Euclidean lattices for high dimensional indexing and searching

Loïc Paulevé

I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1903

EUCLIDEAN LATTICES FOR HIGH DIMENSIONAL INDEXING AND SEARCHING

LOÏC PAULEVÉ

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

IRISA

Institut de Recherche en Informatique et Systèmes Aléatoires
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

Euclidean lattices for high dimensional indexing and searching

Loïc Paulevé *

Systèmes symboliques
Projet Texmex

Publication interne n1903 — Septembre 2008 — 57 pages

Abstract: For similarity based searching, multimedia data are represented by one or more numerical vectors: we search the nearest neighbors of the query. Because of the huge number of these data and their high dimension, classical indexing technics are inefficient. The goal of this internship is to study the use of euclidean lattices for database indexing. Lattices have nice properties: they are spatial quantizers, thereby generate a partition of the space and decoding (quantization step) may be done very quickly. Then, we hope to be able to rapidly find a small space region containing data similar to a given query point, without reading all the database.

Key-words: euclidean lattice, indexing, searching, k nearest neighbors, high dimensional space, Sift descriptor, permutohedron, nearest faces of lattice, k nearest lattice points

(Résumé : tsvp)

From the master internship report, year 2008

Issu du rapport de stage de master, année 2008

* École normale supérieure de Cachan/Antenne de bretagne

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Réseaux géométriques pour l'indexation et la recherche en grandes dimensions

Résumé : À des fins de recherche par similarité, les données multimédia sont représentées par un ou plusieurs vecteurs numériques : on cherche alors les plus proches voisins de la requête. Le volume de telles bases (jusqu'à des milliards de vecteurs) et la grande dimension des vecteurs (autour de 100) rendent les méthodes classiques d'indexation impraticables. Le but de ce stage est d'étudier l'utilisation de réseaux géométriques pour indexer notre base de données. Les réseaux présentent de belles caractéristiques : ce sont des quantificateurs spatiaux, ils forment ainsi une partition implicite de l'espace ; le décodage (quantification) peut être très rapide. On peut ainsi espérer cibler rapidement une zone de l'espace contenant des données similaires à un point requête, sans avoir à parcourir toute la base.

Mots clés : réseau euclidien, indexation, recherche, k plus proches voisins, espace de grande dimension, descripteur Sift, permutoèdre, plus proches faces d'un réseau, k plus proches points de réseau

Table des matières

Introduction

Les données multimédia (images, textes, sons, vidéos) ont deux principales caractéristiques : elles sont nombreuses, et, à des fins de recherche selon une similarité de contenu, sont représentées numériquement dans des espaces de grandes dimensions.

Les propriétés de ces espaces entraînent l'apparition de phénomènes particuliers exigeant des méthodes de recherche au sein des bases qui soient robustes aux grandes dimensions.
De nombreuses stratégies de recherche de données existent. La plupart organisent la base de données autour d'une structure d'arbre. Ainsi, à partir d'une requête, les algorithmes vont naviguer dans cet arbre jusqu'à trouver un emplacement contenant des données similaires. Cette navigation impose de nombreux accès mémoires pour charger les différents noeuds au fur et à mesure du parcours de l'arbre.

Ce stage propose d'étudier une nouvelle approche pour l'indexation et la recherche de données numériques : l'utilisation de réseaux géométriques.

Les réseaux présentent de belles caractéristiques : ce sont des quantificateurs vectoriels, ils forment ainsi une partition implicite de l'espace ; l'étape de quantification (décodage) se fait de manière algébrique et peut être très rapide.
On peut ainsi espérer cibler rapidement une région de l'espace contenant des données similaires à un point requête. Nous évitons alors un parcours complet de la base de données, sans utiliser de procédé de navigation.

Après un tour d'horizon des méthodes d'indexation travaillant directement dans la dimension des données, nous présentons les réseaux géométriques et leurs principales caractéristiques. Nous étudions dans un premier temps l'utilisation d'un réseau unique pour indexer et rechercher des données dans la dimension d'origine.

Chapitre 1

Indexation, recherche par similarité et grandes dimensions

1.1 Définitions

Le but d'un algorithme de recherche est ici de trouver dans une base de données l'ensemble des points similaires à une requête. Les éléments de la base de données et ceux utilisés comme requête sont des vecteurs numériques de grandes dimensions. La similarité est définie en terme de distance. Plus la distance entre deux points est faible, plus ils sont similaires.

Cette recherche de données est divisée en deux phases : une phase d'indexation de la base de données et une phase de recherche.
L'**indexation** consiste à organiser, préparer, la base de données. Cette phase est dite hors-ligne, car elle s'effectue avant toute recherche, et ce une seule fois.
La **recherche** prend un vecteur requête en entrée et utilise un algorithme qui profite pleinement de cette indexation, confinant la recherche de données similaires à une toute petite région de l'espace.

Deux principaux types de recherche existent :
- La recherche à *boule* ε renvoie tous les points contenus dans la boule de rayon ε centrée sur la requête.
- La recherche des k *plus proches voisins* renvoie les k points les proches de la requête.

Le principal inconvénient de la recherche à boule ε est que l'on ne contrôle pas le nombre de résultats renvoyés. À l'inverse, une recherche de k plus proches voisins retourne toujours le nombre de résultats prévu, mais c'est la distance maximum entre les points qui n'est pas contrôlée.
La recherche des k plus proches voisins peut être vue comme une recherche à boule ε_q où

ε_q est la distance entre la requête q et le k-ième plus proche voisin. Ce ε_q dépend complètement de l'emplacement de q dans l'espace des données. Si q est dans une région peu peuplée, ε_q sera très grand, à l'inverse, si q est dans une région très peuplée, ε_q sera très petit.

Notre étude se focalise sur la recherche de similarité par k-plus proches voisins avec la distance euclidienne.

Une implémentation naïve de cette recherche est un algorithme séquentiel et ne requiert aucune phase d'indexation : on compare la requête à tous les points existants dans la base, en conservant les k plus proches.

Pour améliorer la performance des recherches, la plupart des méthodes utilisent un processus de navigation dans les données organisées selon une arborescence, dans le but de confiner la recherche pour n'analyser qu'une faible partie des données. Nous en exposons les principales techniques.

1.2 Hiérarchies de cellules

Pour éviter de comparer notre requête à tous les éléments de la base de données, une stratégie est de cibler une région de l'espace contenant, a-priori, nos plus proches voisins, et d'ignorer les points hors de cette région.

L'idée est alors de regrouper les données en cellules, et d'éliminer, lors de la recherche, celles ne pouvant pas contenir de points appartenant au résultat final, pour ne conserver qu'une liste restreinte de cellules candidates. Les points contenus dans ces cellules seront alors comparés avec la requête, évitant alors de calculer les distances à tous les points de l'espace.

Comme nous comparons la requête à toutes nos cellules, le nombre de cellules doit être très petit devant le nombre de données.

L'élimination de ces cellules se fait à l'aide de règles géométriques simples, basées sur des calculs de distances entre la requête et les bords de la cellule.

Pour chaque cellule, on calcule la distance minimale $dmin$ et maximale $dmax$ au point requête :

$$\text{si } dmin(q, C_i) \geq dmax(q, C_j), \text{ alors éliminer la cellule } C_i$$

Ce processus est illustré par la figure 1.1 avec des cellules rectangulaires.

De même, pendant le processus de recherche, toute cellule dont la distance minimale est plus grande que le k-ième plus proche voisin courant peut être éliminée.

L'efficacité de ces règles depend fortement de la forme des régions pour le calcul des distances, ainsi que de la dimension de l'espace courant. Plus la dimension de l'espace va être grande, plus les cellules tendent à devenir équidistantes. Il devient alors difficile de discriminer les différentes cellules [Weber et al., 1998]. Ainsi, peu de cellules sont éliminées, ce qui entraîne un coût de recherche élevé.

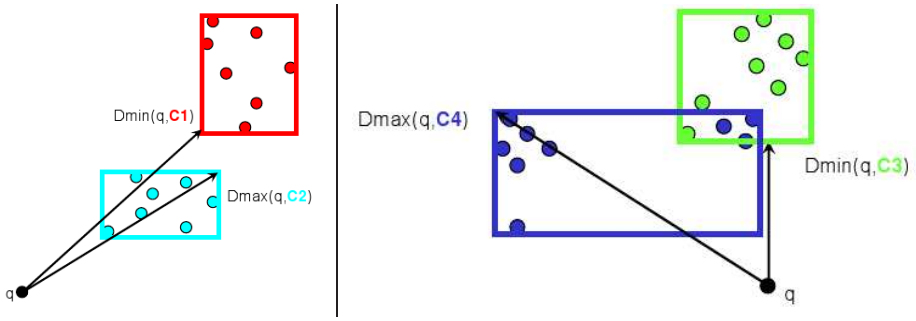

FIG. 1.1 – Illustration des règles de filtrage des cellules. À gauche, $dmin(q, C1) > dmax(q, C2)$, la cellule C_1 peut être éliminée. À droite, cette régle n'est pas applicable : les cellules $C3$ et $C4$ se chevauchent.

Pour avoir un nombre de cellules candidates très réduit, on peut vouloir organiser les cellules hiérarchiquement : les cellules d'un niveau englobent géométriquement celles des niveaux inférieurs.
La recherche se fait en descendant progressivement dans l'arborescence ainsi construite.

L'article [Berrani et al., 2002] présente un panorama de différentes méthodes de création de cellules. Nous commençons par présenter une méthode classique basée sur un partitionnement des données, l'algorithme de R-Tree.

Partionnement des données : le R-Tree

Les données sont regroupées par proximité au sein d'une cellule. Lorsque la contenance de la cellule dépasse un seuil fixé, un algorithme de partionnement divise cette cellule en sous-cellules. Une structure d'arbre est ainsi formée, les données étant dans les cellules "feuilles". Lors de l'insertion d'un nouveau point, trois cas sont possibles :

- Le point appartient à une seule cellule feuille, on l'ajoute à cette cellule.
- Le point appartient à plusieurs cellules feuilles, on l'ajoute dans la cellule de plus petit volume.
- Le point n'appartient à aucune cellule feuille, on agrandit alors la cellule feuille la plus proche ; cette opération tend à accroître le chevauchement

Le partionnement d'une cellule en sous-cellules va directement influer sur la performance de la recherche. Pour pouvoir éliminer le plus de cellules candidates possible, il faut limiter leur chevauchement. De plus, il faut minimiser le volume des cellules : plus une cellule aura un grand volume, plus les données présentes à l'intérieur peuvent être très éloignées, donc peu similaires.

De nombreuses variantes existent, proposant différentes formes de régions et différentes méthodes de partitionnement des cellules.

En grandes dimensions, le volume et le chevauchement entre les cellules devient très important, et il devient ainsi très difficile d'éliminer des cellules à l'aide de règles géométriques simples.

Pour éviter un chevauchement des cellules, nous pouvons utiliser une méthode de partitionnement de l'espace. Nous exposons la principale, l'algorithme de K-D-Tree.

Partionnement de l'espace : le K-D-Tree

Le K-D-Tree est une méthode récursive de partionnement de l'espace. On commence par diviser l'espace en 2 selon une dimension choisie, par exemple celle présentant la plus grande variance. La division se fait selon la distribution des données sur cette dimension, en choisissant un hyperplan séparateur passant par la médiane des points. On applique récursivement cette division sur chaque partie ainsi créée en changeant de dimension, jusqu'à obtenir un nombre de points par cellule satisfaisant.
Un partionnement de l'espace ne crée que des cellules disjointes. L'inconvénient principal est qu'il tend à créer des régions avec de très grand volumes, on perd alors en cohérence spatiale : des points dans une même cellule peuvent être très éloignés.

De manière générale, les propriétés géométriques des espaces de grandes dimensions réduisent l'efficacité de la mise en cellule des points : pour le partionnement de données, le volume des chevauchements devient très important, et pour le partitionnement de l'espace, le nombre de cellules créées augmente exponentiellement selon la dimension.

La forme des régions devient également problématique avec les grandes dimensions :
- Les hyper-rectangles permettent de réduire facilement le chevauchement, mais leur diagonale va augmenter très rapidement avec la dimension : deux points situés aux angles opposés d'une telle cellule sont très éloignés.
- Les hyper-sphères garantissent une homogénéité des distances entre les points, mais il est difficile d'éviter leur chevauchement.

Dans la section suivante, nous abordons plus précisement l'effet des grandes dimensions sur la recherche de plus proches voisins.

1.3 Impact de la dimension

1.3.1 La malédiction

L'article [Weber et al., 1998] étudie le comportement de données *uniformément et indépendamment distribuées* dans des espaces de grandes dimensions.

Plusieurs propriétés intéressantes sont mises en avant :
- un partitionnement de l'espace crée un nombre de cellules exponentiel selon la dimension,
- les cellules résultantes d'un partitionnement de l'espace ont un nombre de cellules voisines exponentiel selon la dimension,
- pour conserver une densité de points lorsque l'on augmente la dimension, il faut augmenter exponentiellement le nombre de points.

Ainsi, les espaces de grandes dimensions sont généralement très creux.

Les méthodes de recherche utilisant les cellules sont comparées à la recherche séquentielle.
- Pour toute méthode de partitionnement, il existe une dimension n au-delà de laquelle une simple recherche séquentielle offre de meilleures performances.
- La complexité de toute méthode de partitionnement tend à être linéaire au nombre de données (toutes les cellules sont visitées et elles contiennent toute une seule donnée, au mieux).

Ainsi, la plupart des processus à base de hiérarchies de cellules deviennent très coûteux (voire impraticables) dans les grandes dimensions.

Nous nous posons maintenant la question de la pertinence des plus proches voisins dans de tels espaces.

1.3.2 Pertinence des plus proches voisins

Une définition d'une recherche de plus proche voisin *instable* est proposée dans [Beyer et al., 1999] : si la distance entre un point requête et la quasi-totalité des autres points est inférieure à $(1 + \epsilon)$ fois la distance à son plus proche voisin, alors la requête est dite instable.

L'article présente un ensemble de conditions sur les lois de distributions des données qui, si elles sont vérifiées, tendent à montrer que la recherche de plus proches voisins est instable.
La recherche en grande dimension reste pertinente lorsque les données sont réparties en clusters : des petites régions de l'espaces ont une densité de points très forte, et ces régions sont distinctes les unes des autres.
Une requête appartenant à une région très peuplée, un cluster, aura un voisinage stable, la recherche est alors pertinente. À l'inverse une requête n'appartenant à aucun cluster sera considérée à égale distance de tous les clusters, les plus proches voisins sont alors instables.

La recherche de plus proche voisins reste pertinente pour des données dont la distribution est très éloignée de l'uniformité, ce qui est le cas de la plupart des bases de données réelles.

1.4 Recherche approximative

Les méthodes approximatives ne lisent qu'une partie des données, et ne renvoient pas toujours les k vrais plus proches voisins d'une requête.
On cherche alors des stratégies efficaces pour trouver au plus vite un plus grand nombre de plus proches voisins possible, en lisant une faible proportion de données.

En s'autorisant à "oublier" des plus proches voisins, on peut imaginer réduire les effets des grandes dimensions qui ralentissent considérablement les processus hiérarchiques.
Nous savons que les méthodes de partitionnement des données souffrent d'un problème de chevauchement des cellules lorsque la dimension augmente. Pour diminuer ce chevauchement, on réduit arbitrairement la taille des cellules. Ceci aura pour effet d'ignorer les points sur les bords des cellules, qui sont potentiellement des plus proches voisins.
Le choix de la réduction est problématique : une réduction trop grande entraînera la perte de trop de données, une trop petite ne permettra pas un gain significatif sur le temps d'exécution.
D'autres techniques estiment la probabilité qu'une cellule contienne le plus proche voisin, seules celles dépassant un seuil sont alors parcourues.

Outre comparer les temps d'éxecutions pour effectuer une recherche, on peut juger la qualité d'une recherche approximative en traçant sa courbe % rappel - % données lues. Le pourcentage de rappel correspond à la proportion des k plus proches voisins retournés par la méthode approximative parmis les k rééls plus proches voisins. La figure 1.2 illustre cette mesure.

Le rapport $\frac{\%\text{ rappel}}{\%\text{ données lues}}$ donne une indication sur la vitesse de découverte des vrais plus proches voisins. Ce rapport est de 1 pour l'algorithme séquentiel.
Un bon algorithme approximatif aura à la fois un rappel et une vitesse de découverte importante.

Ces mesures nous servirons à évaluer l'utilisation des réseaux géométriques appliqués à la recherche approximative de k plus proches voisins.

Le chapitre suivant se consacre à l'étude des réseaux géométriques pour indexer et rechercher des données dans leur dimension originale, sans procédé hiérarchique.

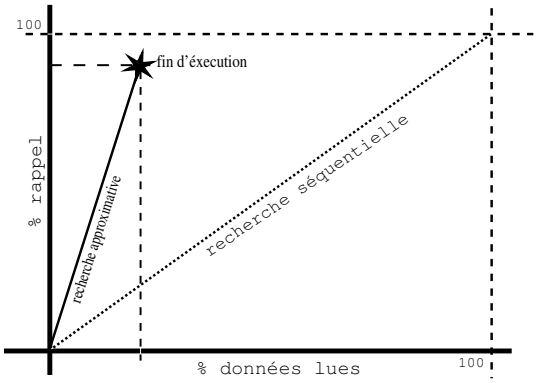

FIG. 1.2 – Illustration des courbes % rappel - % données lues : en pointillés est représenté le comportement moyen de l'algorithme séquentiel ; en trait plein est un exemple de courbe qui pourrait être obtenue par un algorithme approximatif.

Chapitre 2

Indexation et recherche avec un réseau

2.1 Introduction aux réseaux

Nous présentons ici les principales caractéristiques des réseaux. Les réseaux sont traités de manière approfondie dans le livre [Conway and Sloane, 1998].

2.1.1 Définitions

Nous appelons **réseau géométrique** ou **réseau de** $\mathbb{R}^n$ (**lattice** en anglais) un sous-groupe additif discret de $\mathbb{R}^n$ engendré par une base B de m vecteurs $b_1, \cdots, b_m \in \mathbb{R}^n$:

$$\Lambda = \{\eta_1 b_1 + \cdots \eta_m b_m : \eta_i \in \mathbb{Z}\}$$

Λ est le sous-ensemble de $\mathbb{R}^n$ contenant toutes les combinaisons linéaires entières de sa base B.
On dit alors que Λ est de dimension m.

La figure 2.1 montre un exemple de réseau de dimension 2, généré par une base de deux vecteurs v_1, v_2.

La **région de Voronoï** d'un point $P_i \in \Lambda$ est l'ensemble des points de $\mathbb{R}^n$ dont P_i est le plus proche voisin, selon une fonction de distance d :

$$\mathcal{V}_\Lambda(P_i) = \{x \in \mathbb{R}^n : \forall P_j \in \Lambda, d(P_i, x) \leq d(P_j, x)\}$$

Les régions de Voronoï de chaque point ont la même forme. Il s'agit de la *région fondamentale* du réseau.

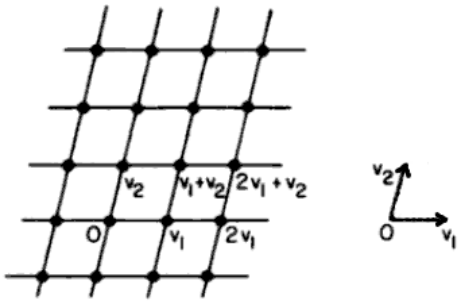

FIG. 2.1 – Un réseau de dimension 2

Le **déterminant** du réseau Λ est défini par $det(\Lambda) = det(BB^t)$. Ce déterminant représente le carré du volume de la région fondamentale de Λ, formé par la base B.

Un réseau peut être vu comme un pavage de $\mathbb{R}^n$ avec une unique tuile, la région fondamentale, centrée sur tous les points du réseau.

2.1.2 Géométrie d'un réseau

Nous nous intéressons à l'aspect géométrique des réseaux. Plus particulièrement, à la géométrie de la région de Voronoï fondamentale formée par un réseau Λ de dimension n.

Pour tout réseau, cette région est un *polytope régulier* de dimension n. Un polytope peut être vu comme une généralisation du polygone à la dimension n. Un polytope est régulier si toutes ses arrêtes ont la même longueur, et tous les angles entre ses faces sont égaux. Les polytopes réguliers sont étudiés dans le livre [Coxeter, 1973].

Une face F d'un polytope de dimension n est un hyperplan de dimension $n-1$, c'est la frontière entre deux points $\alpha, \beta \in \Lambda$:

$$\forall x \in F, d(x, \alpha) = d(x, \beta)$$

Le nombre de faces, de sommets, les angles ainsi que la longueur des arrêtes détermine complètement le réseau. Deux réseaux ayant une région fondamentale identique sont équivalents.

Différentes mesures sont établies pour spécifier un réseau. Pour la distance euclidienne, elles vont principalement comparer la forme de la région à une sphère.

On note V_n le volume d'une sphère de rayon 1 en dimension n.

$$V_n = \frac{\pi^{n/2}}{(n/2)!} = \frac{2^n \pi^{(n-1)/2}((n-1)/2)!}{n!}$$

Le volume d'une sphère de rayon ρ en dimension n vaut alors $V_n \rho^n$.

Compacité

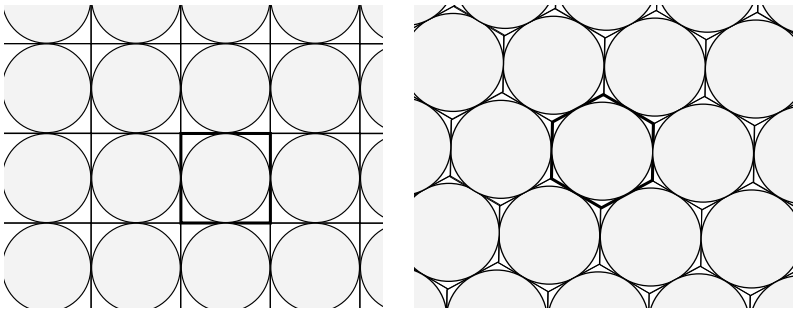

FIG. 2.2 – Le réseau de droite est plus compact que le réseau de gauche

On définit le rayon de compacité ρ comme le rayon de la sphère inscrite dans la région de Voronoï du réseau.
La compacité Δ est la propotion d'espace occupé par les sphères de rayon ρ centrées sur les points du réseaux. La figure 2.2 illustre cette mesure.

$$\Delta = \frac{\text{volume d'une sphère de rayon } \rho}{\text{volume de la région fondamentale}} = V_n \rho^n (det\Lambda)^{-1/2} \leq 1$$

Plus un réseau sera compact, plus la cohérence spatiale des régions formées est forte.

Recouvrement

On définit le rayon de recouvrement R comme le rayon minimal de la sphère englobant la région de Voronoï du réseau.
Le recouvrement Θ est la proportion d'espace recouvert par les sphères de rayon R, centrées sur les points du réseaux. Le recouvrement est supérieur à 1, plus il sera élevé, plus les sphères se chevauchent. La figure 2.3 illustre cette mesure.

$$\Theta = \frac{\text{volume d'une sphère de rayon } R}{\text{volume de la région fondamentale}} = V_n R^n (det\Lambda)^{-1/2} \geq 1$$

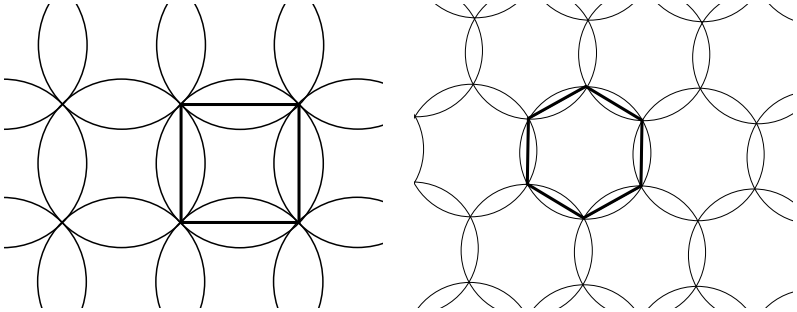

FIG. 2.3 – Le réseau de droite est plus fin que celui de gauche

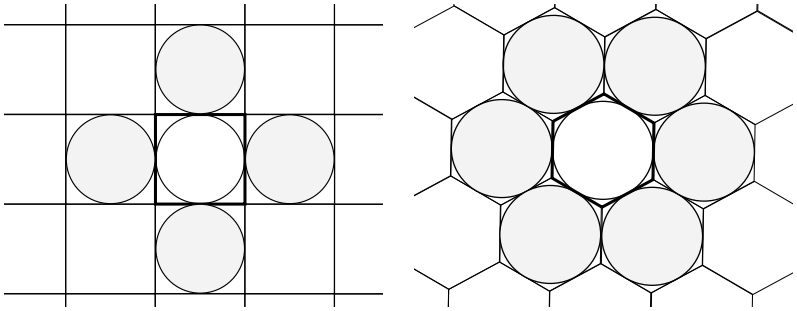

FIG. 2.4 – Le nombre d'embrassements du réseau de gauche est 4, il est de 6 pour celui de droite

Nombre d'embrassements

On centre les sphères de rayon ρ sur les points du réseau. Le nombre d'embrassements (ou kissing number) est le nombre de sphères qui se touchent. La figure 2.4 illustre cette mesure.
On notera que le nombre d'embrassements est inférieur ou égal au nombre de faces de la région.

2.1.3 Recherche du point de réseau le plus proche

Étant donné un réseau Λ de dimension n et un point requête $q \in \mathbb{R}^n$, trouver le point $P \in \Lambda$ le plus proche de q. Il s'agit d'une étape de *quantification vectorielle* que l'on appelera *décodage*.

La recherche du point le plus proche dans un réseau quelconque est abordée notamment dans [Mow, 2003]. La méthode proposée est la suivante :

1. Exprimer q dans la base de Λ
2. Énumérer les points de réseau situés dans une sphère centrée sur q
3. Retourner le point de réseau le plus proche

La complexité d'une telle recherche va dépendre principalement du nombre de points énumérés.

Notre étude se concentrera sur des réseaux particuliers pour lesquels il existe des algorithmes rapides de décodage. On note ϕ_Λ la fonction qui à q associe le point de Λ le plus proche.

Voici la description des réseaux particuliers que nous avons étudiés, ainsi que des algorithmes de décodage simples tirés de l'article [Conway and Sloane, 1982]. La distance euclidienne est notée $l2$; x_i est la i-ème composante du vecteur x.

$\mathbb{Z}^n$ **:** $\mathbb{Z}^n = \{(x_1, \cdots, x_n) : x_i \in \mathbb{Z}\}$ est le *réseau entier* ou *réseau cubique* de dimension n. Le décodage d'un point q s'effectue en arrondissant ses coordonnées à l'entier le plus proche.
L'entier le plus proche de x est noté $\lfloor x \rceil$.

Algorithme 1. $\phi_{\mathbb{Z}^n}$
Entrées: $q \in \mathbb{R}^n$
Sorties: $x \in \mathbb{Z}^n$
  $x \leftarrow (0)^n$
  pour $i = 1$ à n **faire**
    $x_i \leftarrow \lfloor q_i \rceil$
  fin pour

Complexité. $O(n)$

D_n **:** $n \geq 3, D_n = \{(x_1, \cdots, x_n) \in \mathbb{Z}^n : \sum x_i \text{ pair}\}$. Ce réseau de dimension n peut être vu comme un damier (les cases noires appartenant à D_n, les blanches uniquement à $\mathbb{Z}^n$). On commence par décoder le point dans $\mathbb{Z}^n$. Si la somme des coordonnées obtenues n'est pas paire, on arrondira du mauvais côté la coordonnée maximisant la distance à son entier le plus proche.

Algorithme 2. ϕ_{D_n}
Entrées: $q \in \mathbb{R}^n$
Sorties: $x \in D_n$
  $x \leftarrow \phi_{\mathbb{Z}^n}(q)$
  si $\sum_{i=1}^{n} x_i$ n'est pas paire **alors**
    $d \leftarrow q - x$ (* distance des q_i à l'entier le plus proche *)
    $f \leftarrow 0$ (* distance maximale courante *)

pour $i = 1$ à n **faire**
si $|d_i| > |f|$ **alors**
$f \leftarrow d_i$
$c \leftarrow i$
finsi
fin pour
si $f < 0$ **alors**
$x_c \leftarrow x_c - 1$ (* correction de la coordonnée c *)
sinon
$x_c \leftarrow x_c + 1$
finsi
finsi

Complexité. $O(n)$

D_n^* **:** $D_n^* = \mathbb{Z}^n \cup ((\frac{1}{2})^n + \mathbb{Z}^n)$.

Algorithme 3. $\phi_{D_n^*}$
Entrées: $q \in \mathbb{R}^n$
Sorties: $x \in D_n^*$
$x \leftarrow \phi_{\mathbb{Z}^n}(q)$
$x' \leftarrow \phi_{\mathbb{Z}^n}(q - (\frac{1}{2})^n) + (\frac{1}{2})^n$
si $l2(q, x') < l2(q, x)$ **alors**
$x \leftarrow x'$
finsi

Complexité. $O(n)$

D_n^+ **:** $D_n^+ = D_n \cup ((\frac{1}{2})^n + D_n)$; lorsque $n = 8$, ce réseau est également connu sous le nom de E_8, c'est le réseau le plus compact en dimension 8.

Algorithme 4. $\phi_{D_n^+}$
Entrées: $q \in \mathbb{R}^n$
Sorties: $x \in D_n^+$
$x \leftarrow \phi_{D_n}(q)$
$x' \leftarrow \phi_{D_n}(q - (\frac{1}{2})^n) + (\frac{1}{2})^n$
si $l2(q, x') < l2(q, x)$ **alors**
$x \leftarrow x'$
finsi

Complexité. $O(n)$

A_n **:** $A_n = \{(x_0, x_1, \cdots, x_n) \in \mathbb{Z}^{n+1} : \sum x_i = 0\}$. Ce réseau de dimension n se situe sur l'hyperplan $\sum x_i = 0$ dans $\mathbb{R}^{n+1}$. Tout comme pour D_n, on commence par décoder notre point dans $\mathbb{Z}^n$ puis on corrige le résultat jusqu'à ce que notre point soit situé sur l'hyperplan $\sum x_i = 0$:

Algorithme 5. ϕ_{A_n}
Entrées: $q \in \mathbb{R}^{n+1}$
Sorties: $x \in A_n$
$x \leftarrow \phi_{\mathbb{Z}^{n+1}}(q)$
$e \leftarrow \sum_{i=0}^{n+1} x_i$ (* nombre d'erreurs *)
si $e \neq 0$ **alors**
$d \leftarrow q - x$ (* distance des q_i à l'entier le plus proche *)
si $e > 0$ **alors**
$C \leftarrow$ les indices des e plus petites coordonnées de d
pour tout $i \in C$ **faire**
$x_i \leftarrow x_i - 1$
fin pour
sinon
$C \leftarrow$ les indices des e plus grandes coordonnées de d
pour tout $i \in C$ **faire**
$x_i \leftarrow x_i + 1$
fin pour
finsi
finsi
Complexité. $O(n)$

A_n^* **:** $A_n^* = \bigcup_{i=0}^{n}([i] + A_n)$ $\qquad [i] = ((\frac{i}{n+1})^{n+1-i}, (-\frac{n+1-i}{n+1})^i)$

Algorithme 6. $\phi_{A_n^*}$
Entrées: $q \in \mathbb{R}^{n+1}$
Sorties: $x \in A_n^*$
$x \leftarrow \phi_{A_n}(q - [0]) + [0]$
pour $i = 1$ à n **faire**
$x' \leftarrow \phi_{A_n}(q - [i]) + [i]$
si $l2(q, x') < l2(q, x)$ **alors**
$x \leftarrow x'$
finsi
fin pour
Complexité. $O(n^2)$

Un algorithme de décodage de A_n^* en $O(n \log n)$ étapes est présenté dans l'article [Vaughan and Clarkson, 1999].

E_{24} **:** Le *réseau de Leech* est définit pour la dimension 24 seulement. C'est le réseau le plus compact qui puisse exister en dimension 24.
Même si il existe des algorithmes de décodage performants [Be'ery et al., 1989], le

nombre d'opérations à effectuer (de l'ordre de 5000) reste très important devant les réseaux que nous venons de présenter.

2.1.4 Indexation et recherche avec un réseau

Un réseau forme une partition de l'espace. Pour rechercher les plus proches voisins d'une requête q, nous voulons lire uniquement les données appartenant à la même région du réseau que q.
Nous mettons en avant trois phases pour indexer et rechercher des données avec un réseau.

Paramétrage des données

Hormis le choix du réseau, nous pouvons choisir d'ajuster plusieurs paramètres relatifs aux données :

Translation : un réseau contient obligatoirement le point 0^n, l'origine. On ne peut donc pas translater un réseau, mais appliquer la translation opposée à nos données.

Échelle : la taille des régions du réseau. Au lieu de modifier la base du réseau, on peut choisir d'appliquer l'échelle aux données. Ainsi, pour obtenir un réseau d'échelle w, on divise chaque vecteur de notre base par w.

Rotation : tout comme la taille des régions, on peut choisir de changer arbitrairement leur orientation.

Tous ces paramètres peuvent être appliqués directement aux données, sans modifier la base du réseau choisi. On pose TR la fonction qui applique à un vecteur ces transformations successives.

Indexation

La phase d'indexation consiste à créer un index pour un réseau Λ et une fonction TR de transformation des données. Tous les points appartenant à la même région de Voronoï du réseau seront regroupés ensemble, dans la même cellule. L'index est donc la liste des cellules et pour chacune, la liste des points s'y trouvant.

Algorithme 7. `INDEXATION`
Entrées: DB (* base de données *), Λ, TR
Sorties: $INDEX$ (* Index de la base de données DB *)
pour tout $x \in DB$ **faire**
$x' \leftarrow TR(x)$ (* transformation de la donnée *)
$lp \leftarrow \phi_\Lambda(x')$ (* plus proche point du réseau *)
$INDEX(lp) \leftarrow INDEX(lp) \cup \{x\}$ (* Ajout de la donnée à la cellule *)
fin pour

Recherche

En appliquant la même fonction de transformation au point requête q, on calcule directement la région à laquelle appartient q à l'aide des algorithmes présentés dans la section précédente. On effectuera la recherche des plus proches voisins uniquement sur les données appartenant à cette même région.

Algorithme 8. RECHERCHE
Entrées: $INDEX$, Λ, q (* la requête *)
Sorties: $RESULTS$ (* k-plus proches voisins trouvés *)
$q' \leftarrow TR(q)$ (* transformation de la requête *)
$lp \leftarrow \phi_\Lambda(q')$ (* plus proche point du réseau *)
$DB' \leftarrow INDEX(lp)$ (* données appartenant à la même région que la requête *)
$RESULTS \leftarrow$ recherche k-ppv(q, DB') (* recherche parmis les points de la même région *)

Nous allons maintenant détailler la phase de paramétrage et d'indexation, et présenter les résultats de nos premières expériences sur la recherche des k plus proches voisins avec un réseau unique.

2.2 Indexation

Les réseaux sont des quantificateurs vectoriels formant une partition régulière de l'espace. Ainsi, ils sont très bien adaptés aux sources de données considérées uniformes.
La première difficulté sera donc d'obtenir une partition équitable de nos données, non uniformément distribuées. Une cellule contenant trop de points réduira l'efficacité de notre recherche, à l'inverse, les cellules peu peuplées ne permettront pas de renvoyer un résultat pertinent.

Nous cherchons à créer un index tel que :

1. un faible pourcentage de la base est indexé dans des cellules de contenance faible (on notera ce pourcentage **%V**) ;
2. un faible pourcentage de la base est indexé dans la même cellule (on notera ce pourcentage **%P**)

Ces deux mesures nous servirons à juger la qualité d'une indexation par un réseau avec un paramétrage donné.

Les résultats qui suivront ont été établis en considérant comme cellules à contenance faible celles dont la population n'excède pas 10 points.

2.2.1 Jeu de données : base Sift

Les expériences que nous mènerons dans cette étude ont été effectuées sur une base de données de descripteurs locaux *Sift* d'images fixes.

FIG. 2.5 – Extrait du jeu d'images sur lesquels les descripteurs Sift ont été calculés.

Les descripteurs Sift sont des vecteurs de dimension 128 calculés à partir des points d'intérêts d'une image. Ils décrivent chacun une petite région de l'image. Les similarités entre descripteurs Sift se mesurent par la distance eulidienne, notée $l2$.

Notre base de données contient 1 million de descripteurs Sift obtenus sur un jeu de 300 images. La figure 2.5 montre un extrait de ce jeu d'images. Elle est téléchargeable à l'adresse internet
http://lear.inrialpes.fr/people/jegou/data.php.

L'utilisation de descripteurs Sift est très répandue dans le domaine du multimédia, nous pourrons ainsi nous comparer plus facilement à d'autres méthodes.

La dimensionnalité de ces descripteurs nous assure d'être confronté aux difficultés des grandes dimensions, tout en gardant un temps de calcul raisonnable pour des algorithmes linéaires en fonction de la dimension.
De plus, la taille du jeu d'images est suffisamment grande pour occasionner des observations stables.

D'autres jeux de données, utilisant des images et des descripteurs de dimensions différentes, ont également été étudiés. Les résultats et conclusions obtenus sont similaires à ceux que nous présentons ici.

Nous allons maintenant étudier l'indexation de cette base de données Sift par des réseaux avec différents paramètres.

2.2.2 Influence des paramètres d'un réseau

Échelle et translation

Le facteur d'échelle d'un réseau, noté w, permet d'ajuster la taille des régions formées. Par exemple, le réseau $\mathbb{Z}^n$ à l'échelle w formera des régions hyper-cubiques, de longueur d'arrête w, et de volume w^n.

Les cellules créées par un réseau avec un facteur d'échelle w contiendront potentiellement plus de points qu'un réseau avec un facteur d'échelle inférieur, $w' < w$.
Cependant, la translation du réseau va être déterminante. Comme l'illustre la figure 2.6, deux translations d'un même réseau peuvent donner des cellules avec des densités très différentes.

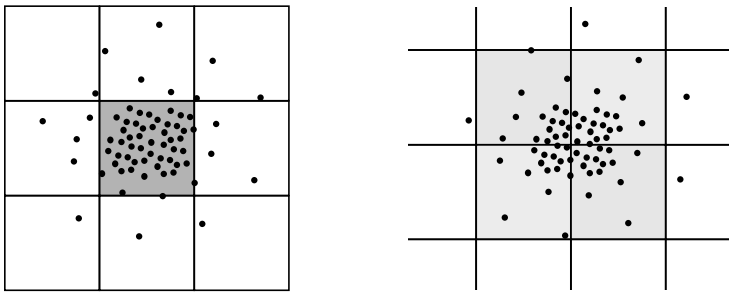

FIG. 2.6 – Illustration de l'effet d'une translation d'un réseau sur la population des cellules : à gauche une seule est très peuplée, en décalant le réseau, on obtient, à droite, 4 cellules avec une densité de points équivalente, bien plus faible.

Pour mettre en avant ce phénomène, on applique au réseau $\mathbb{Z}^{128}$ différentes translations aléatoires. Les composantes des translations sont tirées uniformément dans les valeurs réelles entre 0 et 1, le vecteur aléatoire ainsi construit est ensuite multiplié par une valeur entière tirée uniformément dans l'interval $[10, 1000]$.

Voici les résultats des mesures d'indexations obtenus sur notre jeu de données avec le réseau $\mathbb{Z}^{128}$, à l'échelle $w = 500$, sur 6 translations aléatoires. %V est la proportion de données indexées dans des cellules de moins de 10 points ; %P est la proportion de données indexées dans la cellule la plus peuplée. Les valeurs minimales et maximales sont en gras.

%V	**53.35**	17.48	26.96	27.31	30.62	**14.57**
%P	**0.29**	**4.01**	0.78	0.33	0.59	1.13

On constate bien une large différence entre différentes translations aléatoires. La translation apparaît comme un paramètre déterminant de la qualité d'indexation par un réseau.

La figure 2.7 montre l'influence du facteur d'échelle sur l'indexation de notre base de données par le réseau $\mathbb{Z}^{128}$. Pour chaque échelle, nous calculons la moyenne des mesures obtenues sur 6 translations aléatoires.

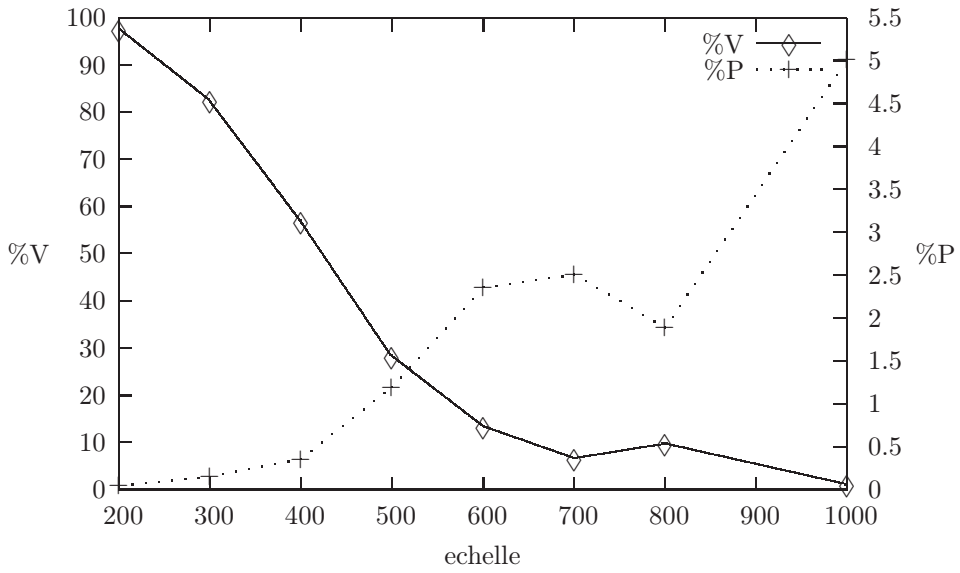

FIG. 2.7 – Évolution des mesures %V et %P en fonction de l'échelle w du réseau $\mathbb{Z}^{128}$

Une petite échelle tend à créer beaucoup de cellules peu peuplées mais, en contre partie, la population maximale des cellule est faible. À l'inverse, une grande échelle réduira le nombre de cellules peu peuplées, mais créera des cellules à forte population.
Le choix d'une échelle résulte d'un compromis entre ces deux mesures.

Choix du réseau

Un réseau détermine la forme des régions de Voronoï dans lesquelles seront indexées nos données. Tous les réseaux créent des régions convexes, des polytopes réguliers qui tendent plus ou moins vers une forme sphérique (voir section 2.1.2). Ces formes sont très comparables, ainsi seul le volume de la région va réellement influer sur la qualité de l'indexation. Le ta-

bleau 2.1 compare la compacité et le volume des différents réseaux étudiés en dimension 128.

	$\mathbb{Z}^{128}$	D_{128}	D^{+}_{128}	D^{*}_{128}	A_{128}	A^{*}_{128}
compacité	2.9410^{-39}	2.7110^{-20}	5.4210^{-20}	5.8810^{-39}	4.7710^{-21}	2.0310^{-38}
volume	1	2	1	0.5	11.36	0.09

TAB. 2.1 – Mesure de compacité et volume des régions pour les différents réseaux étudiés en dimension 128. Pour une meilleure lisibilité, la compacité à été divisée par le volume de la sphère de rayon 1.

Les résultats suivants ont été obtenus avec une échelle $w = 600$, et sont la moyenne des mesures obtenues sur 6 translations aléatoires.

réseau	$\mathbb{Z}^{128}$	D_{128}	D^{*}_{128}	D^{+}_{128}
%V	13.45	9.59	**17.12**	**8.73**
%P	2.36	1.83	**1.25**	**2.52**

Il n'y a pas de différences significatives entre les réseaux. On soulignera qu'il est difficile de comparer plusieurs réseaux sur une même échelle car le volume de leur région fondamentale est différent.

Rotations

Soit n la dimension de notre espace. On tire n vecteurs aléatoirement, que l'on rend orthogonaux entre eux (par le procédé de Gram-Schmidt). On projette chacun de nos points selon ces n vecteurs : on obtient une rotation aléatoire de nos données.

On effectue différentes rotations aléatoires. Pour chaque rotation, on calcule la moyenne des indicateurs d'indexation obtenus avec 6 réseaux translatés aléatoirement. On choisit alors la rotation offrant les meilleurs résultats et nous les comparons avec les résultats obtenus sans rotation. On se limite aux réseau $\mathbb{Z}^{128}$ et D^{*}_{128}.

réseau	$\mathbb{Z}^{128}$	$\mathbb{Z}^{128}$ + rot.	D^{*}_{128}	D^{*}_{128} + rot.
%V	13.45	9.06	17.12	16.90
%P	2.36	1.65	1.25	1.93

La rotation peut donc influer sur la qualité d'indexation de nos données. Toutefois, nous n'observons pas de variance aussi forte qu'avec le paramètre de translation.

2.2.3 Réductions de dimension

Pour mesurer l'influence de la dimension sur la l'indexation des données par un réseau, nous allons réduire la dimension de nos données.

Nous mettons en parallèle trois techniques pour réduire des données de dimension n dans un espace de dimension $n' < n$.

Analyse en composantes principales : on projette nos points sur les n' vecteurs qui expliquent le mieux la dispersion des données.

Sélection de dimensions : on choisit arbitrairement n' dimensions.

Projections aléatoires : on projette nos points sur n' vecteurs tirés aléatoirement.

Les résultats obtenus résultent d'une moyenne sur plusieurs translations aléatoires, la meilleure échelle a été choisie.

Réduction en dimension 24

Le réseau Leech possède la compacité la plus élevée possible dans cette dimension. Toutefois pour des raisons de temps, nous n'avons pas fait d'expériences sur ce réseau et nous nous sommes limités à $\mathbb{Z}^{24}$ et D_{24}^*.

	$\mathbb{Z}^{24}$			D_{24}^*		
	ACP	Sel.	Proj.	ACP	Sel.	Proj.
%V	5.36	12.93	9.21	8.63	8.23	16.10
%P	1.71	1.53	1.66	1.11	1.55	0.99

Réduction en dimension 8

Le réseau $E_8 = D_8^+$ possède la compacité la plus élevée possible dans cette dimension.

	$\mathbb{Z}^8$			D_8^*			E_8		
	ACP	Sel.	Proj.	ACP	Sel.	Proj.	ACP	Sel.	Proj.
%V	4.68	10.00	4.43	6.09	11.80	5.66	3.95	17.06	3.64
%P	0.76	0.75	0.63	0.67	0.70	0.48	0.87	0.74	0.76

On note une réelle amélioration de l'indexation seulement à partir de la dimension 8, où les indicateurs %V et %P sont considérablement plus faibles.
Il est difficile de conclure sur l'impact d'un réseau très compact : E_8 ne semble pas être bien meilleur que les autres réseaux, moins compacts.

Dans cette section, nous avons mis en évidence la difficulté pour obtenir un partitionnement raisonnable de nos données. Un nombre de paramètres important influent sur la qualité d'indexation : l'échelle, la translation, la rotation et le type de réseau.

Cependant, les cellules formées par un réseau ayant toutes la même forme et le même volume, il est difficile de s'adapter aux données non uniformes. Obtenir à la fois une faible proportion de cellules peu peuplées et une faible proportion de données au sein d'une même cellule est très dur en grandes dimensions.
Nous avons également montré que l'indexation s'améliore quand la dimension des données diminue.

La section suivante traite la phase de recherche de plus proches voisins au sein d'un seul réseau.

2.3 Recherche de plus proches voisins

2.3.1 Mesures et protocole

Pour juger l'efficacité d'une recherche de plus proches voisins à l'aide d'un réseau, nous allons mettre en rapport le nombre de réels plus proches voisins trouvés avec le nombre de données auxquelles il a fallu se comparer (voir chapitre 1, section 1.4).

Notre jeu de requêtes est composé de 1000 vecteurs tirés aléatoirement d'un ensemble de descripteurs Sift de variations des images constituant notre jeu de données indexé. Ainsi, les descripteurs des requêtes présentent des réelles similarités avec des points de notre base. Pour chacune de ces requêtes q, les k réels plus proches voisins ont été calculés à l'aide d'un algorithme séquentiel. On note l'ensemble de ces vecteurs $A_k(q)$.
Nous effectuons une recherche des k plus proches voisins dans un réseau donné en suivant l'algorithme `RECHERCHE` présenté en section 2.1.4. Cet algorithme retourne un ensemble de vecteurs, que l'on notera $B_k(q)$. Le taux de rappel, illustré par la figure 2.8 est mesuré ainsi :

$$\%\ \text{rappel}\ k\text{-ppv} = \#(A_k(q) \cap B_k(q))\ \frac{100}{\#A_k(q)}$$

où # est la cardinalité d'un ensemble.

Le nombre de données lues correspond à la population de la cellule dont le contenu à été comparé à la requête.

Les résultats affichés sont la moyenne de ces mesures obtenues avec les 1000 requêtes.

2.3.2 Résultats

Nous présentons les mesures de proportion de données lues (%d), de rappel (%r) et le rapport entre ces deux valeurs (rap.) pour $k = 1$ et $k = 50$. Nous comparons les moyennes obtenues sur plusieurs translations aléatoires de différents réseaux à différentes échelles w.

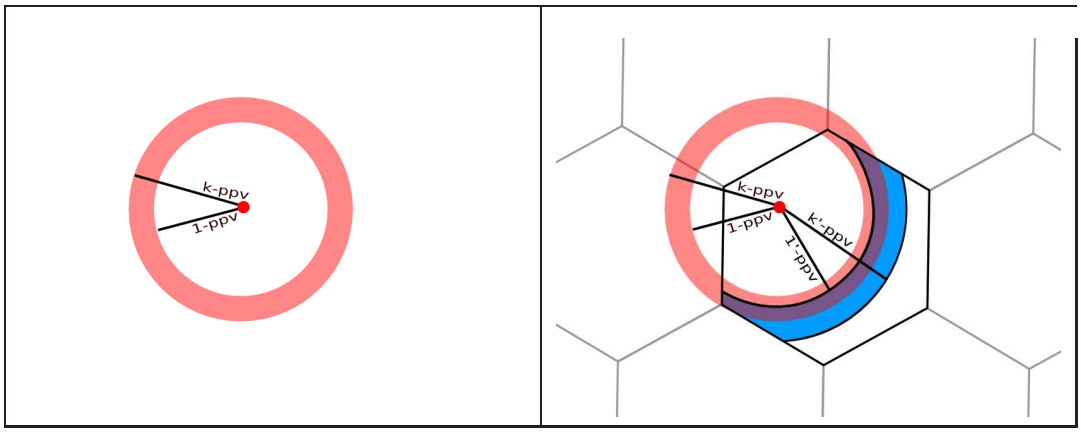

FIG. 2.8 – Sur la figure de gauche est représenté le point requête en rouge et sa "croûte" contenant les k réels plus proches voisins. Lorsque l'on se limite aux points appartenant à la même cellule que la requête (figure de droite), cette croûte est située entre le rayon du plus proche voisin, noté $1'$-ppv, et du k-ième plus proche voisin, noté k'-ppv, parmis ces points. Le rappel est l'intersection de ces deux croûtes, les faux-positifs sont les points situés entre le rayon k-ppv et k'-ppv.

Nous commençons par étudier les réseaux dans la dimension originale des données, à savoir 128.

Dimension originale

Nous avons calculé la distance entre les requêtes et leurs réels plus proches voisins. La rayon de 1-ppv varie entre 46 et 334, le rayon de 50-ppv varie entre 78 et 369.
Nous avons sélectionné les échelles 300, 500, 800, et 1000. Les valeurs minimales de proportion de données lues, et maximales de rappel et rapport sont indiquées en gras.

$k=1$	$w=300$			$w=500$			$w=800$			$w=1000$		
	%d	%r	rap.	%d	%r	rap.	%d	%r	rap.	%d	%r	rap.
$\mathbb{Z}^{128}$	**0.003**	2.6	**967**	**0.06**	7.8	**128**	**0.52**	17.3	**33**	1.33	24.1	18
D^{*}_{128}	**0.003**	2.7	797	0.31	11.7	38	0.97	23.0	24	**1.27**	24.2	**19**
D^{+}_{128}	**0.003**	**2.9**	866	0.16	**13.1**	83	1.54	**28.9**	19	1.89	**33.1**	18

$k=50$	$w=300$			$w=500$			$w=800$			$w=1000$		
	%d	%r	rap.	%d	%r	rap.	%d	%r	rap.	%d	%r	rap.
$\mathbb{Z}^{128}$	**0.003**	1.0	370	**0.06**	4.2	**69**	**0.52**	11.2	**22**	1.33	17.8	13
D^{*}_{128}	**0.003**	1.1	328	0.31	7.4	24	0.97	16.3	17	**1.27**	17.7	**14**
D^{+}_{128}	**0.003**	**1.3**	**370**	0.16	**7.8**	50	1.54	**21.3**	14	1.89	**24.5**	13

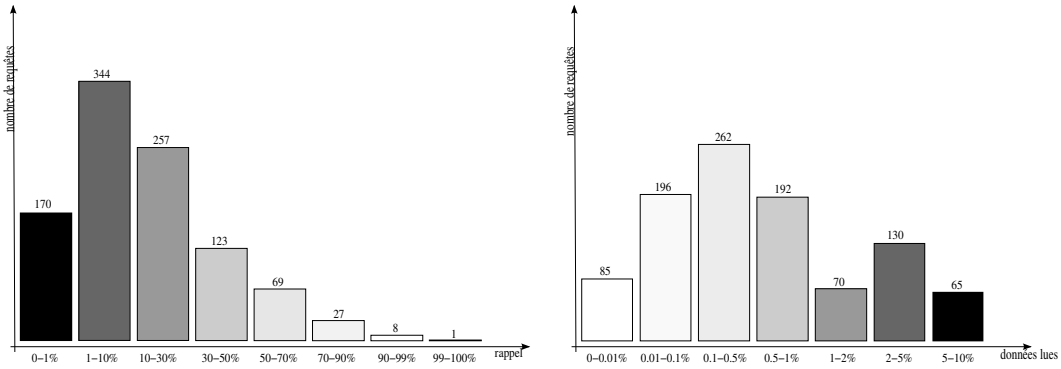

FIG. 2.9 – Histogrammes de distribution du rappel (à gauche) et du nombre de données lues (à droite) pour 1000 requêtes dans le réseau D^{+}_{128}, $w = 1000$

Les taux de rappel obtenus sont faibles, même avec des grands facteurs d'échelles. La figure 2.9 montre que la moitié des requêtes ont un taux de rappel inférieur à 10% : la majorité des plus proches voisins d'une requête sont en dehors de sa région.
La figure 2.10 illustre ce phénomène. Le cas présenté est extrême, mais il permet de se rendre compte de la dégradation du rappel lorsque la dimension augmente. En grandes dimensions, les points proches des frontières d'une cellule auront un taux de rappel très faible.

Nous nous intéressons maintenant au rappel obtenu dans des espaces de plus petites dimensions.

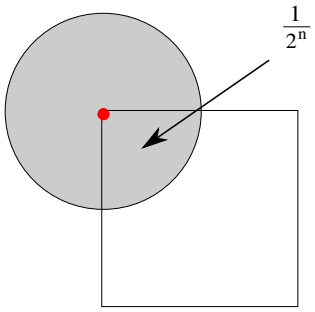

FIG. 2.10 – Cas le plus défavorable : notre requête est proche d'un coin de la cellule, en dimension n. Seul $1/2^n$-ième de la boule de plus proche voisins est récupéré : la dimension dégrade la rappel de façon exponentielle.

Après réduction de dimension

Le jeu de données de dimension 8 a été calculé à partir de la base de données Sift, en projetant les points sur 8 vecteurs aléatoires orthogonaux. Les vecteurs requêtes ont également été projetés dans cet espace, et nous avons recalculé leurs k réels plus proches voisins. Les mesures de rappel présentées sont celles correspondant aux requêtes dans l'espace transformé. Il ne s'agit en aucun cas du rappel pour la requête originale, qui rendrait la comparaison entre dimensions difficile.

Dans cet espace, le rayon de 1-ppv calculé pour les requêtes varie entre 6 et 43, le rayon de 50-ppv varie entre 16 et 61. Les échelles choisies sont 40, 60, 100 et 125. L'indexation à l'échelle 125 par le réseau E_8 créant des cellules trop peuplées, nous omettons ces résultats.

$k=1$	$w=40$			$w=60$			$w=100$			$w=125$		
	%d	%r	rap.	%d	%r	rap.	%d	%r	rap.	%d	%r	rap.
$\mathbb{Z}^8$	0.02	33.2	1589	0.23	46.4	201	2.32	58.5	25	**2.00**	57.9	**29**
D_8^*	**0.01**	31.4	**2732**	**0.18**	52.5	**297**	**1.77**	52.9	**30**	3.57	**73.8**	21
E_8	0.02	**44.9**	1821	0.30	**59.3**	200	2.90	**74.5**	26	-	-	-

$k=50$	$w=40$			$w=60$			$w=100$			$w=125$		
	%d	%r	rap.	%d	%r	rap.	%d	%r	rap.	%d	%r	rap.
$\mathbb{Z}^8$	0.02	17.0	815	0.23	30.7	133	2.32	46.3	20	**2.00**	46.1	**23**
D_8^*	**0.01**	16.1	**1401**	**0.18**	35.8	**203**	**1.77**	42.7	**24**	3.57	**61.8**	17
E_8	0.02	**24.7**	1003	0.30	**43.1**	146	2.90	**62.6**	22	-	-	-

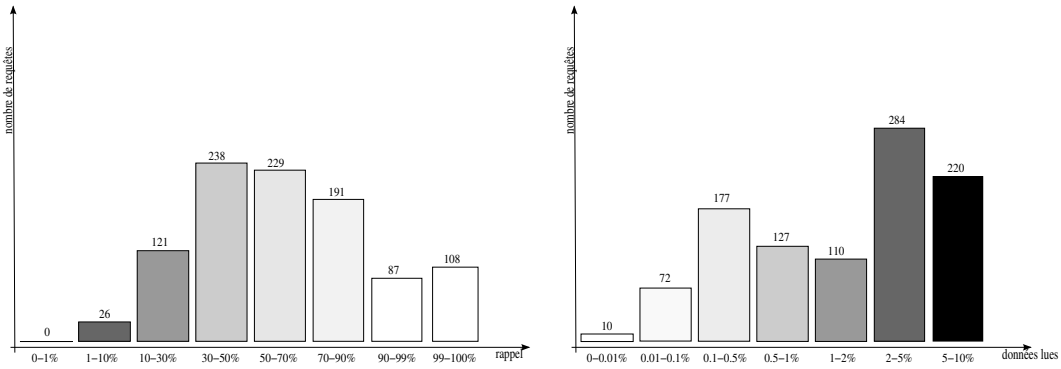

FIG. 2.11 – Histogrammes de distribution du rappel (à gauche) et du nombre de données lues (à droite) pour les requêtes dans le réseau E_8, $w = 100$

L'histogramme de distribution des rappels de la figure 2.11 rend compte de la réduction du phénomène des frontières en dimension réduite.

2.3.3 Première conclusion

La première partie de cette étude a permis de rendre compte de la difficulté d'utiliser les réseaux pour indexer et rechercher des données réelles dans des espaces de grandes dimensions.
Il est difficile d'obtenir une partition équitable de données non uniformes à cause de l'uniformité des régions formées par un réseau.
Le choix du paramétrage d'un réseau résulte d'un compromis entre un nombre réduit de cellules peu peuplées et une faible proportion des données stockées dans la même cellule.

Nous avons mis en avant la dégradation exponentielle du rappel pour des points proches des frontières d'une région lorsque la dimension augmente : la majorité des plus proches voisins d'une requête sont en dehors de sa cellule.
Ainsi, l'utilisation d'un réseau unique, sans adaptation à la requête, donne des résultats médiocres dans des espaces de grandes dimensions. Ces résultats s'améliorent quand la dimension de l'espace diminue.

Nous allons maintenant étudier différentes méthodes pour améliorer ces résultats. Dans un premier temps nous présentons d'autres algorithmes approximatifs de recherche de plus proches voisins, puis nous verrons comment s'en inspirer pour augmenter l'efficacité de nos réseaux.

Chapitre 3

Améliorer la recherche avec les réseaux

3.1 La nécessité de composer

Nous présentons deux méthodes de recherche approximative de plus proches voisins adaptées aux données en grandes dimensions mais qui n'utilisent pas de réseaux. Toutes deux reposent sur l'utilisation de projections dans des espaces de dimension 1. Ces méthodes proposent des approches originales qui résistent aux grandes dimensions, nous allons essayer de nous en inspirer pour améliorer les résultats obtenus avec les réseaux géométriques.

3.1.1 Hiérarchie de projections

Le NV-Tree [Lejsek et al., 2008] est une structure de données arborescente, utilisant des successions de projections sur des espaces de dimension 1.

Les projections se font selon des vecteurs tirés aléatoirement (en assurant l'orthogonalité) ou calculés à partir d'une analyse en composante principale.

La construction de l'index commence par projeter tous les points sur une première droite. Cette droite est segmentée en intervalles construits soit selon une règle de distance fixée entre la borne des intervalles, soit de telle sorte que chaque intervalle contienne le même nombre de points que son voisin. Pour chaque segment, on projette les points lui appartenant sur une nouvelle droite, et on réitère le processus de segmentation. L'algorithme s'arrête quand on atteint le nombre de points voulu par segment. Une structure d'arbre est ainsi créée.

Une requête q sera projetée successivement sur les différentes droites correspondant aux segments à laquelle appartient la requête, jusqu'à tomber sur un segment feuille. On renvoie

alors les points classés autour de la requête projetée.

Il est important de noter qu'aucune distance n'est calculée : on regarde seulement le rang du point projeté sur le segment.

Au final, le NV-Tree a une très bonne adaptation aux données distribuées non uniformément : sa construction garantit des cellules avec un nombre comparable de points, quelle que soit la stratégie de segmentation. Cependant, l'utilisation de projections sur des espaces de dimension 1 entraîne un phénomène d'écrasement des données, des points très éloignées peuvent se retrouver très proches sur la projection. Ceci aura tendance à augmenter le taux de faux-positifs.
De plus, pour des points situés dans des régions peu peuplées, les cellules auront une faible cohérence spatiale : la contenance des segments arrivera rapidement à la taille minimale voulue donc peu de projections différentes seront effectuées, ce qui réduit les chances de séparer des données éloignées.
Enfin, la segmentation des droites pose un problème de frontières : deux points proches peuvent être dans des segments différents. Ce phénomène est néanmoins pris en compte par une redondance optionnelle, où des segments en chevauchement sont créés sur chaque frontière.

On peut également combiner deux ou trois NV-Tree paramétrés avec des projections et des segmentations différentes. Une agrégation de rang permettra de composer les différents résultats obtenus sur chacun des index. Ceci aura pour effet de diminuer le nombre de faux positifs renvoyés, et d'augmenter le rappel.

3.1.2 Différentes fonctions de hashage

La méthode *Locality-Sensitive Hashing* (LSH) originale [Indyk et al., 1997] utilise une multitude de fonctions de hashage spatial vers des espaces de dimension 1.
Une fonction de hashage spatial assure que deux points associés à la même cellule sont proches dans l'espace de départ.

Les fonctions de hashage utilisées se construisent à l'aide de projections sur des droites, tirées généralement aléatoirement, en assurant une certaine orthogonalité. Ces droites sont ensuite segmentées de manière uniforme, chaque segment reçoit un identifiant.

On commence par projetter les données sur m droites différentes, chaque donnée reçoit m identifiants, un par droite.
Parmis ces m identifiants, k sont sélectionnés pour former un point en dimension k. Ce nouveau point est alors projetté sur une nouvelle droite, et reçoit à son tour un identifiant : celui de la cellule dans laquelle la donnée de départ sera indexée.
On répète l fois l'étape de sélection d'identifiants, en choisissant des identifiants différents.

l index sont ainsi créés.

Une requête suit le même cheminement et recevra l'identifiant de l cellules dont le contenu sera analysé.

Tout comme l'indexation avec les réseaux, le LSH se heurte au problème de la non-uniformité des données : des cellules seront très peuplées, d'autres pratiquement vides.
L'utilisation de projections dans des espaces de dimension 1 entraîne un fort écrasement des données, et contribue à renvoyer des faux-positifs. Pour réduire ce phénomène, une solution est d'augmenter le nombre de fonctions de hashage utilisées.

Utiliser plusieurs index permet à la fois d'augmenter la cohérence spatiale des résultats retournés et de diminuer l'effet des frontières des cellules. Cependant, utiliser plusieurs index a tendance à dupliquer les données stockées, il est donc nécessaire de limiter le nombre de fonctions de hashages utilisées.

Des expériences sur des bases de données de descripteurs Sift affichent des taux de rappel et de proportions de données lues équivalent pour les méthodes LSH et NV-Tree [Lejsek et al., 2008].
Pour une base de données Sift de taille comparable à celle utilisée dans notre étude, LSH semble offrir un rappel du plus proche voisin supérieur à 90% en ne lisant que 0.1% de la base de données [Jegou et al., 2008].

Dans un premier temps, nous nous inspirons des mécanismes de ces deux méthodes pour nos réseaux. Pour le côté adaptatif aux données du NV-Tree, nous allons expérimenter un mécanisme d'échelle dynamique des réseaux, pour le côté multiples fonctions de hashage du LSH, nous allons expérimenter la parallélisation de différents réseaux.

Enfin, nous discuterons de possibles améliorations des méthodes présentées en utilisant les réseaux géométriques.

3.2 Échelle adaptative

Le chapitre 2 a mis en évidence la difficulté d'obtenir des cellules de contenance uniforme avec un réseau. Il est difficile d'obtenir à la fois un faible taux de cellules peu peuplées et un faible taux de données indexées dans la même cellule.
Nous avons vu que ces facteurs étaient directement reliés à l'échelle de notre réseau : une échelle importante produira peu de cellules à faible population, une échelle petite produira peu de cellules à forte population.

On propose alors d'utiliser une hiérarchie de réseaux. Un point requête q est d'abord décodé dans un réseau grossier, créant peu de cellules à faible contenance. Si la cellule à laquelle appartient q contient un nombre trop élevé de points, on décode q dans un réseau plus petit. On réitère ce procédé jusqu'à obtenir une cellule contenant un nombre satisfaisant de données, dans laquelle nous effectuerons la recherche.
Malgrès l'utilisation de plusieurs réseaux, cette méthode n'entraîne pas la duplication de données : une donnée est présente dans un seul index.

Les expériences ont été menées sur une hiérarchie de réseaux D_{128}^* utilisant les échelles 1000, 800, 500, 300, avec 3 translations aléatoires pour chacune d'elles, soit un total de 12 réseaux organisés par facteur d'échelle décroissant.
Deux contenances maximales M ont été testées : seule les cellules contenant moins de $M\%$ de la base de données sont parcourues.
Les résultats sont comparés à ceux obtenus sur le réseau D_{128}^* à l'échelle $w = 800$ (colonne ref).

D_{128}^*	ref.	$M \leq 0.1\%$	$M \leq 1\%$
% données lues	0.97	0.03	0.32
% rappel 1-ppv	23.0	10.4	19.7
rapport 1-ppv	24	333	61
% rappel 50-ppv	16.3	4.1	11.4
rapport 50-ppv	17	130	35

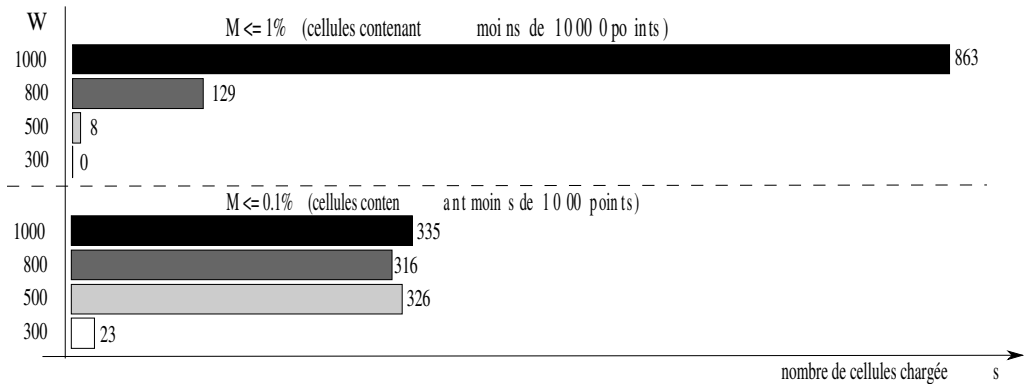

FIG. 3.1 – Histogramme d'utilisation des réseaux pour différentes contenances maximales M

La figure 3.1 rend compte de la profondeur de notre hiérarchie de réseaux atteinte par les requêtes.

On constate une nette diminution dans le nombre de données lues, cependant accompagnée d'une diminution du rappel. Le rapport obtenu est toutefois bien supérieur à celui de référence, ce qui témoigne d'une bonne efficacité.

Nous allons maintenant étudier deux méthodes permettant de minimiser l'effet de frontières des cellules : l'utilisation de réseaux complémentaires, et la navigation à travers les faces les plus proches de la requête.

3.3 Réseaux complémentaires

La chapitre 2 a permis de mettre en avant qu'avec un découpage fixe de l'espace, qui ne dépend pas de la requête, la mesure de rappel est fortement dégradée pour les requêtes arrivant près des frontières de la cellule.

Nous présentons deux méthodes permettant d'augmenter la mesure de rappel en utilisant différents réseaux translatés, dans un premier temps, aléatoirement.

Réseaux centrés

Une première solution pour minimiser le problème de proximité aux bords est de sélectionner un réseau pour lequel la requête arrive le plus proche possible du centre de la region. Par exemple, dans le cas de la figure 3.2, on choisirait la cellule du réseau en pointillés. Nous éloignons ainsi la requête des bords de la cellule, augmentant potentiellement le rappel des plus proches voisins.

Le tableau suivant résume les résultats obtenus en choisissant la cellule dont le centre est le plus proche de notre requête parmis 20 réseaux $\mathbb{Z}^{128}$ translatés aléatoirement, avec différents facteurs d'échelle w. Nous les comparons avec la moyenne des recherches sur chacun de ces réseaux.

	$w = 500$		$w = 800$	
	moy.	centré	moy.	centré
% données lues	0.06	0.19	0.50	2.22
% rappel 1-ppv	7.8	13.5	17.4	29.5
rapport 1-ppv	128	73	33	13
% rappel 50-ppv	4.2	8.4	11.2	20.8
rapport 50-ppv	69	45	22	9

Choisir le réseau le plus centré offre un meilleur rappel : nos résultats affichent une augmentation de 100%.

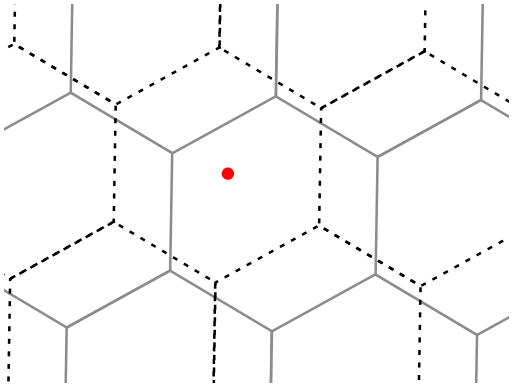

FIG. 3.2 – Deux réseaux complémentaires : on peut choisir les deux cellules contenant le point, ou exclusivement la cellule dont la requête est plus proche du centre.

Cependant, nous constatons une nette augmentation du nombre de données lues.
Une explication possible est que dans les zones peu denses de l'espace, un région centrée sur la requête contient plus de points qu'une non centrée.

Union de réseaux

Pour couvrir au maximum la croûte des plus proches voisins (figure 2.8), on choisit de comparer la requête à l'union des cellules des réseaux la contenant. Sur l'exemple de la figure 3.2, on lira les données dans la cellule contenant la requête du réseau pointillé et du réseau en trait plein.

Les résultats suivants sont issus de l'union de 20 réseaux $\mathbb{Z}^{128}$ décalés aléatoirement sur plusieurs échelles w.

$\mathbb{Z}^{128}$	$w = 300$		$w = 500$		$w = 800$	
	moy.	union	moy.	union	moy.	union
% données lues	0.003	0.05	0.06	1.22	0.52	10.4
% rappel 1-ppv	2.6	25.6	7.8	59.4	17.3	90.7
rapport 1-ppv	967	474	128	49	33	9
% rappel 50-ppv	1.0	12.2	4.2	44.3	11.2	81.9
rapport 50-ppv	370	226	69	36	22	8

On constate une très grande amélioration du rappel. Le nombre de données lues augmente linéairement en fonction du nombre de réseaux utilisés.

L'effet de dégradation du rappel en fonction de la dimension, illustré par la figure 2.10, suggère l'utilisation d'un nombre de réseaux dépendant de la dimension.

Pour rendre compte de ce phénomène, nous avons fait la même expérience sur le jeu de données réduit à 8 dimensions, en utilisant seulement 6 réseaux translatés aléatoirement.
Le tableau suivant résume les résultats obtenus.

$\mathbb{Z}^8$	$w = 40$		$w = 60$		$w = 100$	
	moy.	union	moy.	union	moy.	union
% données lues	0.02	0.11	0.23	1.21	2.32	3.85
% rappel 1-ppv	33.2	86.2	46.4	96.5	58.5	87.6
rapport 1-ppv	1589	762	201	80	25	23
% rappel 50-ppv	17.0	59.4	30.7	84.7	46.3	73.8
rapport 50-ppv	815	525	133	70	20	19

Ainsi, en dimensions réduites, des gains de rappel importants sont obtenus en utilisant moins de réseaux qu'en dimensions supérieures. La figure 3.3 compare les résultats obtenus dans les différentes dimensions.

Décalage calculé

Pour une meilleure complémentarité entre les différentes translations, on pourrait s'aider de la géométrie simple des réseaux pour calculer des décalages garantissant qu'un point proche d'une frontière ou d'un sommet dans un réseau se retrouve proche du centre d'un réseau complémentaire.
On peut imaginer trois types de décalages unitaires : les translations vers les sommets, vers les faces, et en suivant les vecteurs formant la base de notre réseau.
Le nombre de sommets et/ou de faces est généralement exponentiel en fonction la dimension. De plus, pour assurer une bonne efficacité dans les décalages, il faudrait effectuer toutes les combinaisons possibles des translations unitaires, selon plusieurs facteurs.
Ainsi, l'explosion combinatoire du nombre de translations nécessaires ne permet pas de calculer un ensemble de réseaux complémentaires.
Les expériences ont montrés que les résultats sont toujours supérieurs en utilisant des translations aléatoires.

L'union de réseaux complémentaires permet d'obtenir un gain conséquent de rappel, au détriment d'un nombre de données lues multiplié par le nombre de réseaux choisis.
Plus on utilisera de réseaux et plus la dimension des données sera grande, plus le volume de chevauchement entre les cellules sera important. Ainsi nous allons lire plusieurs fois les mêmes données, dupliquées dans les différents index.

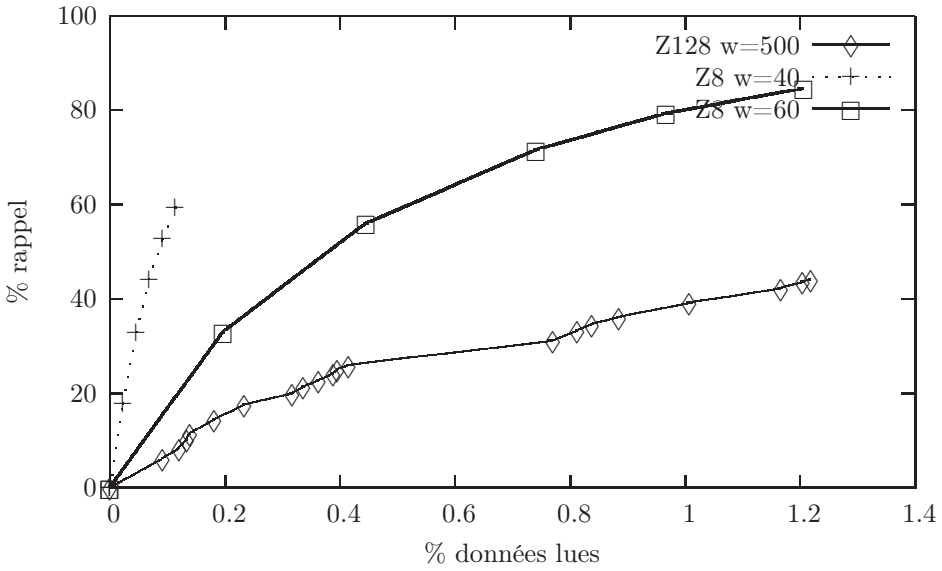

FIG. 3.3 – Un symbole represente une union d'un réseau supplémentaire. Cette courbe met en avant la nécessité d'utiliser plus de réseaux lorsque la dimension augmente.

Dans la section suivante, nous allons voir comment exploiter la connaissance de la géométrie des cellules pour augmenter le rappel, sans utiliser de réseaux complémentaires.

3.4 Les plus proches faces

Pour éviter de lire plusieurs fois les mêmes données à cause du chevauchement des cellule, on peut vouloir explorer certaines cellules voisines.
Le nombre de voisins d'une cellule grandit, en général, exponentiellement en fonction de la dimension. Ainsi regarder toutes les cellules voisines n'est pas une solution envisageable.

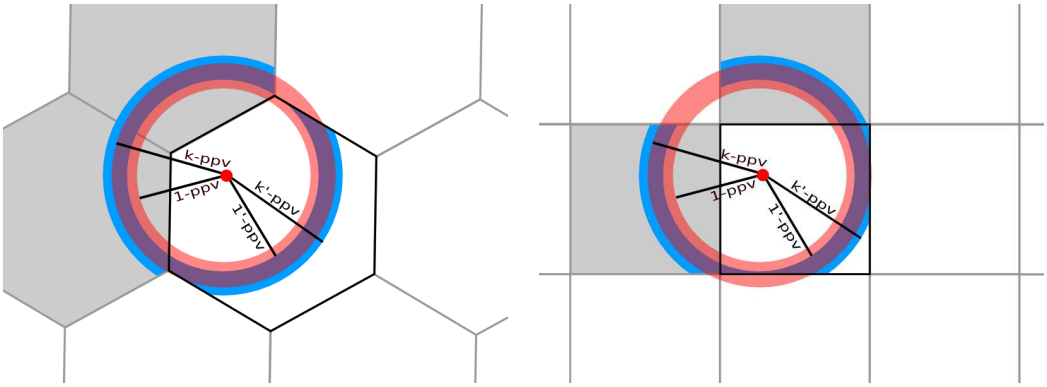

FIG. 3.4 – À gauche est représenté le réseau A_2^*, à droite le réseau $\mathbb{Z}^2$. Les cellules situées derrière les plus proches faces (grisées) contiennent potentiellement une bonne proportion des plus proches voisins de la requête.

La figure 3.4 illustre bien le fait que les cellules situées derrière les plus proches faces de notre requête contiennent potentiellement une bonne proportion des plus proches voisins.
Trouver les plus proches faces d'un point devrait être possible avec n'importe quel réseau. Cependant, nous allons voir que ce calcul devient très vite coûteux, même pour des régions dont la forme est bien spécifiée.
Nous allons étudier plus précisemment la géométrie de 3 réseaux créant des régions aux formes très simples, dont la définition est générale à toute dimension : le réseau $\mathbb{Z}^n$, D_n^* et A_n^*.
Les algorithmes suivant calculent un ensemble d'hyperplans, les faces, décrits par un point appartenant à l'hyperplan et un vecteur normal, de norme 1.

3.4.1 Plus proches faces dans le réseau $\mathbb{Z}^n$

La région formée par le réseau $\mathbb{Z}^n$ est un hypercube de sommets $(\pm\frac{1}{2})^n$. Il possède 2^n sommets et $2n$ faces. Chaque sommet est l'intersection de n faces.

Les faces qui nous intéressent sont celles formant le sommet le plus proche de q. On peut remarquer que les sommets des régions de $\mathbb{Z}^n$ sont exactement le réseau $\mathbb{Z}^n - (\frac{1}{2})^n$ (voir Figure-3.5).

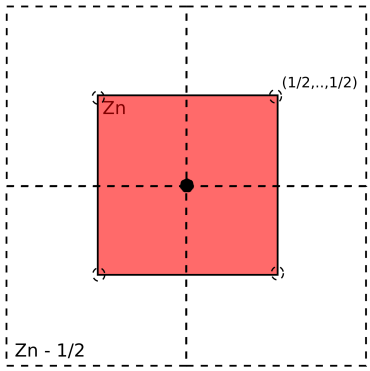

FIG. 3.5 – Illustration du découpage de $\mathbb{Z}^n$ à l'aide de $\mathbb{Z}^n - (\frac{1}{2})^n$

L'algorithme suivant calcule les n plus proches faces.

Algorithme 9. `plus-proches-faces-`$\mathbb{Z}^n$
ENTRÉES: $q \in \mathbb{R}^n$
SORTIES: $\mathcal{H}$ (* Hyperplans proches de q décrits par (point,normale) *)
$y \leftarrow \phi_{\mathbb{Z}^n}(q + (\frac{1}{2})^n) - (\frac{1}{2})^n$
pour $k = 0$ à $n - 1$ **faire**
$\eta \leftarrow (0^k, 1, 0^{n-k-1})$
$\mathcal{H} \leftarrow \mathcal{H} \cup \{(y, \eta)\}$
fin pour

Complexité. $O(n)$ *(*$\phi_{\mathbb{Z}^n} \in O(n)$*)*

3.4.2 Plus proches faces dans le réseau D_n^*

Pour $n \geq 3$,

$$D_n^* = \mathbb{Z}^n \cup (\mathbb{Z}^n + (\frac{1}{2})^n)$$

La région de Voronoï est l'intersection d'un cube de sommets $(\pm\frac{1}{2}^n)$ et du dual de ce cube de sommets $(\pm\frac{n}{4}, 0^{n-1})$ obtenu par construction de la réciproque du cube dans une sphère de rayon $\sqrt{n/4}$ [Coxeter, 1973]. La figure 3.6 illustre cette intersection en dimension 3.

FIG. 3.6 – L'intersection d'un cube et d'un octahèdre forme un octahèdre tronqué, la région fondamentale du réseau D_3^*.

Un cube de dimension n a $2n$ faces et 2^n sommets, son dual a donc $2n$ sommets et 2^n faces. L'intersection produit ainsi un polytope à $2^n + 2n$ faces.

Au lieu de chercher le sommet du polytope le plus proche d'un point q, nous utilisons la même méthode que pour le réseau $\mathbb{Z}^n$: on cherche de quel sommet du cube de départ notre point est le plus proche. Cela revient à trouver le point du réseau $\mathbb{Z}^n - (\frac{1}{2})^n$ le plus proche de q. Les faces les plus proches seront alors :

- une face appartenant au dual du cube, correspondant au sommet le plus proche du cube de départ,
- n faces appartenant au cube (dont l'intersection forme le sommet).

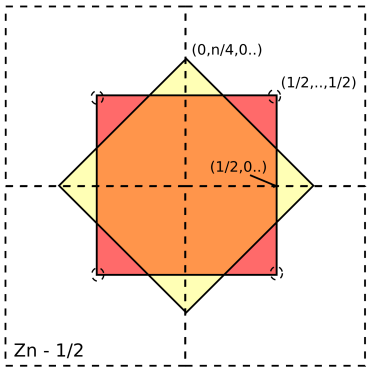

FIG. 3.7 – Illustration du découpage d'une région formée par D_n^*

Soit x un point de $\mathbb{R}^n$ appartenant à la région de Voronoï associée à l'origine du réseau, on note cette région $\mathcal{V}_{D_n^*}(0^n)$. Le sommet du cube le plus proche est donnée par :

$$y = \phi_{\mathbb{Z}^n}(x + (\frac{1}{2})^n) - (\frac{1}{2})^n = \frac{1}{2}(\pm 1, \ldots, \pm 1)$$

L'hyperplan appartenant au dual du cube associé à y a pour normale $\eta = y$ et passe par tous les points de la forme $(O^{i-1}, sign(\eta_i)\frac{n}{4}, 0^{n-i})$ où $sign(\eta_1)$ est le signe de la $i$ème coordonnée de η.
Les n hyperplans des faces du cubes ont pour normale $\eta = (0^k, y_{k+1}, 0^{n-k-1})$ et passent par le point $z = (0^k, y_{k+1}, 0^{n-k-1})$.

L'algorithme suivant résume les étapes pour trouver les $n+1$ plus proches hyperplans d'un point q.

Algorithme 10. `plus-proches-faces-`D_n^*
Entrées: $q \in \mathcal{V}_{D_n^*}(0^n)$
Sorties: $\mathcal{H}$ (* Hyperplans proches de q décrits par (point,normale) *)
$\mathcal{H} \leftarrow$ `plus-proches-faces-`$\mathbb{Z}^n(x)$
$\eta \leftarrow \phi_{\mathbb{Z}^n}(q + \mathbf{\frac{1}{2}}^n) - \mathbf{\frac{1}{2}}^n$
$z \leftarrow (\mathtt{sign}(\eta_1)\frac{n}{4}, \mathbf{0}^{n-1})$
$\alpha \leftarrow \frac{\sqrt{n}}{2}$ (* normalisation *)
$\mathcal{H} \leftarrow \mathcal{H} \cup \{(z, \alpha\eta)\}$

Complexité. $O(n)$ *($\phi_{\mathbb{Z}^n} \in O(n)$)*

3.4.3 Plus proches faces dans le réseau A_n^*

Les sommets du polytope du réseau A_n^* sont toutes les permutations possibles du point σ :

$$(n+1)\sigma = (-\frac{n}{2}, -\frac{n-2}{2}, -\frac{n-4}{2}, \cdots, \frac{n-2}{2}, \frac{n}{2})$$

Ce polytope est un permutohèdre d'ordre $n+1$ sur l'hyperplan $\sum_i \sigma_i = 0$. Il possède $(n+1)!$ sommets et $2^{n+1} - 2$ faces [Coxeter, 1973]. La figure 3.8 montre un permutohèdre d'ordre et d'ordre 4.

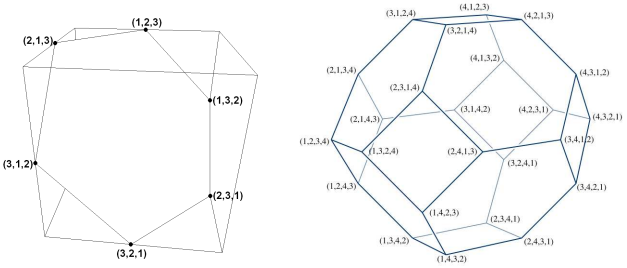

FIG. 3.8 – Permutohèdre d'ordre 3 (à gauche) formant un hexagone, et d'ordre 4 (à droite) formant un octahèdre tronqué

Nous cherchons dans un premier temps le sommet le plus proche de q.

L'algorithme suivant trie un vecteur en notant la position de la coordonnée (exemple : $(\mathbf{9}, \mathbf{6}, \mathbf{7})$ sera transformé en $((2, \mathbf{6}), (3, \mathbf{7}), (1, \mathbf{9}))$) :

Algorithme 11. `ordonne-vecteur`
Entrées: $x \in \mathbb{R}^n$
Sorties: $y \in (\mathbb{N} \times \mathbb{R})^n$
  pour $i = 1$ à n **faire**
    $y_i \leftarrow (i, x_i)$
  fin pour
  `tri de` y `selon sa deuxième composante`

Complexité. $O(n \log n)$

Nous énonçons le théorème principal. Sa démonstration ainsi que celles nécessaires à la suite de cette partie sont détaillées en annexe.

Théorème. *Soit $x \in \mathbb{R}^n$. La permutation π de $u \in \mathbb{R}^n$ est la plus proche de x si et seulement si, pour tout i, j on a $x_i < x_j \Leftrightarrow \pi_i < \pi_j$.*

Algorithme 12. `plus-proche-permutation`
Entrées: $x \in \mathbb{R}^n, u \in \mathbb{R}^n\ \forall i, u_i \leq u_{i+1}$
Sorties: y : plus proche permutation de u de x
  $s \leftarrow$ `ordonne-vecteur` x
  pour $i = 1$ à n **faire**
    $j \leftarrow s_{i1}$ (* position de la coordonnée *)
    $y_j \leftarrow u_i$
  fin pour

Complexité. $O(n \log n)$

L'article [Gaiha and Gupta, 1977] montre que tout sommet d'un permutohèdre d'ordre $n+1$ est l'intersection de n faces qui peuvent être déterminées de la manière suivante :

Soit y un sommet de notre permutohèdre d'ordre $n+1$. On définit L_k comme les positions des k plus petites coordonnées de y.

$$L_k = (p_1, \cdots, p_k), \forall i \leq k, \forall j > k, p_i \leq p_j$$

Soit F l'ensemble des n faces adjacentes à y.

$$F = \{\varphi_{L_k}, 1 \leq k \leq n\}$$

où φ_{L_k} est la face associée à L_k, que l'on représente, dans un premier temps, par l'ensemble de ses sommets :

$$\begin{aligned} \varphi_{p_1,\cdots,p_k} &= \{p \text{ permutation de } y, \forall i \leq k, j > k, p_i < p_j\} \\ \#\varphi_{p_1,\cdots,p_k} &= k!(n+1-k)! \end{aligned}$$

Exemple. Soit $y = (4,3,1,2)$ un sommet du permutohèdre d'ordre 4 issu de toutes les permutations de $(1,2,3,4)$.

$$\begin{aligned} F &= \{\varphi_3, \varphi_{3,4}, \varphi_{2,3,4}\} \\ \varphi_3 &= \{(2,3,\mathbf{1},4),(2,4,\mathbf{1},3),(4,2,\mathbf{1},3)(4,3,\mathbf{1},2),(3,4,\mathbf{1},2),(3,2,\mathbf{1},4)\} \\ \varphi_{3,4} &= \{(3,4,\mathbf{1},\mathbf{2}),(4,3,\mathbf{1},\mathbf{2}),(3,4,\mathbf{2},\mathbf{1}),(4,3,\mathbf{2},\mathbf{1})\} \\ \varphi_{2,3,4} &= \{(4,\mathbf{1},\mathbf{2},\mathbf{3}),(4,\mathbf{1},\mathbf{3},\mathbf{2}),(4,\mathbf{3},\mathbf{1},\mathbf{2}),(4,\mathbf{3},\mathbf{2},\mathbf{1}),(4,\mathbf{2},\mathbf{3},\mathbf{1}),(4,\mathbf{2},\mathbf{1},\mathbf{3})\} \end{aligned}$$

Nous pouvons également décrire une face par l'hyperplan passant par le sommet y et de vecteur normal η_{L_k}. Les faces étant orientées vers le centre 0^{n+1} du permutohèdre, η_{L_k} peut être calculé à partir du barycentre des sommets de la face associée à L_k.

$$\eta_{L_k i} = \frac{1}{k!(n+1-k)!} \sum_{s \in \varphi_{L_k}} s_i$$

Nous pouvons calculer η_{L_k} sans avoir à énumérer φ_{L_k} :

$$\eta_{L_k i} = \frac{(n+1-k)!(k-1)!}{k!(n+1-k)!} \sum_{p \in L_k} y_p = \frac{1}{k} \sum_{p \in L_k} y_p \qquad \text{si } i \in L_k$$

$$\eta_{L_k i} = \frac{k!(n+1-k-1)!}{k!(n+1-k)!} \sum_{p \notin L_k} y_p = \frac{1}{n+1-k} \sum_{p \notin L_k} y_p \qquad \text{si } i \notin L_k$$

L'algorithme suivant résume les étapes necessaires pour trouver les hyperplans des n faces les plus proches d'un point q dans le réseau A_n^*. On rappele que $\sum_i \sigma_i = \sum_i y_i = 0$, d'où

$$\sum_{p \notin L_k} y_p = \sum_i y_i - \sum_{p \in L_k} y_p = - \sum_{p \in L_k} y_p$$

Algorithme 13. `plus-proches-faces-`A_n^*
Entrées: $q \in \mathcal{V}_{A_n^*}(0^{n+1})$
Sorties: $\mathcal{H}$ (* Hyperplans proches de q décrits par (point,normale) *)
  $y \leftarrow$`plus-proche-permutation` $q\ \sigma$
  $z \leftarrow$`ordonne-vecteur` y
  $sum_L \leftarrow 0$
  pour $k = 1$ à n **faire**
    $sum_L \leftarrow sum_L + z_{k2}$ (* $\sum_{p \in L_k} y_k$ *)
    $\alpha \leftarrow sum_L^{-1}(\frac{1}{k} + \frac{1}{n+1-k})^{-1/2}$ (* normalisation *)
    $j \leftarrow 1$ (* position dans z *)
    pour $i = 1$ à $n+1$ **faire**
      si $j \leq k$ et $z_{j1} = i$ **alors**
        $\eta_i \leftarrow sum_L/k$ (* $i \in L_k$ *)
        $j \leftarrow j + 1$
      sinon
        $\eta_i \leftarrow -sum_L/(n+1-k)$ (* $i \notin L_k$ *)
      finsi
    fin pour
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{(y, \alpha\eta)\}$
  fin pour

Complexité. $O(n^2)$

3.4.4 Résultats

Le nombre d'étapes de l'algorithme pour A_n^* étant élevé pour les grandes dimensions, nous l'étudierons uniquement en dimension 8.

Le tableau ci-dessous compare les résultats obtenu sur les réseaux seuls, en utilisant l'union de 20 réseaux translatés aléatoirement, et en parcourant les cellules situées derrière les plus proches faces (ppf).

$n = 128$	$w = 500$			$w = 800$			$w = 1000$		
	%d	%r 50	rap.	%d	%r 50	rap.	%d	%r 50	rap.
$\mathbb{Z}^{128}$	0.06	4.2	69	0.52	11.2	22	1.33	17.8	13
$\mathbb{Z}^{128}$ union	1.22	44.3	36	10.38	81.9	8	21.27	90.8	4
$\mathbb{Z}^{128}$ + ppf	0.24	9.6	40	1.83	21.9	12	4.27	32.6	8
D_{128}^*	0.31	7.4	24	0.97	16.3	17	1.27	17.7	14
D_{128}^* union	6.11	66.7	11	19.34	92.7	5	13.98	83.3	6
D_{128}^* + ppf	1.68	24.0	14	5.38	45.4	7	6.80	47.6	7

L'utilisation des plus proches faces permet d'augmenter le rappel de manière conséquente, bien que les résultats restent bien inférieurs à l'union des réseaux. Cependant, le nombre

de données lues en moyenne est moins élevé, le rapport est toujours supérieur avec les plus proches faces. De plus, nous n'utilisons qu'un seul index.

En utilisant les cellules situées derrière les plus proches faces, on espère récupérer une bonne partie manquante de la croûte des plus proches voisin. Comme le montre la figure 2.10, il faudrait alors obtenir un nombre exponentiel de cellules pour espérer couvrir l'ensemble. Il est donc intéressant de faire la même expérience sur notre jeu de données réduit en dimension 8.

Les résultats suivants comparent les résultats obtenus avec un réseau seul, l'union de 6 réseaux translatés aléatoirement, et en utilisant les plus proches faces.

$n = 8$	$w = 40$			$w = 60$			$w = 100$		
	%d	%r 50	rap.	%d	%r 50	rap.	%d	%r 50	rap.
$\mathbb{Z}^8$	0.02	17.0	815	0.23	30.7	133	2.32	46.3	20
$\mathbb{Z}^8$ union	0.11	59.4	525	1.21	84.7	70	3.85	73.8	19
$\mathbb{Z}^8$ + ppf	0.11	32.2	293	0.94	48.1	51	6.60	61.6	9
D_8^*	0.01	16.1	1401	0.18	35.8	203	1.77	42.7	24
D_8^* union	0.07	60.7	880	1.06	90.0	85	7.07	91.0	13
D_8^* + ppf	0.08	42.5	502	1.06	72.5	68	7.56	66.6	9
A_8^*	0.003	7.4	2766	0.03	20.3	628	0.51	40.6	79
A_8^* union	0.02	30.1	1910	0.20	65.7	333	2.94	93.1	32
A_8^* +ppf	0.02	28.1	1536	0.21	58.5	277	2.62	87.0	33

L'utilisation des plus proches faces dans des espaces de plus petites dimensions apporte un gain de rappel presque comparable à celui obtenu avec l'union des réseaux. Cette approche est donc très efficace quand la dimension est réduite. Ce résultat constitue une contribution de ce stage.

Les courbes rappel/données lues présentées en figure 3.9 comparent l'union des réseau D_n^* et l'utilisation de leurs plus proches faces.
On notera que si la différence de rappel obtenue sur les réseau $\mathbb{Z}^8$ et D_8^* est de l'ordre de 20%, les plus proches faces du réseau A_8^* ne perdent que quelques points de rappel (entre 2 et 7% sur les résultats présentés).
On en déduit que les plus proches faces trouvées dans le réseau A_n^* approximent le mieux la boule des k plus proches voisins. La figure 3.4 permet de rendre compte de l'impact de la forme du réseau sur la qualité des plus proches faces obtenues.
Le réseau A_n^* possède donc une forme intéressante, cependant la complexité de l'algorithme des plus proches faces présenté ne permet pas sa mise en œuvre en grandes dimensions.

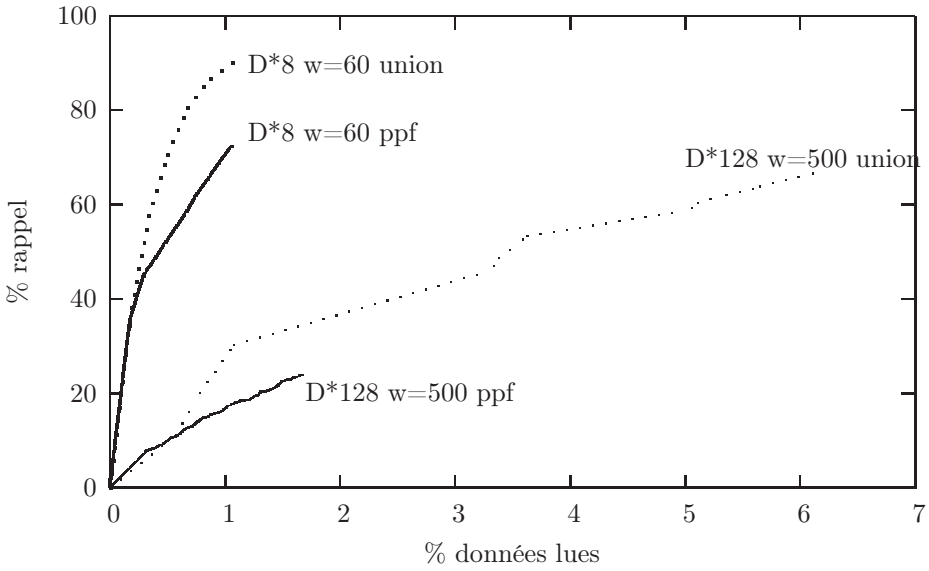

FIG. 3.9 – Comparaisons des courbes %rappel/% données lues pour l'utilisation des plus proches faces et des unions, dans la dimension originale et en dimension 8.

3.5 Apport des réseaux aux autres méthodes

La recherche de plus proches voisins avec des réseaux, même composés, dans la dimension d'origine des données ne donne pas de très bons résultats, mais nous avons mis en évidence leur grande amélioration dans des dimensions réduites.

Les réseaux étant des fonctions de hashage spatial, ils peuvent être utilisés directement dans les algorithmes de type LSH.

À la place d'utiliser des projections sur des espaces de dimension 1, nous pouvons projeter nos données dans des espaces de dimensions supérieures et utiliser un réseau dans cette dimension réduite pour mettre en cellule les données. Nous réduisons ainsi l'effet d'écrasement des données en augmentant la cohérence spatiale des cellules, ce qui permet de réduire le nombre de fonctions de hashages différentes necessaires. De plus, si on utilise des réseaux formant des régions simples, l'utilisation de plus proches faces peut, dans une certaine mesure, se substituer à l'utilisation de fonctions parallèles, et donc réduire encore le nombre de

hashages différents nécessaires.

L'utilisation de hiérarchies de réseaux, présentée en section 3.2, peut être une première solution au problème d'adaptation des cellules aux données non uniformes.

L'article [Jegou et al., 2008] propose l'utilisation du réseau E_8 comme fonctions de hashage de LSH, et montre une certaine amélioration par rapport au LSH basique.
Il serait intéressant de comparer ces résultats avec l'utilisation de réseaux plus simples comme Z_8, D_8^* et A_8^* permettant de combiner des hiérarchies de réseaux avec la navigation dans les plus proches faces.

Conclusion

Ce stage a porté sur l'étude d'une nouvelle technique d'indexation et de recherche par similarité dans des bases de données numériques de grandes dimensions.

Les réseaux sont des quantificateurs spatiaux formant une partition régulière de l'espace. Cette régularité rend l'indexation de données non uniformes difficile.
Nous avons mis en avant l'influence des différents paramètres d'un réseau sur la qualité de cette indexation.

La phase recherche de plus proches voisins a ensuite été détaillée.
Des réseaux particuliers présentent des algorithmes très rapides pour calculer une région de l'espace contenant des données similaires à une requête, rendant praticable leur utilisation directement dans des espaces de grandes dimensions.

En partant de résultats médiocres obtenus avec des réseaux uniques, nous avons mis au point et expérimenté différentes techniques pour améliorer le rappel des plus proches voisins tout en limitant la proportion de données parcourues.

La nette amélioration de la recherche des plus proches voisins en dimensions réduites a permis rendre compte de l'impact négatif de la dimension sur l'efficacité des réseaux.

La compacité des réseaux ne semble pas avoir une influence déterminante sur la recherche des plus proches voisins. De plus, la mise en œuvre des plus proches faces suggère d'utiliser des réseaux aux formes simples, ce qui n'est généralement pas le cas des réseaux très compacts, dont la forme se rapproche le plus d'une sphère.

La connaissance de la géométrie des réseaux a permis d'apporter une approche originale de navigation au sein des cellules avec l'utilisation des plus proches faces. Nous pouvons ainsi améliorer le rappel des plus proches voisins sans avoir recours à plusieurs index, coûteux en espace mémoire. De plus ce procédé s'avère très efficace dans des dimensions réduites, notamment avec le réseau A_n^*.

Il sera intéressant d'approfondir l'intégration des réseaux dans des algorithmes de type LSH avec les différents paramétres et méthodes introduits durant ce stage, et de confronter les résultats obtenus.

Annexes

Plus proches faces du réseau A_n^* - démonstrations

Nous détaillons ici les démonstrations nécessaires à la mise au point de l'algorithme pour trouver les plus proches faces d'un point.

Théorème. *Soit $x \in \mathbb{R}^n$. La permutation π de $u \in \mathbb{R}^n$ est la plus proche de x si et seulement si, pour tout i, j on a $x_i < x_j \Leftrightarrow \pi_i < \pi_j$.*

Démonstration. Soit y une permutation de σ tel qu'il existe a, b vérifiant $x_a < x_b$ et $y_a < y_b$. Soit z une transposition de y définie comme suit :
- $z'_i = y_i$ pour tout $i \neq a, b$
- $z'_a = y_b$
- $z'_b = y_a$

Calculons la différence entre les distances au carré de x à y et z :

$$\begin{aligned} & l_2^2(x,y) - l_2^2(x,z) \\ = \; & (x_a - y_a)^2 - (x_a - z_a)^2 + (x_b - y_b)^2 - (x_b - z_b)^2 \\ = \; & 2x_a(y_b - y_a) + 2x_b(y_a - y_b) \\ = \; & 2x_a(y_b - y_a) + 2(x_a + \delta)(y_a - y_b) \quad \text{avec } \delta > 0 \\ = \; & 2\delta(y_a - y_b) \\ & \Longleftrightarrow l_2(x,y) < l_2(x,z) \end{aligned}$$

Par induction, la permutation de σ verifiant $x_i < x_j \Leftrightarrow y_i < y_j$ pour tout i, j est la plus proche de x. □

Montrons que le barycentre des sommets d'une face est une normale à cette face.
Le permutohèdre est situé sur l'hyperplan $\sum_i x_i = \sum_i \sigma_i$ de l'espace $\mathbb{R}^{n+1}$. Une face du permutohèdre est donc un hyperplan (dimension $n-1$) de ce sous-espace. Elle est à l'intersection de l'hyperplan du permutohèdre et de l'hyperplan ζ_{L_k} de dimension n contenant ses sommets :

$$\zeta_{L_k}.x = \sum_{p \in L_k} y_p \qquad 1 \leq k \leq n$$

Avec

$$\zeta_{L_k i} = \begin{cases} 1 & \text{si } i \in L_k \\ 0 & \text{sinon} \end{cases} \qquad 1 \leq i \leq n+1$$

Nous utiliserons les notations suivantes :

$$\begin{aligned} s &= \sum_i \sigma_i \\ s_{L_k} &= \sum_{p \in L_k} y_p \\ \overline{s}_{L_k} &= \sum_{p \notin L_k} y_p = s - s_{L_k} \\ \overline{k} &= n+1-k \\ c &= (\frac{s}{n+1}^{n+1}) \quad \text{(centre du permutohèdre)} \end{aligned}$$

Pour des raisons de simplicité, nous réordonnons les dimensions de telle sorte que les k premières dimensions correspondent à L_k. Ainsi $\zeta_{L_k} = (1^k, 0^{\overline{k}})$.

L'intersection des hyperplans vérifie les contraintes suivantes :

$$\begin{cases} (1^{n+1}).x = s \\ \zeta_{L_k}.x = s_{L_k} \end{cases} \iff \begin{cases} (0^k, 1^{\overline{k}}) = \overline{s}_{L_k} \\ (1^k, 0^{\overline{k}}) = s_{L_k} \end{cases}$$

Soit η_{L_k} le barycentre des sommets de notre face.

$$\eta_{L_k} = (\frac{s_{L_k}}{k}^k, \frac{\overline{s}_{L_k}}{\overline{k}}^{\overline{k}})$$

On montre que tout point x appartenant à l'intersection de nos hyperplans appartient à l'hyperplan de normale η_{L_k} :

$$
\begin{aligned}
x' &= x - \eta_{L_k} \\
x'_i &= \begin{cases} x_i - \frac{s_{L_k}}{k} & 1 \leq i \leq k \\ x_i - \frac{\overline{s}_{L_k}}{\overline{k}} & k < i \end{cases} \\
x'_i \eta_{L_k i} &= \begin{cases} x_i \frac{s_{L_k}}{k} - \frac{s^2_{L_k}}{k^2} & 1 \leq i \leq k \\ x_i \frac{\overline{s}_{L_k}}{\overline{k}} - \frac{\overline{s}^2_{L_k}}{\overline{k}^2} & k < i \end{cases} \\
x'.\eta_{L_k} &= \sum_{i=1}^{k} x'_i \eta_{L_k i} + \sum_{i=k+1}^{n+1} x'_i \eta_{L_k i} \\
\sum_{i=1}^{k} x'_i \eta_{L_k i} &= \sum_{i=1}^{k} (x_i \frac{s_{L_k}}{k}) - \sum_{i=1}^{k} (\frac{s^2_{L_k}}{k^2}) \\
&= \frac{s_{L_k}}{k} (\sum_{i=1}^{k} x_i) - k \frac{s^2_{L_k}}{k^2} = \frac{s^2_{L_k}}{k} - \frac{s^2_{L_k}}{k} = 0 \\
\sum_{i=k+1}^{n+1} x'_i \eta_{L_k i} &= \sum_{i=k+1}^{n+1} (x_i \frac{\overline{s}_{L_k}}{\overline{k}}) - \sum_{i=k+1}^{n+1} (\frac{\overline{s}^2_{L_k}}{\overline{k}^2}) \\
&= \frac{\overline{s}_{L_k}}{\overline{k}} (\sum_{i=k+1}^{n+1} x_i) - \frac{\overline{s}^2_{L_k}}{\overline{k}} = 0 \\
\text{d'où } x'.\eta_{L_k} &= 0
\end{aligned}
$$

Outils développés

Pour mener à bien toutes ces expériences, nous avons développés un ensemble de modules pour manipuler les réseaux et les données. L'objectif était d'avoir une très grande modularité et souplesse entre les différents composants nécessaires, en un minimum de programmation. Ainsi la quasi-totalité des paramètrages possibles se spécifient par la ligne de commande.

Ces modules ont étés développés en python et utilisent des bibliothèques de calculs scientifiques.

Nous présentons quelques modules développés :

Formats : Nous supportons différents formats de fichiers de données, de requêtes et de stockage d'index. Le paramétrage des différents formats se fait via un fichier de configuration. Nous avons ainsi une transparence sur la façon dont sont stockés les données.

Réseaux : Les réseaux $\mathbb{Z}^n$, D_n, D_n^+, D_n^*, A_n, A_n^* sont implémentés, ainsi que la possibilité de les composer.

Indexation avec les réseaux : Les différents paramètres du réseau, les déformations de l'espace, l'échantillonnage du jeu de données, sont spécifiés en paramètre de la ligne de commande. Cela permet de tester facilement les différents paramètres via un script shell. Des mécanismes de cache ont également été mis en place pour les déformations coûteuses de l'espace.

Recherche avec un réseau : Les différentes stratégies de recherche proposées sont implémentées et se paramètrent via la ligne de commande.

Outils d'analyse : Analyse des jeux de données avant et après transformations (histogrammes des dimensions, etc.), analyse des index (occupation des cellules, distances des points aux centre/faces des régions, etc.), mesures de rappel sur les plus proches voisins. Différents formats de sorties sont utilisés : texte, PDF, courbes gnuplot, dessins SVG.

Il est ainsi possible de tester rapidement, avec un minimum de programmation, l'indexation et la recherche dans un jeu de données avec les réseaux.

Bibliographie

[Be'ery et al., 1989] Be'ery, Y., Shahar, B., and Snyders, J. (1989). Fast decoding of the leech lattice. *Selected Areas in Communications, IEEE Journal on*, 7(6) :959–967.

[Berrani et al., 2002] Berrani, S.-A., Amsaleg, L., and Gros, P. (2002). Recherche par similarités dans les bases de données multidimensionnelles : panorama des techniques d'indexation. *Ingénierie des Systèmes d'Information*, 7(5-6) :9–44.

[Beyer et al., 1999] Beyer, K., Goldstein, J., Ramakrishnan, R., and Shaft, U. (1999). When is "nearest neighbor" meaningful ? *Lecture Notes in Computer Science*, 1540 :217–235.

[Conway and Sloane, 1982] Conway, J. H. and Sloane, N. J. A. (1982). Fast quantizing and decoding algorithms for lattice quantizers and codes. *IEEE Transactions on Information Theory*, 28(2) :227–232.

[Conway and Sloane, 1998] Conway, J. H. and Sloane, N. J. A. (1998). *Sphere Packings, Lattices and Groups*. Springer-Verlag.

[Coxeter, 1973] Coxeter, H. (1973). *Regular Polytopes*. Dover, third edition.

[Gaiha and Gupta, 1977] Gaiha, P. and Gupta, S. K. (1977). Adjacent vertices on a permutohedron. *SIAM Journal of Applied Mathematics*, 32 :323–327.

[Indyk et al., 1997] Indyk, P., Motwani, R., Raghavan, P., and Vempala, S. (1997). Locality-preserving hashing in multidimensional spaces. pages 618–625.

[Jegou et al., 2008] Jegou, H., Amsaleg, L., Schmid, C., and Gros, P. (2008). Query adaptative locality sensitive hashing. *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 825–828.

[Lejsek et al., 2008] Lejsek, H., Ásmundsson, F. H., Jónsson, B. T., and Amsaleg, L. (2008). Nv-tree : An efficient disk-based index for approximate search in very large high-dimensional collections. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Mow, 2003] Mow, H. H. (2003). Universal lattice decoding : Principle and recent advances. *Wireless Communications and Mobile Computing, Special Issue on Coding and Its Applications in Wireless CDMA Systems*, 3(5) :53–569.

[Vaughan and Clarkson, 1999] Vaughan, I. and Clarkson, L. (1999). An algorithm to compute a nearest point in the lattice a_n^*. In Fossorier, M. P. C., Imai, H., Lin, S., and Poli,

A., editors, *AAECC*, volume 1719 of *Lecture Notes in Computer Science*, pages 104–120. Springer.

[Weber et al., 1998] Weber, R., Schek, H.-J., and Blott, S. (1998). A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 194–205.