Techniques de recherche approximatives

Algorithmes et structures pour la recherche par similarité: Séance 3

Alexis Joly
http://www-rocq.inria.fr/ ajoly/

INRIA Rocquencourt, Equipe-projet IMEDIA

April 28, 2008

Outline

Outline

Outline

Chapitre VIII

Locality-Sensitive Hashing

Focus

Données:

- Espace Euclidien d-dimensionel : $\mathbb{R}^d$

Focus

Données:

- Espace Euclidien d-dimensionel : $\mathbb{R}^d$

Objectif: Recherche approximative avec performances bornées

- Temps de recherche sous-linéaire, taille de l'index linéaire

Focus

Données:

- Espace Euclidien d-dimensionel : $\mathbb{R}^d$

Objectif: Recherche approximative avec performances bornées

- Temps de recherche sous-linéaire, taille de l'index linéaire

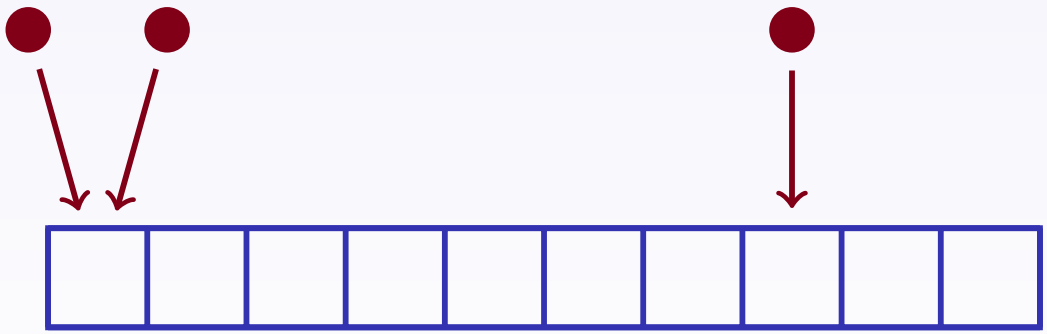
Encore un problème ouvert: Recherche approximative avec temps logarithmique et taille linéaire

Définition de LSH

Indyk&Motwani'98

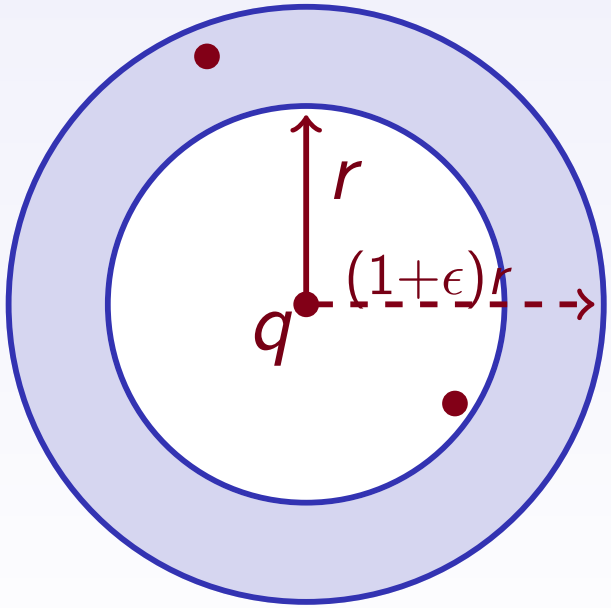
Famille de fonctions de hachage locality-sensitive $\mathcal{H}$ avec paramètres (ϵ, r, P_1, P_2):

- Si $\|p - q\| \leq r$ alors $\mathcal{P}r_{\mathcal{H}}[h(p) = h(q)] \geq P_1$
- Si $\|p - q\| \geq (1 + \epsilon)r$ alors $\mathcal{P}r_{\mathcal{H}}[h(p) = h(q)] \leq P_2$



Définition: requêtes à un rayon près approximatives

Requêtes à un rayon r près $(1+\epsilon)$-approximatives: s'il y a au moins un $p \in S$: $d(q,p) \leq r$ retourne des p' : $d(q,p') \leq (1+\epsilon)r$



Efficacité de LSH

Notation: $\rho = \frac{\log(1/P_1)}{\log(1/P_2)} < 1$

Theorem

Toute famille de fonctions de hachage (ϵ, r, P_1, P_2)-locality-sensitive conduit à un algorithme pour la recherche à un rayon r près $(1+\epsilon)$-approximative avec $\sim n^{\rho}$ de temps de recherche et $n^{1+\rho}$ de taille d'index

Efficacité de LSH

Notation: $\rho = \frac{\log(1/P_1)}{\log(1/P_2)} < 1$

Theorem

Toute famille de fonctions de hachage (ϵ, r, P_1, P_2)-locality-sensitive conduit à un algorithme pour la recherche à un rayon r près $(1+\epsilon)$-approximative avec $\sim n^{\rho}$ de temps de recherche et $n^{1+\rho}$ de taille d'index

Preuve dans les prochains slides

LSH: Construction des index

Function de hachage multidimensionnelle:
$g(p) =< h_1(p), \ldots, h_k(p) >$

LSH: Construction des index

Function de hachage multidimensionnelle:
$g(p) =< h_1(p), \ldots, h_k(p) >$

Preprocessing avec paramètres L, k:

1. Choisir aléatoirement L fonction de hachage composée de k composantes chacune
2. Hacher chaque $p \in S$ dans les L "cases" $g_1(p), \ldots, g_L(p)$

Taille de l'index: $\mathcal{O}(Ln)$

LSH: Recherche

1. Calculer les L empreintes $g_1(q), \ldots, g_L(q)$
2. Aller aux cases correspondantes et retourner les objets contenus
3. En pratique: vérifier explicitement $d(p, q) \leq ? r$ pour les objet retournés

Temps de recherche est $\mathcal{O}(L)$

LSH: Analyse (1/2)

Pour avoir une probabilité d'erreur au plus égale à δ il faut prendre k, L tels que

$$P_2^k n \approx 1 \qquad\qquad L \approx (1/P_1)^k \log(1/\delta)$$

LSH: Analyse (1/2)

Pour avoir une probabilité d'erreur au plus égale à δ il faut prendre k, L tels que

$$P_2^k n \approx 1 \qquad\qquad L \approx (1/P_1)^k \log(1/\delta)$$

Avec ces contraintes on obtient en effet:

$$k = \frac{\log n}{\log(1/P_2)}$$

LSH: Analyse (1/2)

Pour avoir une probabilité d'erreur au plus égale à δ il faut prendre k, L tels que

$$P_2^k n \approx 1 \qquad\qquad L \approx (1/P_1)^k \log(1/\delta)$$

Avec ces contraintes on obtient en effet:

$$k = \frac{\log n}{\log(1/P_2)}$$

$$L = (1/P_1)^{\frac{\log n}{\log(1/P_2)}} \log(1/\delta) = n^{\frac{\log(1/P_1)}{\log(1/P_2)}} \log(1/\delta) = n^{\rho} \log(1/\delta)$$

LSH: Analyse (2/2)

L'espérance du nombre d'objets rencontrés à moins de $(1+\epsilon)r$ de la requête est $P_2^k Ln \approx L$

Pour les vrais voisins à r-près la probabilité d'être hachés dans une même case que q est au moins

$1-(1-(1/P_1)^k)^L \geq 1-(1/e)^{\frac{L}{(1/P_1)^k}} \geq 1-\delta$

LSH: Analyse (2/2)

L'espérance du nombre d'objets rencontrés à moins de $(1+\epsilon)r$ de la requête est $P_2^k Ln \approx L$

Pour les vrais voisins à r-près la probabilité d'être hachés dans une même case que q est au moins

$$1-(1-(1/P_1)^k)^L \geq 1-(1/e)^{\frac{L}{(1/P_1)^k}} \geq 1-\delta$$

Taille de l'index $\mathcal{O}(Ln) \approx n^{1+\rho+o(1)}$
Recherche $\mathcal{O}(L) \approx n^{\rho+o(1)}$

Hashing dans une grille de sphères: idée

1. Projeter l'espace initial dans un espace t-dimensionel (Transformation A)

Hashing dans une grille de sphères: idée

1. Projeter l'espace initial dans un espace t-dimensionel (Transformation A)
2. Couvrir le nouvel espace par U grilles alternées de sphères de rayon w (pas $4w$)

Hashing dans une grille de sphères: idée

1. Projeter l'espace initial dans un espace t-dimensionel (Transformation A)
2. Couvrir le nouvel espace par U grilles alternées de sphères de rayon w (pas $4w$)
3. Hacher les objets p dans le case correspondant à la première boule contenant $A(p)$

Initialization

Paramètres: $t = \log^{2/3} n, w = r \log^{1/6} n, U = 2^{t \log t} \log n$

- Construire la matrice A $(d \times t)$ en prenant chaque élément aléatoirement selon une loi normale $N(0, \frac{1}{\sqrt{t}})$
- Pour chaque $1 \leq i \leq U$ choisir un décalage aléatoire $\bar{v}_i \in [0, 4w]^t$

BG Hashing: Calcul des empreintes d'un objet

1. Calculer $p' = A(p)$
2. Pour $i = 1$ à U vérifier si p' est couvert par la i-ème grille de sphère. Si oui retourne l'index i, le centre de la sphère et stoppe.
3. Si une telle sphère n'existe pas: échec

BG Hashing: Analyse

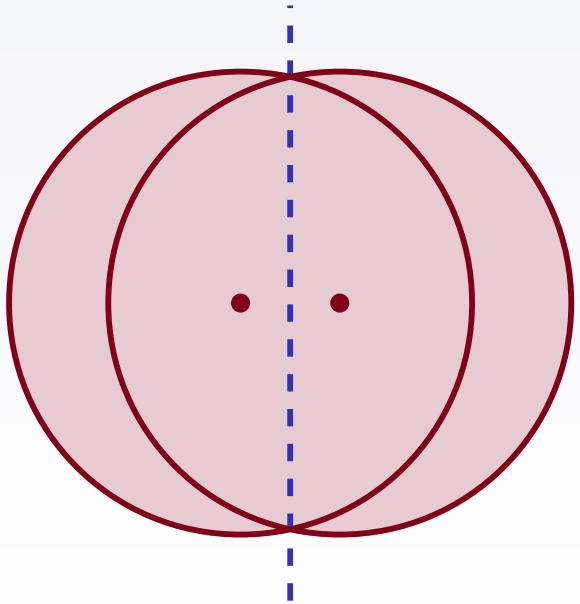
Observation: La probabilité que $\frac{\|Ap-Ap'\|}{\|p-p'\|} \notin [1-\varepsilon, 1+\varepsilon]$ est au plus $\exp(-\varepsilon^2 t)$

BG Hashing: Analyse

Observation: La probabilité que $\frac{\|Ap-Ap'\|}{\|p-p'\|} \notin [1-\varepsilon, 1+\varepsilon]$ est au plus $\exp(-\varepsilon^2 t)$

Etant donnés deux points $p, s \in \mathbb{R}^t : \|p-s\| = \Delta$:

$$Pr[h(p) = h(s)] = \frac{B(p,w) \cap B(s,w)}{B(p,w) \cup B(s,w)}$$



BG Hashing: Résultat final

Après 3 pages de calcul:

$$\rho = \frac{\log(1/P_1)}{\log(1/P_2)} = 1/c^2 + o(1)$$

BG Hashing: Résultat final

Après 3 pages de calcul:

$$\rho = \frac{\log(1/P_1)}{\log(1/P_2)} = 1/c^2 + o(1)$$

Theorem (Andoni & Indyk 2006)

Soit une requête c-approximative à un rayon près r dans un espace d-dimensionnel. Alors pour tout δ il y a un unique algorithme aléatoire avec environ $n^{1/c^2+o(1)}$ de temps de recherche et $n^{1+1/c^2+o(1)}$ de taille d'index. Pour toute requête cet algorithme répond correctement une probabilité de au moins $1-\delta$

Future de LSH

Avantage:

- Temps de filtrage sous-linéaire théorique pour requêtes approximatives à $(1+\epsilon)r$ près, quelles que soit les données

Future de LSH

Avantage:

- Temps de filtrage sous-linéaire théorique pour requêtes approximatives à $(1+\epsilon)r$ près, quelles que soit les données

Inconvénients:

- La probabilité d'erreur ne peut amplifiée que pendant le preprocessing, elle ne peut être inférieure à $1/n$
- Analysis asymptotique de la puissance ρ: dans quelle mesure $n^{1/c^2+o(1)}$ est-il vraiment sous-linéaire ?
- Pour la recherche des plus proches voisins $c = \max \frac{r_{NN}(q)}{r_{FN}(q)}$, où $r_{FN}(q)$ est le voisin le plus éloigné. On aimerait que ce soit proche de 1.

Chapitre IX

Projections aléatoires

Focus

Données:

- Cube de Hamming: $\{0,1\}^d$ avec distance de Hamming

Focus

Données:

- Cube de Hamming: $\{0,1\}^d$ avec distance de Hamming

Objectif: Recherche approximative avec performances bornées

- Temps de recherche logarithmique, taille de l'index polynomiale

Focus

Données:

- Cube de Hamming: $\{0,1\}^d$ avec distance de Hamming

Objectif: Recherche approximative avec performances bornées

- Temps de recherche logarithmique, taille de l'index polynomiale

Encore un problème ouvert: Recherche approximative avec temps logarithmique et taille linéaire

Principe

Problème: Requêtes $(1+\varepsilon)$-approximative à un rayon l-près dans cube de Hamming d-dimensionnel

Principe

Problème: Requêtes $(1+\varepsilon)$-approximative à un rayon l-près dans cube de Hamming d-dimensionnel

- Appliquer une réduction $\{0,1\}^d$ dans $\{0,1\}^k$ telle que les l-voisins tombent généralement à $\delta_1 k$ les uns des autres, tandis que les objets distants de $(1+\varepsilon)l$ tombent à $\delta_2 k$ les uns des autres
- précalculer les $(\frac{\delta_1+\delta_2}{2})k$-voisins pour tout point dans $\{0,1\}^k$
- Pendant la recherche, projeter q et vérifier explicitement tous les $(\frac{\delta_1+\delta_2}{2})k$-voisins précalculés

Pr. Al.: Test du produit scalaire

Test simple:

- Choisir aléatoirement un ensemble de composantes selon une loi binomiale de paramètre $\frac{1}{2l}$, (espérance du nombre de composantes $\frac{d}{2l}$)
- Affecter aléatoirement 0 ou 1 à toutes ces positions, affecter les autres à 0. Soit r le vecteur résultat.
- $h_r(p) = r \cdot p$

Pr. Al.: Test du produit scalaire

Test simple:

- Choisir aléatoirement un ensemble de composantes selon une loi binomiale de paramètre $\frac{1}{2l}$, (espérance du nombre de composantes $\frac{d}{2l}$)
- Affecter aléatoirement 0 ou 1 à toutes ces positions, affecter les autres à 0. Soit r le vecteur résultat.
- $h_r(p) = r \cdot p$

Observation: Il existe des constantes $\delta_1 > \delta_2$

- $H_d(p, s) \leq l \Rightarrow Pr[h(p) = h(q)] \geq \delta_1$
- $H_d(p, s) \geq (1 + \varepsilon)l \Rightarrow Pr[h(p) = h(q)] \leq \delta_2$

Pr. AI.: Prétraitement

Projection par produit scalaire:

- Choisir k tests aléatoires $r_1, \ldots, r_k$
- Projeter tous les p dans $A(p) = h_{r_1}(p) \ldots h_{r_k}(p)$

Pr. AI.: Prétraitement

Projection par produit scalaire:

- Choisir k tests aléatoires $r_1, \ldots, r_k$
- Projeter tous les p dans $A(p) = h_{r_1}(p) \ldots h_{r_k}(p)$

Structure

- Appliquer le produit scalaire à tous les objets de la base
- Pour tout $v \in \{0,1\}^k$ précalculer tous les $(\frac{\delta_1+\delta_2}{2})k$-voisins

Pr. AI.: Recherche

- Calculer $A(q) = h_{r_1}(q) \dots h_{r_k}(q)$
- Retrouver et vérifier explicitement les $(\frac{\delta_1+\delta_2}{2})k$-voisins de $A(q)$

Pr. AI.: Recherche

- Calculer $A(q) = h_{r_1}(q) \dots h_{r_k}(q)$
- Retrouver et vérifier explicitement les $(\frac{\delta_1+\delta_2}{2})k$-voisins de $A(q)$

Analyse:

- Probabilité de rater les vrais l-voisins: $\exp(-\frac{\delta_1-\delta_2}{2\delta_1}k)$
- Probabilité de perdre du temps sur un $(1+\varepsilon)l$-objet: $\exp(-\frac{\delta_1-\delta_2}{2\delta_1}k)$

Pr. AI.: Recherche

- Calculer $A(q) = h_{r_1}(q) \dots h_{r_k}(q)$
- Retrouver et vérifier explicitement les $(\frac{\delta_1+\delta_2}{2})k$-voisins de $A(q)$

Analyse:

- Probabilité de rater les vrais l-voisins: $\exp(-\frac{\delta_1-\delta_2}{2\delta_1}k)$
- Probabilité de perdre du temps sur un $(1+\varepsilon)l$-objet: $\exp(-\frac{\delta_1-\delta_2}{2\delta_1}k)$

Il faut prendre un k logarithmique qui conduit à une taille d'index polynomiale pour $\{0,1\}^k$ dont il faut précalculer les PPV.

Pr. AI.: Formellement

Theorem (Kushilevetz, Ostrovsky, Rabani, 1998)

Soit une requête à un rayon l près $(1+\varepsilon)$-approximative dans le cube de Hamming d-dimensionnel. Alors pour tout μ il y a unique algorithme aléatoire avec $d^2 polylog(d, n)$ temps de recherche et $n^{\mathcal{O}(\varepsilon^{-2})}$ taille d'index. Pour toute requête, cet algorithme répond correctement avec une probabilité de au moins $1-\mu$

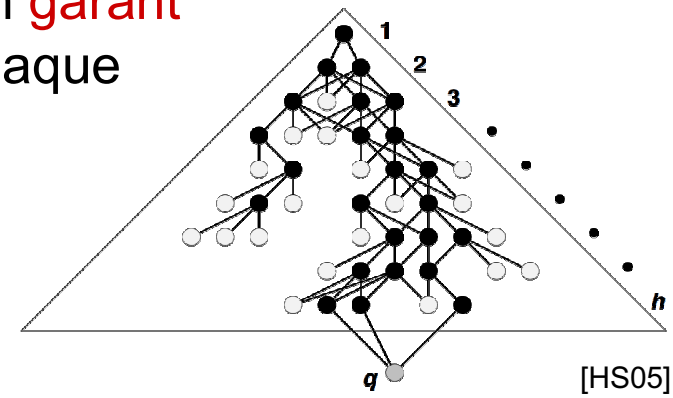
Chapitre X

Etude de SASH

SASH

(*Spatial Approximation Sample Hierarchy* [HS05])

- Méthode conçue pour fournir des réponses approximatives aux requêtes *k*NN, avec un contrôle du temps de réponse
- Structure d'une SASH :
  - Relation bijective entre objets et nœuds
  - N objets dans la base $\Rightarrow$ le niveau l de la SASH contient 2^l nœuds (le niveau $h = \log_2 N - 1$ est le plus bas et contient $N/2$ nœuds)
  - Chaque nœud (hors racine) doit être relié à *p* parents et *c* descendants, $d(o,o')$ est stockée avec le lien (o,o')
  - Chaque nœud (hors racine) doit avoir un garant désigné parmi ses parents (cela rend chaque nœud atteignable depuis la racine)

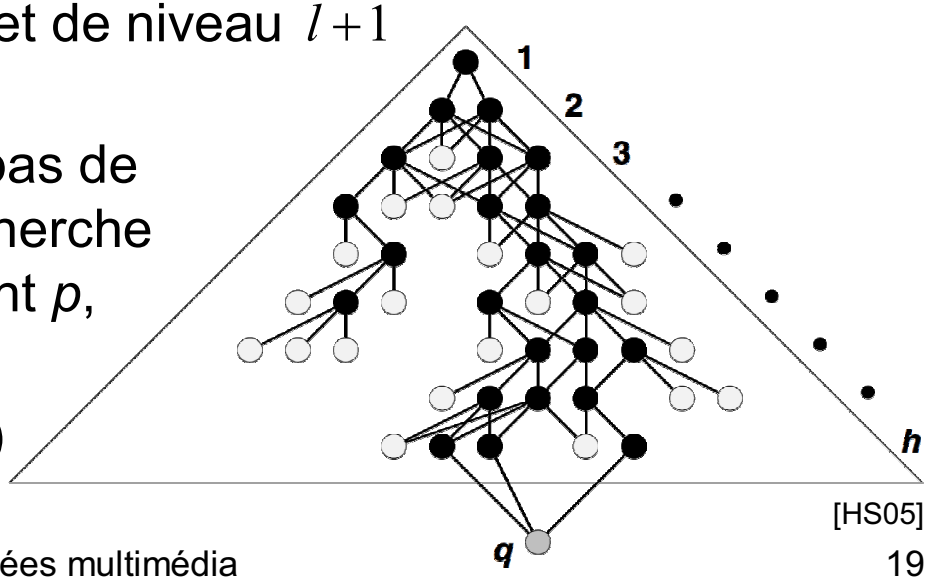


[HS05]

Construction d'une SASH

- Processus récursif : la racine est un objet choisi aléatoirement, le niveau $l+1$ de la SASH est obtenu à partir de la SASH(l) :
  1. Choix aléatoire de 2^l objets non présents dans SASH (l)
  2. Pour chaque objet de niveau $l+1$, choisir à chaque niveau $1 \leq i < l$ jusqu'à *p* plus proches voisins, en partant de la racine ; les parents de l'objet seront les *p* plus proches voisins de niveau l
  3. Pour chaque objet de niveau l, choisir comme descendants les *c* plus proches objets de niveau $l+1$ l'ayant choisi comme parent
  4. Le plus proche parent d'un objet de niveau $l+1$ devient son garant
  5. Si un objet de niveau $l+1$ n'a pas de parent de niveau l, on lui en cherche un en doublant progressivement *p*, puis on lui choisit un garant

  ($c > 2p \Rightarrow \exists$ garant pour tout nœud)



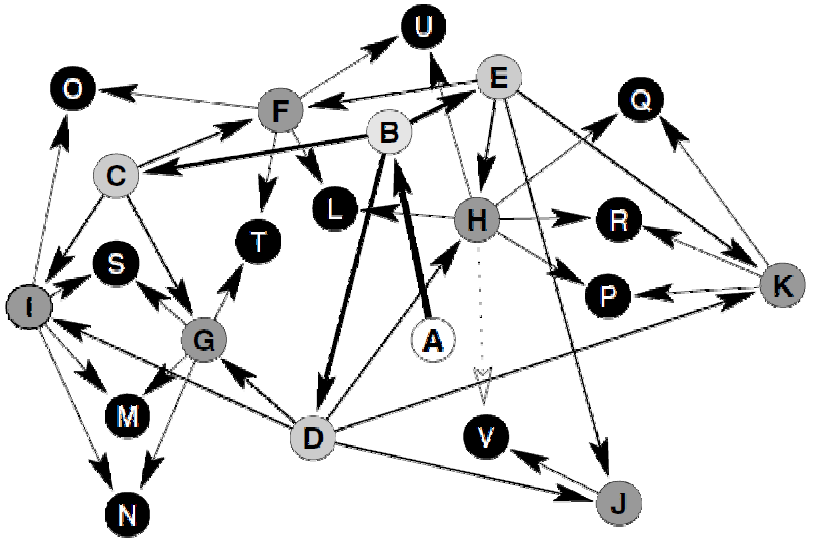
[HS05]

Construction d'une SASH (2)

- Structure essentiellement statique, peut s'accommoder d'insertions : les nouveaux objets sont mis dans un nouveau niveau $h+1$

- Peut être vu comme un index multi-résolution : la résolution augmente quand on descend dans la SASH

SASH sur 22 points 2D :



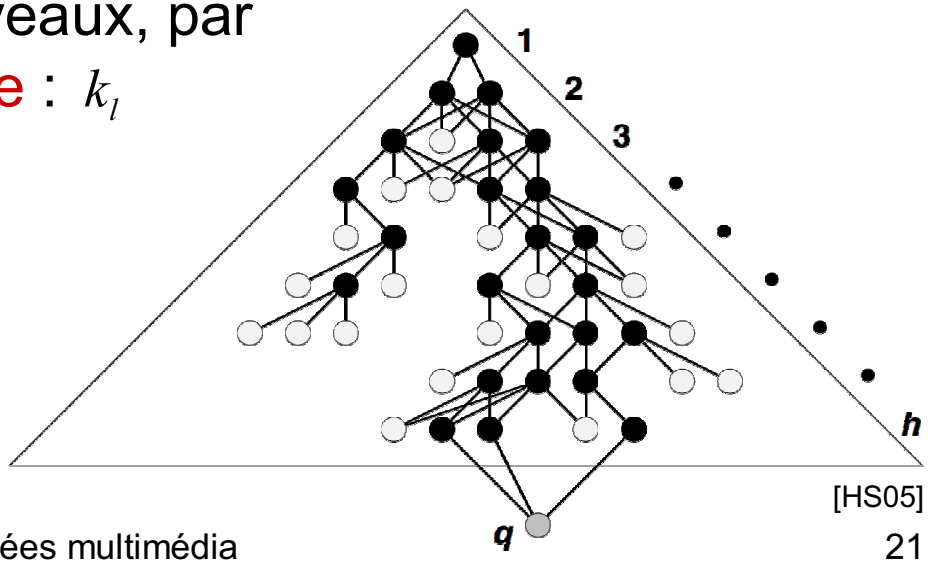
[HS05]

Level	#Nodes	
1	1	$\{A\}$
2	1	$\{B\}$
3	3	$\{C, D, E\}$
4	6	$\{F, G, H, I, J, K\}$
5	11	$\{L, M, N, O, P, Q, R, S, T, U, V\}$

SASH : requête *k* plus proches voisins

- Étant donnée une requête q, la recherche des *k*NN consiste à :
  1. Pour chaque niveau l, obtenir un ensemble de voisins $P_l(q,k_l)$
     1. $P_1(q,k_1) =$ racine
     2. Pour $l>1$, $P_l(q,k_l)$ est composé des k_l plus proches voisins de q parmi les descendants de $P_{l-1}(q,k_{l-1})$
  2. Retourner les k éléments de $\bigcup_{1\le l\le h} P_l(q,k_l)$ les plus proches de q
- Les k_l peuvent avoir tous la même valeur (sélection uniforme) ou des valeurs spécifiques aux niveaux, par exemple (sélection géométrique : k_l augmente avec la résolution)

$$k_l = \max\left\{ k^{1-\frac{h-l}{\log_2 N}},\ \frac{1}{2}pc \right\}$$



[HS05]

SASH : mesure de l'approximation

- Différentes possibilités pour évaluer la qualité des *k*NN approximatifs (U est le résultat retourné, r_i est la distance entre q et le vrai *i* ème plus proche voisin) :

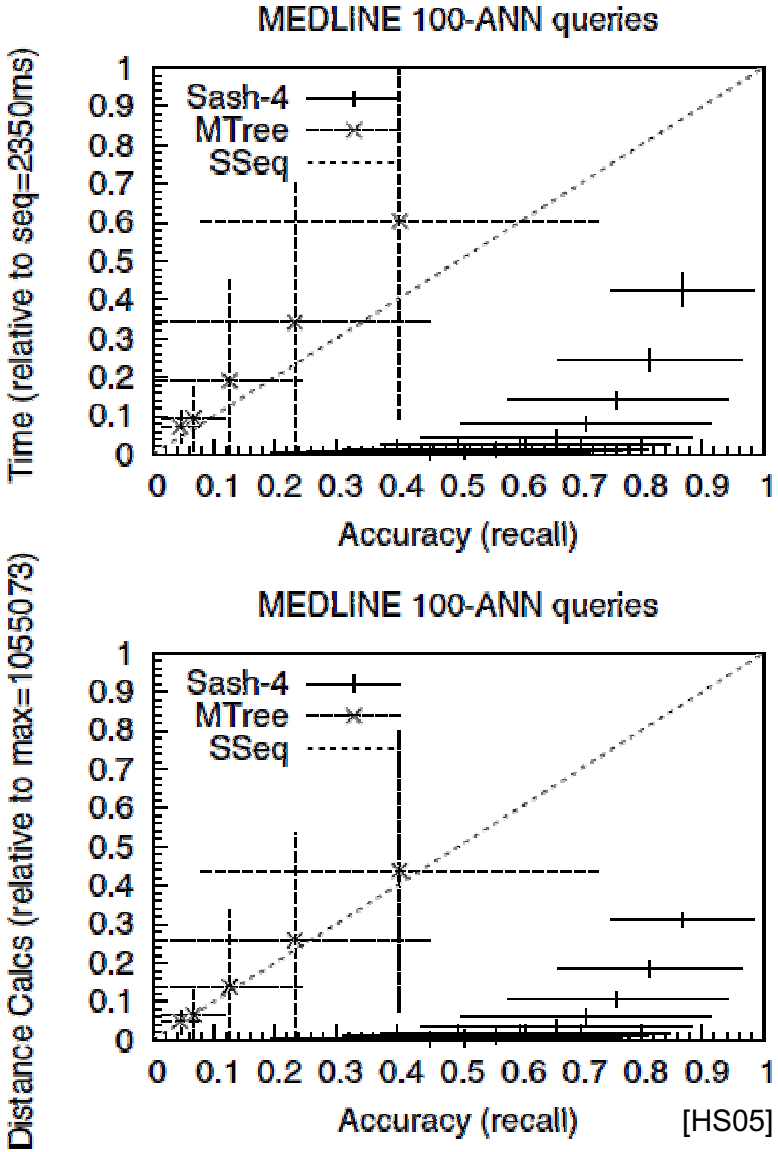
1. $$A_1(q,U) = \frac{\sum_{i=1}^{|U|} d(q,o_i)}{\sum_{i=1}^{|U|} r_i}$$

2. $$A_2(q,U) = \frac{1}{|U|} \sum_{i=1}^{|U|} \frac{d(q,o_i)}{r_i}$$ (voisins ordonnés : distances ↗)

3. $$A(q,U) = \frac{\text{nombre vrais } k\text{NN retrouvés}}{k}$$ (mesure de rappel)

SASH : performances comparatives

- Complexité algorithmique
  - Construction : $p\,c\,N\log_2 N$
  - Requête *k*NN :
    - Sélection uniforme : $c\,k\log_2 N$
    - Sélection géométrique : $k+\log_2 N$
- Exemple :
  - Données MEDLINE : 10^6 textes représentés par des vecteurs de dimension 10^6 (pondération TF-IDF, env. 75 attributs $\neq 0$ par vecteur)
  - $p=4, \quad c=4p$
  - Sélection géométrique
  - Comparaison avec M-*tree*



[HS05]

SASH : conclusion

- Les indexes-arbres classiques permettent de savoir où il n'est pas intéressant d'aller, une SASH indique où il faut chercher en priorité
- Le caractère approximatif des réponses est assumé et fondamental pour la méthode
- La qualité d'approximation n'est pas contrôlable
- Le temps de recherche est contrôlable car très peu dépendant de la requête
- La distribution des distances entre les objets indexés a un impact sur SASH : une variance trop faible diminue la sélectivité et réduit la qualité d'approximation

Exercice

Prouver qu'un nombre $2^{\mathcal{O}(t)}$ de grilles de sphères $(w, 4w)$ choisies aléatoirement est suffisant pour couvir l'espace t-dimensionnel avec une probabilité de $1/2$

En bref

- Locality-sensitive hashing: Utilisation de projections aléatoires définissant une liste de candidats qui sont vérifier explicitement

En bref

- Locality-sensitive hashing: Utilisation de projections aléatoires définissant une liste de candidats qui sont vérifier explicitement
- SASH: Approximation spatiale hiérarchique, pas besoin de l'inégalité triangulaire, utilise uniquement les relations de voisinage

En bref

- Locality-sensitive hashing: Utilisation de projections aléatoires définissant une liste de candidats qui sont vérifier explicitement
- SASH: Approximation spatiale hiérarchique, pas besoin de l'inégalité triangulaire, utilise uniquement les relations de voisinage

Merci de votre attention! Questions?

Références

Y. Lifshits
The Homepage of Nearest Neighbors and Similarity Search
http://simsearch.yury.name

A. Andoni, P. Indyk
Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. FOCS'06
http://web.mit.edu/andoni/www/papers/cSquared.pdf

E. Kushilevitz, R. Ostrovsky, Y. Rabani
Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces. STOC'98
http://www.cs.technion.ac.il/~rabani/pss/Publications/KushilevitzOR98.ps.gz

M. Houle, J. Sakuma
Fast Approximate Similarity Search in Extremely High Dimensional Data Sets ICDE'05