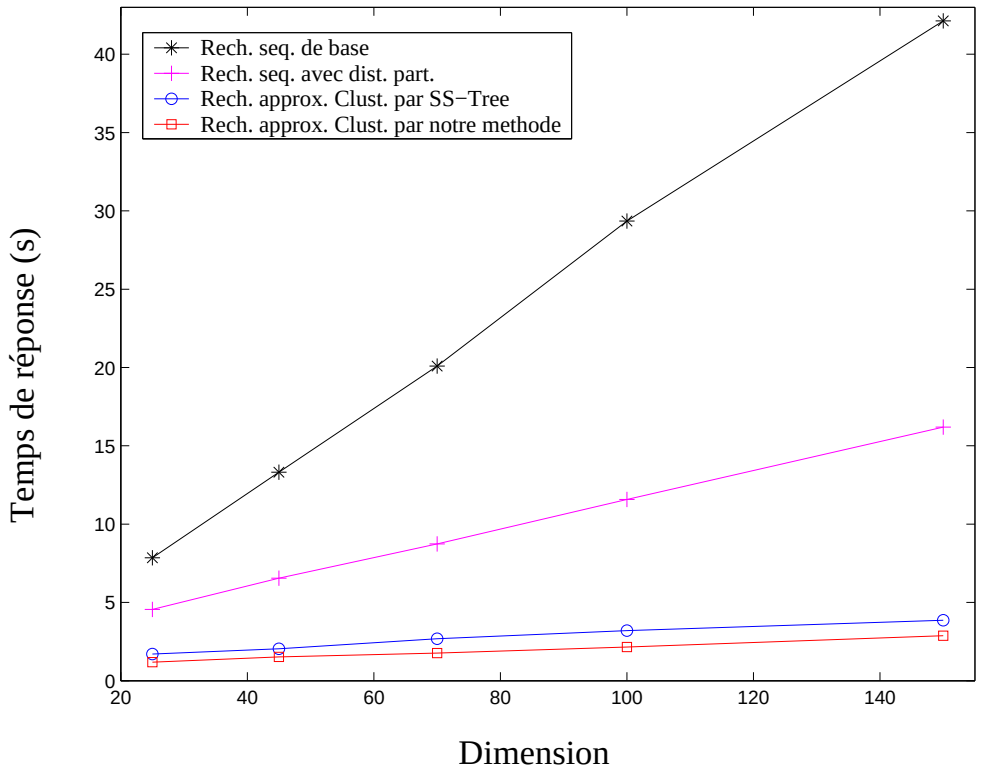


FIG. 6.9: Exp. 6 : BD_3, 30 021 vecteurs, 200 vecteurs requêtes, 20 ppv. Variation du temps de réponse en fonction de la dimension des vecteurs pour la recherche séquentielle de base, la recherche séquentielle avec distance partielle, la recherche approximative ($\alpha = 0,01$) en utilisant le SS-Tree pour le regroupement des données hors ligne et la recherche approximative ($\alpha = 0,01$) en utilisant notre méthode de *clustering* pour le regroupement des données hors ligne.

Dans cette expérience, nous avons utilisé les ensembles de données BD_2 de dimension 24 et les 5 ensembles BD_3 de dimension variant entre 25 et 150. Nous avons construit un SS-Tree à partir de chacun de ces ensembles[3]. Nous avons ensuite extrait de chaque SS-Tree le niveau qui contient un nombre de nœuds qui satisfait la contrainte du choix du nombre de *clusters* (c'est-à-dire entre 1 et 3 fois la racine carrée du nombre de vecteurs de la base). Notons cependant que le SS-Tree ne permet pas de détecter les vecteurs isolés. Les fichiers des vecteurs isolés sont donc vides pour les partitions extraites à partir des SS-Tree.

Nous avons alors recherché les 20 plus proches voisins de 200 vecteurs requêtes dans chacun des ensembles. Cette recherche a été effectuée deux fois, une première fois en utilisant la partition calculée par notre algorithme de *clustering*, et une deuxième fois en utilisant la partition extraite à partir d'un SS-Tree. Plusieurs niveaux d'imprécision différents ont été considérés. Les résultats obtenus sont résumés dans la figure 6.9 et les deux tableaux 6.8 et 6.9.

α	Notre méthode de *clustering* (s)	partitionnement par SS-Tree (s)	Recherche seq. avec dist. partielle (s)	Recherche seq. de base (s)
0	24,57	38,94	58,16	100,25
0,01	10,36	13,79		
0,10	8,01	7,15		
0,20	7,36	5,23		
0,40	6,81	3,22		

TAB. 6.8: Exp. 6 : BD_2, 411 409 vecteurs, 200 vecteurs requêtes, 20 ppv. Impact du *clustering* sur les performances de la recherche.

La figure 6.9 montre l'évolution du temps de réponse en fonction de la dimension pour différentes stratégies de recherche : la recherche séquentielle avec et sans la distance partielle et notre technique pour la recherche de plus proches voisins appliquée sur les deux partitions de l'ensemble de données. Dans cette expérience, α a été fixé à 0,01. Ces courbes montrent que notre méthode est très performante par rapport à la recherche séquentielle même lorsque les *clusters* sont formés aussi grossièrement qu'avec un SS-Tree. En effet, la différence entre le temps de réponse obtenu sur la partition créée à l'aide de notre algorithme de *clustering* et celui obtenu sur la partition extraite à partir d'un SS-Tree est seulement de 0,67 seconde lorsque la dimension est égale à 150. Cette différence représente moins de 1,60 % du temps de réponse de la recherche séquentielle de base dont le temps de réponse est de 42,13 secondes. Le tableau 6.9 donne plus de détails sur les temps pour toutes les dimensions et pour tous les niveaux d'imprécision considérés pour l'ensemble BD_3^*. Les mêmes informations sont données dans le tableau 6.8 pour l'ensemble BD_1.

Globalement, lorsque α est supérieur à 0, les différences entre les temps de réponse obtenus sur la partition extraite à partir de SS-Tree et ceux obtenus sur la partition créée à l'aide de notre algorithme de *clustering* ne sont pas très importantes. Nous remarquons cependant que lorsque $\alpha = 0$, la recherche est nettement plus rapide sur la partition créée

[3]L'implémentation du SS-Tree que nous avons utilisée est celle fournie dans la dernière version du logiciel GiST.

par notre algorithme de *clustering*. L'explication en est très simple. Les *clusters* extraits à partir d'un SS-Tree ne sont pas très adaptés à la recherche de plus proches voisins à cause du chevauchement. Ce problème touche tous les index multidimensionnels et pas seulement le SS-Tree. Il a été notamment discuté dans le chapitre 2. En réduisant les rayons des *clusters* par notre méthode, le taux de chevauchement s'en trouve réduit, ce qui permet d'accélérer la recherche.

Cette expérience a montré la relative indépendance de notre méthode pour la recherche approximative de plus proches voisins par rapport au partitionnement des données réalisé hors ligne. Même avec un partitionnement grossier, les performances de la méthode de recherche restent très bonnes. Cette étude doit toutefois être poussée davantage en considérant d'autres algorithmes de *clustering*.

Dimension	α	Notre méthode de *clustering* (s)	partitionnement par SS-Tree (s)	Recherche seq. avec dist. partielle (s)	Recherche seq. de base (s)
25	0	3,09	4,07	4,55	7,85
	0,01	1,19	1,70		
	0,10	0,97	0,93		
	0,20	0,90	0,69		
	0,40	0,87	0,41		
45	0	5,19	5,79	6,55	13,31
	0,01	1,52	2,04		
	0,10	1,13	1,16		
	0,20	1,06	0,93		
	0,40	0,96	0,73		
70	0	7,12	8,00	8,74	20,09
	0,01	1,77	2,68		
	0,10	1,29	1,95		
	0,20	1,19	1,76		
	0,40	1,09	1,54		
100	0	9,73	10,46	11,57	29,35
	0,01	2,15	3,20		
	0,10	1,59	1,80		
	0,20	1,42	1,38		
	0,40	1,25	0,92		
150	0	13,53	14,20	16,19	42,13
	0,01	2,87	3,86		
	0,10	2,05	2,88		
	0,20	1,79	2,52		
	0,40	1,60	2,17		

TAB. 6.9: Exp. 6 : BD_3, 30 021 vecteurs, 200 vecteurs requêtes, 20 ppv. Impact du *clustering* sur les performances de la recherche.

6.10 Expérience 7 : étude comparative avec CLINDEX

Il existe de nombreuses méthodes pour la recherche approximative de plus proches voisins. Nous en avons donné un panorama dans la section 3.4, page 70. Cependant, il est

très difficile d'effectuer des études comparatives de performances entre ces méthodes et la méthode que nous proposons. En effet, en plus du temps de réponse, il est nécessaire d'évaluer la précision des résultats retournés. Il est également important de distinguer les méthodes permettant de contrôler la précision de la recherche *a priori* de celles qui permettent de retourner un résultat approximatif sans aucune indication sur sa précision.

Dans cette section, nous présentons une étude comparative des performances de notre méthode par rapport à celles de CLINDEX [Li *et al.*, 2002]. Nous avons présenté cette technique dans la section 3.4.1.2, page 71. Nous avons retenu CLINDEX pour cette étude car c'est la seule méthode dont nous avons pu récupérer l'implémentation auprès des auteurs[4]. Malheureusement, CLINDEX est une méthode de recherche approximative sans aucun contrôle de la précision. Elle partitionne les données hors ligne. En ligne, elle considère un nombre fixe de *clusters* proches du vecteur requête. Ce nombre est un paramètre crucial car la précision des résultats retournés en dépend fortement. Plus ce nombre est grand, meilleure est la précision des résultats retournés. Il n'existe, par contre, aucune relation qui explique le lien entre le nombre de *clusters* considérés lors de la recherche et la précision des résultats.

Une manière de comparer notre méthode à CLINDEX est d'effectuer les mêmes recherches de plus proches voisins avec notre méthode et aussi avec CLINDEX en faisant varier leurs paramètres de contrôle, α pour notre méthode, et le nombre de *clusters* considérés pour CLINDEX. Pour chaque ensemble de résultats, nous calculons ensuite la précision effective par rapport à un résultat qui serait retourné par un algorithme de recherche exacte. Cette précision est calculée tout simplement comme la proportion des k-plus proches voisins retrouvés dans le résultat approximatif. Ceci permet d'avoir plusieurs résultats approximatifs avec différentes précisions effectives pour les deux méthodes. Il s'agit ensuite de comparer les nombres de vecteurs lus, les nombres de *clusters* lus et les temps de réponses des deux méthodes pour les expériences qui ont permi d'avoir la même précision effective.

Précision	Clindex			Notre méthode		
moy. effective	# *clusters*	# vecteurs	tps. rech. (s)	# *clusters*	# vecteurs	tps. rech. (s)
0,99	5000	361 630,1	762,21	27,4	22 858,4	10,36
0,95	2500	297 988,2	431,33	13,6	13 982,1	8,80
0,90	600	198 858,2	166,77	8,5	9 887,1	8,01
0,80	120	130 610,6	110,23	4,1	5 697,1	7,36
0,67	20	86 163,5	85,22	2,4	3 522,8	7,02

TAB. 6.10: Exp. 7: BD_2, dimension 24. 200 vecteurs requêtes. Étude comparative entre CLINDEX et notre méthode. Statistiques sur les données lues pendant la recherche et sur les temps de réponse pour des recherches renvoyant des résultats de même précision effective.

Nous avons effectué cette étude en utilisant deux ensembles de données, BD_2 et BD_3^{150}. Pour chacun, nous avons effectué 200 recherches de 20 plus proches voisins. Les résultats obtenus sont donnés dans le tableau 6.10 pour l'ensemble BD_2 et dans le tableau 6.11 pour l'ensemble BD_3^{150}.

Ces tableaux montrent clairement que notre méthode est plus performante que CLINDEX. Elle permet de retrouver des résultats de même précision que les résultats retournés

[4]Dans ce type d'étude, il est important d'utiliser les implémentations des auteurs car elles sont les mieux optimisées et les paramètres internes y sont bien étudiés.

Précision	Clindex			Notre méthode		
moy. effective	# *clusters*	# vecteurs	tps rech. (s)	# *clusters*	# vecteurs	tps rech. (s)
0,97	800	15 226,6	82,64	8,0	2247,5	2,87
0,93	200	9 930,2	52,68	4,6	1403,9	2,37
0,85	60	5 039,2	40,30	2,1	706,0	1,79
0,74	30	3 210,6	35,05	1,2	492,9	1,60

TAB. 6.11: Exp. 7 : BD_3^{150}, dimension 150. 200 vecteurs requêtes. Étude comparative entre CLINDEX et notre méthode. Statistiques sur les données lues pendant la recherche et sur les temps de réponse pour des recherches renvoyant des résultats de même précision effective.

par CLINDEX en lisant beaucoup moins de vecteurs. Notons cependant, que la comparaison du nombre de *clusters* lus par les deux méthodes n'est pas pertinente car les deux méthodes utilisent des algorithmes de *clustering* différents. La différence en temps de réponse est également flagrante. Notre méthode est 73 fois plus rapide que CLINDEX dans le cas de l'ensemble BD_2 lorsque la précision effective est égale à 0,99 (c'est-à-dire quand, en moyenne, 99 % des 20 plus proches voisins sont retrouvés). Cette différence dans les temps de réponse est due à la grande quantité de vecteurs lus pas CLINDEX par rapport à notre méthode. Par ailleurs, il faut également signaler que le code de CLINDEX n'est pas très bien optimisé (e.g. les fichiers de données sont stockés en ASCII).

6.11 Expérience 8 : influence du nombre k de plus proches voisins

Dans toutes les expériences présentées précédemment, nous avons fixé le nombre k de plus proches voisins à rechercher à 20. Il est cependant important d'étudier les performances de notre méthode pour d'autres valeurs de k.

Pour cela, nous avons effectué 200 recherches de k-plus proches voisins dans l'ensemble BD_2, en faisant varier k. Les valeurs de k considérées sont 1, 5, 10, 15, 20, 30 et 50. Le but de cette expérience est d'étudier, d'une part, l'évolution du temps de réponse de la recherche pour différentes valeurs de k, et, d'autres part, l'impact de cette variation sur la précision de la recherche.

Étude du temps de réponse. Les temps de réponse cumulés pour les recherches effectuées sont résumés dans le tableau 6.12. Nous avons effectué les mêmes recherches avec notre méthode pour quatre valeurs de α, et avec la recherche séquentielle, avec et sans la distance partielle.

Nous remarquons que le temps de la recherche séquentielle est indépendant de k, ce qui est tout à fait normal car les mêmes nombres de calculs de distance et d'E/S sont effectués quel que soit k. Cependant, en utilisant notre méthode, ce temps croît légèrement lorsqu'on augmente k. Cela était prévisible car en augmentant k, le champ de recherche autour du vecteur requête s'étend, ce qui a pour effet de sélectionner plus de *clusters* à lire, et donc, d'augmenter le temps de recherche.

La réduction du temps de recherche par rapport à la recherche séquentielle de base reste toutefois élevée quel que soit k. Lorsque α est égal à 0,01, la recherche approximative est

12,66 plus rapide que la recherche séquentielle de base pour $k = 1$. Ce facteur d'accélération est de 10,08 lorsque $k = 15$ et de 9,20 lorsque $k = 50$.

k	Rech. seq. de base	Rech. seq. avec dist. part.	Recherche approx. (α)				
			0	0,01	0,10	0,20	0,40
1	100,18	63,52	17,67	7,91	6,83	6,52	6,39
5	100,28	65,30	20,23	9,26	7,51	6,92	6,67
10	99,93	65,94	20,82	9,78	7,73	7,11	6,76
15	100,08	66,52	21,27	9,92	7,89	7,26	6,77
20	100,73	67,19	21,59	10,16	8,11	7,31	6,85
30	100,43	67,72	21,94	10,43	8,27	7,43	6,93
50	100,19	68,67	22,60	10,88	8,54	7,66	7,07

TAB. 6.12: Exp. 8. BD_2. Influence de nombre k de plus proches voisins recherchés sur le temps de recherche – Temps de réponse cumulés pour 200 vecteurs requêtes et pour différentes stratégies de recherche pour différentes valeurs de k.

Étude de la précision. Pour étudier l'impact du nombre de plus proches voisins k sur la précision de la recherche, nous avons calculé l'imprécision effective pour chaque résultat approximatif. Nous avons procédé de la même manière que dans l'expérience 3 (section 6.6). Nous avons calculé pour chaque résultat approximatif, la proportion des plus proches voisins non retrouvés. L'imprécision effective moyenne est calculée comme la moyenne de cette proportion sur l'ensemble des résultats approximatifs correspondant aux 200 vecteurs requêtes. Les résultats obtenus sont résumés dans le tableau 6.13.

k	α			
	0,01	0,10	0,20	0,40
1	0,01	0,15	0,30	0,35
5	0,01	0,10	0,24	0,36
10	0,01	0,09	0,20	0,38
15	0,01	0,08	0,19	0,36
20	0,01	0,07	0,18	0,36
30	0,01	0,06	0,17	0,35
50	0,02	0,07	0,17	0,37

TAB. 6.13: Exp. 8. BD_2. Étude de l'influence de nombre k de plus proches voisins recherchés sur l'imprécision de la recherche – Imprécision effective moyenne des résultats approximatifs obtenus pour différentes valeurs de k et différentes valeurs de α.

Ce tableau montre que, globalement, le contrôle de l'imprécision est respecté. L'imprécision effective est cependant supérieure à l'imprécision fixée au travers du paramètre α dans quatre cas : lorsque $k = 1$ et $\alpha \in \{0,10, 0,20\}$, $k = 5$ et $\alpha = 0,20$, et, $k = 50$ et $\alpha = 0,01$.

Lorsque $k = 50$ et $\alpha = 0,01$, la recherche approximative n'a pas retrouvé, en moyenne, 2 % des 50 plus proches voisins de chaque vecteur requête alors qu'elle n'était autorisée à rater que 1 % des plus proches voisins. Pour expliquer cela, nous avons isolé les vecteurs requêtes pour lesquels l'imprécision effective est supérieure à l'imprécision autorisée. Nous avons ensuite analysé les distances entre chacun de ces vecteurs et ses 50 plus proches voisins. Nous avons remarqué que ces distances sont très proches au-delà du 40^e plus proche voisin. La différence relative entre les distances qui correspondent à chaque deux plus proches voisins successifs est inférieure à 0,05 %. Ceci explique la perte de certains plus proches voisins lors de la recherche car ces plus proches voisins se trouvent pratiquement à la même distance du vecteur requête. Il est donc très facile d'en rater quelques uns et de les remplacer par d'autres vecteurs. Ces vecteurs ne font pas partie des 50 plus proches voisins. Toutefois, ils sont à une distance du vecteur requête pratiquement identique à celle des plus proches voisins qui auraient dû être retrouvés.

Les trois autres cas où l'imprécision n'a pas été respectée concernent les petites valeurs de k (1 et 5). Pour comprendre la cause de cette perte de précision, nous avons analysé les voisins retournés et leurs distances par rapport aux vecteurs requêtes.

Lorsque $k = 1$ et $\alpha = 0,10$, pour 31 sur 200 vecteurs requêtes, le plus proche voisin n'a pas été retrouvé alors que la recherche était censée le manquer uniquement pour 20 vecteurs requêtes au maximum. Sur les 31 vecteurs retournés au lieu des plus proches voisins, 14 sont les 2^e plus proches voisins et 7 sont les 3^e plus proches voisins. Les distances entre ces vecteurs et les vecteurs requêtes sont très proches des distances entre les vecteurs requêtes et leurs plus proches voisins. De nouveau, cette perte de précision s'explique donc par le fait que les voisins les plus proches du vecteur requête se trouvent tous à pratiquement la même distance du vecteur requête. Il est donc très facile de « rater » le plus proche voisin et de retourner le 2^e ou le 3^e plus proches voisins.

Une façon de renforcer la contrainte de contrôle de la précision dans ces cas extrêmes (trop petites et trop grandes valeurs de k), est de sous-estimer légèrement le paramètre $P(H)$. Nous rappelons que ce paramètre intervient dans le calcul des rayons approximatifs des *clusters*. Le sous-estimer permet d'adopter une approche plus prudente lors de la réduction des rayons des *clusters*. Dans l'ensemble des expériences présentées dans ce chapitre, nous l'avons fixé à 1 sur la base des résultats de l'expérience 1 (section 6.4). Ces résultats ont montré que son estimation par la procédure décrite dans la section 5.3.4, page 124, était très proche de 1. Nous l'avions alors approché par 1.

Pour montrer l'impact $P(H)$ sur la précision, nous avons refait l'expérience de la recherche des k plus proches voisins de 200 vecteurs requêtes en faisant varier k et pour deux valeurs différentes de $P(H)$ (0,95 et 0,90). Notons que nous avons repris le même ensemble de vecteurs et les mêmes vecteurs requêtes. Les résultats de l'évaluation de la précision effective des résultats sont résumés dans les tableaux 6.14.a et 6.14.b.

Ces tableaux montrent bien qu'en augmentant $P(H)$, l'imprécision effective des données est diminuée. Lorsque $P(H) = 0,95$, l'imprécision effective est supérieure à l'imprécision fixée seulement lorsque $k = 1$ et $\alpha = 0,10$ ou $\alpha = 0,20$. Cependant, lorsque $P(H) = 0,90$, l'imprécision effective est toujours inférieure à l'imprécision fixée au travers de α. Ceci montre bien que l'augmentation de α est une solution possible pour traiter le problème de la précision pour les cas extrêmes. Nous remarquons cependant, que l'imprécision effective est beaucoup plus petite que l'imprécision tolérée. Lorsque $P(H) = 0$, un résultat exact est obtenu par des recherches approximatives avec $\alpha = 0,01$ pour plusieurs valeurs de k.

(a) $P(H) = 0,95$

k	α			
	0,01	0,10	0,20	0,40
1	0	0,120	0,265	0,355
5	0	0,061	0,185	0,359
10	0,001	0,045	0,162	0,350
15	0,001	0,043	0,151	0,349
20	0,001	0,041	0,137	0,335
30	0,002	0,037	0,124	0,324
50	0,001	0,048	0,116	0,331

(b) $P(H) = 0,90$

k	α			
	0,01	0,10	0,20	0,40
1	0	0,05	0,18	0,35
5	0	0,03	0,14	0,35
10	0	0,02	0,11	0,33
15	0,0007	0,02	0,10	0,32
20	0,0002	0,02	0,08	0,31
30	0,0003	0,02	0,08	0,30
50	0.0002	0,03	0,09	0,30

TAB. 6.14: Exp. 8. BD_2. Étude de l'influence de nombre k de plus proches voisins recherchés sur l'imprécision de la recherche – Imprécision effective moyenne des résultats approximatifs obtenus pour différentes valeurs de k, différentes valeurs de α et pour deux valeurs de $P(H)$.

Dans ce cas, les réductions de rayons des *clusters* ne sont pas suffisantes pour introduire de l'imprécision dans les résultats et la méthode ne tire pas profit de la totalité de l'imprécision tolérée par l'utilisateur. Par conséquent, l'augmentation de $P(H)$ pour diminuer l'imprécision des résultats doit être réservée aux cas extrêmes, c'est-à-dire pour de très petites ou de très grandes valeurs de k.

6.12 Synthèse

Dans ce chapitre, nous avons présenté les expérimentations réalisées pour l'évaluation des performances de la méthode de recherche de plus proches voisins proposée. Ces expériences ont permis de corroborer les analyses effectuées lors de la présentation de la méthode dans le chapitre 5. Ainsi, nous avons montré que le paramètre $P(H)$ est en général très proche de 1, et ce, pour différents ensembles de données de distributions différentes et de dimensions variables. Nous avons aussi étudié expérimentalement la valeur optimale du nombre de *clusters*, c'est-à-dire, le nombre de *clusters* qui minimise le temps de réponse. Le résultat de cette étude est une heuristique qui stipule que le nombre optimal de *clusters* se situe entre 1 et 3 fois la racine carrée du nombre de vecteurs de la base. Notons cependant que les temps de calcul de l'algorithme de *clustering* sont relativement élevés. Ils constituent la principale difficulté qui limite la taille de la base que peut traiter notre algorithme de recherche. Ce point est inscrit dans les perspectives de ce travail et sera repris en détail dans la conclusion générale. Nous avons également montré que la réduction des rayons de *clusters* est très importante même pour des niveaux d'imprécision très faibles. Ceci permet d'avoir des réductions du temps de réponse considérables tout en effectuant des recherches avec un niveau d'imprécision très faible. Nous avons ensuite évalué l'évolution du temps de réponse lorsque la taille et/ou la dimension des données augmentent. Dans les deux cas, notre méthode est plus performante que la recherche séquentielle même lorsque $\alpha = 0$, c'est-à-dire lorsque la recherche retourne un résultat exact. Dans le cas d'une recherche approximative, le facteur de gain en temps de réponse par rapport à la recherche séquentielle se situe entre 10 et 15. Nous avons aussi étudié la sensibilité de l'algorithme de recherche par rapport à l'algorithme de regroupement des données hors ligne. Pour cela, nous avons

évalué les performances de l'algorithme de recherche en utilisant deux partitions différentes de l'ensemble des données, une partition créée par notre algorithme de *clustering* et une autre extraite à partir d'un SS-Tree. Cette étude a montré que l'algorithme a un impact sur les performances de l'algorithme de recherche particulièrement lorsque $\alpha = 0$. Lorsque α est supérieur à 0, les réductions des rayons des *clusters* permettent de réduire le taux de chevauchement entre les *clusters* et d'augmenter leur compacité même si, initialement, la qualité des *clusters* était très mauvaise en termes de compacité et de séparabilité. Dans ce cas, l'algorithme de recherche est moins sensible à la partition des données. Ce résultat est l'une des conclusions les plus importantes de cette étude. Il est cependant important d'investiguer davantage cette relation. Nous avons aussi effectué une étude comparative entre notre méthode et CLINDEX qui a clairement montré la supériorité de notre méthode *clustering*. Enfin, nous avons étudié l'impact du nombre k de plus proches voisins recherchés sur les performances de notre méthode de recherche. Cette étude a montré que le gain en temps de réponse est toujours très élevé quelle que soit la valeur de k. La précision des résultats est cependant quelque peu dépendante de k. Pour les très petites ou très grandes valeurs de k, l'imprécision effective des résulats était parfois légèrement supérieure à l'imprécision tolérée. Dans ces cas particuliers, cette étude suggère d'adopter une approche plus prudente en sous-estimant la valeur de $P(H)$ utilisée lors du calcul des rayons approximatifs.

Troisième partie

Application à la recherche d'images par le contenu

Introduction

Ayant développé une méthode originale et efficace pour la recherche de plus proches voisins, nous nous intéressons dans cette partie à son intégration et son évaluation dans le cadre d'un système réel pour la recherche d'images par le contenu.

Nous avons choisi comme application la recherche d'images par le contenu pour la détection de copies. L'objectif de cette application est de vérifier si les images présentes sur des sites tiers ne proviennent pas de la collection d'un propriétaire. Sachant que les images piratées subissent plusieurs changements avant d'être publiées, le schéma de description des images doit être robuste face à ces changements. C'est aussi une application dans laquelle l'efficacité de la recherche est une contrainte très importante. En effet, la taille des collections d'images des professionnels se chiffre à plusieurs millions d'images et le nombre d'images suspectes devant être vérifiées quotidiennement est très important.

La motivation du choix de cette application est double. D'une part, c'est une application aux objectifs bien définis. Il s'agit principalement d'être capable de mettre en correspondance une image ayant subi un certain nombre de changements avec l'image originale. Il est donc facile de mettre au point une stratégie d'évaluation automatique du système développé. Il suffit pour cela de simuler artificiellement les changements que peuvent subir les images piratées et de tester la robustesse du système proposé face à ces changements. D'autre part, cette application se base sur l'utilisation d'un schéma de description d'image complexe et extrêmement coûteux, ce qui permet d'évaluer l'efficacité de notre méthode de recherche dans des conditions notoirement difficiles.

Cette partie est composée de deux chapitres. Le premier revient sur le contexte applicatif du problème considéré puis décrit le système de recherche d'images par le contenu que nous proposons pour la détection de copies. Dans le deuxième chapitre, sont présentées les expérimentations réalisées pour la validation du système. Ces expérimentations concernent l'efficacité de la recherche, la robustesse du schéma de description et l'impact de l'imprécision introduite lors de la recherche de plus proches voisins sur le résultat final.

Chapitre 7

Recherche d'images par le contenu pour la détection des copies

7.1 Introduction

La protection du droit d'auteur est un problème clé dont la solution est un préliminaire à la diffusion de collections de photos sur le Web par leur propriétaire. Comme il n'est pas possible d'empêcher la copie d'images sur des sites Web par des personnes techniquement compétentes, les propriétaires d'images comme les agences de photos doivent disposer de moyens pour vérifier si les images présentes sur des sites tiers ne viennent pas de leur propre collection d'images. Ceci est d'autant plus important si le tiers tire profit des images qu'il prétend posséder, au détriment de leur propriétaire légitime.

Le tatouage d'images est la première solution à laquelle on peut penser, mais en faire un argument légal demande une organisation complexe sur le plan technique (de la technique elle-même) et organisationnel (des protocoles, des tiers de confiance...) [Petitcolas *et al.*, 1998, Cox *et al.*, 2002]. Ceci est d'autant plus vrai que les images copiées sont souvent lessivées pour enlever les marques cachées ou rendre leur détection impossible.

Dans ce chapitre, nous proposons un système de détection de copies basé sur un schéma de recherche d'images par le contenu. L'idée est de fournir aux propriétaires d'images des outils pour tester si une image provient de leur stock, et ce sur des bases visuelles seulement. Le but d'un tel système est alors d'être capable d'établir des liens de ressemblance entre des images issues du Web et celles d'une collection possédée par un propriétaire et, au cas où une telle ressemblance existe, de trouver l'image de la collection qui a été copiée.

Pour cela, le système de recherche d'images par le contenu (SRIC) est soumis principalement à trois contraintes.

Robustesse du schéma de description. Il doit être capable de reconnaître une image copiée à partir de l'original, même si la copie a été modifiée : il est probable que les pirates modifient les images qu'ils récupèrent. Ces modifications sont appelées aussi des attaques. Elles peuvent être appliquées à l'image intentionnellement pour rendre son identification difficile, pour enlever un possible tatouage ou bien pour rendre sa détection impossible. Elles peuvent aussi être appliquées pour des raisons éditoriales (par exemple, pour sélectionner la partie d'une photo pertinente par rapport à un

article ou un événement donné). Le SRIC utilisé doit en conséquence être robuste à ces modifications. Les plus fréquentes sont le changement de résolution, la modification légère des couleurs ou du contraste, la rotation, la compression JPEG, la symétrie miroir et l'extraction d'une sous-image. Dans un tel contexte, il ne s'agit donc pas de reconnaître toutes les images de fleurs rouges à partir d'une image requête représentant elle aussi une fleur rouge, mais plutôt de déterminer si une image suspecte est une copie d'une image que l'on détient, et dans l'affirmative, de retrouver de quelle image elle en est une copie. Cela impose d'utiliser une notion de similarité bien plus contraignante et fine que celle utilisée par la plupart des SRICs. On parlera dans la suite d'images semblables plutôt que d'images similaires pour souligner cette différence.

Efficacité de la recherche. Le SRIC ne peut pas fournir de preuves légales de copie, il ne peut qu'émettre des suspicions. La preuve finale du fait qu'il y a eu copie ne peut être fournie que par un expert humain à qui sont transmis les résultats obtenus par le système. Il importe donc que ce dernier ait d'excellents rappel et précision : rappel, car le but du système est bien de détecter les copies et il ne faut pas en rater, précision, car tout faux-positif se traduit par un travail de l'expert dépensé en vain. Le système doit donc fournir des résultats de très haute qualité.

Rapidité de la recherche. Le SRIC doit pouvoir traiter un grand nombre d'images récupérées sur le Web et les confronter à de grandes bases d'images. La réponse à une requête doit donc être rapide. Or, les SRICs sont le plus souvent basés sur l'emploi de techniques d'indexation multidimensionnelle pour effectuer les recherches de plus proches voisins. Dans la première partie de cette thèse, nous avons montré que de tels algorithmes sont très complexes et leurs performances sont plus mauvaises qu'une simple recherche séquentielle et exhaustive en grande dimension. L'aspect rapidité du système apparaît donc à la fois comme crucial et potentiellement bloquant.

Le système que nous proposons dans ce chapitre prend en compte les trois critères décrits ci-dessus. Il utilise une technique robuste de description du contenu visuel des images. Cette technique est basée sur l'extraction de points d'intérêt et la caractérisation du signal autour de ces points par des invariants différentiels locaux [Florack *et al.*, 1994, Schmid & Mohr, 1997, Amsaleg & Gros, 2000, Amsaleg & Gros, 2001]. Elle offre une grande capacité de reconnaissance et d'invariance face aux changements que peuvent subir les images piratées. Le schéma de recherche associé est, par contre, extrêmement coûteux car plusieurs descripteurs sont associés à chaque image. Ce mode de description fait que chaque requête se traduit par plusieurs requêtes de plus proches voisins, autant qu'il y a de descripteurs dans l'image requête. Le système utilise alors la technique de recherche approximative de plus proches voisins que nous avons proposée dans la deuxième partie de cette thèse pour accélérer les recherches. Comme nous l'avons montré, l'approximation faite tend à échanger un peu de précision dans les résultats contre une accélération notoire du temps de recherche (*cf.* chapitre 6).

La première contribution de ce chapitre est la combinaison de ces deux méthodes, celle de description locale et celle de recherche approximative, rendant possible le développement d'un système de détection des copies basé sur une reconnaissance des images copiées par le contenu. Un des points importants est de mesurer l'impact en terme de qualité de

l'approximation réalisée lors de la recherche, et de vérifier que cette qualité reste acceptable pour l'application que nous visons. La deuxième contribution est donc cette vérification tant au plan théorique que pratique.

7.2 Reconnaissance d'images par le contenu

La reconnaissance d'images par le contenu est basée sur la calcul de quantités décrivant les images, quantités appelées descripteurs, et sur l'hypothèse que la mesure de la ressemblance entre les images peut se ramener à celle de la distance entre les descripteurs calculés à partir de ces images. Une grande variété de descripteurs a été proposée [Veltkamp & Tanase, 2000, Smeulders *et al.*, 2000]. Le schéma de description le plus fréquent est celui des descripteurs globaux où un unique descripteur décrit la globalité d'une image. Les histogrammes de couleur ou de niveaux de gris et les corrélogrammes sont des exemples typiques de tels descripteurs [Stricker & Swain, 1994, Huang *et al.*, 1997]. Ces descripteurs sont robustes au sens où ils sont peu affectés par le bruit qui peut affecter le signal. En contrepartie, ils ne sont pas adaptés pour retrouver des parties communes entre deux images et ne sont pas robustes à l'extraction de sous-images (ou recadrage), propriétés indispensables dans le contexte de la détection des copies.

À l'opposé, les méthodes de reconnaissance basées sur des descripteurs locaux offrent naturellement ces propriétés d'invariance, et paraissent donc plus adaptées à notre contexte applicatif. Ces méthodes consistent à calculer de nombreux descripteurs pour une même image, chaque descripteur décrivant une petite partie seulement de l'image, que ce soit autour d'un point ou dans une petite région de l'image. La reconnaissance globale entre deux images est alors jugée en fonction du nombre de descripteurs locaux qui coïncident ou qui sont très proches entre les deux images. Un tel schéma prend en compte les détails des images et permet la prise en compte d'une notion fine de similarité entre les images. On parlera d'images semblables dans ce cas. Ce schéma offre également une bonne robustesse naturellement car il se base sur l'accumulation d'évidences locales. Le résultat final de la recherche d'images semblables à une image requête est la consolidation de plusieurs résultats intermédiaires, chacun étant le résultat de la recherche de plus proches voisins d'un descripteur de l'image requête.

Dans le système que nous présentons ici, nous avons retenu ce deuxième schéma de description. Les descripteurs utilisés appartiennent à la famille des invariants différentiels locaux introduite par [Florack *et al.*, 1994], puis utilisée et évaluée dans le contexte de la reconnaissance d'images en niveaux de gris par [Schmid & Mohr, 1997, Dufournaud *et al.*, 2000b, Schmid *et al.*, 2000]. Leur extension à la couleur a été présentée dans [Deriche *et al.*, 1998, Amsaleg & Gros, 2000, Amsaleg & Gros, 2001]. Ces descripteurs offrent une très grande robustesse aux transformations qu'une image peut subir, et ils peuvent détecter que deux images ont une partie en commun seulement, et cela à une translation, une rotation ou une variation géométrique ou photométrique près.

7.2.1 La famille des descripteurs locaux différentiels

Le calcul de ces descripteurs locaux pour décrire le contenu d'une image est réalisé en trois étapes. (i) On commence par détecter dans l'image des points particuliers, dits points d'intérêt, qui portent une partie importante de l'information contenue dans le signal. Ces

points correspondent à des singularités bidimensionnelles du signal et sont détectés en utilisant l'algorithme de Harris [Harris & Stephens, 1988] amélioré par Schmid [Schmid, 1996]. Le nombre de points détectés dans une image peut varier, car il dépend de la forme du signal de l'image et de la technique de seuillage utilisée (entre 10 et 600 pour une image 512×512) ; (ii) le signal autour de chacun de ces points d'intérêt est caractérisé par sa convolution avec une gaussienne et ses 9 premières dérivées jusqu'à l'ordre 3 ; (iii) enfin, les quantités obtenues (au nombre de 10 pour une image en niveau de gris et de 30 pour une image couleur, 10 par canal de couleur) sont combinées afin d'obtenir les propriétés d'invariance souhaitées en fonction de l'application visée. Il existe plusieurs variantes possibles de ces descripteurs. Ils se différencient par les propriétés d'invariance souhaitées et le type d'images traitées (niveaux de gris/couleur).

Les descripteurs retenus sont ceux présentés dans [Amsaleg & Gros, 2001]. Ils sont de dimension 24 et sont invariants aux translations, aux rotations et aux changements d'illumination décrits en RGB par une matrice diagonale et un vecteur de translation. La mesure de similarité associée à ces descripteurs est la distance euclidienne. Nous les décrivons dans les deux sections suivantes en nous basant sur les descriptions données dans [Schmid, 1996] et [Amsaleg & Gros, 2001].

7.2.1.1 Extraction de points d'intérêt

Les points d'intérêt sont déterminés de façon à ce qu'un point détecté dans une image le soit également, avec la plus grande probabilité possible, dans une autre image semblable à la première. La robustesse d'un détecteur se mesure donc à sa capacité de détection des mêmes points dans deux images semblables (critère de répétabilité). Une multitude de détecteurs de points d'intérêt ont été proposés. C. Schmid en fait un état de l'art dans [Schmid, 1996] et les classe en 3 grandes catégories : les méthodes basées sur les contours, les méthodes basées sur le signal et les méthodes basées sur un modèle théorique du signal. Elle fait également une étude comparative approfondie de la robustesse de ces détecteurs dans [Schmid, 1996, Schmid *et al.*, 2000]. Cette étude montre que le détecteur introduit par Bigün, Granland et Wikland [Bigün *et al.*, 1991], et par Harris et Stephens [Harris & Stephens, 1988] est le plus robuste. Ce détecteur permet de retrouver les singularités 2D de l'image en se basant sur l'étude des valeurs propres de la matrice :

$$e^{-\frac{x^2+y^2}{2\sigma^2}} \otimes \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{7.1}$$

où I_x et I_y sont les convolutions du signal de l'image (I) avec les deux dérivées partielles $\partial G/\partial x$ et $\partial G/\partial y$ d'une fonction gaussienne. Les valeurs propres de cette matrice sont les courbures principales de la fonction d'auto-corrélation du signal. Si ces deux courbures sont grandes, alors le point où la matrice a été calculée est un point de singularité car un changement important du signal selon les deux directions x et y s'y produit. Ce point est donc retenu et est considéré comme un point d'intérêt.

7.2.1.2 Calcul des descripteurs locaux

Dans cette section, nous commençons par la présentation des descripteurs sur des images en niveaux de gris, puis nous décrivons leur extension aux images couleurs.

Une fois l'extraction des points d'intérêt réalisée, des invariants sont calculés en utilisant les dérivées du signal autour de chaque point jusqu'à l'ordre 3. Un invariant est une grandeur qui reste invariante aux différentes transformations que peut subir l'image. L'ensemble des invariants calculés en un point constitue un descripteur.

Invariance à la translation et à la rotation. L'invariance à la translation est implicite. Elle est garantie par le schéma de description qui calcule des invariants autour des points d'intérêt. Si l'image est translatée, les points le sont également et les invariants calculés dans un repère centré sur le point restent donc inchangés. Quant à la rotation, son angle est éliminé algébriquement. La procédure de calcul des invariants se résume en trois étapes :

1. autour de chaque point, on choisit un support pour le calcul des invariants. Afin d'obtenir des quantités invariantes à la rotation de l'image, les supports sont choisis circulaires ;
2. par des convolutions du signal avec des dérivées de la fonction gaussienne, on calcule les dix premières dérivées du signal en chaque point d'intérêt. Si le signal de l'image est $I(x, y)$, on calcul donc $I, I_x, I_y, I_{xx}, I_{xy}, I_{yy}, I_{xxx}, I_{xyy}, I_{xxy}$ et I_{yyy} ;
3. ces dix quantités sont ensuite mélangées afin d'obtenir des invariants à la rotation. Il en résulte alors neuf invariants, écrits ici en notation d'Einstein :

$$\begin{array}{c} I \\ I_i I_i \\ I_i I_{ij} I_j \\ I_{ii} \\ I_{ij} I_{ji} \\ \varepsilon_{ij} \left(I_{jkl} I_i I_k I_l - I_{ikk} I_i I_l I_l \right) \\ I_{iij} I_j I_k I_k - I_{ijk} I_i I_j I_k \\ -\varepsilon_{ij} I_{jkl} I_i I_k I_l \\ I_{ijk} I_i I_j I_k \end{array} \tag{7.2}$$

où ε_{ij} est défini par $\varepsilon_{xy} = -\varepsilon_{yx} = 1, \varepsilon_{xx} = \varepsilon_{yy} = 0$. La notation d'Einstein correspond à la sommation sur les indices répétés : par exemple, $I_i = \sum_i I_i = I_x + I_y$ et $I_{ij} I_{ji} = \sum_i \sum_j I_{ij} I_{ji} = I_{xx} I_{xx} + 2 I_{xy} I_{xy} + I_{yy} I_{yy}$.

Invariance au changement d'illumination. Le modèle de changement d'illumination considéré est un modèle affine : $I' = aI + b$. Même si ce modèle paraît simple *a priori*, il décrit de façon précise les petites variations de l'intensité d'illumination globale. Pour rendre les neuf invariants présentés dans le paragraphe précédent invariants aux changements d'illumination décrits par ce modèle, il suffit d'éliminer les deux premiers et de diviser les sept restants par la puissance adéquate de $I_i I_i$ de façon à éliminer le paramètre a.

Extension aux images couleurs. L'extension des invariants précédemment introduits au cas des images couleurs a été proposée dans [Amsaleg & Gros, 2000,

Amsaleg & Gros, 2001]. Cette extension ne change cependant pas la méthode de calcul des points d'intérêt par le détecteur de Harris. Elle nécessite, cependant, le choix d'un système de représentation de l'information de couleur d'une part, et, d'autre part, la description d'un modèle de variation des valeurs associées à chaque pixel lorsque les conditions d'illumination changent. Il existe de nombreux systèmes de représentation de la couleur (RGB, HSV, Lab, etc. [Poynton, 1997]). Le système retenu est le système RGB non linéaire. Dans ce cas, le signal étant décrit par 30 dérivées en chaque point, 10 dérivées par canal de couleur, la façon dont sont mélangées les dérivées est alors modifiée.

Pour assurer une invariance à la rotation, 3×9 invariants sont calculés en utilisant la formule 7.2. Neuf invariants sont calculés selon chaque canal. Deux autres invariants sont choisis parmi les 3 entités suivantes :

$$R_x G_x + R_y G_y \qquad R_x B_x + R_y B_y \qquad G_x B_x + G_y B_y \tag{7.3}$$

Toutefois, dans la pratique, ces trois entités peuvent être retenues (pour des raisons numériques).

L'invariance photométrique est plus complexe que dans le cas des images en niveaux de gris car de nombreux modèles de changement d'illumination différents peuvent être considérés. Un modèle général s'écrit comme suit :

$$(R', G', B')^T = M(R, G, B)^T + V$$

où M est une matrice 3×3 et V un vecteur de dimension 3. Des modèles particuliers sont obtenus en utilisant une matrice M diagonale ou bien scalaire et un vecteur V nul. En fonction du nombre de paramètres du modèle, la dimension des descripteurs varie entre 18 et 29. Dans [Amsaleg & Gros, 2001], seul le cas où la matrice M est une matrice diagonale a été étudié. Dans ce cas, les invariants peuvent être calculés indépendamment selon chaque canal comme dans le cas des images en niveaux de gris. En rajoutant les trois termes 7.3, les descripteurs résultant sont de dimension 24.

7.2.2 Schéma de recherche basé sur les descripteurs locaux

Dans le schéma de description d'images par descripteurs différentiels locaux, chaque image est décrite par un ensemble de descripteurs (généralement entre 10 et 600 descripteurs pour une image 512×512). Par conséquent, la procédure de recherche des images semblables à une image requête est complètement différente des procédures de recherche plus classiques utilisant des descripteurs globaux. Dans ce dernier cas, il suffit de rechercher les k-plus proches voisins du descripteur de l'image requête pour avoir les k images les plus similaires car chaque image est décrite par un unique descripteur. Dans une recherche basée sur les descripteurs locaux, l'image requête, tout comme les images de la base, est elle aussi décrite par plusieurs descripteurs. Rechercher les images semblables à une image requête génère donc autant de recherches de k-plus proches voisins que de descripteurs dans cette image requête. Chacun des plus proches voisins obtenus donne un vote à l'image de la base dont il est issu. Les images de la base peuvent alors être classées selon le nombre de votes qu'elles ont obtenus. Ce nombre est noté NDS (Nombre de Descripteurs Semblables) dans la suite. NDS donne le nombre de descripteurs en commun (ou proches) entre l'image requête et l'image de la base à laquelle il est associé.

De manière générale, de très nombreuses images reçoivent un, voire quelques votes. Si la base ne contient aucune image semblable à l'image requête, toutes les images qui reçoivent des votes ont des NDSs proches et faibles. À l'opposé, si la base contient des images semblables, la liste des NDSs obtenus se sépare clairement en deux groupes : les images semblables obtiennent des scores élevés qui décroissent en fonction de la ressemblance de ces images avec l'image requête, puis il y a un saut et on trouve ensuite les autres images qui ont des scores faibles. Voici un exemple typique d'une telle liste de NDSs ordonnée de manière décroissante : 444, 258, 142, 42, 17, 14, 10, 9, ..., 9, 8, ..., 8, 7, ...

Il y aurait un grand intérêt à pouvoir déterminer *automatiquement* quelles sont les images semblables à l'image requête. Mais cela n'est pas immédiat. Prendre un nombre fixe d'images (la première, les cinq ou les seize premières en fonction de la taille de l'écran d'affichage) ne permet pas de prendre en compte la redondance possible dans la base et peut dégrader la précision. Par ailleurs, cela implique que le système retourne toujours une réponse et ne peut pas déclarer qu'aucune image n'est semblable à une image donnée. Si toutes les images sont envoyées à un expert et que l'on teste des milliers d'images par jour pour des vérification de copie, cela devient inacceptable.

L'utilisation d'un seuil fixe est aussi problématique car l'ordre de grandeur des NDSs dépend du nombre des descripteurs requêtes d'une part, et aussi de la zone commune entre l'image requête et l'image recherchée d'autre part. Il n'est donc pas possible de fixer *a priori* un seuil qui puisse marcher pour une base donnée indépendamment de l'image requête.

La solution que nous proposons est d'assimiler ce problème à celui de la détection d'une transition dans un signal, et d'utiliser une technique issue du traitement du signal pour détecter cette transition. Pour cela, on trie la liste des NDSs par ordre croissant et on lui applique le test de Page-Hinkley [Page, 1954, Hinkley, 1971, Basseville, 1988]. Le système retourne finalement les images qui ont été séparées des autres par ce test, ou un résultat vide si le test n'a détecté aucune transition dans la liste des NDSs. Nous terminons cette section par la présentation de ce test.

Test de Page-Hinkley. Le test de Page-Hinkley est un test à somme cumulée qui permet de détecter les changements dans un signal. Ce test est connu pour sa robustesse, car il permet de prendre en compte l'intégralité du passé du signal (la liste des scores dans notre cas) à un instant donné. Par ailleurs, il est efficace et peu coûteux, il permet une détection précise de la position du saut, et il peut traiter à la fois les changements abrupts et les changements progressifs sans modification du seuil interne de détection.

Ce test permet de détecter en même temps des changements d'augmentation et de diminution de l'amplitude du signal observé. Dans notre cas, comme la liste des scores est triée par ordre croissant, le but est de détecter uniquement des augmentations significatives des scores, ce qui permet d'isoler les scores élevés correspondant aux images semblables des scores des autres images. Nous présentons donc uniquement la version de ce test permettant de détecter des changements d'augmentation.

Le test de Page-Hinkley se base sur le changement de la moyenne du signal pour détecter ses variations. Pour un signal discret $y_0, y_1, \ldots, y_N$, il s'applique itérativement pour chacune des valeurs y_i dans l'ordre. Si l'on considère que la valeur courante est y_n, il

est donné par :

$$\begin{cases} U_0 = 0 \\ U_n = \sum_{t=1}^{n} (y_t - \mu_0 - \frac{\nu}{2}) \\ U_{n_{min}} = \min_{0 \leqslant t \leqslant n} U_t \\ \qquad \text{alarme si } \; U_n - U_{n_{min}} > \lambda \end{cases}$$

où μ_0 est la moyenne du signal en considérant les valeurs passées $y_0, y_1, \ldots, y_n$ et où ν et λ sont les deux paramètres internes du test. ν peut être interprété comme l'écart de la moyenne toléré *a priori*, et λ est un seuil prédéfini. Lorsque $U_n - U_{n_{min}} > \lambda$, le test émet une alarme de détection de changement dans la moyenne du signal. Le moment estimé du saut est n_{min}, c'est-à-dire celui qui minimise la somme U_k pour $k = 1, \ldots, n$. Après de nombreux essais sur des séquences de scores réels obtenus par des expériences de recherche d'images, nous avons fixé ν à 1 et λ à 0,5.

À titre d'exemple, si l'on applique le test de Page-Hinkley sur les trois listes de scores suivantes :

$L_1 = 25, 19, 19, 18, 17, 17, 16, 15, 14, 13, 13, 12, 12, 12, \ldots, 12, 11, \ldots, 11, 10, \ldots$;

$L_2 = 443, 258, 142, 42, 17, 14, 10, 9, 9, 9, \ldots, 9, 8, \ldots, 8, 7, \ldots$;

$L_3 = 750, 613, 517, 482, 361, 218, 216, 103, 27, 13, 13, 13, 12, 12, 11, 11, \ldots, 11, 10, \ldots$;

les scores jugés significatifs et isolés par ce test sont respectivement :

$R_1 = \{25, 19, 19, 18, 17, 17, 16, 15, 14\}$;

$R_2 = \{443, 258, 142, 42, 17, 14, 10\}$;

$R_3 = \{750, 613, 517, 482, 361, 218, 216, 103, 27\}$;

Nous remarquons que ce test ne détecte pas les sauts en terme d'écart entre deux scores successifs, mais plutôt le changement dans la tendance de l'évolution de la suite des scores. Par exemple, dans la liste L_2, la transition détectée se situe entre les scores 9 et 10 même si la différence entre ces deux scores est plus faible que la différence entre les scores suivants, c'est-à-dire 10 et 14. Ce test considère ainsi la totalité de l'historique de la séquence des scores et permet de détecter le premier moment à partir duquel la tendance de l'évolution des scores (en terme de moyenne) change. Ce test est donc un test conservatif dans la mesure où il tend à garder tous les scores, même petits, du moment où ils se distinguent de la tendance des scores correspondant aux bruits. Ceci favorise donc le rappel au détriment de la précision. Ce point se confirme dans les expériences du chapitre suivant.

7.2.3 Recherche de plus proches voisins

Le schéma de recherche d'images par descripteurs locaux différentiels que nous avons présenté apporte finesse et précision dans la recherche. Toutefois, il modifie en profondeur la méthode de recherche des images semblables. Tout d'abord, il augmente la taille de la base de données : au lieu de stocker un unique descripteur par image, il nécessite d'en stocker un nombre qui peut varier entre 10 et 600 environ pour des images 512×512. La base grossit typiquement d'un facteur 200. Ensuite, ce schéma de description modifie l'algorithme de recherche lui-même. En effet, il ne s'agit plus de rechercher les descripteurs les plus proches d'un unique descripteur requête, mais de mener de nombreuses recherches à partir de tous les descripteurs de l'image requête, induisant autant de recherches de plus proches voisins dans la base qu'il y a de descripteurs requêtes, puis de combiner ces résultats partiels pour obtenir un résultat final.

Le coût de la recherche de plus proches voisins étant intrinsèquement élevé, cela pose un problème de complexité de la méthode et nécessite l'emploi d'une méthode extrêmement rapide. Nous proposons donc d'utiliser la méthode de recherche approximative de plus proches voisins que nous avons proposée dans la deuxième partie de cette thèse. L'utilisation de la recherche approximative est particulièrement appropriée dans ce schéma de recherche d'images. La méthode de description reposant sur une redondance de résultats partiels, il est possible d'autoriser une légère diminution de la qualité des résultats partiels sans dégrader le résultat final, surtout si cette dégradation permet un fort gain en temps de calcul. Néanmoins, il est important d'étudier de manière précise la relation entre l'imprécision de la recherche de plus proches voisins effectuée individuellement pour chaque vecteur requête et l'imprécision du résultat final. Cela fait l'objet de la section suivante.

Par ailleurs, le fait de disposer de plusieurs requêtes en même temps peut être exploité pour réduire le coût des entrées/sorties (E/S). L'idée est d'éviter de lire une même donnée deux fois si celle-ci est requise pour deux vecteurs requêtes par exemple. Cette possibilité d'optimisation est discutée dans la section 7.4.

7.3 Recherche approximative et descripteurs locaux

Avant d'étudier l'impact de l'imprécision introduite lors des recherches de plus proches voisins (effectuées individuellement pour chaque vecteur requête) sur la qualité du résultat final, nous reprenons d'abord en détails, la procédure de vote qui permet de déterminer et d'ordonner les images semblables à l'image requête.

Nous considérons une image requête I_q décrite à l'aide de n descripteurs $(Q_1, ..., Q_n)$. Pour chacun de ces descripteurs, k voisins ont été retrouvés par l'algorithme de recherche approximative. Le niveau d'imprécision a été fixé à α. Chaque voisin d'un descripteur requête attribue un vote à l'image à laquelle il appartient. À la fin de la recherche, c'est-à-dire une fois que tous les voisins de tous les vecteurs requêtes ont été retrouvés, à chaque image est associé le nombre de votes qu'elle a reçus. Ce nombre reflète le degré de similarité entre l'image retrouvée et l'image requête. Nous rappelons qu'il est noté NDS (Nombre de Descripteurs Semblables). L'ensemble des NDSs est ensuite ordonné et seuls les NDSs significatifs sont gardés après l'application du test de Page-Hinkley. Les images dont les NDSs sont supérieurs au seuil déterminé par le test de Page-Hinkley constituent le résultat final.

La figure 7.1 illustre la procédure de vote pour une image requête décrite à l'aide de quatre descripteurs (Q_1, Q_2, Q_3, Q_4). Pour chaque descripteur requête, 5 plus proches voisins ont été retrouvés. Dans cette figure, à chacun de ces plus proches voisins, est associé l'identifiant de l'image à laquelle il appartient. Comme nous l'avons expliqué précédemment, il suffit de compter le nombre de votes par image pour construire la liste des résultats. La partie droite de la figure présente cette liste pour cet exemple. Dans cet exemple, appliquer le test de Page-Hinkley à la liste des NDSs permet d'isoler les images similaires du reste : les images ayant un NDS $\geqslant 3$ sont considérées comme semblables.

Les recherches de plus proches voisins étant effectuées par un algorithme de recherche approximative, la probabilité de ne pas retrouver l'un des k plus proches voisins est inférieure à α. Ne pas retrouver un des k plus proches voisins d'un vecteur requête modifie les NDSs des images semblables dans le résultat final. En termes plus précis, ne pas retrouver

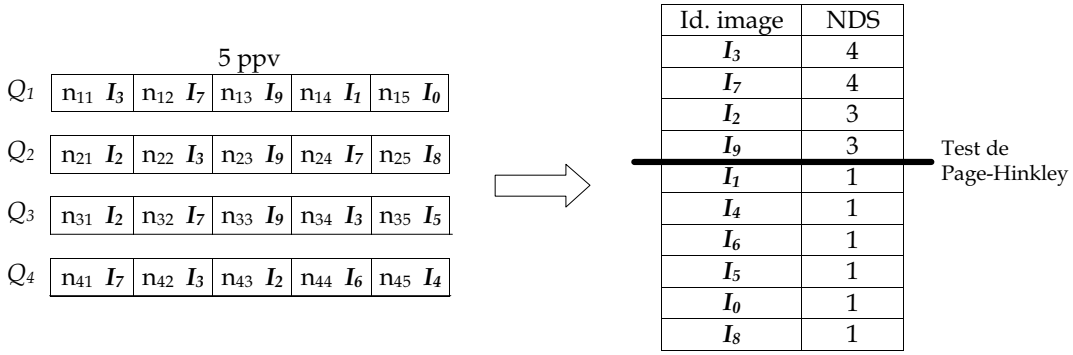


FIG. 7.1: Exemple illustratif de la procédure de vote.

un des k plus proches voisins d'un vecteur requête permet de :

- décrémenter de 1 le NDS de l'image à laquelle appartient le plus proche voisin ignoré ;
- incrémenter de 1 le NDS de l'image décrite par le plus proche voisin approximatif introduit en remplacement du plus proche voisin ignoré.

Cette modification des scores peut entraîner des inversions dans l'ordre des images semblables qui peut nuire plus ou moins gravement à la qualité du résultat final. Si on considère que l'ensemble des images semblables est composé de 10 images, alors l'inversion des deux images les plus semblables ne pose pas problème car toutes les deux restent dans l'ensemble résultat. Par contre, l'inversion de la 10^{e} image avec la 11^{e} qui, elle, ne fait pas partie des images semblables peut entraîner la perte d'une image semblable.

La probabilité d'inverser deux images semblables ordonnées successivement dans la liste du résultat final est liée à la différence entre les nombres de votes qui leurs ont été attribués : plus cette différence est grande et plus la probabilité de les inverser est petite. Pour clarifier ce point, nous considérons deux images I_1 et I_2 semblables à l'image requête, ordonnées successivement dans la liste du résultat final et ayant reçu respectivement NDS_1 et NDS_2 avec une recherche exacte de plus proches voisins. Ces deux images ne peuvent être inversées si la recherche est effectuée par un algorithme de recherche approximative que si :

- au moins δ_1 descripteurs ayant voté pour l'image I_1 avec un algorithme de recherche exacte n'ont pas été retrouvés par la recherche approximative (événement E_1), et,
- au moins δ_2 descripteurs de l'image I_2 ont été introduits par la recherche approximative alors qu'ils ne faisaient pas partie des plus proches voisins lors de la recherche exacte (évènement E_2),
- $\delta_1 + \delta_2 > \text{NDS}\,1 - \text{NDS}_2$.

La probabilité d'inverser les deux images I_1 et I_2 est donc $P(E_1) \cdot P(E_2)$.

Concernant l'évènement E_1, la probabilité de ne pas retrouver δ_1 parmi les k plus proches voisins des descripteurs requêtes est de α^{δ_1} (un plus proche voisin n'a pas été retrouvé pour un sous-ensemble δ_1 de l'ensemble des descripteurs requêtes). Les descripteurs requêtes sont supposés indépendants étant donné qu'ils sont calculés indépendamment sur des points différents de l'image.

La probabilité de E_2 est, cependant, plus difficile à estimer. Les δ_2 descripteurs introduits par la recherche approximative peuvent être divisés en deux sous-ensembles. Un premier sous-ensemble de vecteurs est introduit en remplacement d'une partie des δ_1 plus proches voisins de l'image I_1 qui n'ont pas été retrouvés. L'autre sous-ensemble contient les voisins approximatifs introduits en remplacement de plus proches voisins appartenant à d'autres images. Posons γ le nombre de vecteurs de ce dernier sous-ensemble. $P(E_2)$ est donc égale à α^{γ}. Les plus proches voisins approximatifs appartenant au premier sous-ensemble n'ont pas été pris en compte dans le calcul de $P(E_2)$ étant donné qu'ils ont remplacé une partie des δ_1 plus proches voisins de l'image I_1. Ils ont donc déjà été pris en compte dans le calcul de $P(E_1)$.

Ainsi, la probabilité d'inverser les images I_1 et I_2 dans le résultat final est $\alpha^{\delta_1+\gamma}$. Les valeurs de δ_1 et γ dépendent de la différence entre les NDSs des images I_1 et I_2 : plus cette différence est grande, plus grands sont δ_1 et γ et plus petite est la probabilité d'inverser les images I_1 et I_2 dans le résultat final. Malheureusement, aucune estimation théorique des valeurs de δ_1 et de γ en fonction de α n'est possible. Néanmoins, cette étude a permis de montrer clairement que la probabilité d'inverser deux images dans le résultat final est reliée exponentiellement à α. Par conséquent, elle est beaucoup plus petite que la probabilité de ne pas retrouver un des plus proches voisins d'un vecteur requête. Une petite valeur de α se traduit ainsi par une probabilité presque nulle d'inverser deux images dans le résultat final de la recherche.

Pour illustrer cette analyse, nous présentons un exemple de deux images A et B semblables à une image requête. Nous supposons que ces deux images sont ordonnées successivement dans la liste des images semblables lorsque les recherches de plus proches voisins sont exécutées par un algorithme de recherche exacte. Les scores attribués à ces deux images sont 8 et 4 respectivement. Les descripteurs des images A et B qui ont été retrouvés comme plus proches voisins des descripteurs requêtes sont :

$\mathrm{PPV}_A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\}$ pour l'image A, et,

$\mathrm{PPV}_B = \{b_1, b_2, b_3, b_4\}$ pour l'image B.

Pour que ces deux images soient inversées dans la liste des images semblables si la recherche de plus proches voisins est effectuée par notre algorithme de recherche approximative avec un niveau d'imprécision α, le NDS de A doit augmenter et/ou le NDS de B doit diminuer tel que $\mathrm{NDS}_A < \mathrm{NDS}_B$. Nous supposons que cela a été effectivement le cas lorsque la recherche approximative a été utilisée. À titre d'exemple, les nouveaux NDSs de A et de B sont respectivement 5 et 7, et les ensembles de leurs descripteurs qui ont été retrouvés lors de la recherche de plus proches voisins sont ;

$\mathrm{PPV}'_A = \{a_1, a_2, a_4, a_7, a_8\}$,

$\mathrm{PPV}'_B = \{b_1, b_2, b_3, b_4, b_5, b_6, b_7\}$.

Dans ce cas, la probabilité d'inverser A et B (évènement E) est :

$$P(E) = P(E_1) \cdot P(E_2)$$

où :

$P(E_1)$ est la probabilité de ne pas retrouver les descripteurs a_3, a_5 et a_6 lors de la recherche de plus proches voisins, et,

$P(E_2)$ est la probabilité d'introduire les descripteurs b_5, b_6 et b_7 comme des plus proches voisins lors de la recherche de plus proches voisins.

Comme nous l'avons expliqué précédemment, $P(E_1)$ est facile à calculer. Dans ce cas, il est égale à α^3 car 3 plus proches voisins n'ont pas été retrouvés. Par contre, $P(E_2)$ ne peut être calculé. Sa valeur dépend de la manière dont ont été introduits b_5, b_6 et b_7 dans la liste de plus proches voisins. Si par exemple, b_5 et b_7 ont été introduits en remplacement de descripteurs de l'image A qui n'ont pas été retrouvés (a_3 et a_6), alors $P(E_2)$ est la probabilité de ne pas pas retrouver *un* plus proche voisin d'une image autre que A et d'introduire b_6 à sa place. L'évènement correspondant à l'introduction de b_5 et b_7 ne compte pas dans la calcul de $P(E_2)$ car il est déjà pris en compte dans le calcul de $P(E_1)$. $P(E_2)$ est donc égale à α et $P(E) = \alpha^4$. Si par contre, b_5, b_6 et b_7 ont été introduits en remplacement des plus proches voisins non retrouvés de l'image I_1, alors $P(E_2) = 1$ et $P(E) = P(E_1) = \alpha^3$.

Par ailleurs, plus le nombre de descripteurs de l'image requête est grand, plus faible est l'impact de l'imprécision de la recherche de plus proches voisins sur le résultat final. Lorsque le nombre de descripteurs requêtes est grand, les NDSs des images semblables sont élevés et leur écart par rapport aux NDSs des images non semblables est par conséquent important. Ceci rend l'inversion d'une image semblable et d'une image non semblable dans le résultat final peu probable car la probabilité d'inversion de deux images est inversement proportionnelle à la différence de leurs NDSs (*cf.* analyse ci-dessus).

7.4 Optimisation en utilisant la multiplicité des requêtes

Dans le schéma de recherche par descripteurs locaux, la multiplicité des descripteurs par image rend la procédure de recherche très coûteuse. Autant de recherches de plus proches voisins que de descripteurs dans l'image requête sont nécessaires. Il serait cependant judicieux de réduire le coût global de la recherche en tirant profit de cette multipilicité et ne pas considérer les requêtes de plus proches voisins indépendamment les unes des autres. En particulier, lors de la lecture d'un vecteur de la base pour le comparer à un vecteur requête, si ce vecteur est également pertinent pour d'autres vecteurs requêtes, alors il est préférable de le comparer tout de suite à l'ensemble des vecteurs requêtes pour lesquels il est potentiellement intéressant. Ceci permet d'éviter de le lire plusieurs fois et de réduire ainsi globalement le nombre d'E/S. Cette optimisation est connue dans le domaine des bases de données sous le nom d'optimisation de requêtes multiples. Un état de l'art des travaux proposés dans ce domaine est donné dans [Sellis, 1988]. Dans le cas particulier de la recherche de plus proches voisins, les travaux les plus récents sont ceux proposés par Böhm *et al.* [Böhm *et al.*, 2000] et Braunmüller *et al.* [Braunmüller *et al.*, 2000b].

Dans un algorithme de recherche séquentielle, l'exploitation des requêtes multiples pour réduire le nombre d'E/S est très simple. Il suffit de maintenir une liste de k plus proches voisins pour chaque vecteur requête et de calculer la distance entre chaque vecteur lu et l'ensemble des vecteurs requêtes. Ceci permet de lire la base des descripteurs une seule fois

quel que soit le nombre de descripteurs requêtes. Notons toutefois que le nombre global de calcul de distances reste inchangé. Nous avons implémenté ce mode de recherche et nous l'avons utilisé dans nos expérimentations. Nous l'avons noté « recherche séquentielle optimisée ».

Si l'on veut cependant utiliser la multiplicité des requêtes pour optimiser notre méthode de recherche approximative, l'implémentation n'est pas directe car pour chaque vecteur seule une petite portion de la base des descripteurs est lue. Il faut donc déterminer les *clusters* qui soient pertinents pour plusieurs vecteurs requêtes en même temps. Pour cela, plusieurs heuristiques ont été proposées par Böhm *et al.* [Böhm *et al.*, 2000] et Braunmüller *et al.* [Braunmüller *et al.*, 2000b]. L'idée est de considérer pour chaque vecteur requête, une liste de *clusters* pertinents ordonnés selon leur chance de contenir des plus proches voisins du vecteur requête. L'analyse de l'ensemble des listes de tous les vecteurs requêtes permet de proposer un ordre optimal pour la lecture des *clusters*. Nous présentons ici les trois heuristiques proposées dans [Böhm *et al.*, 2000].

On considère les vecteurs requêtes $Q_1, Q_2, \ldots, Q_n$ et, pour chaque vecteur requête, une liste des *clusters* candidats (LCC_i) ordonnée en fonction de leurs distances minimales au vecteur requête (Q_i).

Technique de la priorité moyenne (`PrioMoy`). Cette première technique consiste à proposer un ordre de lecture des *clusters* en utilisant le rang des *clusters* dans les listes des *clusters* candidats. Le premier *cluster* à lire est celui dont le rang moyen dans toutes les listes où il apparait est minimum. Une fois ce *cluster* lu, il est supprimé des listes des *clusters* candidats et le deuxième *cluster* à lire est déterminé par la même méthode.

Technique de la distance moyenne (`DistMoy`). Cette technique est similaire à la première sauf qu'au lieu de considérer le rang des *clusters* dans les listes des *clusters* candidats, ce sont les distances minimales qui sont utilisées. Ainsi, le premier *cluster* à lire est celui dont la moyenne des distances minimales par rapport aux vecteurs requêtes est la plus petite.

Technique de la priorité maximale (`PrioMax`). Cette technique permet de choisir le *cluster* à lire uniquement parmi les *clusters* apparaissant en tête de liste. Celui qui apparaît le plus souvent en tête de liste est lu en premier.

L'efficacité de ces techniques dépend de leur capacité à réduire le coût des E/S. Cette réduction doit être très importante par rapport au surcoût induit par le temps de calcul propre à ces techniques, c'est-à-dire le temps nécessaire à l'ordonnancement des *clusters*. Les résultats expérimentaux présentés dans [Böhm *et al.*, 2000] montrent que la technique `PrioMax` est la plus efficace. Nous l'avons donc retenue. Son efficacité ainsi que celle de la recherche séquentielle optimisée sont évaluées dans le section 8.5.1 du chapitre suivant.

7.5 Synthèse

Dans ce chapitre, nous avons présenté un système de recherche d'images par le contenu pour la détection de copies. Ce schéma utilise les descripteurs locaux différentiels pour assurer une description robuste des images. Chaque image est décrite par plusieurs descripteurs

possèdant des propriétés d'invariance très intéressantes. Ces propriétés ainsi que la multiplicité des descripteurs par image permettent de mettre en correspondance les images piratées avec les images originales même si les images piratées ont subi certaines modifications. La procédure de recherche associée à ces descripteurs est cependant complexe et extrêmement coûteuse. La taille de la base est multipliée typiquement par un facteur 200 par rapport à un schéma de description globale et chaque requête se traduit par de nombreuses interrogations de plus proches voisins, autant qu'il y a de descripteurs dans l'image requête. Une fois que tous les plus proches voisins sont retrouvés, une fusion des résultats est effectuée. Cette fusion permet de construire une liste d'images ordonnées selon un score de similarité. Cette liste est souvent très grande mais seules quelques images en tête de liste reçoivent des scores élevés et sont considérées comme semblables à l'image requête. La première amélioration que nous avons introduite est l'automatisation de l'identification de l'ensemble des images semblables. Nous avons assimilé le problème à un problème de détection de changement dans un signal et nous avons utilisé le test de Page-Hinkley. Nous avons également utilisé notre méthode pour la recherche de plus proches voisins pour accélérer les recherches. Cette méthode est particulièrement adaptée à ce schéma de recherche car la redondance dans la description des images rend ce schéma peu sensible à l'imprécision de la recherche de plus proches voisins. Nous avons étudié ce point de manière approfondie et nous avons montré que la relation entre l'imprécision de la recherche de plus proches voisins et l'imprécision du résultat final est exponentielle. Plus précisément, lorsque la probabilité de ne pas retrouver un des plus proches voisins exacts est inférieure à α, la probabilité de ne pas retrouver une image similaire est inférieure à α^{β} où β est supérieur à 1 et est d'autant plus grand que l'écart entre les scores des images semblables et les scores des autres images est grand. Dans le chapitre suivant, nous reviendrons sur ce point et nous présenterons les évaluations expérimentales permettant d'appuyer cette étude.

Par ailleurs, nous avons présenté des techniques existantes pour l'optimisation de requêtes multiples. Ces techniques permettent de réduire le coût des E/S en profitant de la présence de plusieurs vecteurs requêtes en même temps. L'efficacité de ces techniques est également évaluée dans le chapitre suivant.

Chapitre 8

Évaluation des performances

Dans ce chapitre, nous présentons une étude de l'évaluation des performances du système de détection de copies d'image proposé dans le chapitre précédent. Nous évaluons sa capacité de reconnaissance et sa robustesse face aux différentes modifications que peuvent subir les images. Nous évaluons également l'efficacité de la recherche approximative et nous étudions l'impact de l'imprécision de la recherche de plus proches voisins sur le résultat final. Comme dans le chapitre 6, pour étudier l'efficacité de la recherche approximative, nous comparons ses performances à celles de la recherche séquentielle.

8.1 Introduction

L'évaluation d'un système de recherche d'images par le contenu est essentielle pour sa validation et son intégration dans une application réelle. Plusieurs travaux se sont intéressés à cette tâche [Dimai, 1999, Müller *et al.*, 2001a, Müller *et al.*, 2001b, Jr. *et al.*, 2002, Vogel & Schiele, 2002]. De manière générale, l'évaluation d'un système de recherche d'images par le contenu (SRIC) concerne à la fois sa capacité de reconnaissance et son efficacité (i.e. sa rapidité). Cette dernière est facile à évaluer. Il suffit de mesurer le temps de réponse moyen du système sur plusieurs requêtes. La reconnaissance est cependant plus difficile à évaluer. Elle requiert la mise au point d'un protocole d'évaluation qui prend en compte les spécificités et les contraintes de l'application considérée.

De manière générale, l'évaluation des capacités de reconnaissance d'un SRIC est effectuée en interrogeant la base avec des images requêtes, et en comparant le résultat obtenu à un résultat de référence. Celui-ci est appelé la vérité terrain. Pour une image requête I_q, il est constitué des images que l'on considère similaires à l'image I_q et que le SRIC est censé retourner. Ici, la notion de similarité est relative et dépend de l'application considérée. Cette façon d'évaluer un SRIC nécessite, d'une part, la création d'une vérité terrain pour un ensemble d'images requêtes et, d'autre part, la définition de métriques permettant de mesurer l'écart entre les résultats retournés et les vérités terrains associées.

Le choix des images requêtes et l'établissement de leurs vérités terrains sont effectués en fonction des objectifs de l'application considérée. Les vérités terrains peuvent être créées soit de façon artificielle en faisant appel aux techniques de synthèse ou de traitement d'images, ou bien construit directement à partir des images de la base [Wenyin *et al.*, 2001]. Dans le premier cas, l'idée est de choisir une image puis de lui appliquer différents types de

transformations possibles (géométriques, photométriques, ajout de bruit,...). Les images transformées constituent alors la vérité terrain de l'image requête. Dans le deuxième cas, des images requêtes sont choisies au hasard à partir de la base d'images. Pour chacune d'entre elles, les images similaires appartenant à la base sont identifiées manuellement. Le très grand nombre des interprétations possibles que l'on peut prêter au contenu d'une même image constitue la principale difficulté de cette dernière méthode.

Les métriques d'évaluation généralement utilisées sont issues du domaine de la recherche d'information [Salton, 1988, Rijsbergen, 1979]. Il s'agit principalement du rappel et de la précision.

Les difficultés de l'évaluation des SRICs concernent principalement l'établissement de la vérité terrain et l'absence de base d'images de référence qui permettent de comparer deux systèmes différents. Néanmoins, lorsque l'application considérée est bien définie, l'évaluation du SRIC est relativement plus facile à effectuer.

Dans le contexte de la détection de copies, l'objectif du SRIC est de retrouver l'image originale de la base à partir de laquelle l'image requête aurait été copiée. Comme il est fort probable que l'image copiée soit modifiée, l'évaluation du système doit se faire par rapport à l'ensemble des modifications qui peuvent être appliquées à l'image piratée. Le choix des images requêtes et la construction de leurs vérités terrains sont donc simples à établir. Il suffit de sélectionner un ensemble d'images à partir de la base, d'en créer des copies et d'appliquer à ces copies différents types de transformations possibles. Les images requêtes sont alors les images copiées puis transformées, et la vérité terrain de chacune est constituée de l'image de la base à partir de laquelle la copie transformée a été générée.

Dans ce chapitre, nous évaluons les performances du SRIC proposé pour la détection de copies. Nous utilisons pour cela deux protocoles d'évaluation. Le premier repose sur l'utilisation de séquences d'images générées artificiellement pour étudier la robustesse des descripteurs face à certains types de changement. Le deuxième étudie plus particulièrement la robustesse des descripteurs dans le contexte de la détection de copies en considérant un ensemble de 79 attaques possibles.

8.2 Méthodologie d'évaluation

L'évaluation des performances du système proposé pour la détection de copies d'images concerne les trois points suivants :

- l'efficacité de la recherche, c'est-à-dire sa rapidité ;
- la robustesse du schéma de description utilisé, c'est-à-dire sa capacité à reconnaître une image copiée même si celle-ci a subi différentes transformations ;
- l'impact de l'imprécision de la méthode de recherche approximative de plus proches voisins utilisée sur la qualité du résultat final.

Le premier et le troisième point sont faciles à évaluer. Pour le premier, il suffit de comptabiliser le temps de la recherche et de la comparer à celui obtenu par une recherche séquentielle. Pour le troisième, il suffit d'effectuer les mêmes requêtes à la fois par notre méthode de recherche approximative et par une méthode de recherche exacte (la recherche séquentielle par exemple). La comparaison des résultats finaux en termes de reconnaissance permet d'évaluer l'impact de l'imprécision de la recherche de plus proches voisins sur le résultat final.

Par contre, pour évaluer la robustesse du schéma de description, nous avons défini deux protocoles d'évaluation qui nous ont permis de réaliser deux campagnes d'évaluation.

1re campagne : utilisation de la base MOVI. Dans cette campagne, nous avons utilisé les 32 séquences de la base MOVI[1]. Chaque séquence représente la même scène (2 ou 3D) prise dans des conditions différentes. Chaque séquence est utilisée pour étudier un changement particulier. Parmi ces séquences, on retrouve par exemple des images représentant la même scène prise dans des conditions d'illumination différentes (changement de l'intensité lumineuse, variation de la composition ou de la position de la source lumineuse, etc.). D'autres séquences ont été générées en modifiant la composition de la scène ou en déplaçant la caméra autour de la scène selon des mouvements plus ou moins complexes. Une description détaillée de ces séquences est donnée dans [Amsaleg & Gros, 2001]. Dans la figure 8.1, les 16 images composant une séquence de la base MOVI sont présentées. Ces images représentent toutes la même scène 2D (une photo de l'intérieur d'une cathédrale) prise sous différentes conditions d'illumination (changement de l'intensité lumineuse).

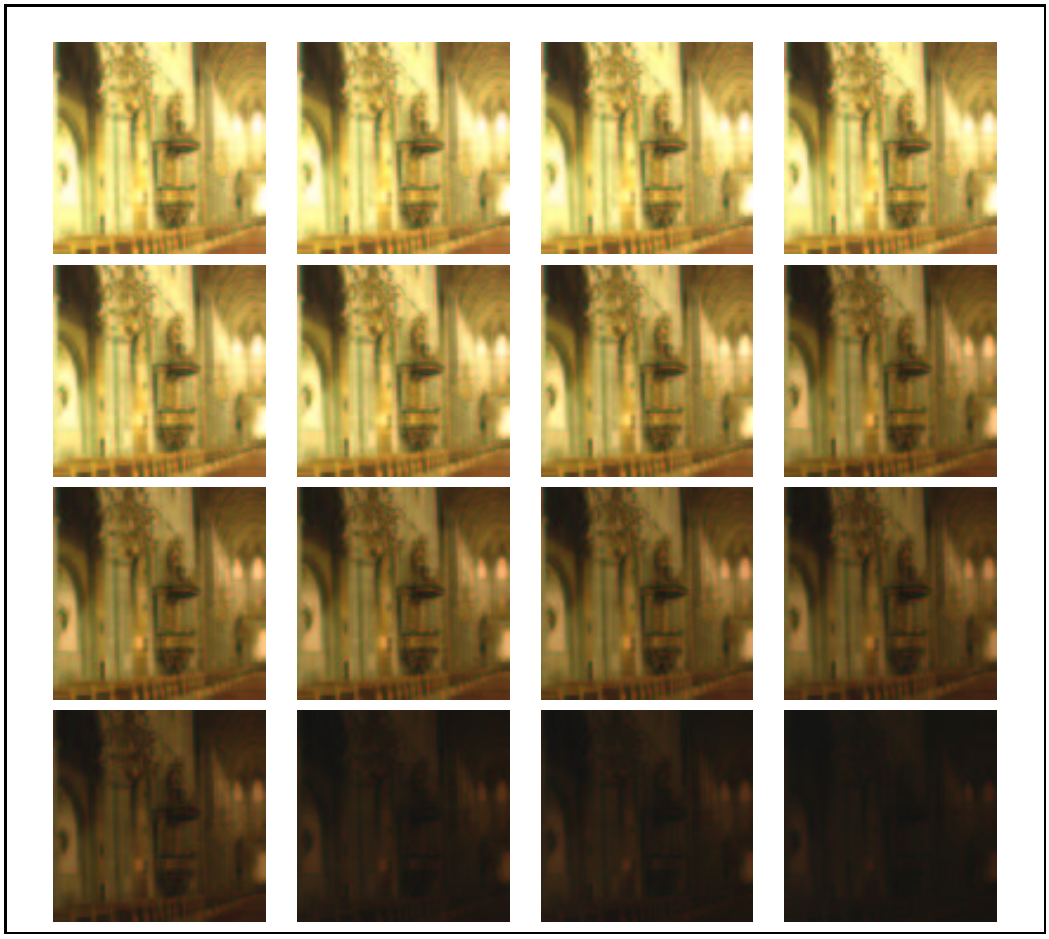


FIG. 8.1: Exemple d'une séquence de la base MOVI.

[1]Disponible en suivant l'URL : `http://www.irisa.fr/texmex/base_images/index.html`

Le protocole d'évaluation associé à cette première campagne procède de la manière suivante. Il choisit une image à partir de chaque séquence et l'utilise pour interroger la base. Dans notre cas, la vérité terrain pour une image requête est parfaitement définie et est constituée de l'ensemble des images qui appartiennent à la même séquence que l'image requête. À la différence de la plupart des SRICs qui permettent seulement d'ordonner les images de la base en fonction de leurs degrés de similarité à l'image requête, l'ensemble résultat retourné par notre système est également parfaitement établi, et ce grâce à l'utilisation du test de Page-Hinkley. Ainsi, il est tout à fait possible de calculer l'écart entre l'ensemble résultat et la vérité terrain en utilisant des métriques traditionnelles telles que le rappel et la précision. Nous rappelons ici leurs définitions.

Définition 6 (Précision et rappel) *Soit q un document requête, $R(q)$ l'ensemble des documents pertinents par rapport à q (sa vérité terrain) et $A(q)$ l'ensemble des documents retournés en résultat par un système de recherche. La précision du résultat $A(q)$ par rapport à la vérité terrain $R(q)$ est donnée par :*

$$p = \frac{|A(q) \cap R(q)|}{|R(q)|}, \tag{8.1}$$

et son rappel par :

$$r = \frac{|A(q) \cap R(q)|}{|A(q)|}, \tag{8.2}$$

où $|E|$ désigne la cardinalité de l'ensemble E.

La précision est la proportion des images pertinentes retournées par rapport à l'ensemble des résultats retournés. Le rappel est quant à lui la proportion des images pertinentes retournés par rapport à l'ensemble des images pertinentes stockées dans la base.

Nous avons également utilisé la F-Mesure (*cf.* équation 8.3) qui permet de combiner la précision et le rappel en une seule mesure [Rijsbergen, 1979].

$$\text{F-Mesure} = \frac{2\,r\,p}{r + p} \tag{8.3}$$

Notons enfin que cette première campagne permet d'évaluer la capacité de reconnaissance des descripteurs de manière générale. Il ne s'agit pas d'une évaluation spécifique par rapport à leur utilisation dans le cas particulier de la détection de copies.

2^{e} campagne : robustesse face aux transformations générées par StirMark. Dans cette deuxième campagne, nous nous sommes plus particulièrement intéressés à la robustesse du système face aux transformations que peuvent subir les images dans le cadre de la détection de copies. Dans ce contexte, il s'agit d'évaluer la capacité du schéma de description à retrouver l'image originale de la base, à partir d'une copie.

La robustesse du schéma de description face à un type de modification particulier se mesure donc par le taux des images copiées qui ont subi cette modification et qui ont été mises en correspondance correctement avec leur image originale. Ce taux est appelé dans la suite, taux de reconnaissance. Pour évaluer notre système, il faut donc reproduire le plus

grand nombre possible de transformations pouvant être appliquées aux images piratées et calculer le taux de reconnaissance pour chaque type de transformation.

Pour cela, nous avons utilisé le logiciel StirMark[2] pour simuler 79 transformations possibles. Ce logiciel a été mis au point pour évaluer la robustesse des méthodes de tatouages d'images. Il permet de générer à partir d'une image, 79 copies, chacune ayant subi un changement particulier. Nous avons considéré l'ensemble de ces transformations générées par StirMark bien que certaines transformations soient spécifiques aux méthodes de tatouages d'images, par exemple, la suppression de lignes et de colonnes de l'image. Cette transformation permet de désynchroniser le signal. Le but est de rendre les possibles marques introduites par un algorithme de tatouages difficile à retrouver.

Les 79 changements considérés sont :

Rot(i) : rotation simple d'angle i. $i \in$ {-2°, -1°, -0,75°, -0,5°, -0,25°, 0,25°, 0,5°, 0,75°, 1°, 2°, 10°, 15°, 30°, 45°, 90°} ;

RotReCE(i) : rotation d'angle i suivie d'un recadrage et d'un changement d'échelle pour sauvegarder la taille originale de l'image. $i \in$ {-2°, -1°, -0,75°, -0,5°, -0,25°, 0,25°, 0,5°, 0,75°, 1°, 2°, 10°, 15°, 30°, 45°, 90°} ;

ChangEch(f) : changements d'échelle d'un facteur f. $f \in \{0{,}5,\ 0{,}75,\ 0{,}9,\ 1{,}1,\ 1{,}5,\ 2\}$;

FilMed : filtrage médian : $2 \times 2, 3 \times 3$ et 4×4 ;

FilGauss : filtrage gaussien : $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$;

Sharp : renforcement de contraste *(sharpening)* : $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$;

RecadCent(p) : recadrage centré de paramètre p. Ce paramètre est le facteur de réduction de la largeur et de la longueur de l'image originale. $100-p$ est donc le pourcentage que représente de la largeur (resp. la longueur) de la sous-image par rapport à celle de l'image originale. $p \in$ { 1 %, 2 %, 5 %, 10 %, 15 %, 20 %, 25 %, 50 %, 75 %} ;

CompJPEG(f) : compression JPEG avec un facteur de qualité f. $f \in$ { 90, 80, 70, 60, 50, 40, 35, 30, 25, 20, 15, 10} ;

Shear(x, y) : cisaillement *(shearing)*. x est le décalage selon la direction X de l'image (% de sa largeur). y est le décalage selon la direction Y (% de sa longueur). $(x, y) \in$ {(0,1), (0,5), (1,0), (5,0), (1,1), (5,5)} ;

SuppLigCol(x, y) : suppression de x lignes et de y colonnes de l'images. $(x, y) \in$ {(1,1), (1,5), (5,1), (5,17), (17,5)} ;

TransGeomLin(a, b, c, d) : transformation géométrique linéaire : $\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} \begin{vmatrix} x \\ y \end{vmatrix}$. $(a, b, c, d) \in$ {(0,010, 0,013, 0,009, 1,011), (1,007, 0,010, 0,010, 1,012), (1,013, 0,008, 0,011, 1,008)} ;

DistorGeomAlea : distortion géométrique aléatoire.

[2] Page web : `http://www.petitcolas.net/fabien/watermarking/stirmark/`

8.3 Contexte des expérimentations

Étant donné que les expériences effectuées dans ce chapitre s'intéressent également à la capacité de reconnaissance des descripteurs, il est important d'utiliser une base images contenant uniquement des images fixes. Les images issues de séquences de vidéo sont fortement similaires et risquent de biaiser les expérimentations.

Pour cela, nous avons utilisé la plus grande base d'images fixes dont nous disposons et qui contient 8934 images[3]. Nous y avons rajouté la base MOVI, celle-ci étant nécessaire pour effectuer la première campagne d'évaluation. Ainsi, la base résultante contient 9544 images. Ces images ont été décrites à l'aide de 2 623 824 invariants différentiels locaux (descripteurs de dimension 24). En moyenne, chaque image a été décrite par 275 descripteurs.

Tous les algorithmes de recherche ont été implémentés en C++. Ils ont été exécutés sur un PC sous Linux disposant d'un processeur Intel Pentium 4 cadencé à 2,4 GHz et de 1 giga octets de mémoire centrale. Les temps d'exécution des algorithmes de recherche reportés dans la suite ont été obtenus grâce à la fonction `getrusage()`. Notons toutefois que ces temps de réponse ne sont pas directement comparables à ceux présentés dans le chapitre 6 car le contexte matériel est différent. Les expérimentations présentées dans le chapitre 6 ont été réalisées sur une station de travail Sun Blade 100 sous SunOS 5.7.

8.4 Plan des expérimentations

Avant de présenter les évaluations concernant la capacité de reconnaissance de notre système, nous commençons par l'étude de l'amélioration des temps de réponse par les heuristiques d'optimisation relatives à la multiplicité des requêtes (*cf.* section 7.4, page 172). L'objectif est d'avoir une idée sur l'efficacité de ces heuristiques indépendamment de la capacité de reconnaissance des descripteurs.

La suite des expériences est consacrée à l'évaluation de la robustesse des descripteurs ainsi que l'étude de l'impact de l'imprécision des plus proches voisins sur le résultat final. Elle est divisée en trois parties. La première correspond à la première campagne d'évaluation qui utilise la base MOVI ; la deuxième étudie la robustesse du système proposé face à toutes les attaques simulées par le logiciel StirMark ; enfin, dans la troisième partie, l'étude concernant la robustesse face au recadrage est approfondie, le recadrage étant une attaque très fréquente et très importante dans ce contexte.

Dans toutes les expériences qui suivent, nous avons fixé le nombre de plus proches voisins recherchés pour chaque descripteur requête à 30. Ce paramètre dépend de la taille de la base d'images et de la variabilité des descripteurs. De manière générale, il n'existe pas de critère pour fixer automatiquement ce paramètre. Le choix de la valeur de 30 a été effectué suite à l'étude expérimentale dans laquelle nous avons étudié l'impact de ce paramètre sur la qualité du résultat final. Cette étude est présentée à la fin de la section 8.5.2.

[3]Nous tenons à souligner ici la difficulté d'avoir de grandes bases d'images fixes à cause des problèmes de *copyright*.

8.5 Résultats expérimentaux

8.5.1 Expérience 1 : la multiplicité des requêtes

Dans cette première expérience, nous étudions la réduction du temps de réponse apportée grâce à l'optimisation par requêtes multiples. Cette optimisation a été discutée dans la section 7.4. Nous avons préféré commencer par cette expérience afin de sélectionner les stratégies de recherche qui seront utilisées dans la suite des expériences.

Nous rappelons que l'optimisation par requêtes multiples s'appuie sur des heuristiques afin de réduire le nombre d'E/S effectuées lors de la recherche. Elle exploite la présence de plusieurs requêtes simultanément pour éviter de lire une même donnée plusieurs fois si celle-ci est pertinente pour plusieurs requêtes. Elle s'applique aussi bien à la recherche séquentielle qu'à notre méthode de recherche approximative. Nous évaluons donc la réduction du temps de recherche apportée par ces heuristiques dans le cas de la recherche séquentielle et de notre méthode.

Recherche séquentielle. L'algorithme de la recherche séquentielle qui utilise la multiplicité des requêtes a été présenté dans la section 7.4. Nous l'avons noté « recherche séquentielle optimisée ». Dans cet algorithme, quel que soit le nombre de descripteurs requêtes, la base est lue une seule fois. Cette optimisation couplée avec le principe de la distance partielle permet donc d'avoir quatre variantes de l'algorithme de recherche séquentielle :

RechSeq : la recherche séquentielle sans distance partielle et sans optimisation par requêtes multiples ;

RechSeqAvecDP : la recherche séquentielle avec distance partielle mais sans optimisation par requêtes multiples ;

RechSeqOpt : la recherche séquentielle optimisée sans distance mais avec optimisation par requêtes multiples ;

RechSeqOptAvecDP : la recherche séquentielle optimisée avec distance partielle et avec optimisation par requêtes multiples.

Pour évaluer les performances de ces quatre algorithmes, nous avons tiré aléatoirement 20 images de la base. Pour chacune, nous avons recherché les images similaires en utilisant les quatre stratégies de recherche. Nous avons utilisé l'ensemble de ces quatre stratégies afin de pouvoir distinguer la réduction du temps de réponse apportée par chaque optimisation.

Les temps de réponse moyen pour les 20 requêtes et pour les quatre stratégies de recherche utilisées sont résumés dans le tableau suivant :

	RechSeq	RechSeqAvecDP	RechSeqOpt	RechSeqOptAvecDP
Temps de réponses (s)	192,14	158,63	89,31	62,42

Ces temps de réponse montrent que l'optimisation par requête multiple permet de réduire le temps de réponse. Dans le cas des algorithmes RechSeqOpt et RechSeqOptAvecDP, la base a été lue une seule fois pour chaque image requête, alors qu'elle été lue autant de fois qu'il y a de descripteurs dans chaque image requête pour les algorithmes RechSeq et RechSeqAvecDP, c'est-à-dire en moyenne 448 fois par image. Le gain en nombre d'E/S est donc de $1 - \frac{1}{448}$ ($\approx 99\,\%$). Notons cependant que le gain en temps de réponse n'est pas proportionnel au gain en termes d'E/S. Ceci est dû au fait que le coût de la recherche de

plus proches voisins a une forte composante CPU, c'est-à-dire que le temps de recherche consacré à des calculs est important (calculs de distance).

Par ailleurs, nous remarquons que le gain en terme de temps de réponse varie selon que la distance partielle est utilisée ou non. Dans le cas où l'option distance partielle n'est pas utilisée, il est de 53,52 % (le temps de recherche moyen passe de 192,14 à 89,31 secondes). Il est cependant de 60,65 % dans le cas contraire. Cette différence de gain est due à la répartition du coût de la recherche entre le CPU et les E/S. De manière générale, le temps de recherche est dominé par les calculs de distances. Lorsque la distance partielle est utilisée, le coût de ces calculs est réduit. La part des E/S dans le temps global de la recherche est donc augmentée. Ainsi, en utilisant l'optimisation par requêtes multiples qui, rappelons-le, ne permet de réduire que les E/S, la réduction du temps de réponse est plus forte lorsque la distance partielle est utilisée.

Recherche approximative. Dans le cas de notre méthode de recherche, l'implémentation de l'optimisation par requête multiple n'est pas aussi simple que pour la recherche séquentielle. Il est nécessaire de maintenir une liste de *clusters* candidats pour chaque descripteur requête, et de trouver un ordre optimal pour la lecture des *clusters*. Nous avons présenté dans la section 7.4, page 172, trois heuristiques qui ont été proposées par [Böhm *et al.*, 2000] pour l'ordonnancement des *clusters*. Nous avons implémenté seulement la technique de la priorité maximale (`PrioMax`). Dans les expérimentations présentées dans [Böhm *et al.*, 2000], cette technique est nettement plus performante que les deux autres.

Pour évaluer les performances de cette technique, nous avons repris les 20 images requêtes précédemment utilisées et nous avons effectué des recherches en utilisant notre méthode de recherche approximative avec et sans cette optimisation. Nous avons effectué ces recherches pour 5 valeurs différentes de α (0, 0,01, 0,10, 0,20, 0,40). Les résultats obtenus sont résumés dans le tableau 8.1. Ce tableau montre que la technique `PrioMax` détériore le

α	Temps de réponse (s)	
	sans `PrioMax`	avec `PrioMax`
0	70,33	421,91
0,01	26,60	59,86
0,10	18,62	29,52
0,20	16,88	22,27
0,40	15,35	16,75

TAB. 8.1: Temps de réponse moyen de la recherche. 20 images requêtes – Notre méthode de recherche approximative de plus proches voisins avec et sans la technique `PrioMax`. Résultats pour 5 valeurs différentes de α.

temps de réponse. Dans ce cas, le coût propre à la fonction qui détermine le prochain *cluster* à lire est plus important que le gain en termes d'E/S. À titre d'exemple, pour une image requête contenant 409 descripteurs, le temps de réponse de notre méthode sans `PrioMax` est de 26,4 secondes lorsque $\alpha = 0,01$. Globalement, lors de l'exécution des 409 requêtes de 30 plus proches voisins, 106 430 868 vecteurs ont été lus. Avec la fonction `PrioMax`, ce nombre est seulement de 37 074 676 vecteurs (soit une réduction de 65,17 %) alors que le temps de

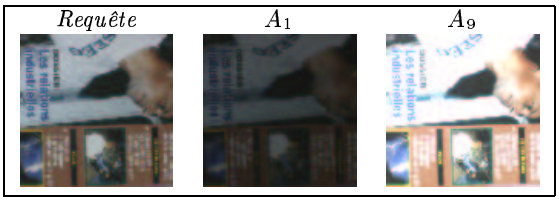
réponse est passé à 57,66 secondes (soit une augmentation de 118,41 %). À la lumière de ces résultats, cette technique n'a donc pas été retenue pour le reste des expériences.

Notons par ailleurs que lorsque $\alpha = 0$, c'est-à-dire lorsque la recherche est exacte, le temps de réponse de notre méthode est supérieur à celui de la recherche séquentielle optimisée avec distance partielle. Dans ce contexte, celle-ci devient donc une méthode extrêmement performante.

Synthèse. En résumé, l'optimisation basée sur les requête multiples permet de réduire le nombre d'E/S effectuées lors de la recherche. Elle apporte un gain significatif à la recherche séquentielle. Elle détériore cependant les performances de notre méthode de recherche. Par conséquent, nous reportons dans les expériences suivantes uniquement les temps de réponse des quatre stratégies de recherche séquentielle et de notre méthode de recherche approximative pour des valeurs de α non nulles.

8.5.2 Expérience 2 : évaluation de la robustesse en utilisant la base MOVI

Cette expérience correspond à la première campagne d'évaluation décrite dans la section 8.2. Son but premier est d'étudier la robustesse des descripteurs locaux dans un cadre plus général que celui de la détection de copies. Elle permet également d'analyser l'impact de l'imprécision des plus proches voisins sur le résultat final et d'étudier les temps de réponse des différentes stratégies de recherche utilisées.



TAB. 8.2: Exemple d'une image requête et deux de ses images semblables.

Algo. de recherche		Temps de réponse (s)	NDS								# Faux nég.	# Faux pos.
			A_5	A_3	A_6	A_2	A_1	A_7	A_8	A_9		
RechSeq		245,83	613	517	482	361	218	216	103	27	0	0
RechSeqAvecDP		196,18	613	517	482	361	218	216	103	27	0	0
RechSeqOpt		119,64	613	517	482	361	218	216	103	27	0	0
RechSeqOptAvecDP		82,2	613	517	482	361	218	216	103	27	0	0
α	0,01	36,39	617	515	481	361	220	211	102	26	0	6
	0,10	24,64	595	495	455	349	207	201	94	24	0	3
	0,20	22,08	542	433	401	312	182	180	83	22	0	5
	0,40	20,07	381	315	292	251	142	132	64	16	0	3

TAB. 8.3: Résultats de la recherche : temps de réponse et NDSs des images semblables retrouvées pour différentes stratégies de recherche.

Afin d'illustrer le protocole d'évaluation utilisé dans cette première campagne, nous présentons tout d'abord une expérience d'interrogation à l'aide d'une seule image requête.

Cette image est tirée d'une séquence représentant la même scène prise sous différentes intensités lumineuses (*cf.* tableau 8.2). Cette image a été recherchée selon la procédure décrite dans la section 7.2.2, page 166. La recherche de plus proches voisins a été exécutée par les quatre versions de la recherche séquentielle de base et par l'algorithme de recherche approximative avec différents degrés d'imprécision (α = 0,01, 0,10, 0,20, 0,40).

Les résultats obtenus sont résumés dans le tableau 8.3. Ils montrent la capacité des invariants locaux différentiels utilisés à absorber ce type de variation. En effet, toutes les images appartenant à la même séquence que l'image requête ont été retrouvées. Ils montrent aussi que la recherche approximative n'a qu'un impact mineur sur la qualité du résultat final alors qu'elle permet de réduire considérablement le temps de réponse. Un facteur d'accélération de 6,75 (resp. 12,25) a été observé par rapport à la recherche séquentielle de base (RechSeq) lorsque $\alpha = 0,01$ (resp. 0,40). Notre méthode reste également plus performante que la meilleure version de la recherche sequentielle (RechSeqOptAvecDP). Son facteur d'accélération par rapport à RechSeqOptAvecDP est de 2,26 (resp. 4,10) lorsque $\alpha = 0,01$ (resp. 0,40) tout en retournant un résultat d'un excellent rappel (0 faux négatifs). Quelques faux positifs ont été introduits. Ils ont été cependant ordonnés après les images pertinentes dans la liste des images semblables. Dans ce cas, la recherche approximative a relevé les scores de certaines images non pertinentes. Le test de Page-Hinkley les a donc considérées comme des scores élevés. À titre d'exemple, lorsque $\alpha = 0,10$, la liste des scores ordonnés est : $595, 495, 455, 349, 207, 201, 94, 24, 15, 14, 13, 12, 12, 12, 12, 11, \ldots, 11, 10, \ldots$. Le test de Page-Hinkley a gardé, en plus des images de la séquence, les images qui ont reçu les scores 15, 14 et 13. Comme nous l'avons expliqué dans la section 7.2.2, page 167, ceci est dû au fait que le test de Page-Hinkley permet plutôt de détecter les changements dans la tendance générale de la liste des scores et ne se base pas sur les écarts entre ceux-ci. De manière générale, ceci a pour effet de favoriser le rappel au détriment de la précision.

Algo. de recherche		Tps de rep. moy. (s)	% moy. de reduc. du tps de rep. moy. par rap. à RechSeq	Precision moy.	Rappel moy.	F-Mesure moy.
RechSeq		172,44	-	0,65	0,93	0,77
RechSeqAvecDP		141,60	17,88	0,65	0,93	0,77
RechSeqOpt		80,34	53,41	0,65	0,93	0,77
RechSeqOptAvecDP		55,14	68,02	0,65	0,93	0,77
α	0,01	23,79	86,20	0,64	0,92	0,75
	0,10	16,61	90,37	0,65	0,91	0,75
	0,20	15,03	91,13	0,65	0,90	0,75
	0,40	13,69	92,06	0,65	0,89	0,75

TAB. 8.4: 32 images requêtes : précision moyenne, rappel moyen et accélération moyenne pour différentes stratégies de recherche.

Nous avons ensuite effectué cette même expérience pour l'ensemble des 31 séquences restantes de la base MOVI afin de tester la robustesse du système par rapport à l'ensemble des variations. Les moyennes des résultats obtenus sont résumées dans le tableau 8.4. Les taux moyens de rappel et de précision sont tous les deux élevés, ce qui montre bien la grande capacité de ces descripteurs à reconnaître même les images ayant subi des transformations géométriques et photométriques complexes. En moyenne, 92% des images semblables aux images requêtes sont toujours retrouvées. Elles représentent 65% de l'ensemble des images

retournées. Nous avons également noté que pour 23 images requêtes sur l'ensemble des 32, toutes les images semblables ont été retrouvées (rappel = 1).

Ces résultats ont permis aussi de corroborer l'étude présentée dans la section 7.3, qui étudie l'impact de l'imprécision de la recherche de plus proches voisins sur le résultat final. La recherche approximative a permis de réduire considérablement le temps de recherche en sauvegardant la qualité du résultat final par rapport à la recherche séquentielle. Des réductions du temps de recherche variant entre 85 et 91% ont été réalisées par rapport à la recherche séquentielle de base tout en gardant à peu près le même taux de reconnaissance que la recherche séquentielle (une légère diminution de la F-Mesure, 0,75 contre 0,77 pour la recherche exacte).

Choix du nombre optimal de plus proches voisins. Dans toutes les expériences présentées ici, nous avons fixé le nombre k de plus proches voisins recherchés pour chaque descripteur requête à 30. Cette valeur a été choisie expérimentalement.

Intuitivement, une petite valeur de k permet de retrouver uniquement les descripteurs très proches des descripteurs requêtes. Dans ce cas, seuls les descripteurs des images très similaires à l'image requête sont retrouvés. Dans le résultat final, on retrouve ainsi uniquement les images très similaires à l'image requête. Par conséquent, une très bonne précision est obtenue mais le rappel risque d'être mauvais. Par contre, lorsqu'une grande valeur est affectée à k, l'ensemble des plus proches voisins d'un descripteur requête contient des voisins très proches qui correspondent aux descripteurs des images similaires à l'image requête, mais aussi, des voisins « éloignés ». Ceux-ci appartiennent à des images qui ne sont pas similaires à l'image requête. Ils voteront toutefois pour les images auxquelles ils appartiennent. Dans le résultat final, on retrouve donc les images similaires à l'image requête mais probablement aussi d'autres images non similaires. Par conséquent, un très bon rappel est obtenu mais la précision risque d'être plus mauvaise.

Pour vérifier cette analyse, nous avons refait l'expérience de l'étude de la robustesse des descripteurs en utilisant les séquences MOVI mais en variant le nombre de plus proches voisins recherchés pour chaque descripteur requête. Nous avons utilisé uniquement la recherche séquentielle car le but de cette expérience n'est pas d'étudier les temps de réponse ni la précision de la recherche de plus proches voisins.

Nous avons considéré 7 valeurs différentes de k: 10, 20, 30, 40, 50, 75 et 100. Les résultats obtenus sont résumés dans tableau 8.5. Ces résultat confirme bien notre analyse. En augmentant k, la précision diminue alors que le rappel augmente. Un bon compromis est obtenu lorsque $k = 30$.

8.5.3 Expérience 3 : évaluation de la robustesse face aux attaques générées par StirMark

Dans cette section, nous résumons les résultats de la deuxième campagne d'évaluation. Cette campagne est décrite dans la section 8.2. Nous avons simulé 79 variations possibles sur 20 images de la base. Nous avons ensuite interrogé la base en utilisant chacune des images modifiées. Le but est de tester si notre système est capable de retrouver l'image originale à partir de laquelle l'image requête a été copiée. Pour une variation donnée, la proportion des images requêtes qui ont pu être mises en correspondance correctement avec leurs images originales est calculée. Ce taux est appelé taux de reconnaissance. Il reflète la robustesse de

k	Precision moy.	Rappel moy.	F-Mesure moy.
10	0,71	0,89	0,79
20	0,65	0,90	0.75
30	0,65	0,93	0,77
40	0,61	0,94	0,74
50	0,60	0,94	0,73
75	0,57	0,94	0,71
100	0,51	0,95	0,66

TAB. 8.5: Variation de k : précisions moyennes, rappels moyens et F-mesure moyennes pour la recherche de 32 images requêtes de base MOVI.

notre schéma face à la variation étudiée. Nous avons exécuté chaque recherche par les quatre algorithmes de la recherche séquentielle et par notre méthode de recherche approximative pour trois valeurs de α (0,01, 0,20 et 0,40). Les résultats obtenus sont résumés dans les deux tableaux 8.6 et 8.7. Nous n'avons pas reporté les temps de réponse car les facteurs d'accélération par rapport à la recherche séquentielle de base restent identiques à ceux reportés dans l'expérience précédente. Ces tableaux montrent bien que, globalement, le schéma de description est très robuste aux variations générées par StirMark. Il montre également que le caractère approximatif de notre méthode de recherche n'a quasiment aucun impact sur la précision finale de la recherche.

Cependant, si on analyse ces tableaux par type d'attaques, on voit que la robustesse des descripteurs à la rotation se confirme bien. Pour des angles variant entre -2 et 90°, le taux de reconnaissance est pratiquement toujours égal à 1. Il reste élevé mais relativement plus faible lorsqu'un recadrage et un changement d'échelle sont appliqués en plus de la rotation. Ces descripteurs sont également pratiquement invariants par rapport aux différents filtres appliqués (médians et gaussiens), au renforcement de contraste, aux cisaillements, à la suppression de lignes et de colonnes de l'image, aux transformations géométriques et aux recadrages. Pour cette dernière variation, le taux de reconnaissance est égal à 90 % lorsque l'image requête ne représente que 6,25 % de l'image originale (RecadCent(75)). Il est égal à 1 dès que l'image requête représente plus de 25 % de l'image originale (RecadCent(50)).

Ce schéma reste cependant limité par rapport au changement d'échelle. Au delà d'un changement d'échelle de facteur supérieur à 1,50 ou bien inférieur à 0,50, le taux de reconnaissance chute en dessous de 70 %. Ceci est dû au fait que, dans la version des invariants locaux différentiels que nous avons utilisée pour réaliser ces expériences, l'approche multi-échelle n'a pas été utilisée. Cette approche a été proposée dans [Schmid, 1996] puis revisitée dans [Dufournaud *et al.*, 2000b, Dufournaud *et al.*, 2000a]. Elle permet d'assurer une robustesse au changement d'échelle allant jusqu'à un facteur de 6.

Type d'attaque	Taux de reconnaissance			
	Rech. exacte	Rech. approx (α)		
		0,01	0,20	0,40
Rot(-0,25)	1	1	1	1
Rot(-0,50)	1	1	1	1
Rot(-0,75)	1	1	1	1
Rot(-1,00)	1	1	1	1
Rot(-2,00)	1	1	1	1
Rot(0,25)	1	1	1	1
Rot(0,50)	1	1	1	1
Rot(0,75)	1	1	1	1
Rot(1,00)	1	1	1	1
Rot(10,00)	1	1	1	1
Rot(15,00)	1	1	1	1
Rot(2,00)	1	1	1	1
Rot(30,00)	0,95	0,95	1	0,95
Rot(45,00)	1	1	1	0,95
Rot(5,00)	1	1	1	1
Rot(90,00)	1	1	1	1
RotReCE(-0,25)	1	1	1	1
RotReCE(-0,50)	1	1	1	1
RotReCE(-0,75)	1	1	1	1
RotReCE(-1,00)	1	1	1	1
RotReCE(-2,00)	1	1	1	1
RotReCE(0,25)	1	1	1	1
RotReCE(0,50)	1	1	1	1
RotReCE(0,75)	1	1	1	1
RotReCE(1,00)	1	1	1	1
RotReCE(10,00)	1	1	1	0,9
RotReCE(15,00)	0,9	0,9	0,8	0,55
RotReCE(2,0)	1	1	1	1
RotReCE(30,00)	0,4	0,4	0,35	0,25
RotReCE(45,00)	0,25	0,25	0,2	0,2
RotReCE(5,00)	1	1	1	1
RotReCE(90,00)	0,6	0,6	0,45	0,5
ChangEch(0,50)	0,65	0,65	0,575	0,55
ChangEch(0,75)	0,95	0,95	0,9	0,875
ChangEch(0,90)	1	1	1	1
ChangEch(1,10)	1	1	1	1
ChangEch(1,50)	0,7	0,7	0,6	0,45
ChangEch(2,00)	0,675	0,675	0,6	0,575
FilMed(2x2)	1	1	1	1
FilMed(3x3)	1	1	1	1
FilMed(4x4)	1	1	1	0,95
FilGauss	1	1	1	1

TAB. 8.6: Étude de la robustesse du schéma de description face aux attaques simulées par StirMark. Taux de reconnaissance moyen sur 20 images requêtes.

(Suite du tableau 8.6)

Type d'attaque	Taux de reconnaissance			
	Rech. exacte	Rech. approx (α)		
		0,01	0,20	0,40
Sharp	1	1	1	1
RecadCent(1)	1	1	1	1
RecadCent(10)	1	1	1	1
RecadCent(15)	1	1	1	1
RecadCent(2)	1	1	1	1
RecadCent(20)	1	1	1	1
RecadCent(25)	1	1	1	1
RecadCent(5)	1	1	1	0,975
RecadCent(50)	1	1	1	0,95
RecadCent(75)	0,9	0,9	0,9	0,9
CompJPEG(10)	1	1	0,95	0,85
CompJPEG(15)	1	1	1	1
CompJPEG(20)	1	1	1	1
CompJPEG(25)	1	1	1	1
CompJPEG(30)	1	1	1	1
CompJPEG(35)	1	1	1	1
CompJPEG(40)	1	1	1	1
CompJPEG(50)	1	1	1	1
CompJPEG(60)	1	1	1	1
CompJPEG(70)	1	1	1	1
CompJPEG(80)	1	1	1	1
CompJPEG(90)	1	1	1	1
Shear(0, 1)	1	1	1	1
Shear(0, 5)	1	1	1	1
Shear(1, 0)	1	1	1	1
Shear(1, 1)	1	1	1	1
Shear(5, 0)	1	1	1	1
Shear(5, 5)	1	1	1	1
SuppLigCol(17, 5)	1	1	1	1
SuppLigCol(1, 1)	1	1	1	1
SuppLigCol(1, 5)	1	1	1	1
SuppLigCol(5, 17)	1	1	1	1
SuppLigCol(5, 1)	1	1	1	1
TransGeomLin(1,007, 0,010, 0,010, 1,012)	1	1	1	1
TransGeomLin(1,010, 0,013, 0,009, 1,011)	1	1	1	1
TransGeomLin(1,013, 0,008, 0,011, 1,008)	1	1	1	1
DistorGeomAlea	1	1	1	1

TAB. 8.7: Étude de la robustesse du schéma de description face aux attaques simulées par StirMark. Taux de reconnaissance moyen sur 20 images requêtes.

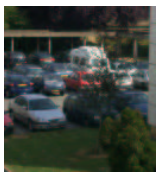
8.5.4 Expérience 4 : évaluation de la robustesse face au recadrage

Le recadrage est une attaque très fréquente et extrêmement importante. Les images des professionnelles de la photo sont souvent de haute résolution. Elle peuvent être recadrées pour créer de nouvelles photo tout à fait exploitable. Il est donc important d'assurer une bonne robustesse face à ce type d'attaque.

Nous avons vu dans l'expérience précédente que notre schéma était très robuste au recadrage lorsque celui-ci est appliqué tout seul, c'est-à-dire, lorsque les fragment générés par recadrage ne subissent aucune autre variation supplémentaire. Néanmoins, il serait intéressant d'étudier le cas où ces fragments subissent eux aussi d'autres variations. Pour cela, nous avons généré 6 fragments pour chacune des 20 images requêtes puis nous leurs avons appliqué d'autres variations supplémentaires à l'aide de StirMark.

Les 6 fragments considérés pour chaque image sont des sous-images dont les tailles en terme de surface représentent 6,25, 9, 16, 25, 36 et 56,25 % par rapport aux images originales. Les proportions 6,25, 25, et 56,25 correspondent respectivement à RecadCent(75), RecadCent(50) et RecadCent(25).[4] Les autres proportions ont été générées afin d'avoir des sous-images de tailles intermédiaires, le but étant d'affiner la granularité de l'étude.

Le tableau 8.8 présente un exemple d'une image et les six fragments (ou sous-images) associés.

Image originale	Fragments requêtes associés
	6,25% 9% 16% 25% 36% 56,25%

TAB. 8.8: Exemple d'une image et les six fragments requêtes associés.

% de l'image orig.	Taux de reconnaissance			
	Rech. exacte	Rech. approx (α)		
		0,01	0,20	0,40
6,25	0,90	0,90	0,90	0,90
9	0,95	0,95	0,95	0,95
16	0,95	0,95	0,95	0,95
25	1	1	1	0,95
36	1	1	1	1
56,25	1	1	1	1

TAB. 8.9: Taux de reconnaissance moyen pour différentes tailles de fragments requêtes et différentes stratégies de recherche – Fragments originaux.

[4]Le paramètre de la fonction RecadCent correspond au facteur de réduction (%) de la largeur et de la longueur de l'image originale.

Nous avons appliqué ensuite 5 variations différentes sur chaque fragment requête :
- un renforcement de contraste *(Sharpening)* ;
- un cisaillement (1,1),(5, 5) *(Shearing)* ;
- des rotations de -2, -1, -0,75, -0,50, -0,25, 0,25, 0,50, 1, 10, 15, 30 et 45° ;
- un filtrage médian(3×3) ;
- un filtrage gaussien.

Chaque fragment a été utilisé pour interroger la base en suivant la même procédure décrite dans la section précédente. Une moyenne du taux de reconnaissance a été ensuite calculée sur l'ensemble des fragments de même proportion et ayant subi le même type de changement.

Les tableaux 8.10.(a), 8.10.(b),... ,8.10.(h) donnent les résultats obtenus pour une partie des expériences réalisées. Nous donnons dans le tableau 8.9, le taux de reconnaissance pour différentes stratégies de recherche lors de l'interrogation de la base en utilisant les fragments originaux (n'ayant subi aucune transformation). Il s'agit des mêmes résultats que ceux présentés dans l'expérience précédente (tableau 8.7).

Les tableaux 8.10.(a) à 8.10.(g) présentent les résultats obtenus pour la recherche de fragments ayant subi des variations. Naturellement, les taux de reconnaissance dans ces cas là sont inférieurs par rapport aux taux de reconnaissance obtenus pour les fragments n'ayant subi aucune transformation. Néanmoins, ils restent très bons dans le cas de variations de type renforcement de contraste, cisaillement, convolution avec un filtre gaussien et convolution avec un filtre médian. Ils sont, par contre, nettement moins bons pour les fragments ayant subi une rotation de plus de 30° et plus particulièrement pour les petits fragments (6,25% et 9%). Dans ces deux derniers cas, nous remarquons que le taux de reconnaissance se dégrade considérablement lorsque le taux d'imprécision augmente. Ceci est dû principalement au petit nombre de descripteurs associés à ces fragments. En effet, seulement 4,4 (resp. 8,88) descripteurs en moyenne ont été calculés pour décrire le fragment dont la proportion est 6,25% (resp. 9%) dans le cas de la rotation à 45°. Comme nous l'avons expliqué dans la section 7.3, page 7.3, plus le nombre de descripteurs de l'image requête est élevé, moins l'imprécision de la recherche de plus proches voisins a d'impact sur le résultat final.

Ce problème n'est cependant pas une vraie limitation du système, car, en pratique, soit la taille des fragments est trop petite et, par conséquent, le fragment n'a aucune utilité pratique, soit sa taille est suffisamment grande, auquel cas le nombre de descripteurs qui leurs sont associés sera élevé et le problème ne se pose pas.

Pour conclure cette expérience, nous avons calculé les taux de reconnaissance moyens pour l'ensemble des variations considérées (voir tableau 8.10.(h)). Ces résultats confirment les conclusions précédemment énoncées concernant la robustesse des descripteurs (des taux moyens de reconnaissance relativement élevés) et la sensibilité mineure de l'imprécision de la recherche de plus proches voisins sur la qualité des résultats finaux. Nous retrouvons aussi les problèmes de dégradation de la qualité des résultats lorsque les fragments sont trop petits.

% de l'image orig.	Taux de reconnaissance			
	Rech. exacte	Rech. approx (α)		
		0,01	0,20	0,40
6,25	0,90	0,90	0,90	0,80
9	0,90	0,90	0,90	0,80
16	0,95	0,95	0,95	0,95
25	1	1	1	0,95
36	1	1	1	1
56,25	1	1	1	1

(a) Fragments + filtre gaussien

% de l'image orig.	Taux de reconnaissance			
	Rech. exacte	Rech. approx (α)		
		0,01	0,20	0,40
6,25	0,75	0,75	0,7	0,55
9	0,85	0,8	0,75	0,75
16	0,95	0,95	0,85	0,75
25	0,95	0,95	0,85	0,85
36	0,95	0,95	0,95	0,9
56,25	1	1	1	1

(b) Fragments + filtre médian

% de l'image orig.	Taux de reconnaissance			
	Rech. exacte	Rech. approx (α)		
		0,01	0,20	0,40
6,25	0,55	0,55	0,50	0,30
9	0,90	0,90	0,90	0,65
16	0,90	0,90	0,90	0,85
25	0,90	0,90	0,90	0,90
36	1	1	1	1
56,25	1	1	1	1

(c) Fragments + Rotation (-2°)

% de l'image orig.	Taux de reconnaissance			
	Rech, exacte	Rech. approx (α)		
		0,01	0,20	0,40
6,25	0,50	0,45	0,40	0,20
9	0,75	0,75	0,65	0,40
16	0,90	0,90	0,90	0,85
25	0,90	0,90	0,90	0,90
36	0,95	0,95	0,95	0,90
56,25	1	1	1	1

(d) Fragments + Rotation (10°)

% de l'image orig.	Taux de reconnaissance			
	Rech. exacte	Rech. approx (α)		
		0,01	0,20	0,40
6,25	0,10	0,10	0,05	0
9	0,35	0,35	0,15	0,15
16	0,70	0,70	0,65	0,50
25	0,90	0,90	0,75	0,50
36	0,95	0,95	0,85	0,75
56,25	1	1	0,95	0,90

(e) Fragments + Rotation (45°)

% de l'image orig.	Taux de reconnaissance			
	Rech. exacte	Rech. approx (α)		
		0,01	0,20	0,40
6,25	0,45	0,50	0,35	0,15
9	0,50	0,50	0,40	0,25
16	0,65	0,65	0,55	0,30
25	0,70	0,70	0,65	0,50
36	0,95	0,90	0,90	0,85
56,25	1	1	0,90	0,80

(f) Fragments + *Sharpening*

% de l'image orig.	Taux de reconnaissance			
	Rech. exacte	Rech. approx (α)		
		0,01	0,20	0,40
6,25	0,75	0,75	0,60	0,10
9	0,85	0,85	0,75	0,70
16	0,90	0,90	0,85	0,80
25	0,95	0,95	0,95	0,95
36	1	1	1	1
56,25	1	1	1	1

(g) Fragments + *Shearing* (5,5)

% de l'image orig.	Taux de reconnaissance			
	Rech. exacte	Rech. approx (α)		
		0,01	0,20	0,40
6,25	0,61	0,60	0,52	0,36
9	0,78	0,78	0,72	0,63
16	0,88	0,88	0,86	0,79
25	0,92	0,92	0,89	0,83
36	0,98	0,98	0,96	0,94
56,25	0,99	0,99	0,98	0,97

(h) Résultats moy. sur toutes les variations

TAB. 8.10: Taux de reconnaissance moyen pour différentes tailles de fragments requêtes *transformés* et différentes stratégies de recherche.

8.6 Synthèse

Dans ce chapitre, nous avons présenté une évaluation expérimentale du système de recherche d'images pour la détection de copie que nous avons proposé dans la chapitre

précédent. Nous avons commencé par évaluer l'accélération de la recherche apportée par l'optimisation basée sur les requêtes multiples. Il s'est avéré que cette optimisation qui, ne concerne que les E/S, a permis d'améliorer considérablement les performances de la recherche séquentielle. Elle a cependant détérioré les performances de notre méthode car le surcoût de son exécution est trop élevé comparativement au gain apporté. Cette étude a permis également de montrer que le recherche séquentielle optimisée utilisant de plus l'option distance partielle, est une méthode très compétitive. Elle permet de réduire le temps de recherche de plus de 68 % par rapport à la recherche séquentielle de base.

Dans la deuxième partie de ces expérimentations, nous avons évalué à la fois la rapidité de la recherche, la capacité de reconnaissance des descripteurs utilisés ainsi que l'impact de l'imprécision des plus proches voisins sur le résultat final. Nous avons utilisé pour cela les deux protocoles d'évaluation décrits dans la section 8.2. Les résultats obtenus ont montré une grande robustesse des descripteurs face aux variations modélisées par les séquences de la base MOVI et, à la majeure partie des attaques simulées par le logiciel StirMark. Nous avons remarqué cependant que ces descripteurs n'étaient pas robustes aux changements d'échelle de facteurs supérieurs à 1,5 et inférieurs à 0,5. Ceci était prévisible car les descripteurs utilisés dans ces expériences ne sont pas invariants aux changements d'échelle. Pour assurer cette invariance, il faut utiliser la version multi-échelle de ces descripteurs qui a été proposée dans [Dufournaud *et al.*, 2000a]. L'extension de cette étude aux autres variantes de ces descripteurs (multi-échelle compris) et leur évaluation dans un contexte de très grandes bases d'images est l'une des perspectives du présent travail.

Conclusion générale

Dans ce travail de thèse, nous nous sommes intéressés principalement à la recherche par similarité dans le contexte des applications de recherche d'images par le contenu. Nous avons pris en compte les contraintes liées à l'efficacité de la recherche mais aussi les spécificités des descripteurs d'images. Nous avons alors développé une méthode originale et efficace pour la recherche approximative de plus proches voisins. Nous l'avons ensuite intégrée puis évaluée dans le cadre d'une application réelle de recherche d'images par le contenu.

Dans cette conclusion générale, nous présentons tout d'abord une synthèse des travaux effectués, puis nous esquissons plusieurs perspectives dans le prolongement de cette étude.

Synthèse des travaux effectués

Les deux principales contributions de nos travaux concernent la recherche de plus proches voisins au sein d'ensembles de descripteurs d'images, et son application à la recherche d'images par le contenu.

Recherche de plus proches voisins

Notre but premier était de développer une méthode efficace pour la recherche de plus proches voisins qui prend en compte l'ensemble des spécificités des descripteurs d'images (grande dimension, grand volume, aucune hypothèse sur la distribution des vecteurs...). Pour atteindre cet objectif, nous avons tout d'abord étudié de manière approfondie les méthodes existantes et leurs limites en prêtant une attention particulière au problème de la malédiction de la dimension. Cette étude a montré que les performances des techniques existantes se dégradent exponentiellement lorsque la dimension augmente. Cette dégradation des performances est particulièrement manifeste lorsque la recherche est exacte. Une manière de pallier le problème de la dimension est alors d'effectuer des recherches approximatives. L'imprécision introduite lors de la recherche est compensée par une forte réduction du temps de réponse. Plusieurs méthodes reposant sur ce principe ont été développées. Ces méthodes possèdent cependant plusieurs inconvénients. Soit elles ne permettent aucun contrôle de la précision, soit ce contrôle est effectué au travers de nombreux paramètres complexes et difficiles à fixer par un utilisateur non spécialisé. De plus, nombre d'entre elles ne peuvent effectuer que des recherches *du* plus proche voisin. Sur la base de cette étude, nous avons décidé de nous placer dans le contexte de la recherche approximative. Cependant, nous nous sommes fixés principalement quatre critères que doit satisfaire la méthode

développée, critères que nous pensons indispensables pour que la recherche approximative soit utilisable en pratique. Ces quatre critères sont :

- les paramètres de contrôle doivent être intuitifs ;
- le contrôle de la précision doit être fiable ;
- l'imprécision introduite doit être compensée par une forte réduction du temps de réponse ;
- la méthode doit être capable d'effectuer des recherches de k-plus proches voisins.

La méthode proposée satisfait l'ensemble de ces critères. Elle permet de contrôler la précision de la recherche de façon probabiliste et à l'aide d'un seul paramètre appelé le niveau d'imprécision et noté α. Ce paramètre correspond à la probabilité maximale de ne pas retrouver un des plus proches voisins dans le résultat de la recherche.

Notre méthode repose sur le principe classique des méthodes de recherche de plus proches voisins qui consiste à regrouper, hors ligne, les vecteurs en paquets et de les englober dans des cellules de formes géométriques simples, puis d'exploiter en ligne, les propriétés des cellules pour en sélectionner seulement les paquets qui ont le plus de chance de contenir les plus proches voisins recherchés. Il est bien établi que l'efficacité des méthodes de recherche de plus proches voisins dépend fortement de la compacité et de la séparabilité des cellules. Si ces deux propriétés ne sont pas satisfaites, le taux de chevauchement entre cellules est important. Par conséquent, les règles de filtrage utilisées lors de la recherche deviennent inefficaces et ne permettent pas de confiner la recherche à un petit sous-ensemble de paquets.

Pour prendre en compte ces deux propriétés, nous avons adapté et utilisé un algorithme de *clustering* pour le regroupement des vecteurs. Nous avons choisi la forme hypersphérique pour les cellules des *clusters*. Cependant, pour réduire davantage le taux de chevauchement entre cellules, nous avons considéré pour chaque *cluster*, une hypersphère de rayon sous-estimée, c'est-à-dire qui n'englobe pas la totalité des vecteurs du *cluster*. Ceci introduit de l'imprécision lors de la recherche car certains vecteurs se trouvant à l'extérieur de l'hypersphère ne sont pas pris en compte lors de l'application des règles de filtrage. Pour contrôler cette imprécision, nous avons établi un lien entre le paramètre de contrôle (le niveau d'imprécision α) et les rayons des *clusters*. Nous avons ensuite proposé une méthode permettant de calculer pour chaque *cluster*, un rayon approximatif en fonction du niveau d'imprécision. Pour considérer plusieurs niveaux d'imprécision, il suffit de calculer et de mémoriser pour chaque *cluster*, plusieurs rayons approximatifs, chacun correspondant à un niveau d'imprécision donné.

Pour évaluer notre méthode, nous avons mené une étude expérimentale étendue portant à la fois sur l'efficacité de la méthode et la fiabilité du processus de contrôle de la précision. Nous avons également étudié l'impact de l'algorithme de *clustering* sur les performances de la méthode et nous avons réalisé une étude comparative entre notre méthode et CLINDEX. Cette étude a montré que notre méthode est très fiable car globalement, l'imprécision des résultats retournés est inférieure à l'imprécision fixée au travers du paramètre α. Elle permet des facteurs d'accélération allant jusqu'à 15 par rapport à la recherche séquentielle et elle est beaucoup plus performante que CLINDEX.

Par ailleurs, nous avons également proposé deux autres modes de recherche possibles exploitant les principes de base de la recherche approximative. Il s'agit de la recherche incrémentielle de base et de la recherche incrémentielle avec indication sur l'amélioration possible des résultats.

Application à la recherche d'images par le contenu

Disposant d'une méthode de recherche de plus proches voisins efficace, nous nous sommes intéressés dans un deuxième temps à son application à la recherche d'images par le contenu. L'objectif est d'évaluer son efficacité ainsi que l'impact de l'imprécision qu'elle introduit lors dans la recherche dans un cadre applicatif réel. Pour cela, nous avons choisi comme application, la recherche d'images par le contenu pour la détection de copies. L'objectif de cette application est de vérifier si une image provenant d'un site tiers n'est pas une copie d'une image d'un propriétaire et ce, en se basant uniquement sur la similarité visuelle entre images. C'est une application qui offre de très bonnes possibilités d'évaluation qualitative et quantitative de la recherche. Elle se base également sur un schéma de description locale des images très robuste mais extrêmement coûteux (descripteurs locaux différentiels [Schmid, 1996, Amsaleg & Gros, 2001]). Elle représente ainsi un cadre approprié pour l'évaluation de notre méthode de recherche.

Le résultat de la recherche dans un schéma de description locale étant la fusion des résultats de plusieurs recherches de plus proches voisins, nous avons tout d'abord étudié théoriquement l'impact de l'imprécision introduite par notre méthode de recherche sur la qualité du résultat final. Cette étude a montré que l'imprécision des plus proches voisins n'a qu'un impact mineur sur le résultat final. Si le niveau d'imprécision de la recherche de plus proches voisins est fixé à α, la probabilité de permuter deux images similaires dans le résultat final est de α^{γ}, où $\gamma >> 1$.

Nous avons ensuite réalisé un travail d'évaluation qui portait à la fois sur la capacité de reconnaissance des descripteurs, et sur l'impact de l'imprécision de la recherche de plus proches voisins sur le résultat final.

Cette étude a montré la robustesse des descripteurs face aux changements étudiés. Elle a également montré que l'imprécision de la recherche de plus proches voisins n'a qu'un impact mineur sur la qualité des résultats finaux, confirmant ainsi les résultats de l'étude théorique.

Perspectives

Les perspectives que nous envisageons dans le prolongement de ce travail de thèse s'articulent autour des points suivants.

Retour sur les descripteurs

Dans ce travail, nous avons pris en compte les propriétés et les spécificités des descripteurs lors de la conception de notre méthode de recherche approximative, et nous ne les avons, à aucun moment, remis en cause. Il serait, cependant, pertinent de s'intéresser également aux descripteurs et à leur calcul afin de proposer des simplifications qui pourraient réduire les difficultés rencontrées lors de la recherche (la grande dimension notamment). En général, l'unique contrainte prise en compte lors de la conception des méthodes de description est la capacité de reconnaissance.

L'idée est alors de reconsidérer le calcul des descripteurs en prenant en compte également l'aspect efficacité. Il s'agit de réaliser un compromis entre capacité de reconnaissance et efficacité. Pour cela, il faut tout d'abord mettre au point un protocole

d'évaluation de la capacité de reconnaissance des descripteurs. Celui-ci permet de quantifier, de façon précise, l'apport en terme de capacité de reconnaissance, des différents choix effectués lors de la mise au point des descripteurs (la dimension, la mesure de similarité...). On pourrait alors, sur la base des résultats d'une campagne d'évaluation utilisant ce protocole, décider, par exemple, de réduire la dimension des descripteurs en éliminant les composantes n'ayant qu'un impact mineur sur la qualité en terme de reconnaissance. Ceci permet de réduire le coût de la recherche par similarité. Notons que cette réduction est fondamentalement différente d'une réduction de la dimension par des méthodes d'analyse de données. En effet, les méthodes d'analyse de données utilisent les propriétés statistiques des descripteurs pour réduire leur dimension sans se soucier explicitement de l'usage qui est fait des descripteurs. Généralement, il est difficile d'établir un lien entre la perte d'information mesurée en terme statistique et l'impact de la réduction de la dimension sur la capacité de reconnaissance (*cf.* section 2.5.1.3, page 51).

Par ailleurs, l'intégration de notre méthode de recherche de plus proches voisins dans le protocole d'évaluation permet d'alléger le coût des campagnes d'évaluation, ce qui permet d'effectuer plus de tests et de considérer des bases de très grande taille. Ainsi, ce protocole pourrait également être utilisé pour étudier d'autres paramètres intervenant lors de la conception des descripteurs, et qui n'ont pas forcément d'impact sur l'efficacité de la recherche (e.g. le choix des espaces couleurs). L'efficacité de ce protocole permet de considérer des bases d'images de taille très grande et de lever ainsi le biais lié à la taille des bases d'images de test.

Recherche approximative et *clustering*

La méthode de recherche approximative proposée utilise un algorithme de *clustering* pour le regroupement des vecteurs lors ligne. Pour cela, nous avons adapté l'algorithme BIRCH pour effectuer cette tâche. Bien que cet algorithme soit de complexité linéaire et ne nécessite qu'une seule lecture de la base de vecteurs, son temps d'exécution est exorbitant. C'est le seul facteur qui limite la taille des bases pouvant être considérées par notre méthode. Il est donc important d'alléger le coût de cette phase afin de permettre le passage à l'échelle de notre méthode de recherche. Pour ce faire, il est tout d'abord nécessaire de bien comprendre l'impact du *clustering* sur les performances de la recherche. Dans l'expérience 6 (section 6.9, page 148), une première étude a été menée en ce sens. Elle a montré que la recherche est peu sensible à l'algorithme de *clustering*. Par conséquent, un algorithme de *clustering* effectuant un partitionnement grossier mais extrêmement rapide de la base pourrait être envisageable. Néanmoins, il est important d'approfondir cette étude et de définir de manière plus précise la relation entre les propriétés du partitionnement de la base et les performances de la recherche. Ceci permettrait de définir les conditions suffisantes que doit satisfaire l'algorithme de *clustering* afin de garantir un bon niveau de performance lors de la recherche.

Par ailleurs, nous envisageons d'autres solutions pour réduire le coût de cette phase. Il s'agit de la parallélisation de cette phase et son exécution sur une grappe de machine. Cette solution nécessite la parallélisation de l'algorithme de *clustering* dans la mesure du possible, ou bien l'utilisation ou l'adaptation d'un algorithme de *clustering* parallèle exis-

tant [Judd *et al.*, 1998, Xu *et al.*, 1999, Arlia & Coppola, 2001]. Cette solution est intéressante d'autant plus que, dans les systèmes réels, les bases d'images sont souvent distribuées sur plusieurs machines.

Passage à l'échelle

Les expérimentations effectuées dans ce travail ont montré l'efficacité de notre méthode de recherche. Ces expérimentations ont, cependant, été réalisées sur des bases d'images de taille relativement modeste comparativement aux bases d'images réelles dont le nombre d'images se chiffre à plusieurs millions (e.g. les bases des agences de photos). Nous envisageons, par conséquent, d'étendre ces expérimentations à des bases d'images de plus grande taille. Cela permet de tester à la fois le passage à l'échelle de la méthode de recherche et la capacité de reconnaissance des descripteurs. Dans un tel contexte, de nouveaux problèmes de reconnaissance et de rapidité risquent d'apparaître.

Recherche incrémentielle

Dans la section 5.4, page 127, nous avons proposé deux extensions possibles de la méthode de recherche approximative permettant deux modes de recherche supplémentaires. Il s'agit de la recherche incrémentielle de base et la recherche incrémentielle avec indication sur l'amélioration possible des résultats. Ce sont deux modes de recherche interactive très intéressants dans les applications pratiques. Nous ne les avons cependant pas expérimentés. Il serait donc très important d'en évaluer les performances dans le cadre d'une application réelle.

Gestion des insertions

Dans la méthode de recherche de plus proches voisins, nous n'avons proposé aucune fonctionnalité permettant de prendre en compte l'ajout de nouveaux descripteurs dans la base. La seule façon naïve d'effectuer cela est de mettre les descripteurs rajoutés dans un fichier spécial qui est parcouru séquentiellement lors de la recherche. Lorsque la taille de ce fichier devient conséquente, le coût de la recherche risque d'être très coûteux. Il est alors nécessaire de reprendre l'ensemble des descripteurs de la base et d'effectuer, de nouveau, un partitionnement des descripteurs suivi d'un calcul des rayons approximatifs.

Dans le cas d'une base d'images dans laquelle les opérations d'ajout sont peu fréquentes, cette solution peut tout à fait être envisageable. Par contre, dans une base d'images dont le contenu évolue régulièrement, cette solution risque d'être extrêmement coûteuse. Il serait donc nécessaire, d'une part, de doter l'algorithme de *clustering* d'une procédure de mise à jour afin de maintenir dynamiquement une partition de la base, et, d'autre part, de mettre au point une stratégie de mise à jour des rayons approximatifs des *cluster*. Celle-ci met à jour le rayon d'un *cluster* si un ou plusieurs descripteurs y sont insérés.

Bibliographie

[Amsaleg & Gros, 2000] Amsaleg, L. & Gros, P. (2000). A robust technique to recognize objects in images, and the DB problems it raises. Rapport de recherche 4081, INRIA.

[Amsaleg & Gros, 2001] Amsaleg, L. & Gros, P. (2001). Content-based retrieval using local descriptors: Problems and issues from a database perspective. *Pattern Analysis and Applications, Special Issue on Image Indexation*, 4:108–124.

[Arlia & Coppola, 2001] Arlia, D. & Coppola, M. (2001). Experiments in parallel clustering with dbscan. In *7th International Euro-Par Conference (Parallel Processing), Manchester, UK*, pages 326–331. LNCS 2150.

[Barnett, 2002] Barnett, V. (2002). *Sample Survey, Principles and Methods, 3rd edition.* Arnold.

[Basseville, 1988] Basseville, M. (1988). Detecting changes in signals and systems – A survey. *Automatica*, 24(3):309–326.

[Beckmann *et al.*, 1990] Beckmann, N., Kriegel, H.-P., Schneider, R., & Seeger, B. (1990). The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the* ACM SIGMOD *International Conference on Management of Data, Atlantic City, New Jersey, USA*, pages 322–331.

[Bellman, 1961] Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour.* Princeton University Press.

[Bennett *et al.*, 1999] Bennett, K. P., Fayyad, U., & Geiger, D. (1999). Density-based indexing for approximate nearest-neighbor queries. In *Proceedings of the 5th* ACM *International Conference on Knowledge Discovery and Data Mining, San Diego, California, USA*, pages 233–243.

[Bentley, 1979] Bentley, J. L. (1979). Multidimensional binary search in database applications. IEEE *Transactions on Software Engineering*, 4(5):333–340.

[Berchtold *et al.*, 1998a] Berchtold, S., Böhm, C., & Kriegel, H.-P. (1998a). Improving the query performance of high-dimensional index structures by bulk load operations. In *Proceedings of the 6th Conference on Extending Database Technology, Valencia, Spain*, pages 216–230.

[Berchtold *et al.*, 1998b] Berchtold, S., Böhm, C., & Kriegel, H.-P. (1998b). The pyramid-technique: Towards breaking the curse of dimensionality. In *Proceedings of the* ACM SIGMOD *International Conference on Management of Data, Seattle, Washington, USA*, pages 142–153.

[Berchtold *et al.*, 1996] Berchtold, S., Keim, D. A., & Kriegel, H.-P. (1996). The x-tree : An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on Very Large Data Bases, Mumbai (Bombay), India*, pages 28–39.

[Bercken *et al.*, 1997] Bercken, J. V. D., Seeger, B., & Widmayer, P. (1997). A generic approach to bulk loading multidimensional index structures. In *Proceedings of the 23rd International Conference on Very Large Data Bases, Athens, Greece*, pages 406–415.

[Beyer *et al.*, 1999] Beyer, K. S., Goldstein, J., Ramakrishnan, R., & Shaft, U. (1999). When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory, Jerusalem, Israel*, pages 217–235. Springer.

[Beyer & McCreight, 1972] Beyer, R. & McCreight, E. M. (1972). Organization and maintenance of large ordered indices. *Acta Informatica*, 1(3):173–189.

[Bigün *et al.*, 1991] Bigün, J., Granlund, G. H., & Wiklund, J. (1991). Multidimensional orientation estimation with applications to texture analysis and optical flow. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 13(8):775–790.

[Böhm *et al.*, 2000] Böhm, C., Braunmüller, B., & Kriegel, H.-P. (2000). The pruning power: Theory and heuristics for mining databases with multiple k-nearest-neighbor queries. In *Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery*, pages 372–381.

[Borg & Groenen, 1997] Borg, I. & Groenen, P. (1997). *Modern multidimensional Scaling. Theory and Applications*. Springer Series in Statistics.

[Braunmüller *et al.*, 2000a] Braunmüller, B., Ester, M., Kriegel, H.-P., & Sander, J. (2000a). Efficiently supporting multiple similarity queries for mining in metric databases. In *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA*, pages 256–270.

[Braunmüller *et al.*, 2000b] Braunmüller, B., Ester, M., Kriegel, H.-P., & Sander, J. (2000b). Efficiently supporting multiple similarity queries for mining in metric databases. In *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA*, pages 256–267.

[Brown, 1956] Brown, G. (1956). *Modern Mathematics for the Engineer*. McGraw-Hill, New York.

[Cardoso, 1999] Cardoso, J.-F. (1999). High-order contrasts for independent component analysis. *Neural Computation*, 11(1):157–192.

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

[Chakrabarti & Mehrotra, 2000] Chakrabarti, K. & Mehrotra, S. (2000). Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proceedings of the 26th International Conference on Very Large Data Bases, Cairo, Egypt*, pages 89–100.

[Chu *et al.*, 2002] Chu, S.-C., Roddick, J. F., & Pan, J. (2002). An efficient k-medoids-based algorithm using previous medoid index, triangular inequality elimination criteria, and partial distance search. In *Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery, Aix-en-Provence, France*, pages 63–72.

[Ciaccia & Patella, 2000] Ciaccia, P. & Patella, M. (2000). Pac nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA*, pages 244–255.

[Ciaccia *et al.*, 1997] Ciaccia, P., Patella, M., & Zezula, P. (1997). M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases, Athens, Greece*, pages 426–435.

[Comaniciu & Meer, 2002] Comaniciu, D. & Meer, P. (2002). Mean shift: A robust approach towards feature space analysis. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619.

[Comon, 1994] Comon, P. (1994). Independent component analysis, a new concept? *Signal Processing, Special Issue on High Order Statistics*, 36(3):287–314.

[Cox *et al.*, 2002] Cox, I. J., Miller, M. L., & Bloom, J. A. (2002). *Digital Watermarking*. Morgan Kaufmann Publishers.

[Cox & Cox, 2001] Cox, T. F. & Cox, M. A. A. (2001). *Multidimensional Scaling, Second Edition*. Chapman & Hann/CRC.

[de Sousa *et al.*, 2002] de Sousa, E. P. M., Traina, C., Traina, A. J. M., & Faloutsos, C. (2002). How to use fractal dimension to find correlations between attributes. In *First Workshop on Fractals and Self-similarity in Data Mining: Issues and Approaches (in conjunction with 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining), Edmonton, Alberta, Canada*, pages 26–30.

[Demartines, 1994] Demartines, P. (1994). *Analyse de données par réseaux de neurones auto-organisés*. PhD thesis, Institut National Polytechnique de Grenoble.

[Demartines & Hérault, 1997] Demartines, P. & Hérault, J. (1997). Curvilinear component analysis: A self-organizing neural network for non linear mapping of data sets. IEEE *Transactions on Neural Networks*, 8(1):148–154.

[Deriche *et al.*, 1998] Deriche, R., Gouet, V., & Montesinos, P. (1998). Differential invariants for color images. In *Proceedings of the 14th International Conference on Pattern Recognition, Brisbane, Australia*.

[Dimai, 1999] Dimai, A. (1999). Assessment of effectiveness of content based image retrieval systems. In Huijsmans, D. P. & Smeulders, A. W. M., editors, *Proceedings of the International Conference on Visual Information and Information Systems (VISUAL), Amsterdam, The Netherlands*, pages 525–532.

[Donoho, 2000] Donoho, D. L. (2000). High-dimensional data analysis: The curses and blessings of dimensionality. In *The American Mathemathical Society Conference "Math Challenges of the 21st Century"*. Available at http://www-stat.stanford.edu/~donoho/.

[Dufournaud *et al.*, 2000a] Dufournaud, Y., Schmid, C., & Horaud, R. (2000a). Appariement d'images à des échelles différentes. In *Actes du 12e Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle, Paris, France*, volume 2, pages 327–336.

[Dufournaud *et al.*, 2000b] Dufournaud, Y., Schmid, C., & Horaud, R. (2000b). Matching images with different resolutions. In *Proceedings of the Conference on Computer Vision and Pattern Recognition, Hilton Head Island, South Carolina, USA*, volume 1, pages 612–618.

[Ester *et al.*, 1996] Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowladge Discovery and Data Mining, Portland, Oregon, USA*, pages 226–231.

[Ester *et al.*, 1995] Ester, M., Kriegel, H.-P., & Xu, X. (1995). Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. In *Proceedings of the 4th International Symposium on Large Spatial Databases, New York, USA*.

[Everitt & Rabe-Hesketh, 1997] Everitt, B. S. & Rabe-Hesketh, S. (1997). *The Analysis of proximity Data*. Arnold.

[Faloutsos, 1996] Faloutsos, C. (1996). *Searching Multimedia Databases by Content*. Kluwer Academic Publishers.

[Faloutsos & Lin, 1995] Faloutsos, C. & Lin, K.-I. (1995). Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the* ACM SIGMOD *International Conference on Management of Data, San Jose, California, USA*, pages 163–174.

[Ferhatosmanoglu *et al.*, 2000] Ferhatosmanoglu, H., Tuncel, E., Agrawal, D., & El Abbadi, A. (2000). Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the 9th* ACM *International Conference on Information and Knowledge Management, McLean, VA, USA*, pages 202–209.

[Ferhatosmanoglu *et al.*, 2001] Ferhatosmanoglu, H., Tuncel, E., Agrawal, D., & El Abbadi, A. (2001). Approximate nearest neighbor searching in multimedia databases. In *Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany*, pages 503–511.

[Florack *et al.*, 1994] Florack, L., ter Haar Romeny, B., Koenderink, J., & Viergever, M. (1994). General intensity transformation and differential invariants. *Journal of Mathematical Imaging and Vision*, 4(2):171–187.

[Fodor, 2002] Fodor, I. K. (2002). A survey of dimension reduction techniques. Technical Report UCRL-ID-148494, Lawrence Livermore National Laboratory. dim + VE.

[Gerbrands, 1981] Gerbrands, J. J. (1981). On the relationships between SVD, KLT and PCA. *Pattern Recognition*, 14(1-6):375–381.

[Gionis *et al.*, 1999] Gionis, A., Indyk, P., & Motwani, R. (1999). Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases, Edinburgh, Scotland, UK*, pages 518–529.

[Goldstein & Ramakrishnan, 2000] Goldstein, J. & Ramakrishnan, R. (2000). Contrast plots and p-sphere trees: Space vs. time in nearest neighbor searches. In *Proceedings of the 26th International Conference on Very Large Data Bases, Cairo, Egypt*, pages 429–440.

[Grabmeier & Rudolph, 2002] Grabmeier, J. & Rudolph, A. (2002). Techniques of cluster algorithms in data mining. *Data Mining and Knowledge Discovery*, 6(4):303–360.

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

[Halkidi *et al.*, 2001] Halkidi, M., Batistakis, Y., & Vazirgiannis, M. (2001). On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2–3):107–145.

[Harris & Stephens, 1988] Harris, C. G. & Stephens, M. J. (1988). A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference, Manchester, England*, pages 147–151.

[Hastie *et al.*, 2001] Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The Elements of Statistical Learning. Data Mining, Inference and Prediction.* Springer Series in Statistics.

[Henrich, 1998] Henrich, A. (1998). The lsd^{h}-tree: An access structure for feature vectors. In *Proceedings of the 14th International Conference on Data Engineering, Florida, USA*, pages 362–369.

[Henrich *et al.*, 1989] Henrich, A., Six, H.-W., & Widmayer, P. (1989). The lsd tree: Spatial access to multidimensional point and nonpoint objects. In Apers, P. M. G. & Wiederhold, G., editors, *Proceedings of the 15th International Conference on Very Large Data Bases, Amsterdam, The Netherlands*, pages 45–53. Morgan Kaufmann.

[Hentschel & Procaccia, 1983] Hentschel, H. & Procaccia, I. (1983). The infinite number of generalized dimensions of fractals and strange attractors. *Physica 8D*, pages 435–444.

[Hermes *et al.*, 2002] Hermes, L., Zölle, T., & Buhmann, J. M. (2002). Parametric distributional clustering for image segmentation. In Heyden, A., Sparr, G., Nielsen, M., and Johansen, P., editors, *Proceedings of the 7th European Conference on Computer Vision, Copenhagen, Danemark*, pages 577–591.

[Hinkley, 1971] Hinkley, D. (1971). Inference about the change-point from cumulative sum tests. *Biometrika*, 58:509–523.

[Hinneburg & Keim, 1998] Hinneburg, A. & Keim, D. A. (1998). An efficient approach to clustering in multimedia databases with noise. In *Proceedings of the 4th International Conference on Knowladge Discovery and Data Mining, Portland, Oregon, USA.*

[Hinneburg & Keim, 1999] Hinneburg, A. & Keim, D. A. (1999). Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *Proceedings of the 25th International Conference on Very Large Data Bases, Edinburgh, Scotland, UK*, pages 506–517.

[Hjaltason & Samet, 1995] Hjaltason, G. R. & Samet, H. (1995). Ranking in spatial databases. In Egenhofer and Herring, editors, *Proceedings of the 4th International Symposium on Spatial Databases, Portland, Maine, USA*, pages 83–95.

[Hotelling, 1933] Hotelling, H. (1933). Analysis of complex statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441, 498–520.

[Huang *et al.*, 1997] Huang, J., Kumar, S. R., Mitra, M., Zhu, W., & Zabih, R. (1997). Image indexing using color correlograms. In *Proceedings of the Conference on Computer Vision and Pattern Recognition, Puerto Rico, USA*, pages 762–768.

[Hubert, 1973] Hubert, L. (1973). Monotone invariant clustering. *Psychometrika*, 38(1):47–62.

[Hutflesz *et al.*, 1988] Hutflesz, A., Six, H.-W., & Widmayer, P. (1988). Twin grid files: space optimizing access schemes. In *Proceedings of the* ACM SIGMOD *International Conference on Management of Data, Chicago, IL USA*, pages 183–190.

[Hyvärinen, 1999a] Hyvärinen, A. (1999a). Fast and robust fixed-point algorithms for independent component analysis. IEEE *Transactions on Neural Networks*, 10(3):626–634.

[Hyvärinen, 1999b] Hyvärinen, A. (1999b). Survey on independent component analysis. *Neural Computing Surveys*, 2:94–128.

[Hyvärinen & Oja, 2000] Hyvärinen, A. & Oja, E. (2000). Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4-5):411–430.

[Indyk & Motwani, 1998] Indyk, P. & Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of 13th Annual* ACM *Symposium on Theory of Computing, Dallas, Texas, USA*, pages 604–613.

[Inselberg, 1985] Inselberg, A. (1985). The plane with parallel coordinates, special issue on computational geometry. *The Visual Computer*, 1:69–97.

[Inselberg & Dimsdale, 1990] Inselberg, A. & Dimsdale, B. (1990). Parallel coordinates: A tool for visualizing multidimensional geometry. In *Visualization'90*.

[Jain & Murty, 1999] Jain, A. K. & Murty, M. N. (1999). Data clustering: A review. ACM *Computing Surveys*, 31(3):265–323.

[Jeffrey, 1995] Jeffrey, A. (1995). *Handbook of Mathematical Formulas and Integrals*. Academic Press Limited.

[Jr. *et al.*, 2002] Jr., J. A. B., Fahmy, G., & Panchanathan, S. (2002). A method for evaluating the performance of content-based image retrieval systems based on subjectively determined similarity between images. In Lew, M. S., Sebe, N., and Eakins, J. P., editors, *Proceedings of the International Conference on Image and Video Retrieval, London, UK*, pages 356–366.

[Judd *et al.*, 1998] Judd, D., McKinley, P. K., & Jain, A. K. (1998). Large-scale parallel data clustering. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 20(8):871–876.

[Jutten, 1987] Jutten, C. (1987). *Calcul neuromiétique et traitement du signal - Analyse en composantes indépendantes*. PhD thesis, Institut National Polytechnique de Grenoble - France.

[Kamel & Faloutsos, 1993] Kamel, I. & Faloutsos, C. (1993). On packing r-trees. In *Proceedings of the 2nd* ACM *International Conference on Information and Knowledge Management, Washington, DC, USA*, pages 490–499.

[Kamel & Faloutsos, 1994] Kamel, I. & Faloutsos, C. (1994). Hilbert r-tree: An improved r-tree using fractals. In *Proceedings of the 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile*, pages 500–509.

[Karhunen, 1947] Karhunen, K. (1947). Uber lineare methoden in wahrscheinlichkeitsrechnung. *Ann. Acad. Sci. Fenn. 37*.

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

[Keim & Hinneburg, 1999] Keim, D. A. & Hinneburg, A. (1999). Clustering techniques for large data sets – from the past to the future. In *Tutorial Notes for* ACM SIGKDD *International Conference on Knowledge Discovery and Data Mining, San Diego, California, USA*, pages 141–181.

[Knuth, 1997] Knuth, D. E. (1997). *Art of Computer Programming, Volume 2: Seminumerical Algorithms*, pages 135–136. Addison-Wesley.

[Korn *et al.*, 2001] Korn, F., Pagel, B.-U., & Faloutsos, C. (2001). On the "dimensionality curse" and the "self-similarity blessing". IEEE *Transactions on Knowledge and Data Engineering*, 13(1):96–111.

[Lance & Williams, 1967] Lance, G. N. & Williams, W. T. (1967). A general theory of classication sorting strategies. 2. clustering systems. *Computer Journal*, 10:271–277.

[Lawder & King, 2001] Lawder, J. K. & King, P. J. H. (2001). Querying multi-dimensional data indexed using the hilbert space-filling curve. *SIGMOD Record*, 30(1):19–24.

[Lebart *et al.*, 1984] Lebart, L., Morineau, A., & Warwick, K. M. (1984). *Multivariate Descriptive Statistical Analysis*. Willey Series in probability and mathematical statistics.

[Li *et al.*, 2002] Li, C., Chang, E., Garcia-Molina, H., & Wiederhold, G. (2002). Clustering for approximate similarity search in high-dimensional spaces. IEEE *Transactions on Knowledge and Data Engineering*, 14(4):792–808.

[Lin *et al.*, 1994] Lin, K.-I., Jagadish, H. V., & Faloutsos, C. (1994). The tv-tree: An index structure for high-dimensional data. *VLDB journal*, 3(4):517–542.

[Loève, 1948] Loève, M. (1948). *Fonctions aléatoires de second ordre*. Hermann, Paris.

[Mandelbrot, 1982] Mandelbrot, B. (1982). *The Fractal Geometry of Nature*. San Francisco; Freeman.

[Mandelbrot, 1984] Mandelbrot, B. (1984). *Les objets fractales (2^e édition)*. Flammarion.

[Müller *et al.*, 2001a] Müller, H., Müller, W., & Squire, D. M. (2001a). Automated benchmarking in content-based image retrieval. In *Proceedings of the* IEEE *International Conference on Multimedia and Expo, Tokyo, Japan*, pages 22–25.

[Müller *et al.*, 2001b] Müller, H., Müller, W., Squire, D. M., Marchand-Maillet, S., & Pun, T. (2001b). Performance evaluation in content-based image retrieval: Overview and proposals. *Pattern Recognition Letters*, 22(5):593–601.

[Nievergelt *et al.*, 1984] Nievergelt, J., Hinterberger, H., & Sevcik, K. C. (1984). The grid file: An adaptable, symmetric multikey file structure. ACM *Transactions on Database Systems*, 9(1):38–71.

[Page, 1954] Page, E. (1954). Continous inspection schemes. *Biometrika*, 41:100–115.

[Petitcolas *et al.*, 1998] Petitcolas, F. A. P., Anderson, R. J., & Kuhn, M. G. (1998). Attacks on copyright marking systems. In Aucsmith, D., editor, *Second Workshop on Information Hidding, Portland, Oregon, USA*, pages 218–238.

[Poynton, 1997] Poynton, C. A. (1997). Frequently asked questions about color.

[Rijsbergen, 1979] Rijsbergen, C. V. (1979). *Information Retrieval*. Butterworth.

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

[Roussopoulos *et al.*, 1995] Roussopoulos, N., Kelley, S., & Vincent, F. (1995). Nearest neighbor queries. In *Proceedings of the* ACM SIGMOD *International Conference on Management of Data, San Jose, California, USA*, pages 71–79.

[Roussopoulos & Leifker, 1985] Roussopoulos, N. & Leifker, D. (1985). Direct spatial search on pictorial databases using packed r-trees. In *Proceedings of the* ACM SIGMOD *International Conference on Management of Data, Austin, Texas, USA*, pages 17–31.

[Salton, 1988] Salton, G. (1988). *Automatic Text Processing.* Addison-Wesley.

[Samet, 1990] Samet, H. (1990). *The Design and Analysis of Spatial Data Structures.* Addison-Wesley.

[Saporta, 1990] Saporta, G. (1990). *Probabilités, analyse des données et statistique.* Editions TECHNIP.

[Schmid, 1996] Schmid, C. (1996). *Appariement d'images par invariants locaux de niveaux de gris.* PhD thesis, Institut National Polytechnique de Grenoble, GRAVIR – IMAG – INRIA Rhône–Alpes.

[Schmid & Mohr, 1997] Schmid, C. & Mohr, R. (1997). Local grayvalue invariants for image retrieval. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 19(5):530–534.

[Schmid *et al.*, 2000] Schmid, C., Mohr, R., & Bauckhage, C. (2000). Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2):151–172.

[Scott & Thompson, 1983] Scott, D. & Thompson, J. R. (1983). Probability density estimation in higher dimensions. In *The 15th Symposium on the Interface, North Holland-Elsevier, Amsterdam, New York, Oxford.*, pages 173–179.

[Seeger & Kriegel, 1990] Seeger, B. & Kriegel, H.-P. (1990). The buddy-tree: An efficient and robust access method for spatial data base systems. In *Proceedings of the 16th International Conference on Very Large Data Bases, Brisbane, Queensland, Australia*, pages 590–601.

[Sellis, 1988] Sellis, T. K. (1988). Multiple-query optimization. ACM *Transactions on Database Systems*, 13(1):23–52.

[Sellis *et al.*, 1987] Sellis, T. K., Roussopoulos, N., & Faloutsos, C. (1987). The r^+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases, Brighton, England*, pages 507–518.

[Silverman, 1986] Silverman, B. (1986). *Density Estimation for Statistics and Data Analysis.* Chapman and Hall.

[Smeulders *et al.*, 2000] Smeulders, A. W. M., Worring, M., Santini, S., Gupta, A., & Jain, R. (2000). Content-based image retrieval at the end of the early years. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380.

[Stricker & Swain, 1994] Stricker, M. & Swain, M. (1994). The capacity of color histogram indexing. In *Proceedings of the Conference on Computer Vision and Pattern Recognition, Seattle, Washington, USA.*

[Traina Jr. *et al.*, 2000] Traina Jr., C., Traina, A. J. M., Wu, L., & Faloutsos, C. (2000). Fast feature selection using fractal dimension. In *The XV Brazilian Database Symposium*, pages 158–171, Joao Pessoa, Paraiba, Brazil.

[Veillon, 1984] Veillon, F. (1984). Utilisation de la couleur pour une visualisation en 5 dimensions. Technical Report 472, Informatique et Mathématiques Appliquées de Grenoble (IMAG).

[Veltkamp & Tanase, 2000] Veltkamp, R. C. & Tanase, M. (2000). Content-based image retrieval systems: A survey. Technical Report UU-CS-2000-34, Department of Computing Science, Utrecht University.

[Verleysen, 2003] Verleysen, M. (2003). Learning high-dimensional data. In V. Piuri, M. Gori, S. A. & Goras, L., editors, *Limitations and future trends in neural computation*, pages 141–162. IOS Press.

[Vogel & Schiele, 2002] Vogel, J. & Schiele, B. (2002). On performance characterization and optimization for image retrieval. In Heyden, A., Sparr, G., Nielsen, M., and Johansen, P., editors, *Proceedings of the 7th European Conference on Computer Vision, Copenhagen, Danemark*, pages 49–66.

[Weber & Böhm, 2000] Weber, R. & Böhm, K. (2000). Trading quality for time with nearest neighbor search. In *Proceedings of the 7th Conference on Extending Database Technology, Konstanz, Germany*, pages 21–35.

[Weber *et al.*, 1998] Weber, R., Schek, H.-J., & Blott, S. (1998). A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases, New York City, New York, USA*, pages 194–205.

[Wenyin *et al.*, 2001] Wenyin, L., Su, Z., Li, S., Sun, Y.-F., & Zhang, H. (2001). A performance evaluation protocol for content-based image retrieval algorithms/systems. In *Proceedings of the* IEEE *Workshop on Empirical Evaluation Methods in Computer Vision, Hawaii, USA.*

[White & Jain, 1996] White, D. A. & Jain, R. (1996). Similarity indexing with the ss-tree. In *Proceedings of the 12th International Conference on Data Engineering, New Orleans, Louisiana, USA*, pages 516–523.

[Wu *et al.*, 2000] Wu, P., Manjunath, B. S., & Shin, H. D. (2000). Dimensionality reduction for image search. In *Proceedings of the 7th* IEEE *International Conference on Image Processing, Vancouver, Canada.*

[Xu *et al.*, 1999] Xu, X., Jäger, J., & Kriegel, H.-P. (1999). A fast parallel clustering algorithm for large spatial databases. *Data Mining and Knowledge Discovery*, (3):263–290.

[Zhang *et al.*, 1996] Zhang, T., Ramakrishnan, R., & Livny, M. (1996). Birch: An efficient data clustering method for very large databases. In *Proceedings of the* ACM SIGMOD *International Conference on Management of Data, Montreal, Canada*, pages 103–114.

Références de l'auteur

Revues scientifiques

L. Amsaleg, P. Gros, S.-A. Berrani. – Robust Object Recognition in Images and the Related Database Problems. *Special issue of the Journal of Multimedia Tools and Applications*, 23(3), 2004.

S.-A. Berrani, L. Amsaleg, P. Gros. – Recherche approximative de plus proches voisins : application à la reconnaissance d'images par descripteurs locaux. *RSTI - Technique et Science Informatiques. Indexation par le contenu visuel*, 22(9):1201–1230, 2003.

S.-A. Berrani, L. Amsaleg, P. Gros. – Recherche par similarité dans les bases de données multidimensionnelles : panorama des techniques d'indexation. *RSTI - Ingénierie des systèmes d'information. Bases de données et multimédia*, 7(5-6):9-44, 2002.

Congrès internationaux

S.-A. Berrani, L. Amsaleg, P. Gros. – Robust Content-Based Image Searches for Copyright Protection. *Proc. of the* ACM *International Workshop on Multimedia Databases, MMDB'03*, La Nouvelle Orléans, Louisiane, USA. Novembre 2003.

S.-A. Berrani, L. Amsaleg, P. Gros. – Approximate Searches: k-Neighbors+Precision. *Proc. of the 12th* ACM *International Conference on Information and Knowledge Management, CIKM'03*, La Nouvelle Orléans, Louisiane, USA. Novembre 2003.

L. Amsaleg, P. Gros, S.-A. Berrani. – A Robust Technique to Recognize Objects in Images, and the DB Problems it Raises. *Proc. of the Workshop on Multimedia Information Systems, MIS'01*, Capri, Italie. Novembre 2001.

A. Rahmoun, S.-A. Berrani. – A Genetic-based Neuro-Fuzzy Generator: NEFGEN. *Proc. of the* ASC/IEEE *International Conference on Computer Systems and Applications, CSA'00*, Beyrouth, Liban. Juin 2000.

Congrès nationaux

S.-A. Berrani, L. Amsaleg, P. Gros. – Recherche d'images par le contenu pour la détection des copies. *14^e Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle, RFIA'04*, Toulouse, France. Janvier 2004.

S.-A. Berrani, L. Amsaleg, P. Gros. – Probabilistically Controlling the Precision of Approximate Nearest-Neighbor Searches. *Actes des 19^e journées Bases de Données Avancées, BDA'03*, Lyon, France. Octobre 2003.

Brevet

S.-A. Berrani, L. Amsaleg, P. Gros. – Nearest Neighbor Searches in Multidimensional Databases. *Déposé par Thomson.*

Séminaire invité

S.-A. Berrani. – Recherche approximative des k-plus proches voisins dans les bases de données multimédia. *Journée AS fouille de données multimédia sur le thème « Analyse de données, Statistique et Apprentissage pour la Fouille d'images »*, Paris, France. 1er avril 2003.

Résumé

Cette thèse s'intéresse aux systèmes de recherche d'images par le contenu, où le but est de rechercher les images les plus similaires à une image requête fournie en exemple. Pour cela, il est d'abord nécessaire de décrire les images stockées dans la base. L'objectif est de traduire la similarité visuelle entre images en une simple notion de proximité spatiale entre descripteurs. Ainsi, pour retrouver les images similaires à l'image requête, il suffit de retrouver les descripteurs les plus proches du descripteur de l'image requête. Ce mode de recherche, appelé recherche de plus proches voisins, est cependant extrêmement coûteux même lorsque des techniques d'indexation multidimensionnelle sont utilisées. Les performances de celles-ci se dégradent exponentiellement lorsque la dimension des descripteurs augmente (phénomène de la « malédiction de la dimension »). Il s'avère toutefois que l'on peut fortement réduire le coût de ces recherches en effectuant des recherches approximatives. Le principe est alors de négocier une réduction du temps de réponse contre l'introduction d'imprécisions durant la recherche.

Ce travail reprend ce principe et propose une nouvelle méthode de recherche approximative de plus proches voisins qui permet un contrôle fin et intuitif de la précision de la recherche. Ce contrôle s'exprime au travers d'un seul paramètre qui indique la probabilité maximale de ne pas retrouver un des plus proches voisins recherchés.

Dans le but d'évaluer rigoureusement ses performances dans le cadre d'un système réel, la méthode proposée a été ensuite intégrée dans un système de recherche d'images pour la détection de copies. Les expérimentations effectuées montrent que la méthode proposée est efficace, fiable et son imprécision n'a qu'un impact mineur sur la qualité des résultats finaux.

Mots clefs : indexation multimédia, recherche approximative de plus proches voisins, malédication de la dimension, recherche d'images par le contenu.

Abstract

This thesis is concerned with content-based image retrieval (CBIR) systems. These systems retrieve the images that are the most similar to a query one within an image database. This requires to describe automatically the images such that the visual similarity between two images becomes proportional to the distance between their descriptors. Retrieving the most similar images of the query image is therefore achieved by searching for the nearest-neighbors of the descriptor of the query image. This search mode is however inherently expensive when the database is very large and even when multidimensional indexing techniques are used. The dimensionality curse problem makes these techniques inefficient in high dimensional spaces. It is possible, however, to reduce the retrieval cost by searching for the approximate neighbors of the query descriptors instead of searching for the exact result.

In this thesis, this principle is used and a new approach for performing efficient approximate nearest-neighbor searches in high-dimensional databases is proposed. This approach allows a fine and intuitive control over the precision of the search by setting the maximum probability to miss one of the exact nearest neighbors.

In addition, this method is used to perform similarity searches within a CBIR system for image copy identification. The idea is to evaluate the method within a real system. This evaluation shows that it is efficient, reliable and its imprecision has only a little impact on the final results.

Keywords: Multimedia Indexing, Approximate Nearest-Neighbor Searches, Dimensionality Curse, Content-Based Image Retrieval.