Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Licence 3 Informatique

Algorithmique IV

Géométrie Algorithmique

Lesly-Ann Daniel
Fabien Nicot

12 avril 2016

Table des matières

1 Introduction

Le but de ce projet est d'implémenter les problèmes suivants : **calcul de la paire de points la plus proche dans un ensemble de points**, et **calcul du nombre d'intersections entre deux ensembles de segments**. Chaque problème est implémenté selon trois approches différentes : **force brute**, **diviser pour régner** et **sweep-line**. L'objectif est de comparer les performances de chacune de ces approches sur des ensembles de plus en plus grand. Pour chaque partie les différents algorithmes sont donnés et les résultats de chaque approche sont comparés.

2 Paire de points la plus proche

2.1 Force Brute

Algorithm 1 Paire la plus proche BF

Input : P : ensemble de points
Output : (p, q) : paire de points la plus proche

```
min ← +∞
for p ∈ P do
  for q ∈ P, p ≠ q do
    if distance(p, q) < min then
      min ← distance(p, q)
      (pmin, qmin) ← (p, q)
    end if
  end for
end for
return (pmin, qmin)
```

2.2 Diviser pour régner

Algorithm 2 Paire la plus proche DPR

Input : P : ensemble de points triés
Output : (min, (p, q)) : paire de points la plus proche

```
# Condition d'arrêt
if P comporte moins de 100 éléments then
  return Paire_la_plus_proche_BF(P)
end if
# Division du problème
(mg, (pg, qg)) ← Paire_la_plus_proche_DPR(PartieGauche(P))
(md, (pd, qd)) ← Paire_la_plus_proche_DPR(PartieDroite(P))
(min, (pmin, qmin)) ← min((mg, (pg, qg)), (md, (pd, qd)))
# Réunion
k ← milieu(P)
g ← min(g) : distance(g.x, (k-1).x) < min
d ← max(d) : distance(k.x, g.x) < min
for p ∈ P : p.x ≥ g.x ET p.x ≤ (k-1).x do
  for q ∈ P : q.x ≥ k.x ET q.x ≤ d.x do
    if distance(p, q) < min then
      (min, (pmin, qmin)) ← (distance(p, q), (p, q))
    end if
  end for
end for
return (min, (pmin, qmin))
```

2.3 Sweep-Line

Algorithm 3 Paire la plus proche SL

Input : P : ensemble de points triés
Output : (p, q) : paire de points la plus proche

```
min ← +∞
for p ∈ P do
  q ← p + 1
  for q ∈ P : distance(p.x, q.x) < min do
    if distance(p, q) < min then
      min ← distance(p, q)
      (pmin, qmin) ← (p, q)
    end if
  end for
end for
return (pmin, qmin)
```

2.4 Comparaison des performances

Chaque résultat est une moyenne sur 5 exécutions de l'algorithme. On voit que l'approche Brute-Force est la moins efficace avec une complexité $T(n) = \theta(n^2)$.

Pour les méthodes Sweep-Line et DPR, la complexité est de l'ordre de $T(n) = \theta(n * log(n))$. En effet, le tri en $T(n) = \theta(n * log(n))$ détermine dans les deux cas la classe de complexité. Néanmoins, en pratique, c'est l'approche Sweep-Line qui donne les meilleurs résultats. Voir graphique partie A.1.

3 Nombre d'intersections entre deux ensembles de segments

3.1 Force Brute

Algorithm 4 Nombre d'intersections BF

Input : V : ensemble de segments verticaux, H : ensemble de segments horizontaux
Output : n : nombre d'intersections

```
n ← 0
for v ∈ V do
  for h ∈ H do
    if intersect(v, h) then
      n ← n + 1
    end if
  end for
end for
return n
```

3.2 Diviser pour régner

Dans l'algorithme suivant on travaille avec un ensemble de points P trié selon la coordonnée x. On peut diviser P en trois sous-ensembles :

- G(P) : extrémités gauches de P dont l'extrémité droite correspondante n'est pas dans P
- D(P) : extrémités droites de P dont l'extrémité gauche correspondante n'est pas dans P
- V(P) : points correspondant à un segment vertical de P

Algorithm 5 Nombre d'intersections DPR

Input : P : ensemble trié des points correspondant aux extrémités des segments
Output : (P, n) : nombre d'intersections

```
# Condition d'arrêt
if P ne comporte qu'un seul élément p then
  if p est une extrémité gauche then
    return G(P) = {p}, D(S) = ∅, V(P) = ∅
  else if p est une extrémité droite then
    return G(S) = ∅, D(P) = {p}, V(P) = ∅
  else if p est un segment vertical then
    return G(S) = ∅, D(P) = ∅, V(P) = {p}
  end if
end if
# Division du problème
(P1, n1) ← Nombre_intersections_DPR(PartieGauche(P))
(P2, n2) ← Nombre_intersections_DPR(PartieDroite(P))
# Réunion
G(P) ← [G(P1)\D(P2)] ∪ G(P2)
G(P) ← [D(P2)\G(P1)] ∪ D(P1)
V(P) ← V(P1) ∪ V(P2)
n ← n1 + n2
# Il ne reste ici qu'à vérifier les coordonnées y pour les ensemble suivants, on pourrait aussi trier les points
selon y pour accélérer l'opération
n ← n + intersect([G(P1)\D(P2)], V(P2)
n ← n + intersect([D(P2)\G(P1)], V(P1)
return n
```

3.3 Sweep-Line

Plutôt que d'implémenter la méthode sweep-line vue en cours, nous avons décidé d'implémenter une autre méthode basé sur le principe sweep-line et qui correspond mieux au problème précis où nous n'avons que des segments verticaux et horizontaux. Ainsi, le principe de fonctionnement est très similaire à la recherche de la paire de points la plus proche.

Algorithm 6 Nombre d'intersections SL

Input : V : ensemble de segments verticaux, H : ensemble de segments horizontaux
Output : n : nombre d'intersections

```
n ← 0
for h ∈ H do
  for v ∈ V : h.x1 ≤ v.x ET h.x2 ≥ v.x do
    # Il ne reste ici qu'à vérifier la coordonnée y
    if intersect(v, h) then
      n ← n + 1
    end if
  end for
end for
return n
```

3.4 Comparaison des performances pour des tailles de segment aléatoires

Chaque résultat est une moyenne sur 5 exécutions de l'algorithme. On voit que l'approche Brute-Force est la moins efficace avec une complexité $T(n) = \theta(n^2)$. L'algorithme Sweep-Line n'est pas loin derrière avec un complexité proche de $T(n) = \theta(n^2)$. La méthode DPR est la plus efficace avec tout de même une complexité qui semble polynomiale. Voir graphique partie A.2.1.

3.5 Influence de la taille des segments sur le temps d'exécutions

Les tests sont réalisés sur des fenêtres de 500*500.

La taille des segments influe peu sur le temps d'exécution de la méthode Brute-Force, on aura toujours une complexité en $T(n) = \theta(n^2)$. Elle néanmoins a une influence sur le nombre de comparaisons qui seront effectuées. En effet 4 comparaisons sont nécessaires pour savoir si deux segments se coupent or, si la première est fausse, les autres ne sont pas effectuées. Pour des tailles de segment plus petites on a donc plus de chance de s'arrêter plus tôt.

En revanche pour les approches Sweep-Line et DPR la taille des segments influe sur la classe de complexité. Pour de grandes tailles de segment comme 300 ou 150 on a une complexité $T(n) = \theta(n^2)$ alors que sur des petites tailles comme 5 ou 0.1, on se rapproche du $T(n) = \theta(n*log(n))$ (toujours imputé au tri des tableaux). Voir graphique partie A.2.2.

3.6 Comparaison des méthodes Sweep-Line et DPR pour de petits segments

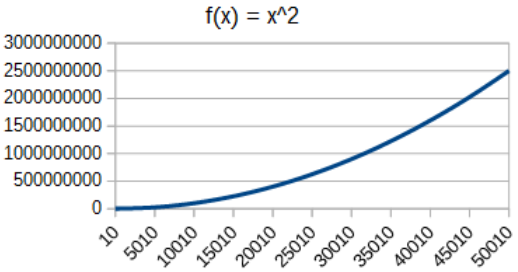
Il semble que plus la taille des segments est petite, plus il devient avantageux d'opter pour l'approche Sweep-Line. En effet pour des tailles de segments de 5, le DPR se montre plus performant avec un complexité $T(n) = \theta(n * log(n))$ alors que l'approche Sweep-Line semble encore s'exécuter en temps polynomial. En revanche pour un taille de segment de 0.1 l'approche Sweep-Line s'exécute en $T(n) = \theta(n * log(n))$ avec de meilleures performances que le DPR. Voir graphique partie A.2.3.

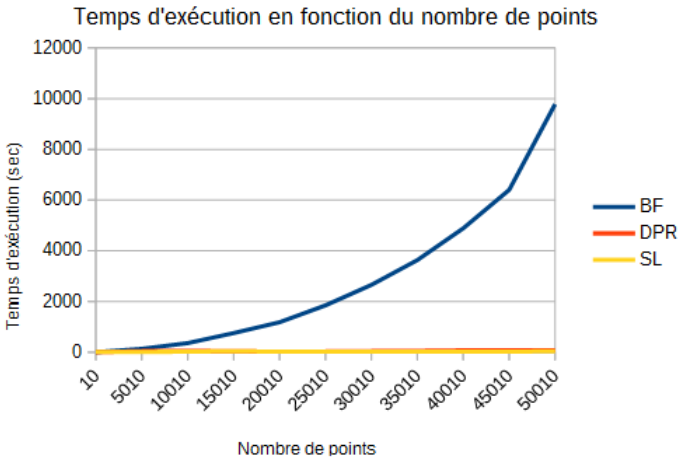
4 Conclusion

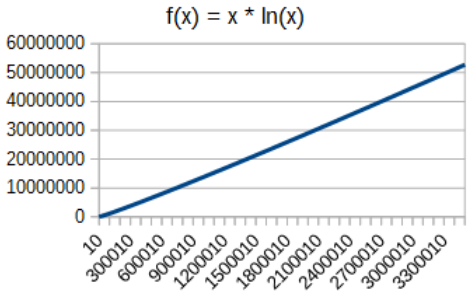
Ce projet nous a permis de concrétiser les différentes techniques vues en cours et d'appréhender de nouvelles méthodes de résolution de problème. Nous avons pu aussi découvrir le logiciel Processing qui nous a permis de représenter facilement nos problèmes en facilitant l'utilisation d'une interface graphique pour la représentation de formes géométriques.

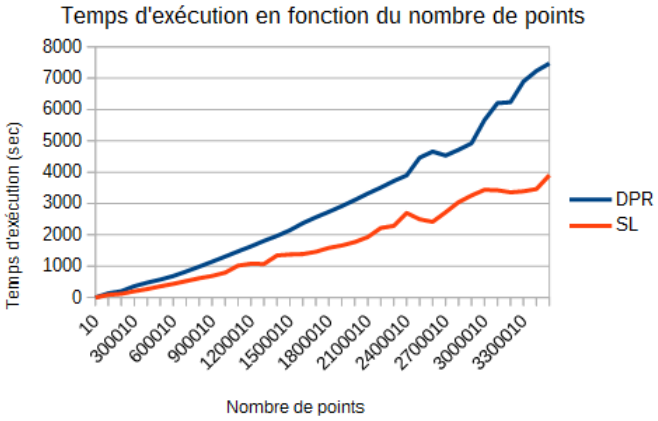
A Annexe

A.1 Recherche de la plus petite paire de points



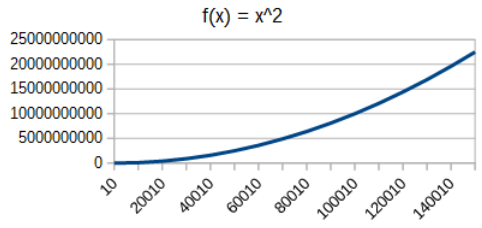


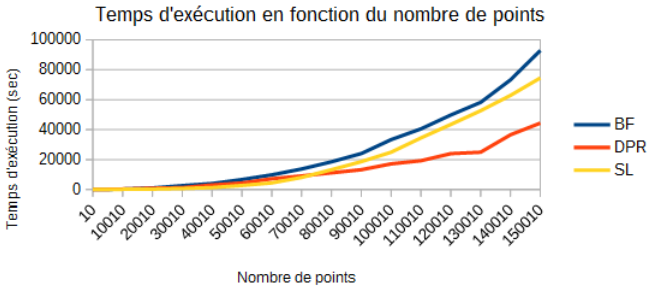




A.2 Recherche du nombre d'intersections

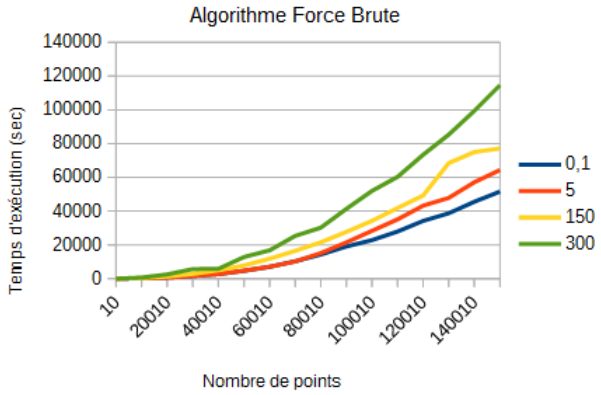
A.2.1 Taille de segment aléatoire

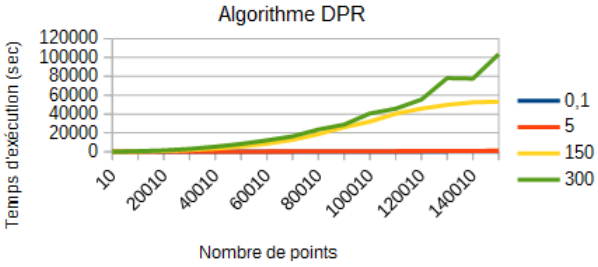


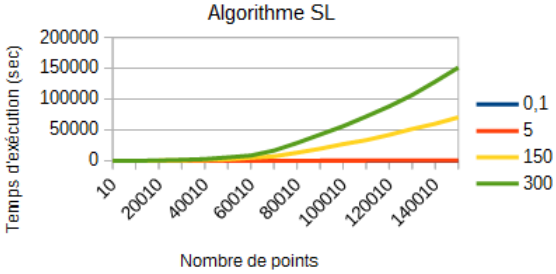


A.2.2 Influence de la taille des segments

Temps d'exécution en fonction du nombre de points pour des longueurs de segment différentes

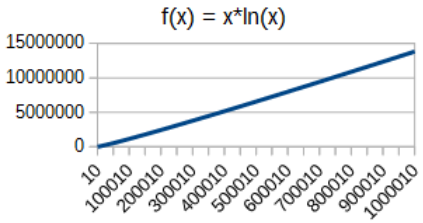


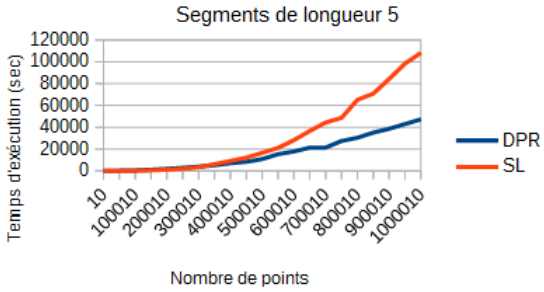


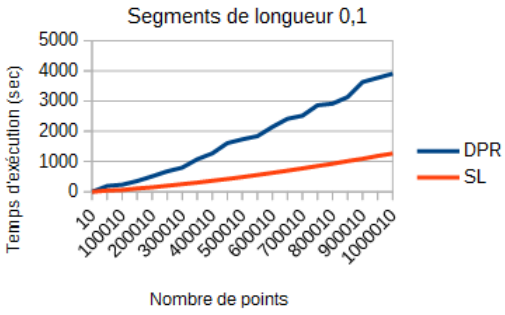


A.2.3 Comparaison des méthodes Sweep-Line et DPR pour de petits segments

Temps d'exécution en fonction du nombre de points pour des longueurs de segment fixe







B Bibliographie

Description du principe utilisé pour l'intersection de segments en DPR : `https://electures.informatik.uni-freiburg.de/portal/download/4003/21331/01_introduction_26_10_ann.pdf`