Bases de données multimédia
VI – Indexation de vecteurs

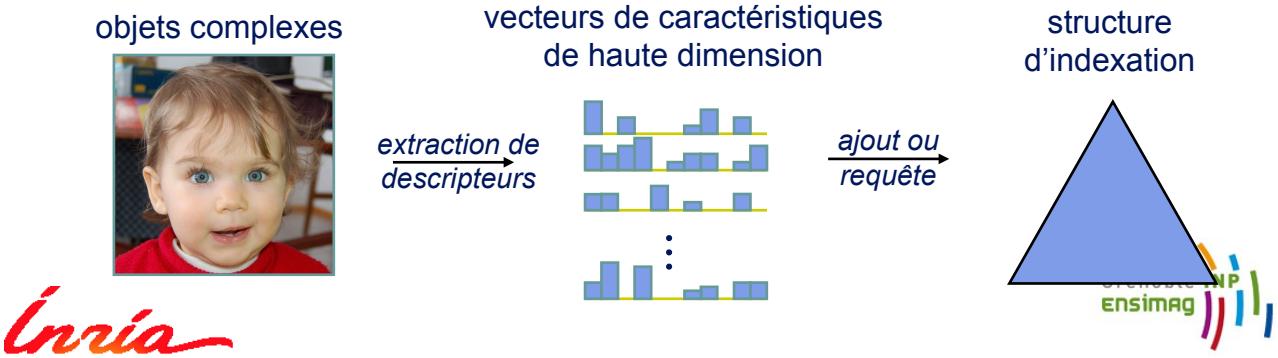
ENSIMAG

2014-2015

Matthijs Douze & Karteek Alahari

Indexation

- Problème : rechercher **efficacement** des vecteurs de caractéristique (=descripteurs) proches au sein d'une base de descripteurs
  → on voudrait éviter de faire une comparaison exhaustive
- Opérations supportées : recherche (critique), ajout/suppression (+ ou – critique) selon l'application
- Deux types de recherche
  - Recherche à ε
  - Recherche des *k* plus proches voisins



Plan

- Préliminaires
- Indexation mono-dimensionnelle
- Indexation multi-dimensionnelle
- Perspectives

Notations

- base de n vecteurs : $Y = \{ y_i \in \mathbb{R}^d \}_{i=1..n}$

- vecteur requête : q in $\mathbb{R}^d$

on note $N(q) \subset Y$ les voisins de q
→ les vecteurs “similaires”

- Le voisinage est relatif à une distance (ou une similarité/dissimilarité)

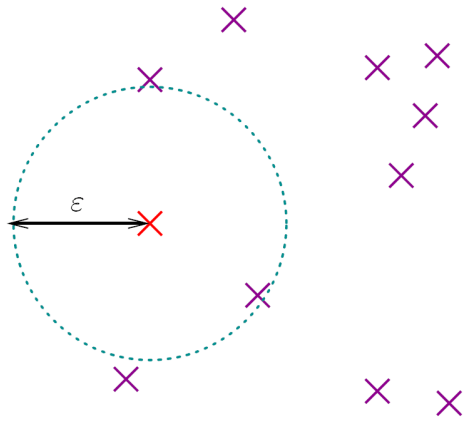
→ on se concentre sur la distance Euclidienne

$$d(x,y) = ||x-y||_2$$

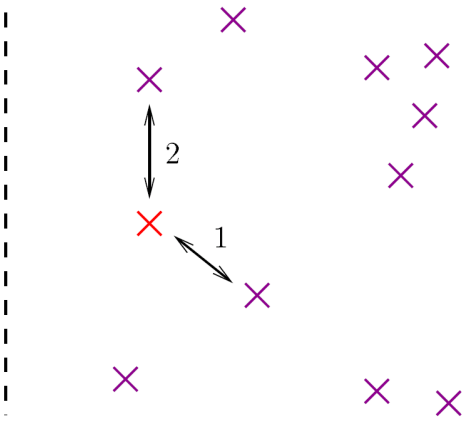
Qu'est ce qu'un "voisin" ?

- un vecteur qui répond à une propriété de proximité absolue ou relative

recherche à ε



recherche des *k* plus proches voisins $_{k=2}$

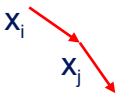


$N_\varepsilon(q) = \{ y_i \in Y : d(y_i,q) < \varepsilon \}$

$N_k(q) = k\text{-arg-min}_i\, d(y_i,q)$

Remarque : le voisinage au sens des k-ppv n'est pas symétrique

- $X = \{ x_i \in \mathbb{R}^d \}_{i=1..n}$
- on peut avoir $x_j \in N_k(x_i)$ et $x_j \notin N_k(x_i)$
- Ex: une spirale échantillonnée

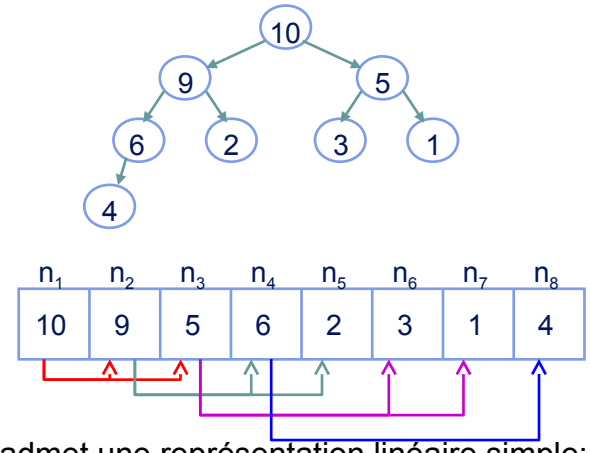


Algorithme naif : complexité

- Si la base Y n'est pas organisée (=indexée), calculer N_k ou N_ε nécessite le calcul de l'ensemble des distances $d(q,y_i)$
  - $O(n \times d)$
- Pour $N_k(q)$, il faut de plus trouver effectuer l'opération k-arg-$\min_i$ $d(q,y_i)$
- Exemple: on veut 3-argmin {1,3,9,4,6,2,10,5}
  - Méthode naïve 1: trier ces distances → O(n log n)
  - Méthode naïve 2: maintenir un tableau des k-plus petits éléments, mis à jour pour chaque nouvelle distance considérée → O(n k)
- Intuitivement, on peut faire mieux…

Max-heap

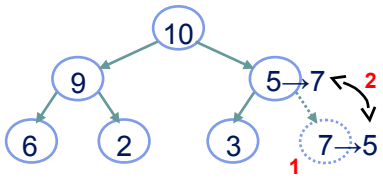
- *Binary max heap*
  - structure d'arbre binaire équilibré
  - dernier niveau rempli de gauche à droite
  - à chaque noeud *n* on associe une valeur v(*n*)
  - propriété : si n_i est un noeud fils de n_j, alors $v(n_i) < v(n_j)$



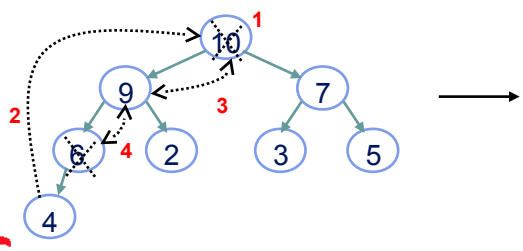
- Remarque : il admet une représentation linéaire simple: n_i a pour père $n_{i/2}$

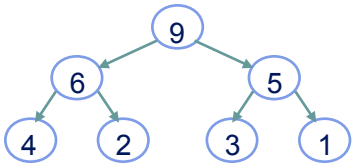
Max-heap : opérations élémentaires

- **`heap_push`**: ajout inconditionnel d'un élément dans le heap
  - ajouter le noeud k+1, et y placer le nouvel élément
  - mise à jour (pour garder les propriétés d'ordre) :
    - l'élément inséré remonte : inversion avec son parent s'il est plus grand
    - jusqu'à vérification de la relation d'ordre
    - complexité en O(log k) au pire, mais O(1) en pratique



- **`heap_pop`**: suppression inconditionnelle de la plus grande valeur
  - on supprime le noeud racine et on le remplace par l'élément du noeud k
  - mise à jour :
    - on descend l'élément : inversion avec le fils le plus grand
    - jusqu'à ce que la relation d'ordre soit vérifiée





Algorithme: Max-heap pour chercher les k plus petites valeurs

- Pb: on cherche k-argmin$_i$ $\{a_1,\ldots a_i,\ldots a_n\}$
- Initialisation du heap à l'arbre vide
- Pour i=1..n,
  - si l'arbre n'est pas encore de taille k → `heap_push`
  - si l'arbre est déjà de taille k, on compare à la racine
    - si $a_i \geq$ racine → on passe à l'élement suivant
    - sinon
      - `heap_pop`
      - `heap_push`
- Exemple: 3-argmin {1,3,9,4,6,2,10,5}

Algorithme: Max-heap pour chercher les k plus petites valeurs

- Complexité : O(n log k) au pire, bien meilleure en moyenne
  - un nouveau $n^{ième}$ élement doit être plus grand que le noeud racine
  - probabilité de k/i de faire effectivement un push
    → décroit rapidement

- Peut être utilisé pour trier (algorithme *heapsort*).

Plan

- Préliminaires
- Indexation mono-dimensionnelle
- Indexation multi-dimensionnelle
- Perspectives

Indexation mono-dimensionnelle

- Application phare : les bases de données (au sens classique du terme)

- Différents types d'opérations de recherche
  - recherche d'un point/vecteur : PAM (point access method)
  - recherche de structures spatiales plus complexes : SAM (spatial access method)
    Ex: recherche d'une valeur dans un intervalle, ou du plus proche voisin d'une valeur

- Les structures de références :
  - Hashing
  - B+ tree

Hashing

- Principe:
  - définir une fonction x → *k* (x), appelée clé de hachage

$$k(x) = ((\sum_i r_i x_i) \bmod P) \bmod m$$

  - on ne se compare qu'aux éléments ayant la même clé

- Excellent pour un accès de type point (PAM)

  SELECT name FROM PERSON WHERE name = 'herve'

- Peu adapté aux requêtes comparatives (type intervalle, donc SAM) du type

  SELECT taille FROM PERSON WHERE taille > 1.70 and taille < 1.90

  → dans ce cas, complexité en O(n)

- Typiquement, 1-2 I/O par requête pour les requêtes PAM

B+ tree (début)

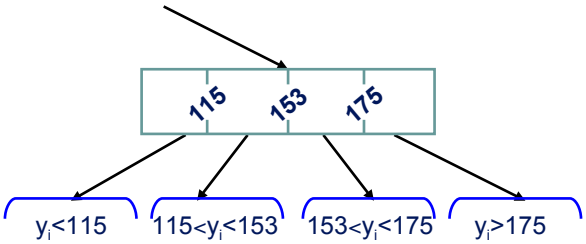
- Dérivé du B-tree (différence soulignée plus tard)
- Utilisation d'un ordre sur les éléments
- Pour requêtes exactes ou requête sur des plages

- Permet d'optimiser les requêtes du type

  SELECT taille FROM PERSON WHERE taille > 1.70 and taille < 1.90
- Dans ce cas, plus lent que le hashing

B+ tree : structure

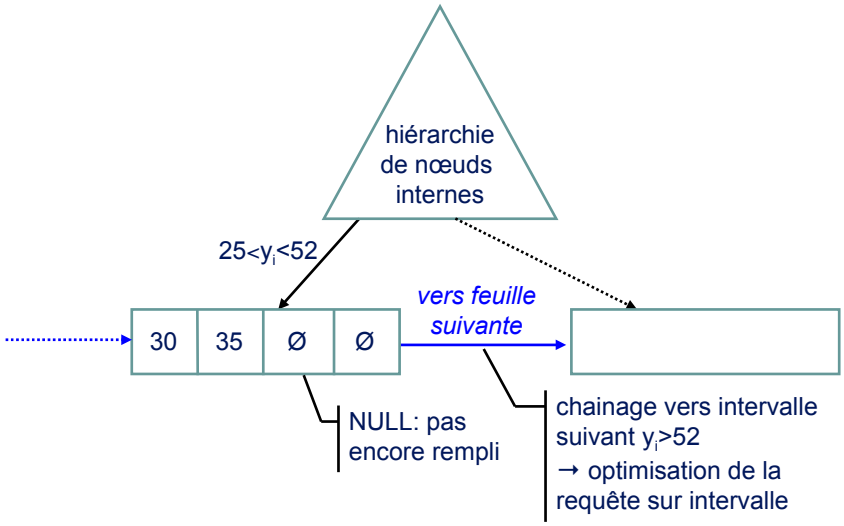
- Structure arborescente avec deux types de noeuds
- Noeuds index (noeuds internes à l'arbre)
  - contiennent des pointeurs vers d'autres noeuds ou des feuilles
  - ces noeuds ordonnent les sous-noeuds par des valeurs de séparation



- Les entrées (valeurs de séparation) ne sont pas nécessairement remplies
- Paramètres :
  - B : maximum d'entrées dans un noeud
  - B_m : minimum
- Usuellement B = 2 B_m

B+ tree : feuille

- Feuilles : contiennent les valeurs (et non les valeurs de séparation)



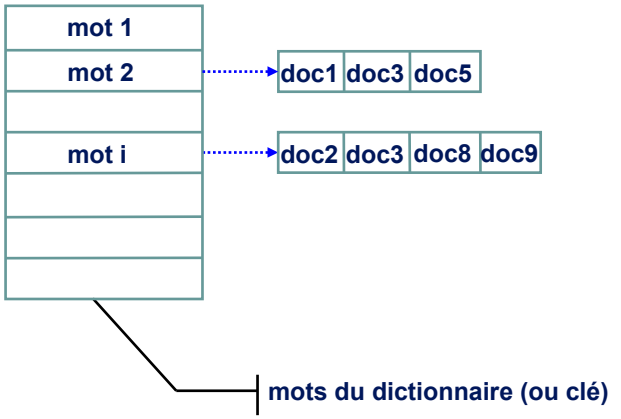
- La recherche dans un B+tree se fait par une descente dans l'arbre
  - en temps logarithmique

B+ tree : propriétés

- Recherche d'une valeur donnée, {y_i in Y : y_i = a}
  - on cherche la feuille contenant la valeur
  - on se compare aux éléments
- Recherche sur intervalle [a,b]
  - recherche de L correspondant à a
  - utilisation du chaînage horizontal pour récupérer toutes les entrées jusqu'à b
- Complexité
  - insertion/suppression/recherche en temps logarithmique (amorti)
  - effectuer une recherche à ε est en O (log + $|N_\varepsilon(q)|$)
- Utilisation de l'espace paginé :
  - 50% au moins pour $B=2B_m$
  - 67% pour des nombres aléatoires (uniformes)

Fichier inversé (inverted file)

- Cette structure liste des éléments qui ont une valeur donnée pour un attribut
- Bases de données: utilisé pour les clés secondaires (doublons attendus)
- Exemple courant d'utilisation : requêtes sur documents textuels (mails, …)



- La requête consiste à récupérer la liste des documents contenant le mot
  - coût proportionnel au nombre de documents à récupérer

Recherche de documents textuels (1)

- Modèle vectoriel
  - on définit un dictionnaire de mots de taille d
  - un document texte est représenté par un vecteur $f=(f_1,\ldots,f_i,\ldots f_d) \in \mathbb{R}^d$
  - chaque dimension i correspond à un mot du dictionnaire
  - f_i = fréquence du mot dans le document

- en pratique, on ne garde que les mots discriminants dans le dictionnaire
  - "le", "la", "est", "a", etc, sont supprimés, car peu discriminants

- Ces vecteurs sont creux
  - dictionnaire grand par rapport au nombre de mots utilisés

Recherche de documents textuels (2)

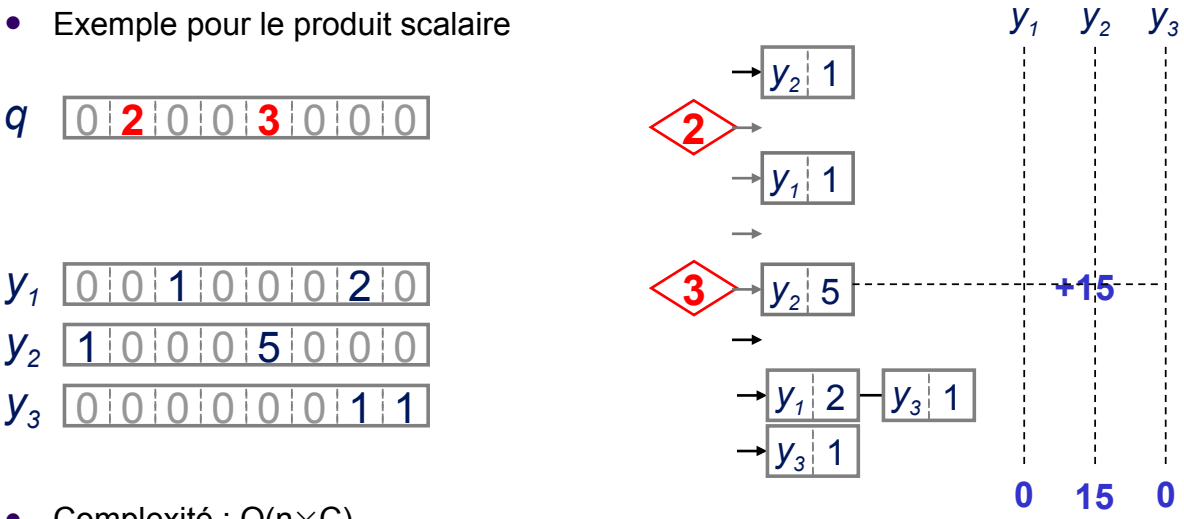
- Exemple
  - dictionnaire = {“vélo”, “voiture”, “déplace”, “travail”, “école”, “Rennes”}
  - espace de représentation: $\mathbb{R}^6$
  - “Rennes est une belle ville. Je me déplace à vélo dans Rennes”

    f=(1/4,0,1/4,0,0,1/2)^t

- Recherche de document = trouver les vecteurs les plus proches
  - au sens d’une mesure de (dis-)similarité, en particulier le produit scalaire

Fichier inversé : distances entre vecteurs creux

- requête q et base Y: vecteurs creux
- Fichier inversé: calcul efficace du produit scalaire (en fait, toute distance L_p)
- Exemple pour le produit scalaire



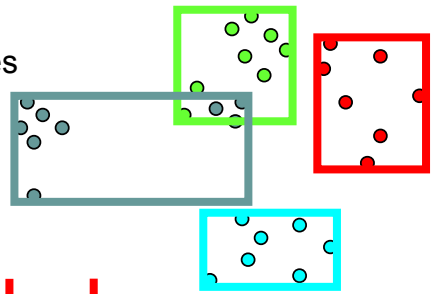
- Complexité : O(n×C),
  - C : espérance du nombre de positions non nulles en commun, très petit (=d s^2 si uniforme et indépendant)

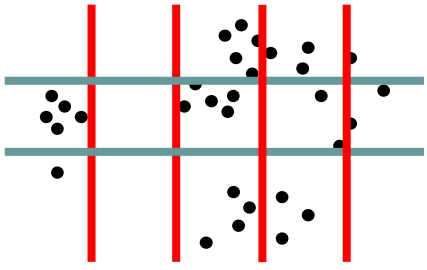
Plan

- Préliminaires
- Indexation mono-dimensionnelle
- Indexation multi-dimensionnelle
  - préliminaires
  - R-tree
  - KD-tree
  - la malédiction de la dimension
  - LSH (Locality-Sensitive Hashing)
  - Quantifieur produit
- Perspectives

Préliminaires

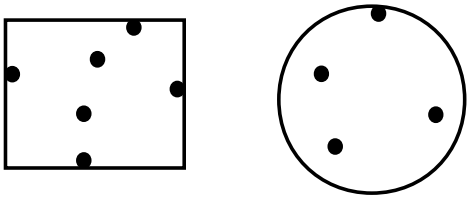
- Pour d $\geq$ 2, on perd l'ordre total
  - compromet la recherche en temps logarithmique
- Nécessité d'organiser l'espace pour éviter de se comparer à tout les vecteurs : groupement en cellules
- Deux approches pour l'organisation des données
  - à partir des données
    (comme pour le B-tree)
  - pré-défini
    comme pour le hachage)





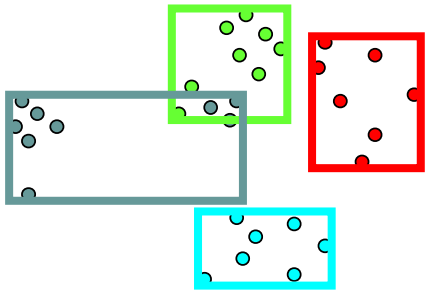
Organiser à partir des données

- Utilisation d'hyper-rectangles ou d'hyper-sphères enveloppants



- Hyper-rectangles plus facile à indexer, mais meilleure compacité des hyper-sphères
  - la diagonale de l'hyper-rectangle est égale à 5.47 pour d=30 et un volume de 1
  - le diamètre de l'hyper-sphère est égale à 2.86 → filtrage plus efficace

- Selon la méthode:
  - recouvrement autorisé
  - ou non



Inria

Grenoble INP ensimag

Sur l'utilisation de cellules

- Bien moins de cellules que de descripteurs
  - trouver les cellules non vides les plus proches d'un descripteur requête est relativement efficace
  - donc ce pré-filtrage de la plus grande partie des données apporte *a priori* un gros gain de performance

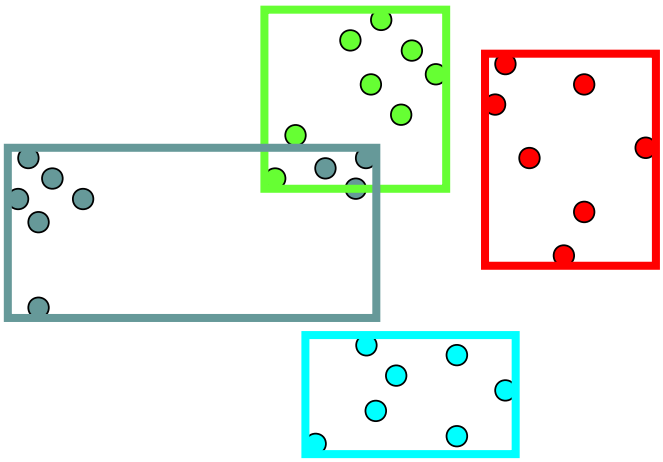
- Peu de paramètres à associer à chaque cellule
  - hyper-rectangle englobant : deux points
  - sphère : centre et rayon

- Qu'est-ce que l'on exploite ?
  - l'inégalité triangulaire

Inria

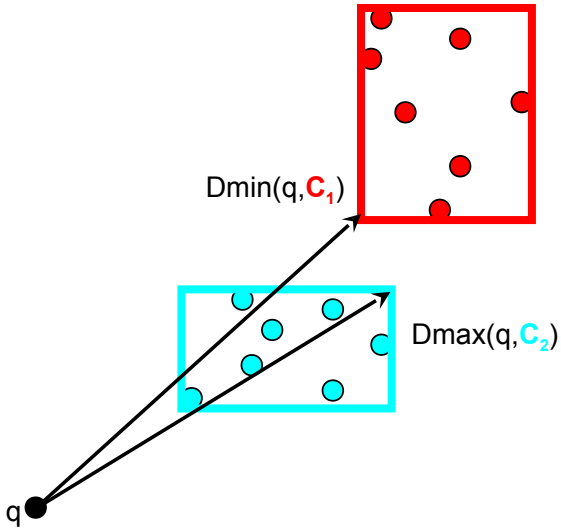
Grenoble INP ensimag

Exemple d'utilisation de cellules englobantes



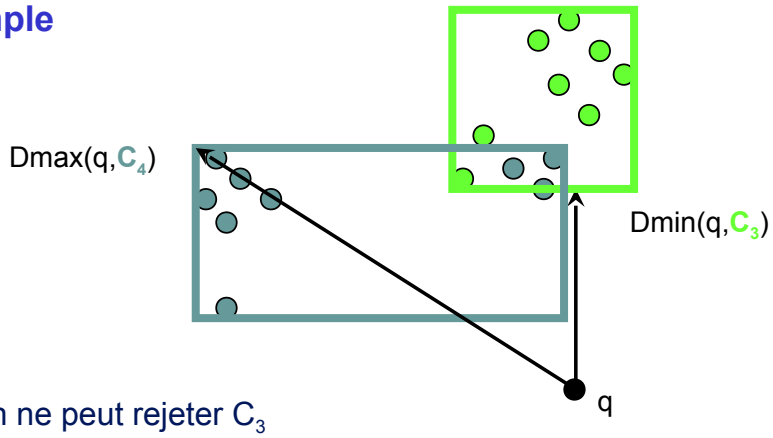
- Recherche des 4 plus proches voisins d'une requête
  - on connaît les rectangles
  - mais pas encore les points contenus dans ces rectangles

Exemple d'utilisation de cellules englobantes



- On peut rejeter C_1 sans analyser son contenu

Exemple



- On ne peut rejeter C_3
  - On doit analyser son contenu

- Il est préférable de commencer par les cellules les plus proches
  - on obtient des distances à des descripteurs, plus proches voisins temporaires.
  - ces distances sont moins lâches que celles déterminées par les bornes des cellules.
  - cela permet d'écarter davantage de cellules qui auraient pu être pré-sélectionnées dans un premier temps

Inria

Grenoble INP ensimag

Première structure de référence: R-Tree

- Guttman, 1984, ACM SIGMOD

- "Extension" des B/B+-Tree aux espaces multi-dimensionnels
  - arbre équilibré
  - chaque noeud représente un rectangle englobant les rectangles des fils
  - hiérarchie de rectangle

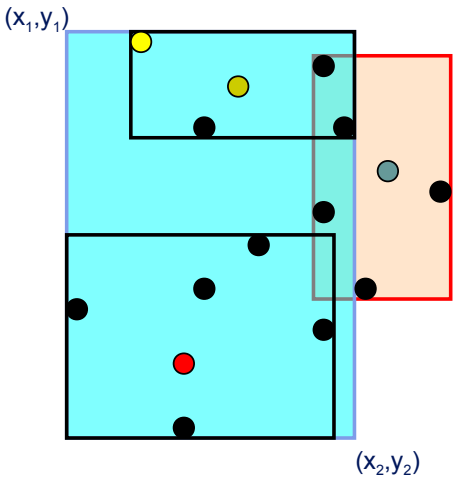
- **Notation** : REM = rectangle englobant minimum
  (*MBR=minimum bounding rectangle*)

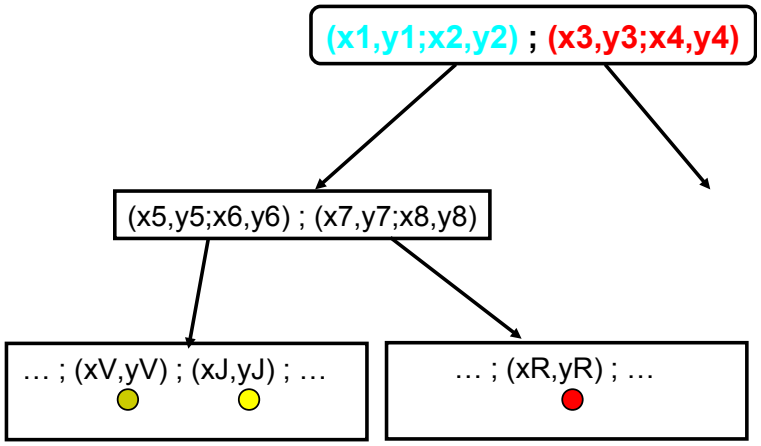
- Arbre équilibré, rectangles englobant minima, hiérarchie de rectangles

Inria

Grenoble INP ensimag

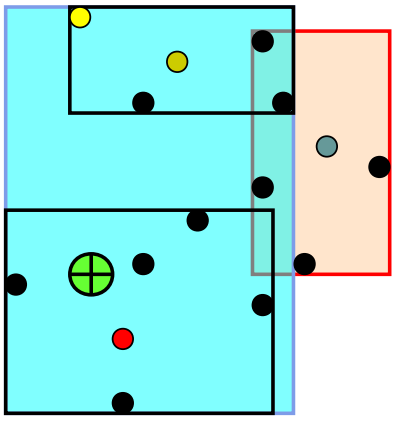
R-Tree

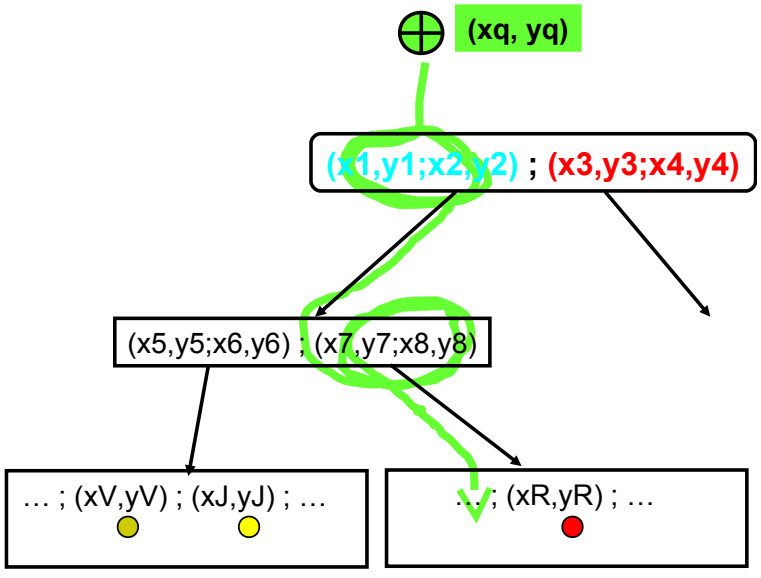




Un rectangle sur un niveau englobe les rectangles des noeuds fils

R-Tree (suite)





La recherche procède en descendant l'arbre en utilisant les coordonnées des rectangles

Mais la requête peut tomber dans plusieurs régions à la fois (≠B+tree)

R-Tree : insertion

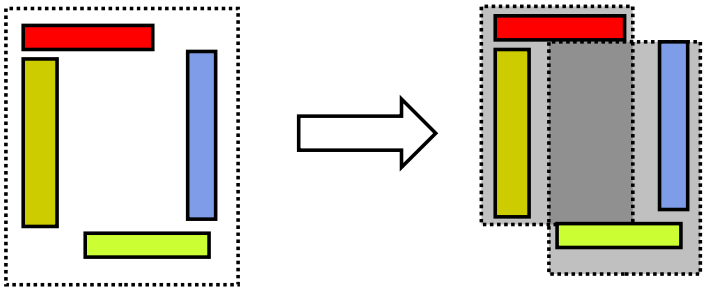
- Insertion dynamique de vecteurs
- Les arbres sont traversés du noeud racine vers les feuilles

Pour l'insertion, à chaque niveau

- v ∈ 1 REM → aucun problème
- v ∈ plusieurs REM : insertion dans le plus petit
- v ∈ aucun REM : insertion le REM qui grossira le moins

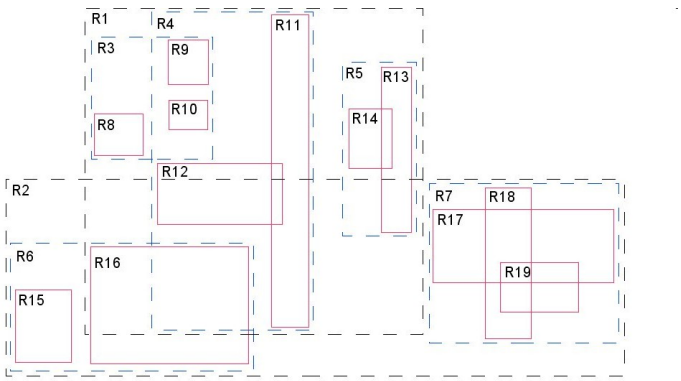
- Les noeuds et les feuilles ont une capacité d'accueil constante
  - Ils doivent être coupés en cas de dépassement de capacité

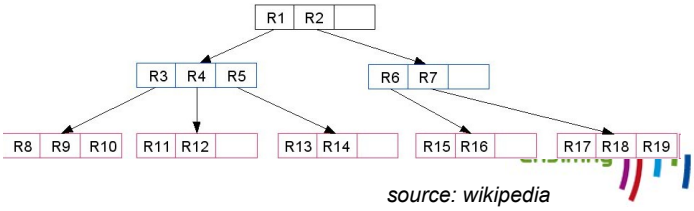
Pb:



R-Tree : conclusion

- La structure en arbre rend la descente théoriquement rapide (log)
- mais peu efficace en grande dimension car autorisation de l'overlapping : peu de rectangles sont filtrés (parcours exhastif de l'arbre)
- Intéressant pour les systèmes d'information géographique (2D)





source: wikipedia

Kd-Tree

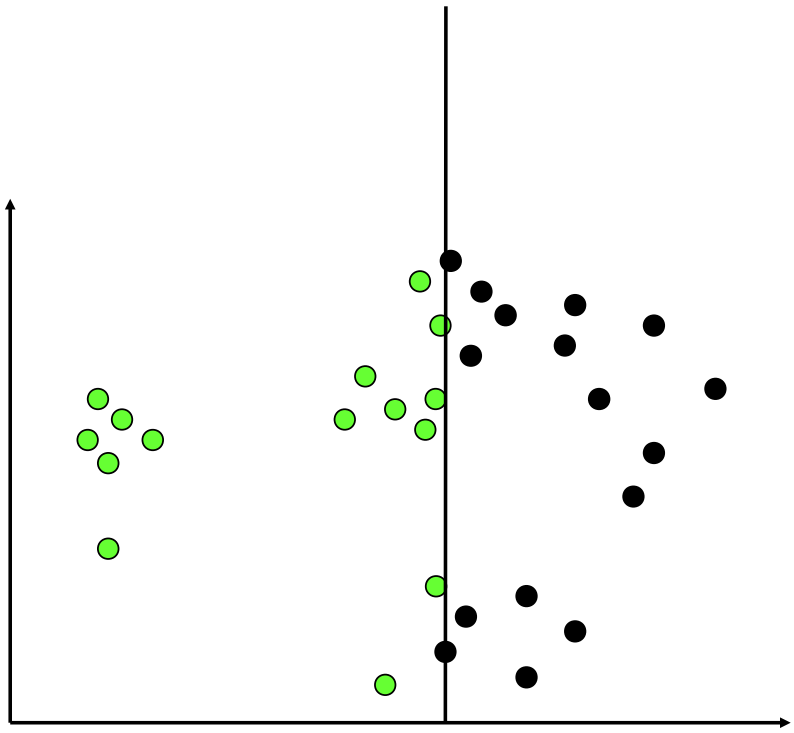
- Technique (populaire) de partitionnement de l'espace des vecteurs

- Espace découpé récursivement en utilisant des hyperplans
  - parallèles aux axes (base de représentation initiale du vecteur)
  - point de découpage = valeur médiane sur l'axe considéré
  - choix du prochain axe : plus grande variance résiduelle
    - mais variantes

Inria

Grenoble INP ensimag

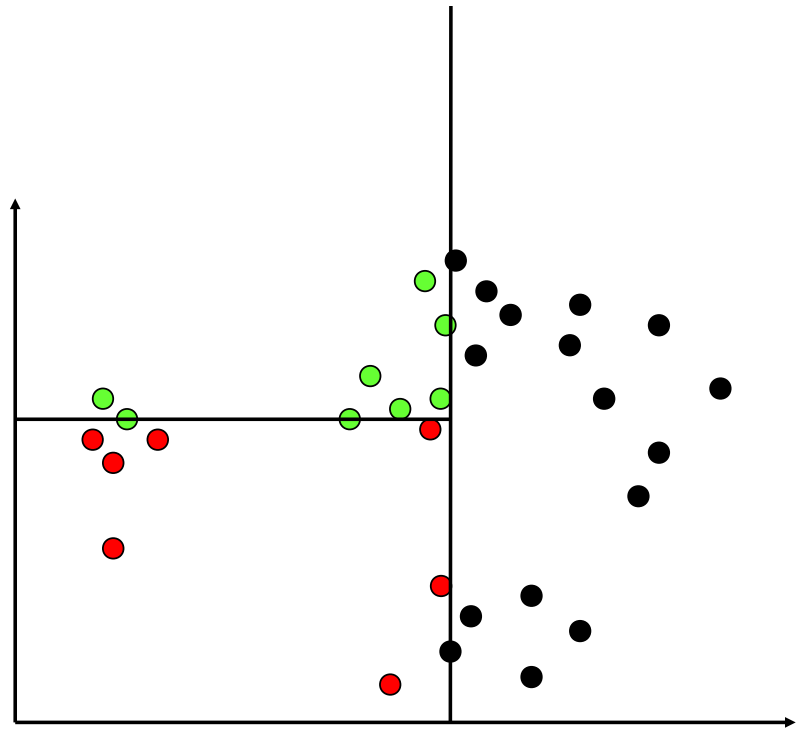
Kd-Tree : construction



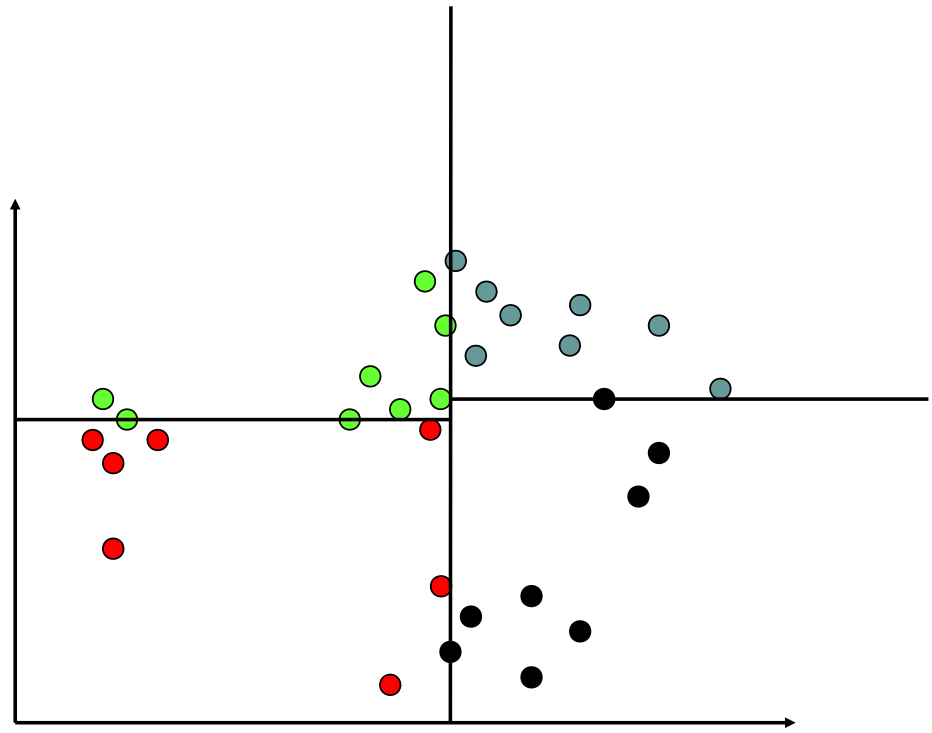
Inria

Grenoble INP ensimag

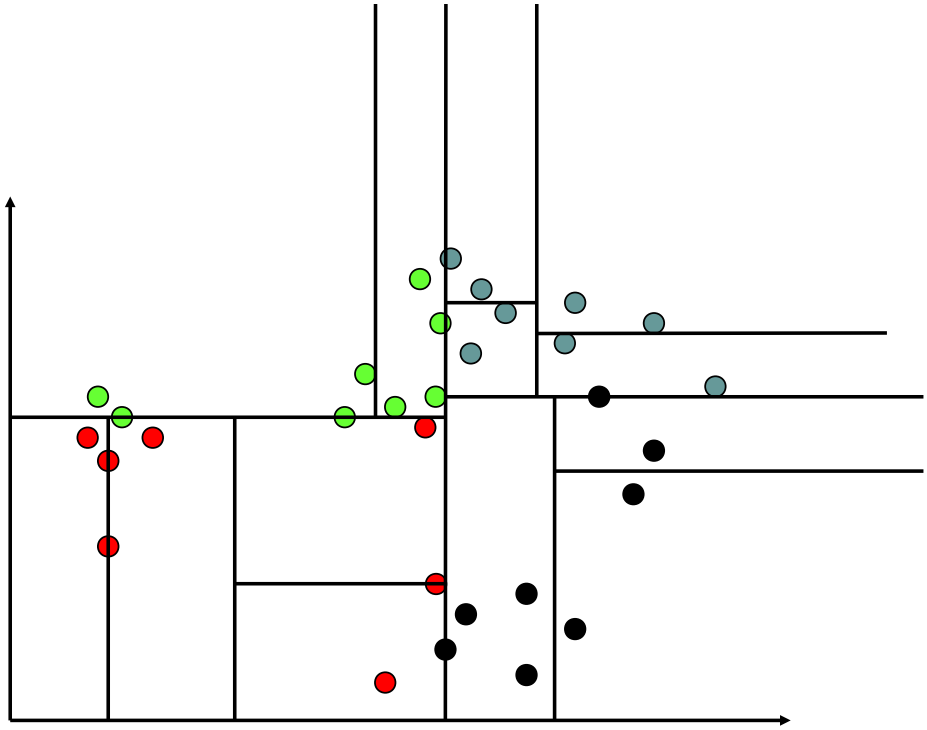
Kd-Tree : construction



Kd-Tree : construction



Kd-Tree : construction



Kd-Tree

- C'est une partition de l'espace
  - toutes les cellules sont disjointes (pas de recouvrement)
  - cellules de tailles initialement équilibrées (par construction)
  - mais se déséquilibrent si insertion de vecteurs additionnels

- peu efficace en grande dimension (ne filtre que peu de cellules)

Kd-Tree : recherche du ppv

- arbre de recherche
  - descente dans l'arbre (log) pour trouver la cellule ou tombe le point
- on cherche le ppv dans la cellule
  → donne la meilleure distance r
- recherche de l'intersection entre l'hypersphère de rayon r et l'hyperplan séparant la cellule
  - si non vide, analyse de l'autre côté de l'hyperplan
  - si un point plus proche est trouvé, on met à jour r
- on analyse toutes les cellules de l'arbre qu'on l'on ne peut filtrer
  - dans le pire des cas, parcours total de l'arbre

Inria

Grenoble INP ensimag

Kd-tree : variantes

- Elles sont nombreuses…
- BSP Tree (Binary Space Partitioning)
  - la subdivision n'est pas parallèle au système de coordonnées
- Variations
  - où on coupe une dimension donnée
  - quelle dimension est coupée
  - arbre équilibré ou non…

Inria

Grenoble INP ensimag

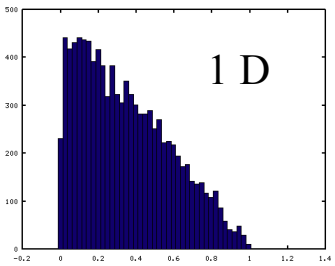
Malédiction de la dimension!

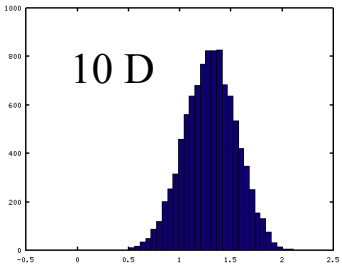
- ou « fléau de la dimension » (*The dimension curse*)

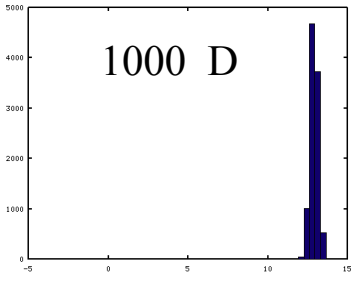
- Quelques propriétés surprenantes
  - *vanishing variance*
  - phénomène de l'espace vide
  - proximité des frontières

Vanishing variance

- La distance entre paires de points tend à être identique quand d augmente
  - le plus proche voisin et le point le plus éloigné sont à des distances qui sont presque identiques
  - exemple avec données uniformes:



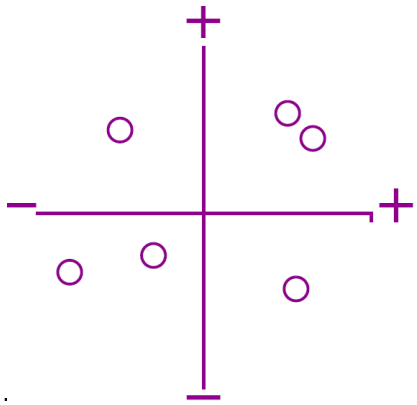




- Conséquence
  - le plus proche voisin devient très instable
  - inégalité triangulaire peu efficace pour filtrer
- Les jeux de données uniformes ne peuvent pas être indexés efficacement
  - c'est moins vrai pour des données naturelles (ouf !)

Phénomène de l'espace vide

- Cas d'école : partition de l'espace selon le signe des composantes



- d=100 → 1.26 10^{30} cellules >> n
- Très peu de cellules sont remplies
  - pour une partition pourtant grossière…

- Ce phénomène est appelé « phénomène de l'espace vide »
  - difficulté pour créer une partition
    - une bonne répartition des points
    - avec une bonne compacité

Inria

Grenoble INP ensimag

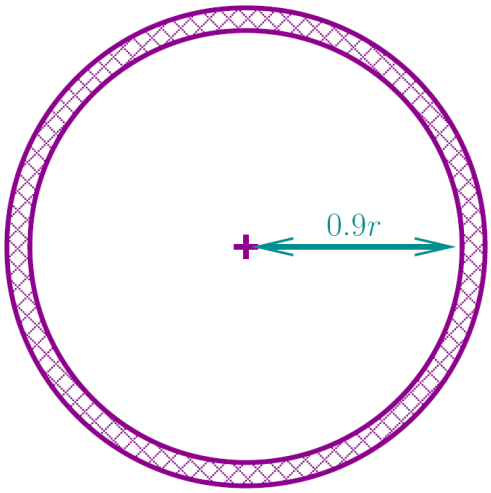
Tout les vecteurs sont près des frontières

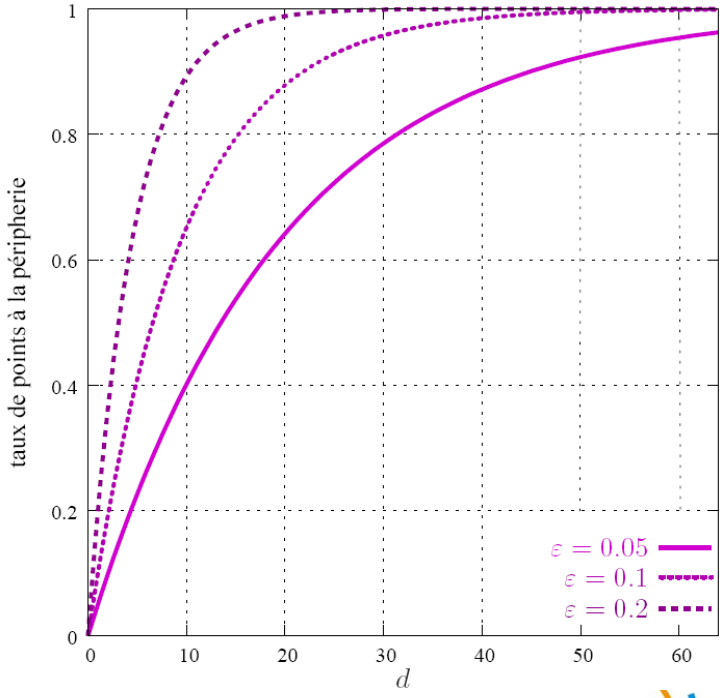
- Pour un partitionnement de l'espace
  - les vecteurs sont très proches des surfaces de séparation avec une très grande probabilité
  - le plus proche voisin d'un point appartient à une cellule différente avec une grande probabilité

Inria

Grenoble INP ensimag

Tout les vecteurs sont près des frontières



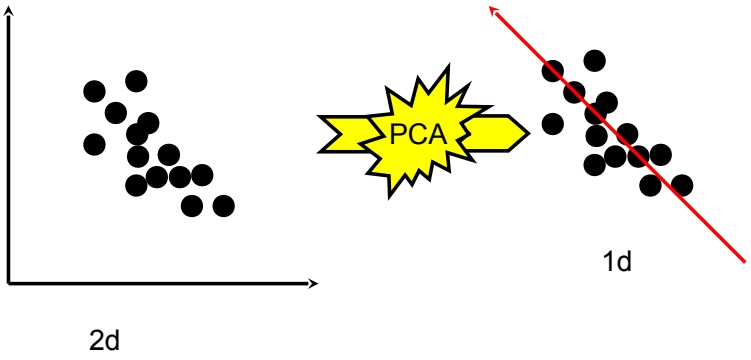


Réduction de la dimensionnalité

- “La dimension gagne toujours” (*Bill Triggs*)
- Réduisons la!
- Approche la plus courante : analyse en composantes principales (ACP)
  - PCA en anglais : *Principal Component Analysis*

ACP

- Analyse des relations statistiques entre les différentes composantes
- Pour être capable de reproduire la plus grande partie de l'énergie d'un vecteur avec un nombre plus faible de dimension
  - élimination des axes peu énergétiques



ACP

- Il s'agit d'un changement de base
  - translation (centrage des données) + rotation
- 4 étapes (off-line)
  - centrage des vecteurs
  - calcul de la matrice de covariance
  - calcul des valeurs propres et vecteurs
  - choix des *d'* composantes les plus énergétiques (+ grandes valeurs propres)
- Pour un vecteur, les nouvelles coordonnées sont obtenues par centrage et multiplication du vecteur par la matrice *d' x d* des vecteurs propres conservés

ACP

- Moins de dimensions, donc meilleur comportement des algorithmes de recherche
- Si la réduction préserve la plupart de l'énergie
  - les distances sont préservées
  - donc le + proche voisin dans l'espace transformé est aussi le plus proche voisin dans l'espace initial
- Limitations
  - utile seulement si la réduction est suffisamment forte
  - lourdeur de mise à jour de la base (choix fixe des vecteurs propres préférable)
  - peu adapté à certains types de variétés (idéal : sous-espace linéaire)

Inria

Grenoble INP ensimag

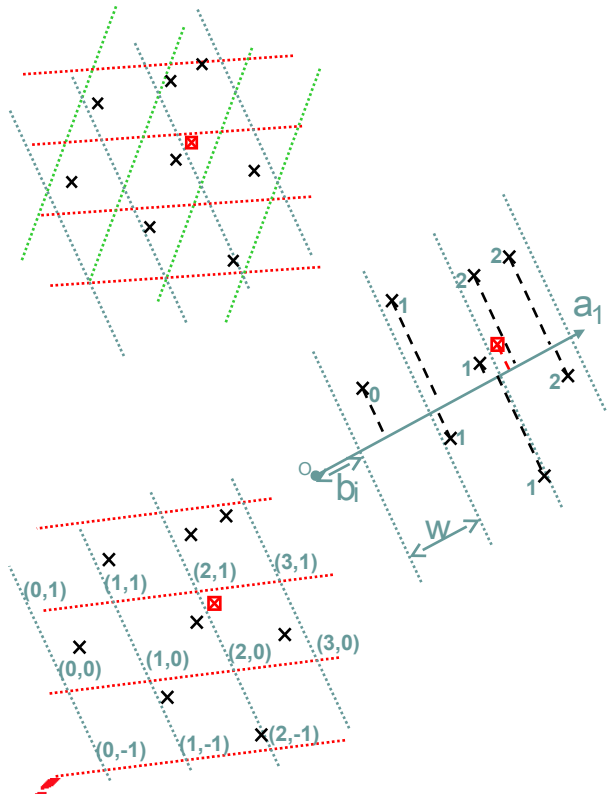
LSH (1999)

- Constat d'échec: pas d'algorithme de recherche de ppv efficace en grande dimension
  - pour d>15, il vaut mieux faire une recherche exhaustive !
- Nouvel objectif : trouver nos voisins en probabilité
  - que l'on peut faire varier en fonction du compromis pertinence/coût recherché
  - c'est la recherche *approximative* de plus proches voisins
  - échec lorsque le plus proche voisin n'est pas significativement plus proche
    → pas forcément grave en pratique
- Explication au tableau

Inria

Grenoble INP ensimag

Euclidean Locality Sensitive Hashing (E2LSH)



1) Projection sur *m* directions aléatoires

$$h_i(x) = \lfloor \frac{x.a_i - b_i}{w} \rfloor$$

2) Construction de fonctions de hachage: concatenation de *k* index h_i par fonction de hash

$$g_j(x) = (h_{j1}, \ldots, h_{jk})$$

3) Pour chaque g_j, calcul de deux valeurs de hash

- fonctions de hash universelles: $u_1(.)$, $u_2(.)$

- stockage de l'identifiant *id* du vecteur dans une table de hashage

Et la recherche d'images?

- LSH marche correctement, mais pas à très grande échelle
  - nécessite beaucoup de hachages distincts de l'espace
  - coût mémoire prohibitif en grande dimension
  - temps de recherche linéaire avec le nombre de hachage

- Utilisable jusqu'à (disons), 100×10^6 descripteurs (100000 images)

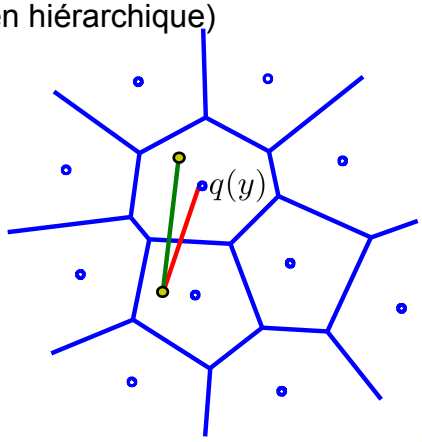
- Le domaine de la recherche approximative est très actif
  - outil de base dans de nombreux domaines/applications

Améliorer la finesse de la quantification: le quantificateur produit

- On a un pb d'approximation de distances entre requête x et vecteur y de la base

$$d(x, y) \approx d(x, q(y))$$

- il faut une quantification plus fine que quelques milliers de centroides
  - k-means coûteux
  - assignement coûteux
  - stockage des centroïdes coûteux (même en hiérarchique)
- Solution: le quantifieur produit (PQ)



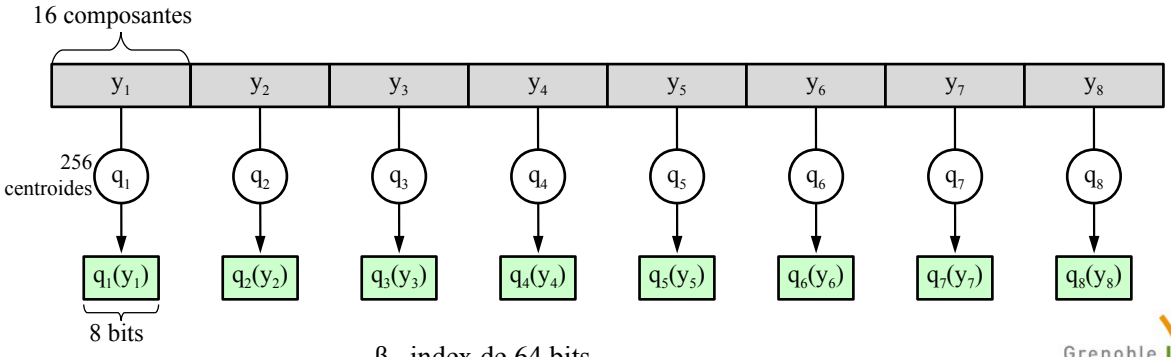
Quantificateur produit prour recherche de plus proches voisins

- Vecteur coupé en *m* sous-vecteurs: $y \rightarrow [y_1|\ldots|y_m]$
- Sous-vecteurs quantifiés séparément:

$$q(y) = [q_1(y_1)|\ldots|q_m(y_m)]$$

chaque *qi* a un petit vocabulaire

- Exemple: y = 128 D en 8 sous-vecteurs de dimension 16



ß index de 64 bits

Quantificateur prodiut : calcul de distance asymétrique (ADC)

- Calculer la distance dans le domaine compressé

$$d(x,y)^2 \approx \sum_{i=1}^{m} d(x_i, q_i(y_i))^2$$

- Pour calculer la distance entre la requête et beaucoup de codes
  - calculer $d(x_i, c_{i,j})^2$ la distance avec tous les centroides

  → stockés dans tables de look-up
  - pour chaque code: accumuler les distances élémentaires

- Chaque code de 64 bits nécessite 8 additions seulement.

Inria

Grenoble INP ensimag