Etude du M-tree et autres utilisations de l'inégalité triangulaire

Algorithmes et structures pour la recherche par similarité: Séance 2

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

INRIA Rocquencourt, Equipe-projet IMEDIA

April 25, 2008

Plan

Plan

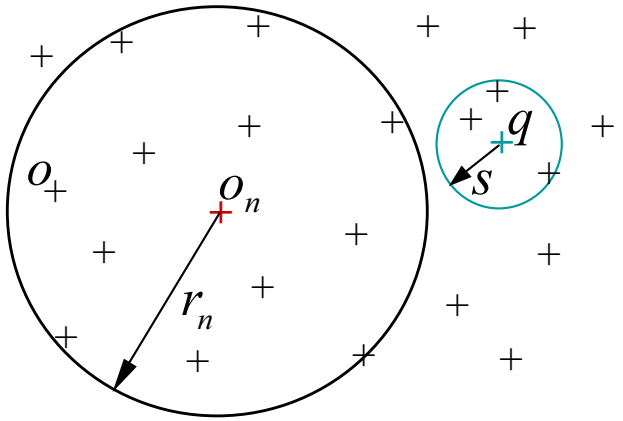
Plan

Chapitre V

Etude détaillée du M-Tree

M-*tree* : principe d'indexation

- Arbres de recherche basés sur le partitionnement des données
  1. Regroupement hiérarchique des données à partir de proximités
  2. Lors de la recherche, rejet des sous-arbres qui ne satisfont pas la contrainte de proximité
- Principe de l'indexation par M-*tree* [CPZ97]
  - Regroupement hiérarchique basé sur la métrique d
  - Critère de rejet issu de l'inégalité triangulaire :



Sous-arbre $N(o_n)$: objets o tels que $d(o,o_n) \leq r_n$

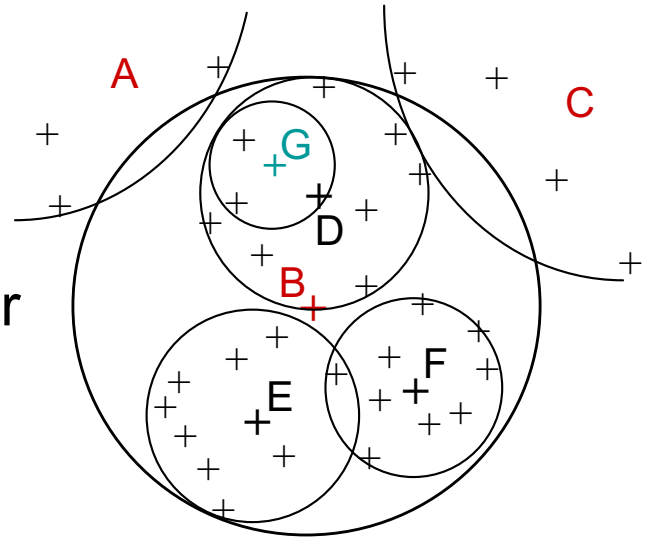
Requête q de rayon s

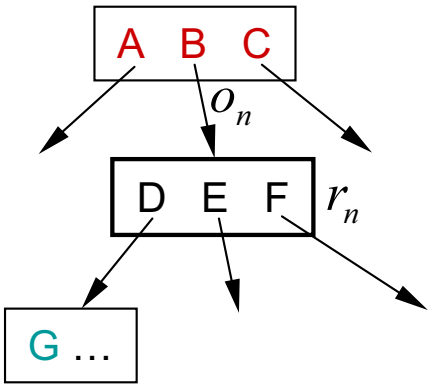
Montrons que si $d(q,o_n) > r_n + s$, alors tout le sous-arbre $N(o_n)$ peut être rejeté :

l'inégalité triangulaire implique $d(q,o) \geq d(q,o_n) - d(o,o_n)$, mais $d(o,o_n) \leq r_n$ et donc $d(q,o) > r_n + s - r_n = s$

Structure d'un M-*tree*

- M-*tree* : introduit dans [CPZ97]
- Chaque nœud $N(o_n)$ du M-*tree* :
  - Est associé à une région caractérisée par
    un « objet de routage » o_n
    et un « rayon de couverture » r_n
    tels que $d(o, o_n) \leq r_n$ pour tout objet o de la région
  - Contient entre C/2 et C entrées
  - C ≤ (taille 1 page sur disque) / (taille max. stockage 1 entrée)
  - Superposition partielle entre
    régions de même profondeur





Structure d'un M-*tree* (2)

- Une entrée d'un nœud terminal
  - stocke un objet quelconque o
  - permet de calculer $d(q,o)$
  - conserve $d(o,o_n)$, la distance à l'objet de routage parent o_n
- Une entrée d'un nœud intermédiaire
  - stocke un objet de routage o_e pour un nœud de niveau inférieur
  - conserve le rayon de couverture pour cet objet de routage o_e
  - permet de calculer $d(q,o_e)$
  - conserve $d(o_e,o_n)$, la distance à l'objet de routage parent o_n (à l'exception des entrées du nœud racine)

→ Code source C++ libre : http://www-db.deis.unibo.it/Mtree/

M-*tree* : *sphere query*

- Test d'acceptation d'un nœud :
  - Soit le nœud ayant comme objet de routage o_e, de rayon de couverture r_e
  - Si $d(q,o_e) \leq r_e + s$ alors le nœud (avec le sous-arbre dont il est la racine) est conservé pour exploration ultérieure
- Possibilité supplémentaire d'éviter des calculs de distances (*fast pruning* [CPZ97]) :
  - Parcours préfixe de l'arbre $\Rightarrow$ si o_n est l'objet de routage parent de o_e, alors $d(q,o_n)$ est calculée avant de s'intéresser à o_e
  - En conséquence, si $|d(q,o_n) - d(o_e,o_n)| > r_e + s$ alors on trouve directement que $d(q,o_e) > r_e + s$ (donc on rejette le nœud ayant o_e comme objet de routage), sans avoir à calculer $d(q,o_e)$

M-*tree* : *sphere query* (2)

```
procedure SphereQuery(N(o_n):node, q:query, s:range)
begin
  if N(o_n) not leaf then
    for each o_e entry of N(o_n) do    //o_n routing object of N(o_n)
      if |d(q,o_n) − d(o_e,o_n)| ≤ r_e + s then
        compute d(q,o_e);
        if d(q,o_e) ≤ r_e + s then SphereQuery(N(o_e), q, s); endif
      endif
    endfor
  else
    for each o entry of N(o_n) do
      if |d(q,o_n) − d(o,o_n)| ≤ s then
        compute d(q,o);
        if d(q,o) ≤ s then AddAccepted(o); endif
      endif
    endfor
  endif
end
```

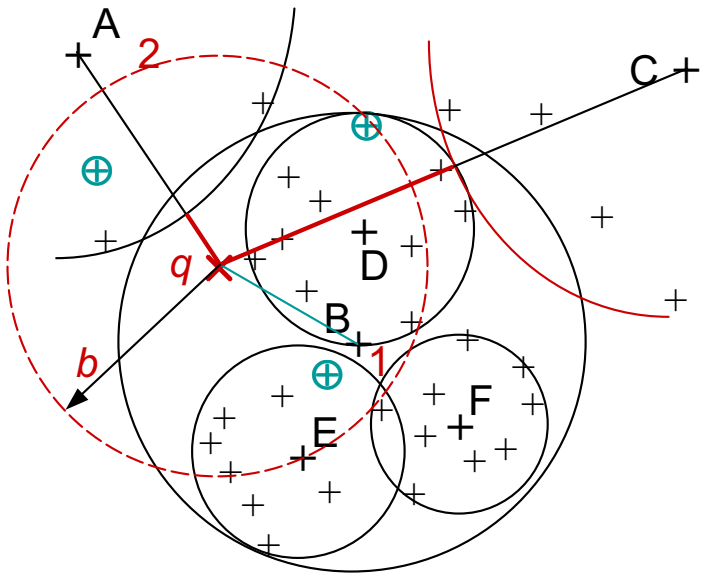
M-*tree* : requête *k* plus proches voisins

```
procedure KNNSearch(N0:root, q:query, k:nbNeighbours)
begin
    CandidateList := {N0};
    KNN := random selection of k objects;
    b := max_{oi ∈ KNN} {d(q,oi)};
    while notEmpty(CandidateList) and b > max{0, d(q,oh) − rh} do   (1)
        Remove(head(CandidateList));
        NodeSearch(head(CandidateList), q, b);
    endwhile
end
```

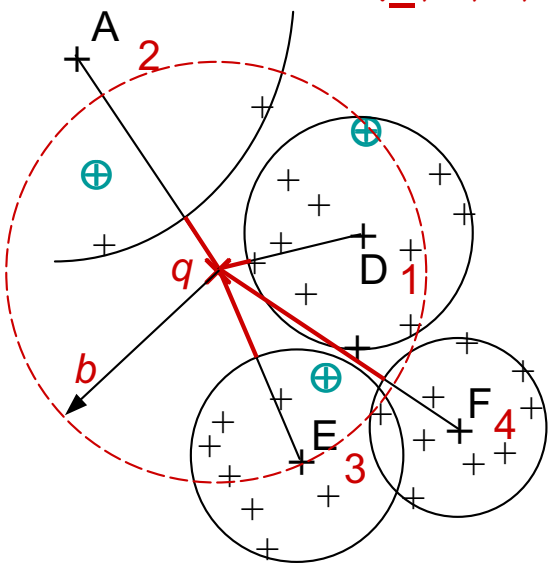
- **CandidateList** est triée en ordre croissant de $\max\{0, d(q,o_e)-r_e\}$
- Lors d'ajouts/remplacements, **KNN** est triée en ordre croissant de $d(q,o_i)$
- La valeur b sert de rayon de recherche dynamique dans **NodeSearch()**
- **NodeSearch()** : comme **SphereQuery()**, mais actions différentes !
- o_h est l'objet de routage de **head(CandidateList)**

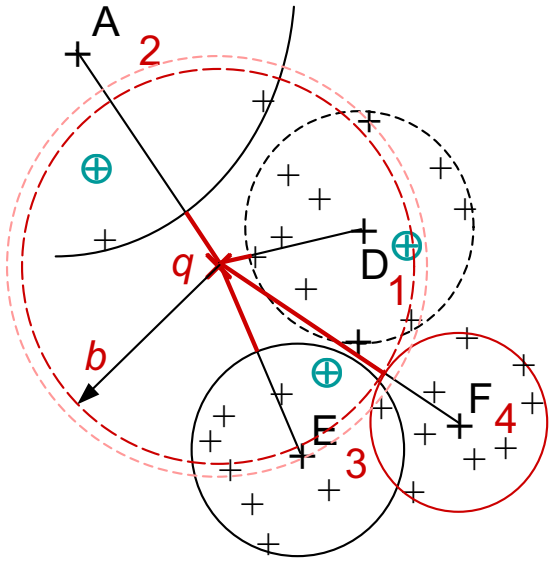
M-*tree* : requête *k* plus proches voisins

Initialisation : `KNN:=(⊕)` , `CandidateList:=(B,A)`

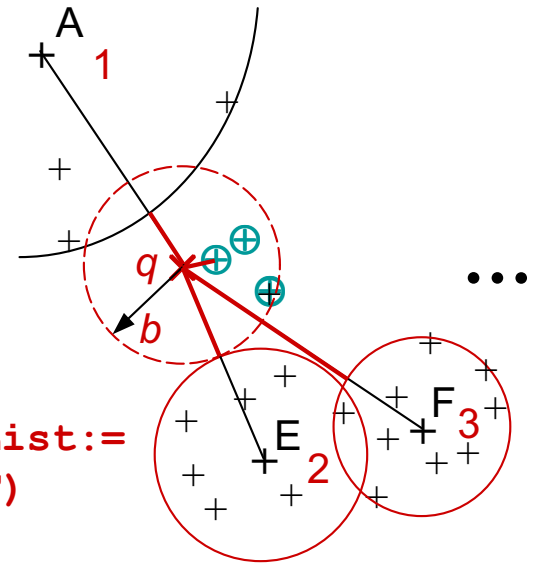


`CandidateList:=(D,A,E,F)`





`CandidateList:=(A,E,F)`



M-*tree* : requête *k* plus proches voisins

```
procedure NodeSearch(N(o_n):node, q:query, b:bound)
begin
  if N(o_n) not leaf then
    for each o_e entry of N(o_n) do
      if |d(q,o_n) − d(o_e,o_n)| ≤ r_e + b then
        compute d(q,o_e);                                   ②
        if d(q,o_e) ≤ r_e + b then AddToCandidateList(N(o_e)); endif
      endif                                                 ③
    endfor
  else
    for each o entry of N(o_n) do
      if |d(q,o_n) − d(o,o_n)| ≤ b then
        compute d(q,o);
        if d(q,o) ≤ b then AddToKNN(o); b := max_{o_i ∈ KNN} {d(q,o_i)}; ④ endif
      endif
    endfor
  endif
end
```

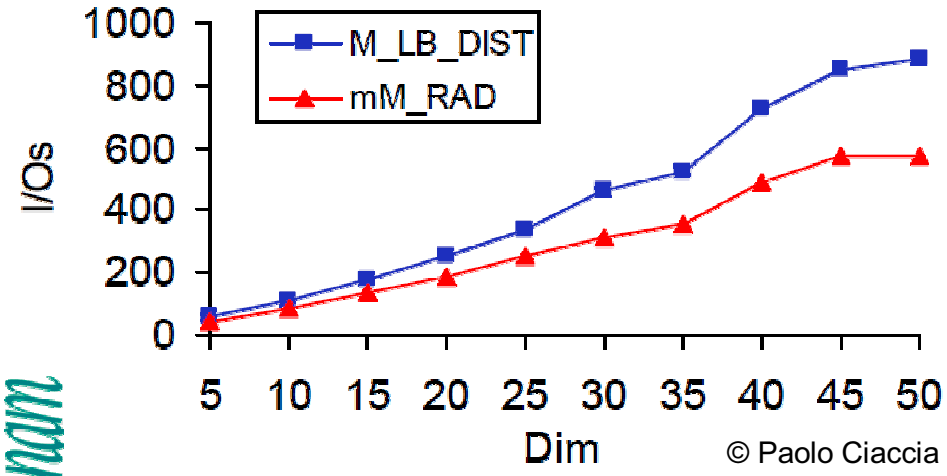
Construction M-*tree* : insertion

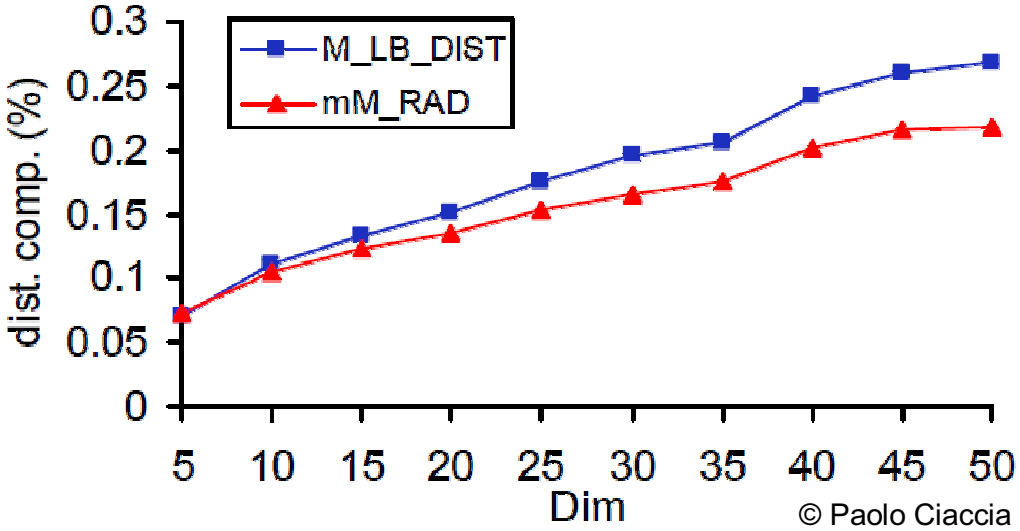
- Construction M-*tree* : insertions successives des objets
  - Chaque nouvel objet est inséré à partir de la racine, en choisissant à chaque niveau le nœud fils le plus approprié
  - L'insertion peut provoquer le dépassement de la capacité d'un nœud, qui devra donc être divisé de façon appropriée
  - Le dépassement de capacité peut se propager vers la racine

- Méthode d'insertion
  - Dans chaque nœud traversé, on affecte le nouvel objet o à l'objet de routage o_e dont il produit la plus faible augmentation du rayon de couverture r_e ; la valeur de r_e est actualisée

Construction M-*tree* : division nœud

- Parmi les critères de division (*split*) d'un nœud o_n :
  1. Choisir comme objets de routage des 2 nœuds résultants (qui remplacent l'ancien) l'objet le plus proche de o_n et l'objet le plus éloigné (M_LB_DIST)
  2. Minimiser le plus grand des rayons de couverture des 2 nœuds résultants (mM-RAD) : plus complexe (construction plus lente de l'arbre) mais plus efficace (recherche plus rapide)





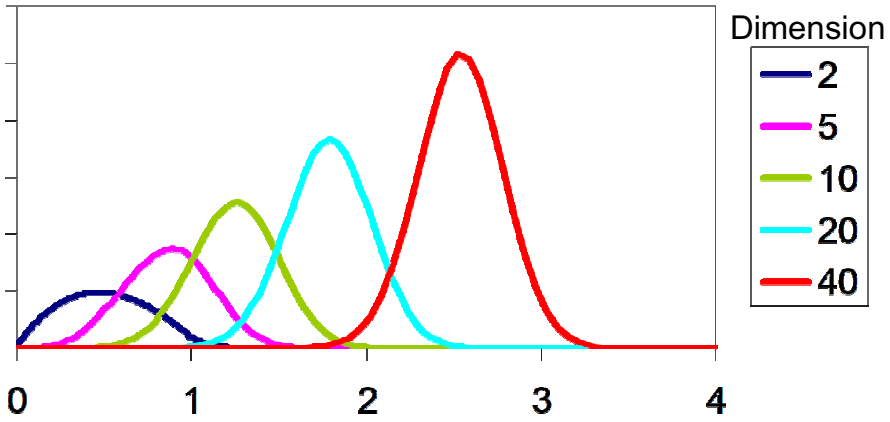
M-*tree* et malédiction de la dimension

- Données vectorielles à distribution uniforme : la variance de la distribution des distances diminue quand la dimension augmente

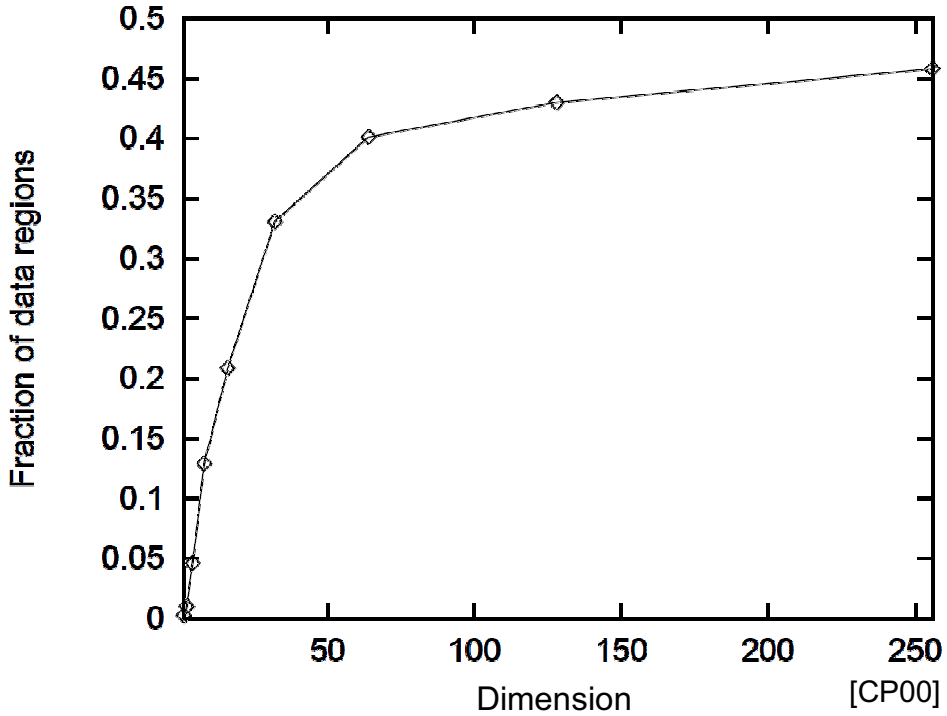
⇒ Augmentation rapide de la superposition entre régions associées aux nœuds d'un même niveau

⇒ Diminution rapide de l'efficacité de l'indexation

Données uniformes : variation (avec la dimension) de la distribution des distances



Données uniformes : variation (avec la dimension) de la fraction des régions contenant la requête



[CP00]

M-*tree* : recherche approximative

- On ne considère ici que 2 approches : AC-NN, PAC-NN
- *Approximately correct* (AC) *kNN search* ([AMNSW98])
  - Nature de l'approximation : pour chaque objet o retourné, $d(q,o) \leq (1+\varepsilon)\, r_k$ où r_k est la distance entre la requête et le vrai *k*-ième NN, et $\varepsilon > 0$ est l'erreur relative maximale admise

    $\Rightarrow$les *k* objets retournés ne sont pas les *k* plus proches voisins de la requête, mais *k* voisins proches (dans le sens défini ci-dessus)
  - Comment la mettre en œuvre : dans ① ② ③ on remplace b par $b/(1+\varepsilon)$ $\Rightarrow$ arrêt précoce et élimination de candidats car toute amélioration limitée à ε est inutile
  - Résultats : en général $\varepsilon_{\text{effectif}} << \varepsilon$ (qui est souvent difficile à fixer), mais le gain en rapidité reste modéré
  - Même si *k*NN déjà trouvés, la recherche continue (inutilement) si **`CandidateList`** non vide et $b/(1+\varepsilon) > \max\{0, d(q,o_h) - r_h\}$

M-*tree* : recherche approximative (2)

- *Probably AC NN search* (PAC-NN, [CP00], k = 1)
  - Intuition : bases difficiles = voisins éloignés, le plus de temps sera souvent perdu en cherchant à diminuer b en-dessous de la distance à laquelle se trouve le plus proche voisin

  → Rendre l'arrêt introduit par AC-NN encore plus précoce, de façon contrôlable, à partir de l'estimation d'une borne inférieure pour $d(q,o)$

  → Introduire l'arrêt en ④ si $b \leq (1+\varepsilon)\, r_{q,\delta}$ (en plus des adaptations de la méthode précédente, AC-NN)
    - $r_{q,\delta}$ défini par $P\{\exists o : d(q,o) \leq r_{q,\delta}\} \leq \delta$ → on doit estimer la distribution des distances
    - Lorsqu'on s'arrête car $b \leq (1+\varepsilon)\, r_{q,\delta}$, la probabilité de ne pas avoir trouvé le NN (car il existe un point tel que $d(q,o) \leq r_{q,\delta}$) est $\leq \delta$
  - Principe applicable à d'autres méthodes d'indexation !

M-*tree* : conclusion

- Transposition des indexes multidimensionnels à base d'arbres à de simples espaces métriques
- Comparaison avec le SS-*tree*
  - Un M-*tree* connaît seulement la structure métrique de l'espace
  - Une « sphère » correspondant à un nœud du M-*tree* est nécessairement centrée sur un objet de la base (*routing object*)
- La distribution des distances entre les objets indexés a un fort impact sur le M-*tree* ; pour des données vectorielles, on retrouve la malédiction de la dimension
- Quand l'application l'accepte, une recherche approximative (PAC-NN) améliore l'efficacité
- Quand la variance de la distribution des distances s'effondre, rechercher les *k*NN a encore un intérêt ?

Chapitre VI

Marches dans des graphes

Algorithme d'Orchard

Preprocessing: Orchard'91

Pout tous les objets $p_i \in S$ construire une liste $L(p_i)$ de tous les autres objets triés par leur similarité à p_i

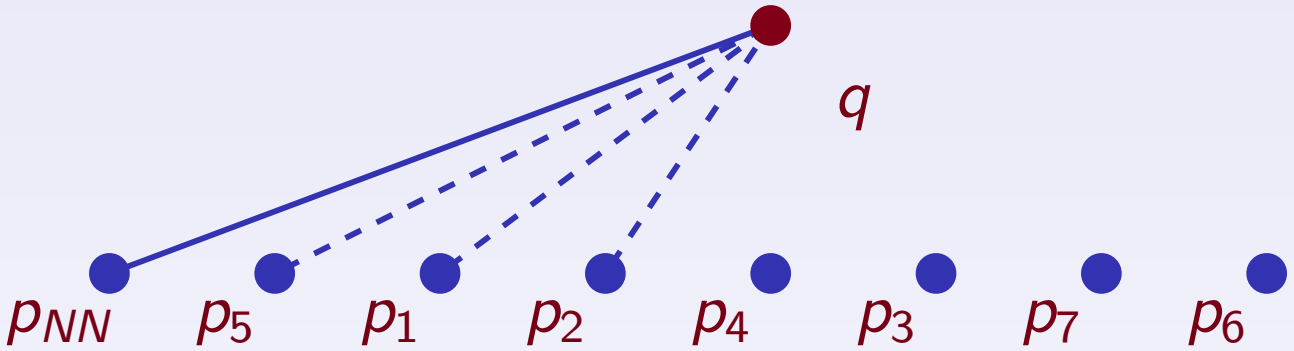
Algorithme d'Orchard

Preprocessing: Orchard'91

Pout tous les objets $p_i \in S$ construire une liste $L(p_i)$ de tous les autres objets triés par leur similarité à p_i

Observation: Construction statique très coûteuse, stockage quadratique

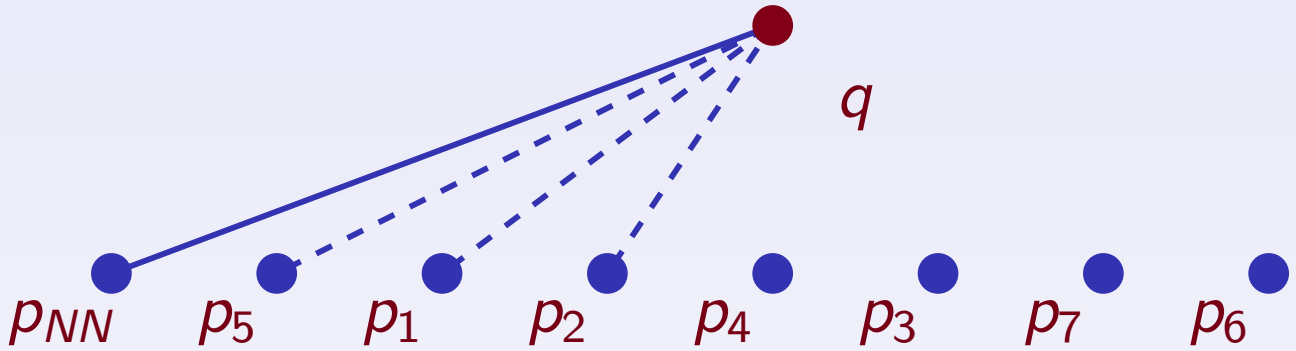
Algorithme d'Orchard



Traitement requête:

- Commencer avec un objet aléatoire p_{NN}

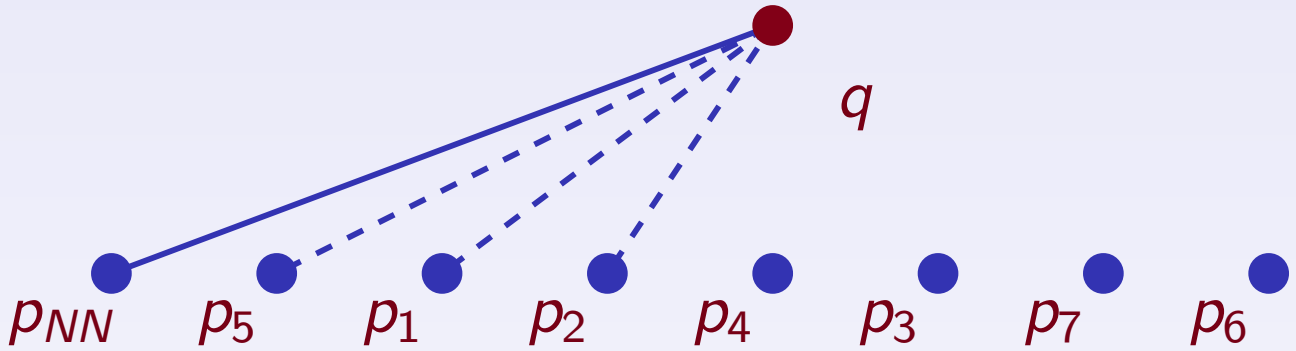
Algorithme d'Orchard



Traitement requête:

- Commencer avec un objet aléatoire p_{NN}
- Inspecter les membres $p' \in L(P_{NN})$ de gauche à droite

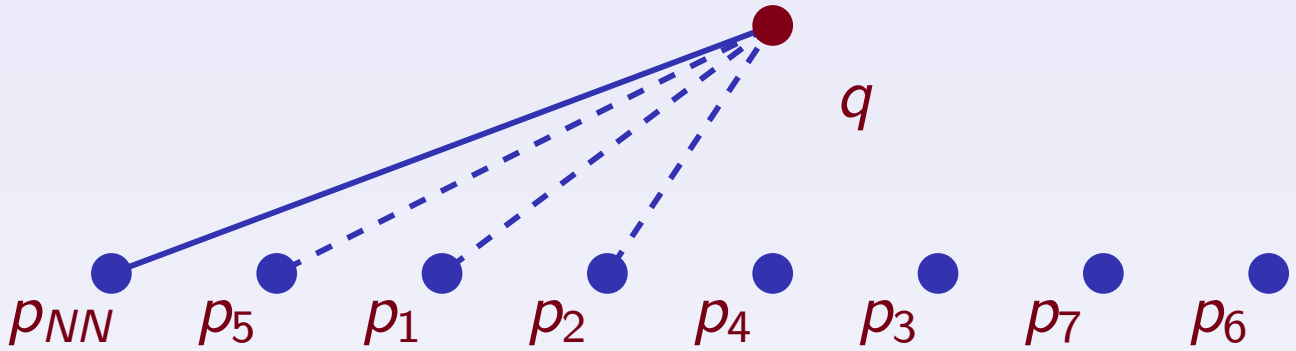
Algorithme d'Orchard



Traitement requête:

- Commencer avec un objet aléatoire p_{NN}
- Inspecter les membres $p' \in L(P_{NN})$ de gauche à droite
- Si $d(p', q) < d(p_{NN}, q)$, update $p_{NN} := p'$

Algorithme d'Orchard



Traitement requête:

- Commencer avec un objet aléatoire p_{NN}
- Inspecter les membres $p' \in L(P_{NN})$ de gauche à droite
- Si $d(p', q) < d(p_{NN}, q)$, update $p_{NN} := p'$
- Condition d'arrêt: On trouve un p' tel que $d(p', q) \geq 2d(p_{NN}, q)$

Algorithme d'Orchard hiérarchique

- Choisir aléatoirement $S_1 \subset S_2 \subset \ldots S_k = S$ avec $|S_i|/|S_{i-1}| \approx \alpha > 1$
- Lancer l'algorithme d'Orchard sur S_1
- Pour $i = 2$ à k appliquer l'algorithme d'Orchard sur S_i en utilisant le résultat précédent comme objet de départ

Algorithme d'Orchard hiérarchique

- Choisir aléatoirement $S_1 \subset S_2 \subset \ldots S_k = S$ avec $|S_i|/|S_{i-1}| \approx \alpha > 1$
- Lancer l'algorithme d'Orchard sur S_1
- Pour $i = 2$ à k appliquer l'algorithme d'Orchard sur S_i en utilisant le résultat précédent comme objet de départ

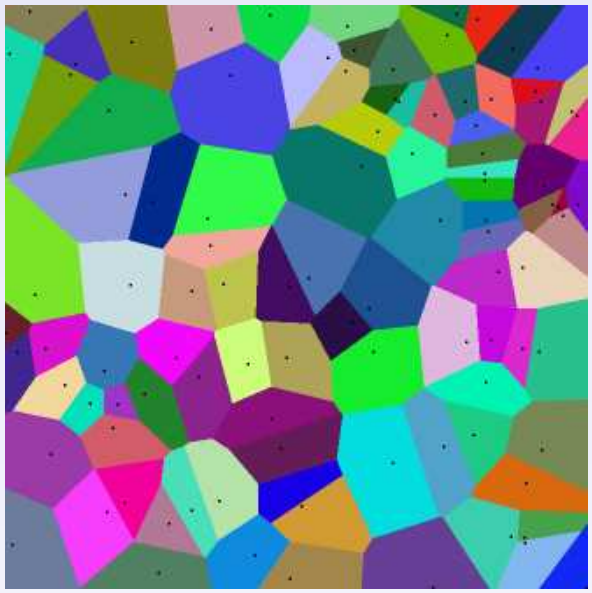
Inspiré par l'algo classique 1D skip list Pugh'90

Graphe de Delaunay, espace Euclidien

Graphe de Delaunay:

Construire le diagramme de Voronoi pour l'ensemble S (couteux, statique).

Créer des liens entre toutes les paires d'objets dont les cellules sont adjacentes

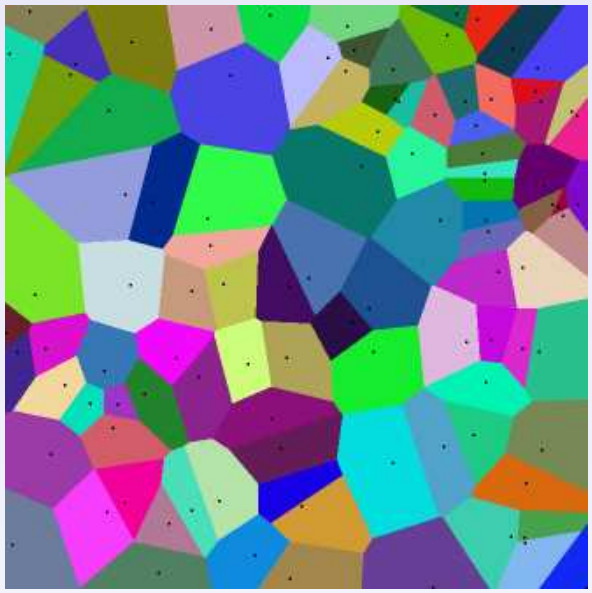


Graphe de Delaunay, espace Euclidien

Graphe de Delaunay:

Construire le diagramme de Voronoi pour l'ensemble S (couteux, statique).

Créer des liens entre toutes les paires d'objets dont les cellules sont adjacentes



Algorithme de recherche:

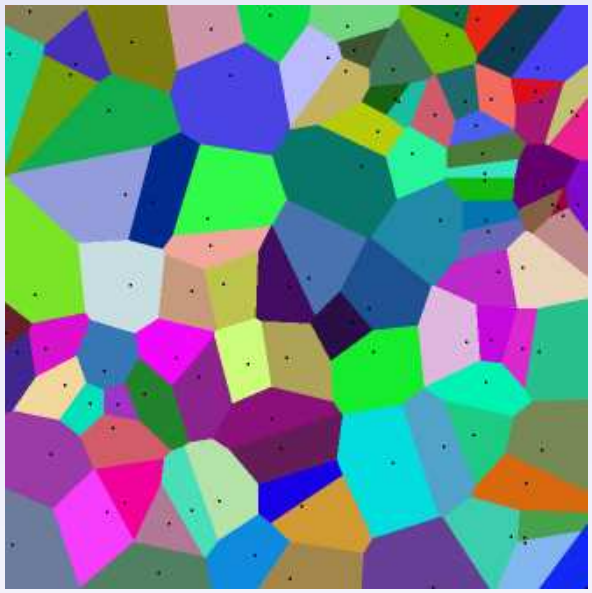
- Commencer avec un point aléatoire p_{NN}

Graphe de Delaunay, espace Euclidien

Graphe de Delaunay:

Construire le diagramme de Voronoi pour l'ensemble S (couteux, statique).

Créer des liens entre toutes les paires d'objets dont les cellules sont adjacentes



Algorithme de recherche:

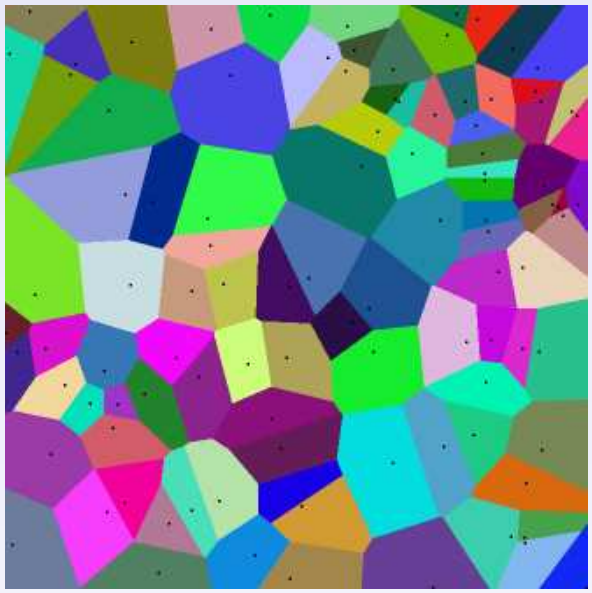
- Commencer avec un point aléatoire p_{NN}
- Inspecter les voisins dans le graphe de Delaunay

Graphe de Delaunay, espace Euclidien

Graphe de Delaunay:

Construire le diagramme de Voronoi pour l'ensemble S (couteux, statique).

Créer des liens entre toutes les paires d'objets dont les cellules sont adjacentes



Algorithme de recherche:

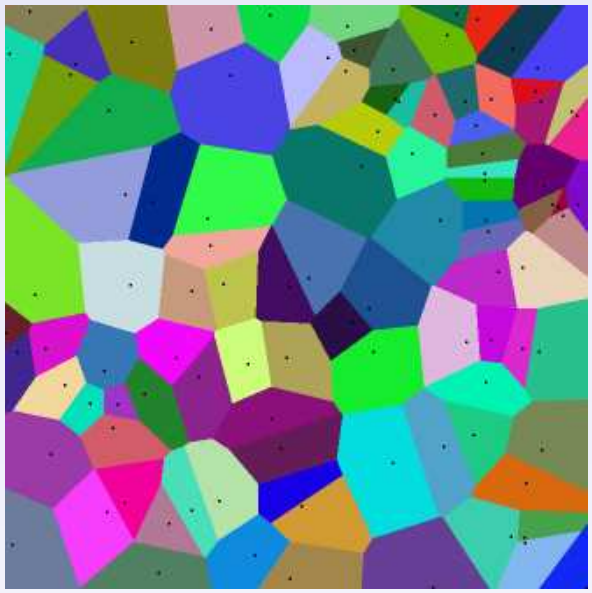
- Commencer avec un point aléatoire p_{NN}
- Inspecter les voisins dans le graphe de Delaunay
- Si on rencontre un p' tel que $d(p', q) < d(p_{NN}, q)$, update $p_{NN} := p'$

Graphe de Delaunay, espace Euclidien

Graphe de Delaunay:

Construire le diagramme de Voronoi pour l'ensemble S (couteux, statique).

Créer des liens entre toutes les paires d'objets dont les cellules sont adjacentes



Algorithme de recherche:

- Commencer avec un point aléatoire p_{NN}
- Inspecter les voisins dans le graphe de Delaunay
- Si on rencontre un p' tel que $d(p', q) < d(p_{NN}, q)$, update $p_{NN} := p'$
- Sinon retourne p_{NN}

Graphe de Delaunay, cas général ?

Supposons que l'on ait un espace métrique avec la matrice des distances pour toute paire de points.
Comment le graphe de Delaunay pourrait-il être défini ?

Graphe de Delaunay, cas général ?

Supposons que l'on ait un espace métrique avec la matrice des distances pour toute paire de points. Comment le graphe de Delaunay pourrait-il être défini ?

Navarro, 2002: Pour une matrice de distance quelconque, toutes les paires d'objets peuvent être adjacentes :-(

Spatial Approximation Tree: Construction

Navarro'99:

- Choisir un objet aléatoire p pour la racine
- Technique de partition:
  - Inspecter tous les objets par ordre de similarité avec p pour sélectionner un ensemble d'enfants $Ch(p)$
  - Lorsqu'un p' est plus proche de p que des enfants déjà choisis dans $Ch(p)$ ajouter p' à $Ch(p)$
  - Tous les autres objets p'' sont mis dans le sous-arbre de l'enfant de $Ch(p)$ le plus proche de p''
- Répéter récursivement

Spatial Approximation Tree: Construction

Navarro'99:

- Choisir un objet aléatoire p pour la racine
- Technique de partition:
  - Inspecter tous les objets par ordre de similarité avec p pour sélectionner un ensemble d'enfants $Ch(p)$
  - Lorsqu'un p' est plus proche de p que des enfants déjà choisis dans $Ch(p)$ ajouter p' à $Ch(p)$
  - Tous les autres objets p'' sont mis dans le sous-arbre de l'enfant de $Ch(p)$ le plus proche de p''
- Répéter récursivement

Observation: Le rayon de couverture d'un arbre fils est inférieure au rayon de couverture de l'arbre père

SA-Tree: Recherche

- Commencer à la racine p et $p_{NN} = p$
- Pour chaque noeud inspecté:

  conserve le candidat global p_{NN}

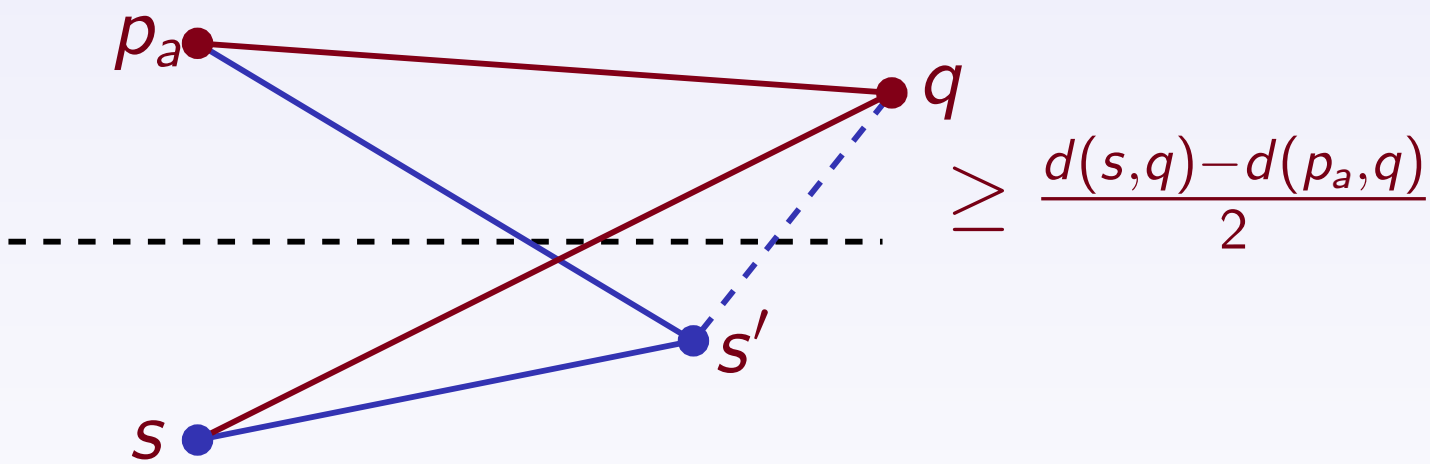
  ET p_a, objet le plus proche de q parmi les ancêtres et les frères
- Parcours en profondeur
- Traitement du noeud courant p':
  - Calculer la distance à q de tous les enfants de p' (mise à jour p_a et p_{NN})
  - Aller à l'enfant p'' si $d(q, p'') < d(q, p_a) + 2d(q, p_{NN})$

SA-Tree: Validité

Observation: Pour un noeud fixe s, p_a un de ses ancêtres ou frères et s' un objet dans le sous-arbre. Alors s' est plus proche de s que de p_a.

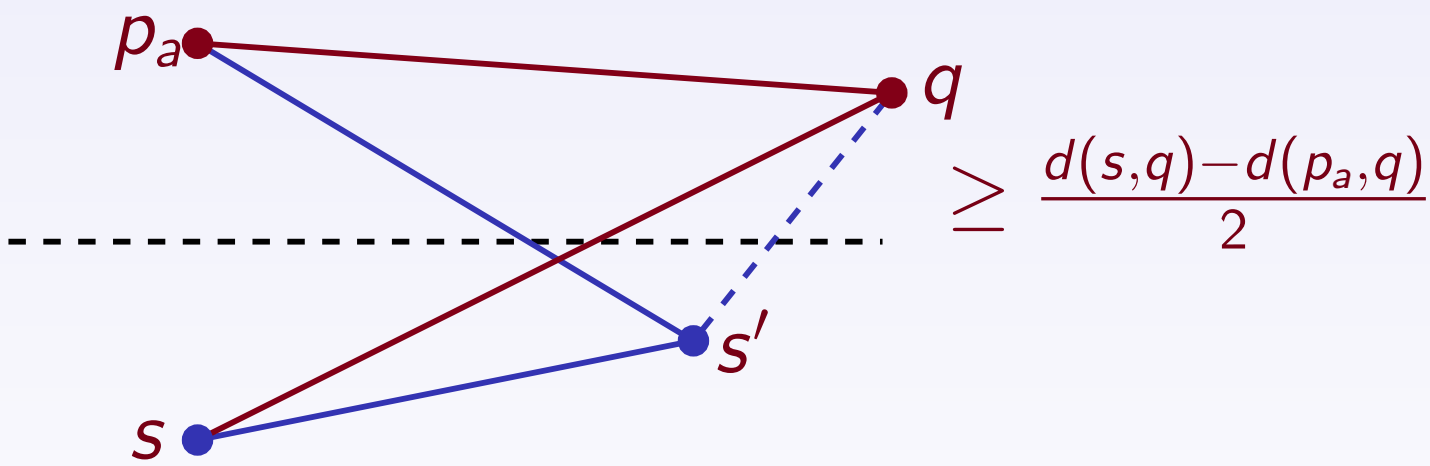
SA-Tree: Validité

Observation: Pour un noeud fixe s, p_a un de ses ancêtres ou frères et s' un objet dans le sous-arbre. Alors s' est plus proche de s que de p_a.



SA-Tree: Validité

Observation: Pour un noeud fixe s, p_a un de ses ancêtres ou frères et s' un objet dans le sous-arbre. Alors s' est plus proche de s que de p_a.



S'il existe un s' tel que $d(s', q) < r_{NN}$ alors $d(s, q) < d(p_a, q) + 2r_{NN}$

Chapitre VII

Techniques matricielles

Approximating and Eliminating Search Algorithm

Preprocessing: Vidal'86

Calculer la matrice $n \times n$ des distances entre les objets de S (statique, quadratique)

Approximating and Eliminating Search Algorithm

Preprocessing: Vidal'86

Calculer la matrice $n \times n$ des distances entre les objets de S (statique, quadratique)

Traitement d'une requête:

- Maintenir un ensemble C de candidats , initiallement $C := S$
- Pour tout $p \in C$ conserver la borne inférieure $d_{min}(q, p)$
- Boucle principale:
  - Choisir $p \in C$ avec la borne inf. la plus petite, calculer $d(q, p)$, update $p_{NN}, r_{NN} = d(q, p_{NN})$ si nécessaire
  - Approximer: update de la borne inf. de C utilisant l'inégalité $d(q, p') \geq d(q, p) + d(p, p')$
  - Eliminer: Supprimer tous les éléments de C dont la borne inf. est plus grande que r_{NN}

AESA linéaire

Avantages de AESA: petit nombre de calculs de distances
Désavantages: Stockage et calcul des autres opérations

AESA linéaire

Avantages de AESA: petit nombre de calculs de distances
Désavantages: Stockage et calcul des autres opérations

AESA linéaire: Micó, Oncina, Vidal'94

Calculer une matrice $n \times m$ de m objets choisis comme pivots

AESA linéaire

Avantages de AESA: petit nombre de calculs de distances
Désavantages: Stockage et calcul des autres opérations

AESA linéaire: Micó, Oncina, Vidal'94

Calculer une matrice $n \times m$ de m objets choisis comme pivots

Recherche à un rayon R près:

- Calcul des distances requête-pivot
- Calcul de la borne inf. pour tous les objets non-pivot
- Eliminer les objets avec une borne inf supérieure à R
- Vérification des objets non-pivot restant

TLAESA

Une combinaison du bisector tree et de AESA linéaire

Structure: Micó, Oncina, Carrasco'96

Bisector tree standrd

Et en plus, m pivots

Les distances entre les pivots et tous les autres objets sont précalculés

TLAESA

Une combinaison du bisector tree et de AESA linéaire

Structure: Micó, Oncina, Carrasco'96

Bisector tree standrd

Et en plus, m pivots

Les distances entre les pivots et tous les autres objets sont précalculés

Traitement requête:

Calcul des distances requête-pivot

Parcours Depth-first ou Best-first du bisector tree

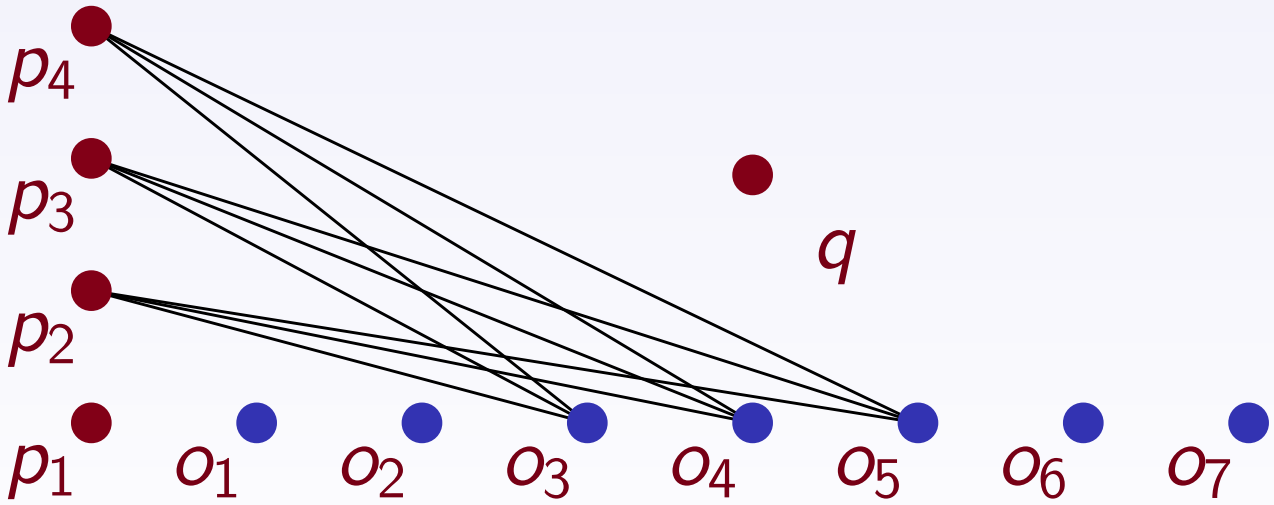
Condition additionnelle pour éliminer le sous-arbre de certains objets s:

$$\exists i : \quad |d(p_i, s) - d(p_i, q)| \geq r_c(s) + r_{NN}$$

Algorithme de Shapiro (1/2)

Structure: Shapiro'77

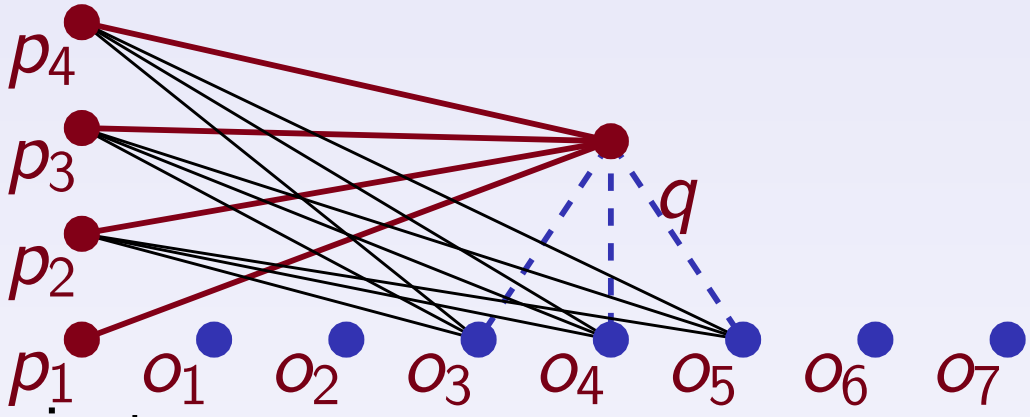
Matrice $n \times m$ des distances (pivots $p_1, \ldots, p_m$)
Objets non-pivot triés par leur distance
au premier pivot p_1 : $o_1, \ldots, o_n$



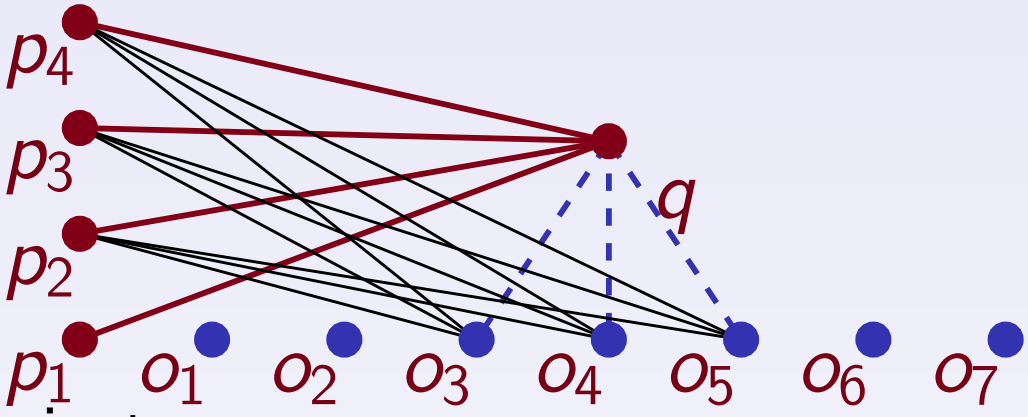
Algorithm de Shapiro (2/2)

Traitement requête:

Calcul des distances requête-pivot



Algorithm de Shapiro (2/2)

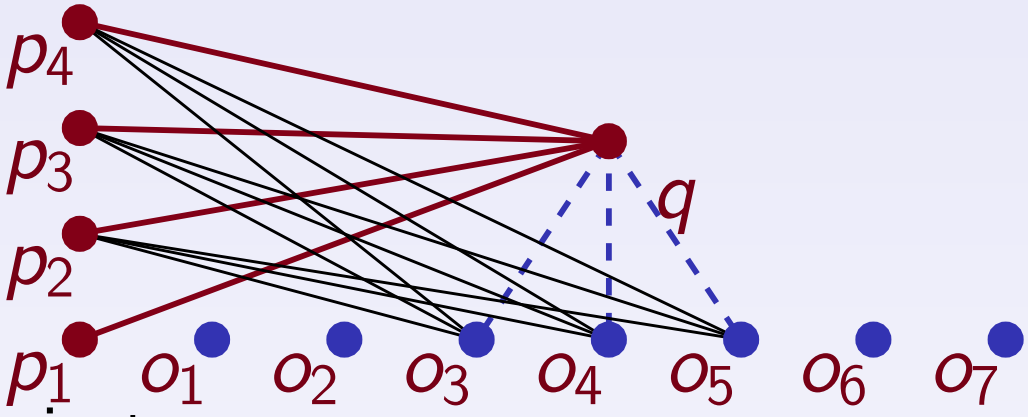


Traitement requête:

Calcul des distances requête-pivot

Commencer avec un o_i tel que $d(p_1, o_i) \approx d(p_1, q)$

Algorithm de Shapiro (2/2)



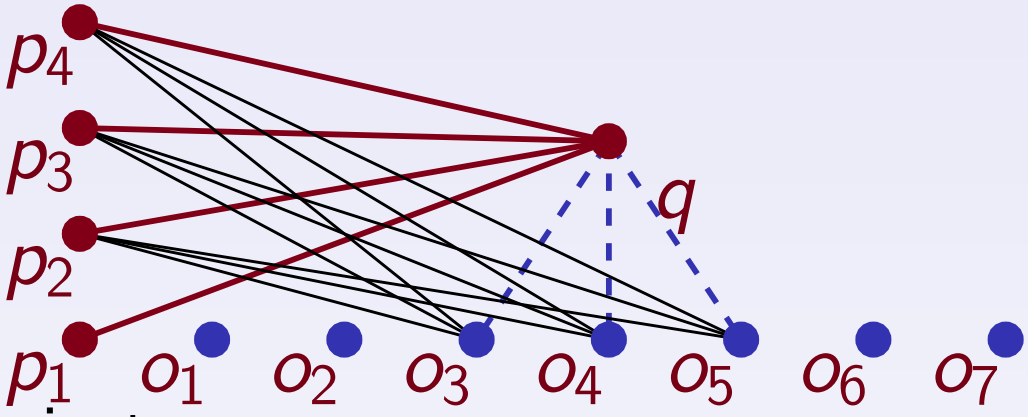
Traitement requête:

Calcul des distances requête-pivot

Commencer avec un o_i tel que $d(p_1, o_i) \approx d(p_1, q)$

Inspecter les autres objets dans l'ordre $i-1, i+1, i-2, i+2, ...$

Algorithm de Shapiro (2/2)



Traitement requête:

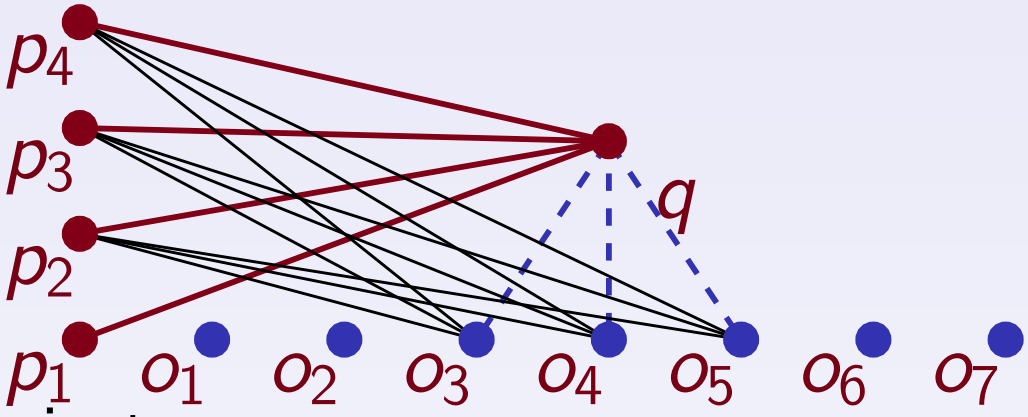
Calcul des distances requête-pivot

Commencer avec un o_i tel que $d(p_1, o_i) \approx d(p_1, q)$

Inspecter les autres objets dans l'ordre $i-1, i+1, i-2, i+2, ...$

Lorsqu'on rencontre un meilleur candidat, on change le centre d'inspection

Algorithm de Shapiro (2/2)



Traitement requête:

Calcul des distances requête-pivot

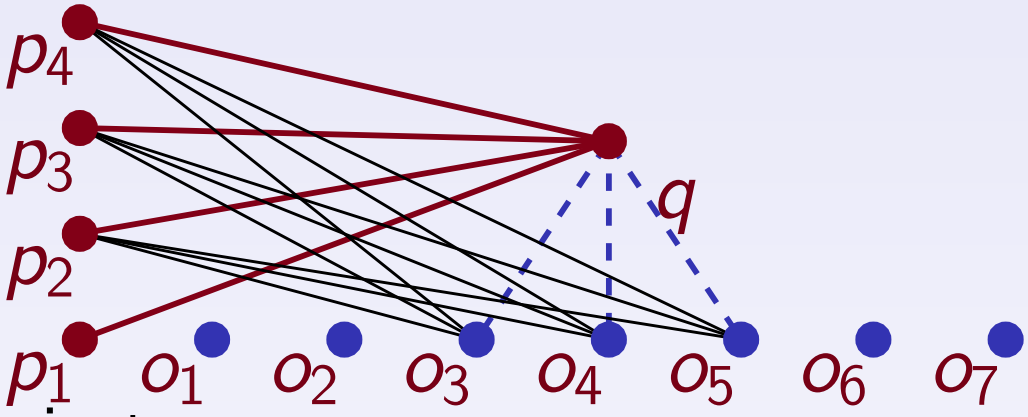
Commencer avec un o_i tel que $d(p_1, o_i) \approx d(p_1, q)$

Inspecter les autres objets dans l'ordre $i-1, i+1, i-2, i+2, ...$

Lorsqu'on rencontre un meilleur candidat, on change le centre d'inspection

Utilise des flags pour éviter de les doubles inspections

Algorithm de Shapiro (2/2)



Traitement requête:

Calcul des distances requête-pivot

Commencer avec un o_i tel que $d(p_1, o_i) \approx d(p_1, q)$

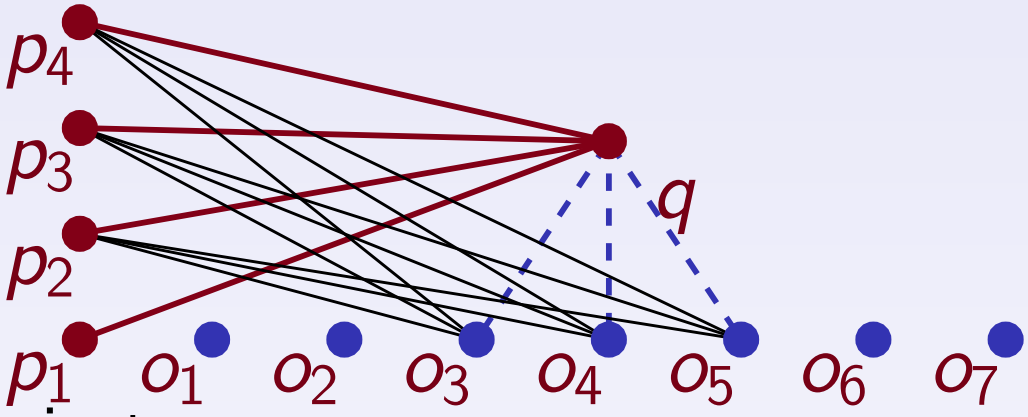
Inspecter les autres objets dans l'ordre $i-1, i+1, i-2, i+2, ...$

Lorsqu'on rencontre un meilleur candidat, on change le centre d'inspection

Utilise des flags pour éviter de les doubles inspections

Utiliser tous les pivots pour éliminer certains objets (comme AESA)

Algorithm de Shapiro (2/2)



Traitement requête:

Calcul des distances requête-pivot

Commencer avec un o_i tel que $d(p_1, o_i) \approx d(p_1, q)$

Inspecter les autres objets dans l'ordre $i-1, i+1, i-2, i+2, ...$

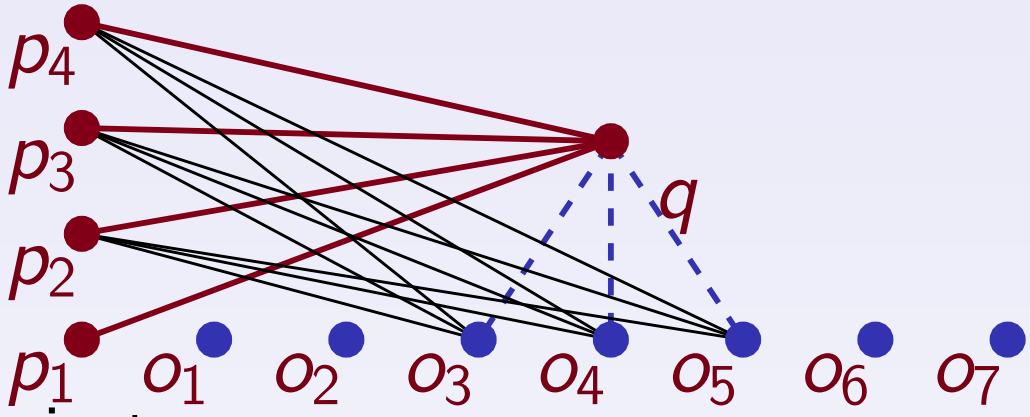
Lorsqu'on rencontre un meilleur candidat, on change le centre d'inspection

Utilise des flags pour éviter de les doubles inspections

Utiliser tous les pivots pour éliminer certains objets (comme AESA)

Condition d'arrêt: $|d(p_1, o_i) - d(p_1, q)| \geq r_{NN}$

Algorithm de Shapiro (2/2)



Traitement requête:

Calcul des distances requête-pivot

Commencer avec un o_i tel que $d(p_1, o_i) \approx d(p_1, q)$

Inspecter les autres objets dans l'ordre $i-1, i+1, i-2, i+2, ...$

Lorsqu'on rencontre un meilleur candidat, on change le centre d'inspection

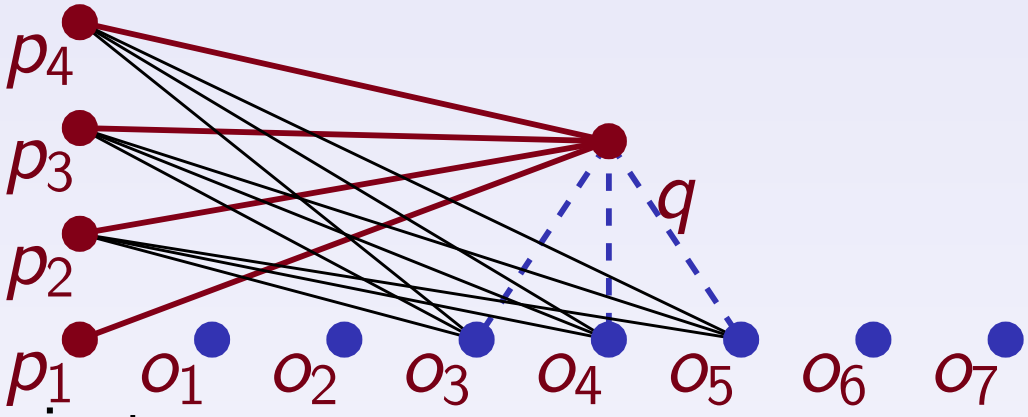
Utilise des flags pour éviter de les doubles inspections

Utiliser tous les pivots pour éliminer certains objets (comme AESA)

Condition d'arrêt: $|d(p_1, o_i) - d(p_1, q)| \geq r_{NN}$

En fait, il s'agit d'un mélange entre LAESA et Orchard

Algorithm de Shapiro (2/2)



Traitement requête:

Calcul des distances requête-pivot

Commencer avec un o_i tel que $d(p_1, o_i) \approx d(p_1, q)$

Inspecter les autres objets dans l'ordre $i-1, i+1, i-2, i+2, ...$

Lorsqu'on rencontre un meilleur candidat, on change le centre d'inspection

Utilise des flags pour éviter de les doubles inspections

Utiliser tous les pivots pour éliminer certains objets (comme AESA)

Condition d'arrêt: $|d(p_1, o_i) - d(p_1, q)| \geq r_{NN}$

En fait, il s'agit d'un mélange entre LAESA et Orchard
Mais publié avant les deux autres: 1977 vs 1991 et 1992!

Exercices

Prouver la validité de la condition d'arrêt de l'algorithme d'Orchard

Exercices

Prouver la validité de la condition d'arrêt de l'algorithme d'Orchard

Prouver la validité de la condition d'arrêt du SA-tree

En bref

- M-tree équivalent du B-tree pour la recherche par similarité dans des espaces métriques en général (en moins efficace :-( )

En bref

- M-tree équivalent du B-tree pour la recherche par similarité dans des espaces métriques en général (en moins efficace :-( )
- Orchard: Précalul de la base ordonnée pour tous les points, Change de points lorsqu'on trouve une meilleur solution

En bref

- M-tree équivalent du B-tree pour la recherche par similarité dans des espaces métriques en général (en moins efficace :-( )
- Orchard: Précalul de la base ordonnée pour tous les points, Change de points lorsqu'on trouve une meilleur solution
- Spatial Approximation tree: Emulation du graphe de Delaunay pour les espaces métriques en général

En bref

- M-tree équivalent du B-tree pour la recherche par similarité dans des espaces métriques en général (en moins efficace :-( )
- Orchard: Précalul de la base ordonnée pour tous les points, Change de points lorsqu'on trouve une meilleur solution
- Spatial Approximation tree: Emulation du graphe de Delaunay pour les espaces métriques en général
- AESA: use every inter-object distance to get a lower bound on unchecked distances

En bref

- M-tree équivalent du B-tree pour la recherche par similarité dans des espaces métriques en général (en moins efficace :-( )
- Orchard: Précalul de la base ordonnée pour tous les points, Change de points lorsqu'on trouve une meilleur solution
- Spatial Approximation tree: Emulation du graphe de Delaunay pour les espaces métriques en général
- AESA: use every inter-object distance to get a lower bound on unchecked distances

Merci de votre attention! Questions?

Références

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

P. Zezula, G. Amato, V. Dohnal, M. Batko
Similarity Search: The Metric Space Approach. Springer, 2006.
http://www.nmis.isti.cnr.it/amato/similarity-search-book/

E. Chávez, G. Navarro, R. Baeza-Yates, J. L. Marroquín
Searching in Metric Spaces. ACM Computing Surveys, 2001.
http://www.cs.ust.hk/~leichen/courses/comp630j/readings/acm-survey/searchinmetric.pdf

G.R. Hjaltason, H. Samet
Index-driven similarity search in metric spaces. ACM Transactions on Database Systems, 2003
http://www.cs.utexas.edu/~abhinay/ee382v/Project/Papers/ft_gateway.cfm.pdf