Paire de points la plus proche ou *Closest Pair*

Utilisation du paradigme « diviser pour régner » en OCaml

Lucas Tabary

1 Introduction

On considère un ensemble de points (qu'on nommera *nuage*) du plan $\mathbb{R}^2$ muni d'un repère orthonormé $(O;\ \vec{\imath},\ \vec{\jmath})$ de n points deux à deux distincts, $n \geqslant 2$. Les points sont indexés (arbitrairement) et on peut écrire le nuage sous la forme $\mathcal{N} = \{M_1,\ M_2, \ldots,\ M_n\}$. On recherche la distance minimale entre deux points de ce nuage. On représentera dans la suite de l'exercice un nuage en mémoire avec le type : `(float * float) array`. On utilisera les primitives `fst` et `snd` pour récupérer respectivement l'abscisse et l'ordonnée des points, ou bien la compréhension `x, y = m`, où `m` est la variable informatique représentant un point du nuage.

Par la suite, on utilisera la fonction `distance m1 m2` définie ci-dessous, renvoyant le carré de la distance entre deux points `m1` et `m2` du nuage.

```
let distance m1 m2 =
  let dx = (fst m2) -. (fst m1) and dy = (snd m2) -. (snd m1) in
    dx *. dx +. dy *. dy
```

On envisagera dans un premier temps un algorithme naïf pour en évaluer la complexité avant de considérer un algorithme basé sur le paradigme de « diviser pour régner ».

```
val l : (float * float) array =
  [|(1., 2.1); (4.5, -1.6); (5.4, 1.2); (1.1, 2.3)|]
# fst l.(1);;
- : float = 4.5
```

Figure 1 – Exemple de manipulation d'un nuage

2 Algorithme naïf

2.1 Cas général

On ne cherchera pas à démontrer la terminaison ni la correction des fonctions suivantes. Construisons dans un premier temps une fonction `distance_i nuage i n` qui, pour un nuage `nuage` donné, détermine la distance du `i`-ème point de l'ensemble à ceux d'indice supérieur. On construit préalablement une fonction `min_ref r n` qui, à une référence `r` attribue `n` si `n < !r` et ne fait rien sinon, ce qui permet de réduire la taille de la fonction et de gagner en lisibilité, tout comme la fonction `dist j`.

```
let min_ref r n =
  if n < !r then r := n

let distance_i nuage i n =
  let dist j = distance nuage.(i) nuage.(j) in
  let mini = ref (dist (i + 1)) in
  for k = i + 2 to n - 1 do
```

```
8       min_ref mini (dist k)
9     done;
10    !mini
```

On peut dès lors construire la fonction `distance_mini nuage` qui renvoie le carré [1] de la distance minimale entre deux points du nuage `nuage`, en évaluant toutes les distances mesurables entre des points du nuages. On réalise ainsi la mesure de la distance du premier point au $n-1$ suivants, puis du deuxième aux $n-2$ restants, etc.

```
1  let distance_mini nuage =
2    let n = Array.length nuage in
3    let mini = ref (distance_i nuage 0 n) in
4    for i = 1 to n - 2 do
5      min_ref mini (distance_i nuage i n)
6    done;
7    !mini
```

Établissons maintenant l'expression de la complexité de `distance_mini nuage`. On considérera n la taille du nuage `nuage` comme taille de donnée et l'appel à `distance m1 m2` comme opération élémentaire ; on notera $C(n)$ la complexité évaluée. La fonction `distance_i nuage i n` réalise un appel à `distance m1 m2` puis une répétitive indexée de $i+2$ à $n-2$ avec un appel à chaque tour. Enfin la fonction distance mini réalise un appel à `distance_i nuage i n` pour $i=0$ et ensuite une répétitive indexée de 1 à $n-1$ réalisant un appel à `distance_i nuage i n`. Finalement :

$$C(n) = \left(1+\sum_{k=2}^{n-1}1\right)+\sum_{i=1}^{n-2}\left(1+\sum_{k=i+2}^{n-1}1\right) = \sum_{i=0}^{n-2}\left(1+\sum_{k=i+2}^{n-1}1\right) = \sum_{i=0}^{n-2}\sum_{k=i+1}^{n-1}1 = \sum_{i=0}^{n-2}(n-1-(i+1)+1)$$
$$= \sum_{i=0}^{n-2}(n-1-i) = (n-1)(n-1)-\sum_{i=0}^{n-2}i = (n-1)^2-\frac{(n-2)(n-1)}{2} = \frac{n(n-1)}{2}$$

$$\therefore C(n) \underset{+\infty}{=} O\big(n^2\big)$$

2.2 Cas d'un nuage en dimension 1

On suppose les points alignés et le tableau `nuage` trié par abscisses croissantes. Puisqu'ils sont alignés, on a pour un point M_i du nuage :

$$\forall j \in [\![i+2,\, n]\!],\ M_iM_j = M_iM_{i+1} + M_{i+1}M_j > M_iM_{i+1}$$

Ainsi une distance entre deux points non adjacents dans le tableau ne peut être le minimum et il n'y a, alors, qu'à vérifier les distances entre les points adjacents.

```
1  let distance_mini_1D nuage =
2    let n = Array.length n in
3    let dist i = distance nuage.(i) nuage.(i + 1) in
4    let mini = ref (dist 0) in
5    for i = 1 to n - 2 do
6      min_ref mini (dist i)
7    done;
8    sqrt !mini
```

Un seul appel à `distance m1 m2` est réalisé directement dans une répétitive indexée qui effectue environ n tours de boucle. Puisque aucun appel n'est réalisé ailleurs, on a bien $C(n) = O(n)$.

1. On renvoie le carré et non la valeur (qu'on peut obtenir en utiliser `sqrt`), afin de réutiliser la fonction ensuite

3 Diviser pour régner

3.1 Principe et application

On suppose maintenant qu'on possède deux listes `nuage_h` et `nuage_v` contenant les points du nuage `nuage`, triées selon l'ordre lexicographique sur $\mathbb{R}^2$, l'une en considérant d'abord les abscisses et l'autre les ordonnées. On applique le principe de « diviser pour régner » en recherchant les minima dans des tableaux de tailles strictement inférieures — on découpe le nuage en sous-nuages. Écrivons d'abord la fonction `partition nuage` qui renvoie deux tableaux de tailles strictement inférieures, pour un nuage `nuage` ordonné (on choisira `nuage_h`). On découpera en deux nuages de tailles $\left\lfloor \frac{n}{2} \right\rfloor$ et $n - \left\lfloor \frac{n}{2} \right\rfloor$, soit

$$n - \left\lfloor \frac{n}{2} \right\rfloor = \begin{cases} n/2 & \text{n pair} \\ (n+1)/2 & \text{n impair} \end{cases} \quad \text{et} \quad \left\lfloor \frac{n}{2} \right\rfloor = \begin{cases} n/2 & \text{n pair} \\ (n-1)/2 & \text{n impair} \end{cases}$$

```
let partition nuage =
  let n = Array.length nuage in
  let nuage_g = Array.make (n / 2) (0.0, 0.0)
  and nuage_d = Array.make (n - (n / 2)) (0.0, 0.0) in
  for k = 0 to n - 1 do
    if k < n / 2 then nuage_g.(k) <- nuage.(k)
    else nuage_d.(k - (n / 2)) <- nuage.(k)
  done;
  nuage_g, nuage_d
```

Par ailleurs, il est nécessaire de partitionner le nuage verticale aussi selon le positionnement par rapport à x_{med}, qui correspond à l'abscisse du point $M_{\lfloor \frac{n}{2} \rfloor}$. On construit dans ce but la fonction `tri_vertical nuage_v seuil` suivante.

```
let tri_vertical nuage_v seuil =
  let n = Array.length nuage_v in
  let g = ref [||] and d = ref [||] in
  for k = 0 to n - 1 do
    let m = nuage_v.(k) in
    let choix = if (fst m) < seuil then g else d in
    choix := Array.append !choix [|m|]
  done;
  !g, !d
```

Dans le cas où les nuages sont suffisamment petits ($n < 4$), on applique de nouveau l'algorithme naïf. Sinon, on procède récursivement et on partitionne encore. Une fois qu'on a déterminé les minima des nuages à gauche (d_g) et à droite (d_d), on pose $\delta = \min(d_g, d_d)$ et on isole les points de l'ensemble B tel que $B = \{(x,y) \in \mathcal{N},\ |x - x_{med}| < \delta\}$, où x_{med} désigne l'abscisse du point $M_{\lfloor \frac{n}{2} \rfloor}$ à la limite des deux nuages.

Deux points dans le même demi-plan (d'équation $x < x_{med}$ ou $x \geqslant x_{med}$) ne peuvent être à une distance strictement plus petite que δ, par définition de δ. Par ailleurs, pour $M_1(x_1, y_1), M_2(x_2, y_2) \in \mathcal{N}$ on a :

$$M_1M_2^2 = \underbrace{(x_2 - x_1)^2}_{\geqslant 0} + \underbrace{(y_2 - y_1)^2}_{\geqslant 0} \Longrightarrow \begin{cases} M_1M_2 \geqslant \sqrt{(x_2 - x_1)^2} = |x_2 - x_1| \\ M_1M_2 \geqslant \sqrt{(y_2 - y_1)^2} = |y_2 - y_1| \end{cases}$$

On considère de plus $x_1 < x_{med}$, $M_1 \notin B \Longrightarrow x_1 < x_{med} - \delta$ et $x_2 \geqslant x_{med}$, d'où $x_2 \geqslant x_{med} > x_1 + \delta \Longrightarrow x_2 - x_1 > \delta \Longrightarrow M_1M_2 \geqslant |x_2 - x_1| > \delta$. L'autre cas ($x_2 > x_{med} + \delta$) est symétrique. Ainsi deux points qui ne sont pas dans B ne peuvent former un minimum. À l'inverse, des points de B ont pu ne pas être comparés car ils n'étaient pas dans le même nuage et sont pourtant plus proches (cas des deux points rouges, 2). On déterminera ainsi le minimum sur l'ensemble de ces points.

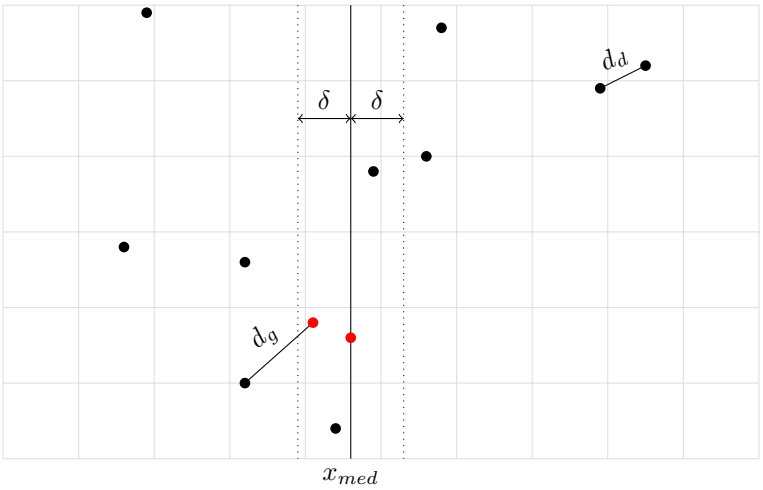

FIGURE 2 – Exemple de découpage

On établit la fonction `restriction_bande nuage_v xmed delta` qui, à un nuage trié selon les ordonnées `nuage_v`, associe la bande B définie précédemment.

```
let restriction_bande nuage_v xmed delta =
  let bande = ref [||] in
  for k = 0 to (Array.length nuage_v) - 1 do
    let m = nuage_v.(k) in
    if abs_float ((fst m) -. xmed) < delta then
      bande := Array.append !bande [|m|]
  done;
  !bande
```

On cherche à réduire la longueur de la bande étudiée. On considère alors $M_1(x_1, y_1)$, $M_2(x_2, y_2) \in B$ avec $y_2 > y_1 + \delta \Longrightarrow y_2 - y_1 > \delta$. Alors $M_1M_2 \geqslant |y_2 - y_1| > \delta$. Ce couple de points ne peut donc pas correspondre au minimum.

Soit un carré de longueur $\delta/2$ situé dans un demi-plan ($x < x_{med}$ ou $x \geqslant x_{med}$). Toute mesure à l'intérieur de ce carré ne peut excéder la longueur de la diagonale, soit $\sqrt{2} \cdot \frac{\delta}{2}$. On suppose ainsi qu'il existe au moins deux points M_1 et M_2 du nuage $\mathcal{N}$ dans ce carré. Automatiquement, $M_1M_2 < \sqrt{2}\frac{\delta}{2} = \frac{\delta}{\sqrt{2}} < \delta$. Ceci est absurde par minimalité de la distance δ sur les demi-plans. Il existe donc au plus un point du nuage dans un tel carré.

La bande réduite étant un rectangle de dimension $(\delta + \delta) \times (\delta)$, il est possible d'y construire 8 carrés de côté $\delta/2$ comme sur la figure 3.

Ces carrés étant situés dans des demi-plans distincts, ils possèdent au plus 1 point : soit 8 points au total dans cette zone, c'est-à-dire 7 points distincts du point étudié. On n'a donc qu'à déterminer au plus les distances du point au 7 points qui possèdent une ordonnée directement supérieure à la sienne. On considère alors une fonction minimum qui correspond à l'algorithme naïf appliqué initialement mais limité à 7 itérations, notée `minimum_global bande delta`.

```
let minimum_global bande delta =
  let n = Array.length bande in
  let dist k j = distance bande.(k) bande.(j) in
  let mini = ref (dist 0 1) in
  for k = 0 to (n - 2) do
    for j = k + 1 to min (k + 7) (n - 1) do
```

```
      min_ref mini (dist k j);
    done;
  done;
  min delta !mini
```

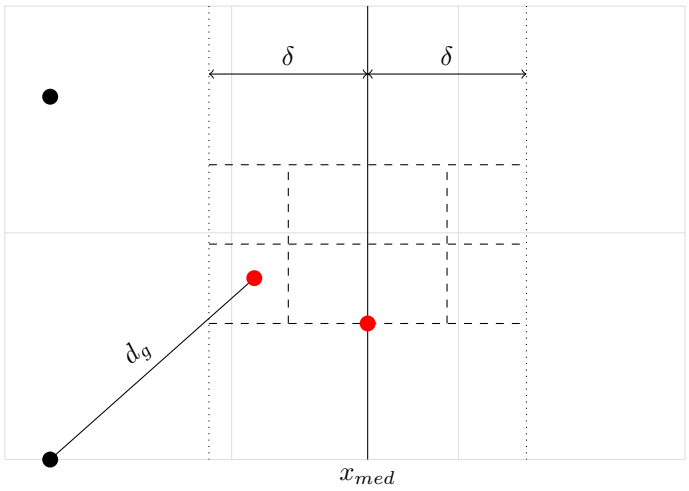

FIGURE 3 – Construction des carrés de longueur $\delta/2$ (reprise de l'exemple précédent)

On établit finalement la fonction `distance_mini_dpr nuage_h nuage_v` qui associe au même nuage, trié selon les abscisses et les ordonnées, le carré [2] de la distance minimale entre deux points.

```
let rec distance_mini_dpr nuage_h nuage_v =
  let n = Array.length nuage_h in
  if n < 4 then
    distance_mini nuage_h
  else
    let nuage_g, nuage_d = partition nuage_h in
    let xmed = fst nuage_d.(0) in
    let nuage_vg, nuage_vd = tri_vertical nuage_v xmed in
    let dg = distance_mini_dpr nuage_g nuage_vg
    and dd = distance_mini_dpr nuage_d nuage_vd in
    let delta = min dg dd in
    let bande = restriction_bande nuage_v xmed (sqrt delta) in
    if Array.length bande = 1 then
      delta (* delta est nécessairement le minimum *)
    else
      minimum_global bande delta
```

3.2 Calcul de la complexité — Pire des cas

On considère un nuage de taille n, qui sera fixé comme taille de donnée. L'opération élémentaire est ici le nombre d'appels à la fonction `distance`. On suppose d'abord que $n = 2^k$, $k \in \mathbb{N}$. Ainsi à chaque appel sur un nuage de taille k, les deux sous-nuages créés par `partition` sont de tailles $\frac{k}{2} \in \mathbb{N}$ chacun.

On constate qu'aucun appel à `distance` n'est effectué par les fonctions `restriction_bande`, `partition` et `tri_vertical`. Reste ainsi la fonction `minimum_global`. On suppose dans le pire

2. Encore une fois, cela facilite les appels récursifs car les calculs sont effectués à partir de cette quantité, et non de la véritable distance.

des cas que la bande `bande` correspond au nuage entier `nuage_v`, de taille n. La fonction présente deux répétitives indexées, la première effectuant $n-2+1$ tours de boucles et la deuxième au pire 7, ceci provenant d'un résultat démontré précédemment. Cette seconde boucle effectuant un seul appel à `distance` par itération, on détermine la complexité temporelle notée $C_1(n)$ de `minimum_global` par :

$$C_1(n) = 1 + \sum_{k=0}^{n-2} \sum_{j=k+1}^{k+7} 1 = 1 + \sum_{k=1}^{n-1} 7 = 1 + 7(n-1) = 7n - 6$$

Enfin, deux appels à `distance_mini_dpr` sont réalisés dans la fonction-même, sur des données de tailles $\frac{n}{2}$. On obtient alors :

$$C(n) = 2C\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 7n - 6 \Longrightarrow C(n) - 6 = 2C\left(\left\lfloor \frac{n}{2} \right\rfloor\right) - 12 + 7n = 2\left[C\left(\left\lfloor \frac{n}{2} \right\rfloor\right) - 6\right] + 7n$$

En considérant la suite $(C(n)-6)_{n\in\mathbb{N}}$, on reconnaît une suite dichotomique de paramètres $p = 2$, $\lambda = 7$, $\gamma = 1$. Or, $\log_2 p = 1 = \gamma$. On en conclut :

$$C(n) - 6 \underset{+\infty}{=} O\big(n^1 \log_2 n\big) \Longrightarrow C(n) \underset{+\infty}{=} O(n \ln n)$$

Nombres d'appels à `distance` dans plusieurs cas et évaluation du temps

Le cas de l'algorithme diviser pour régner correspond à une valeur moyenne

$n =$	10	100	500	1 000	2 000	10 000	50 000
Algorithme naïf							
Opérations	45	4 950	124 750	499 500	1 999 000	49 995 000	1 249 975 000
Temps d'éxécution	24 μs	269 μs	10 ms	26 ms	103 ms	2, 85 s	85 s
Diviser pour régner							
Opérations	20	550	4 872	11 500	26 500	176 767	1 137 049
Temps d'éxécution	56 μs	120 μs	900 μs	4 ms	12 ms	258 ms	7, 4 s