Recherche des plus proches voisins par des arbres k d

Motivation : Recherche des PPV → Utile en apprentissage supervisé pour former des groupes de points.

Définitions : On appelle **point** un élément de $\mathbb{N}^d$, où $d \in \mathbb{N}^*$ est appelée la **dimension**. La distance entre deux points $(x_1, \dots, x_d)$ et $(y_1, \dots, y_d)$ est définie par : $d((x_1, \dots, x_d), (y_1, \dots, y_d)) = (x_1 - y_1)^2 + \dots + (x_d - y_d)^2$.

Problème : k - PPV

Entrée : $p_1, \dots, p_n \in \mathbb{N}^d$, $p \in \mathbb{N}$, $k \in [\![1; n]\!]$

Sortie : k plus proches voisins de p parmi $p_1, \dots, p_n$.

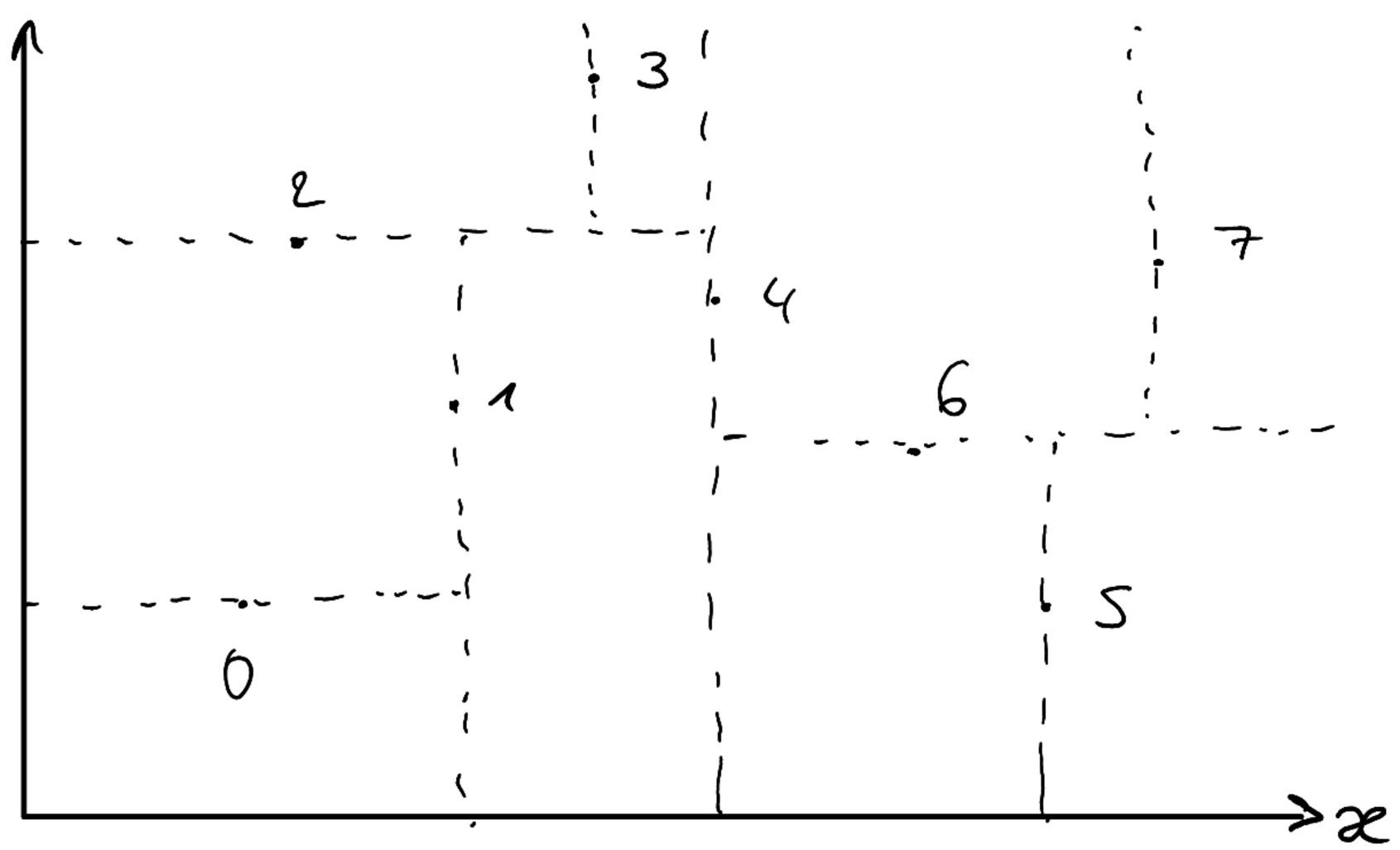
Solutions naïves :

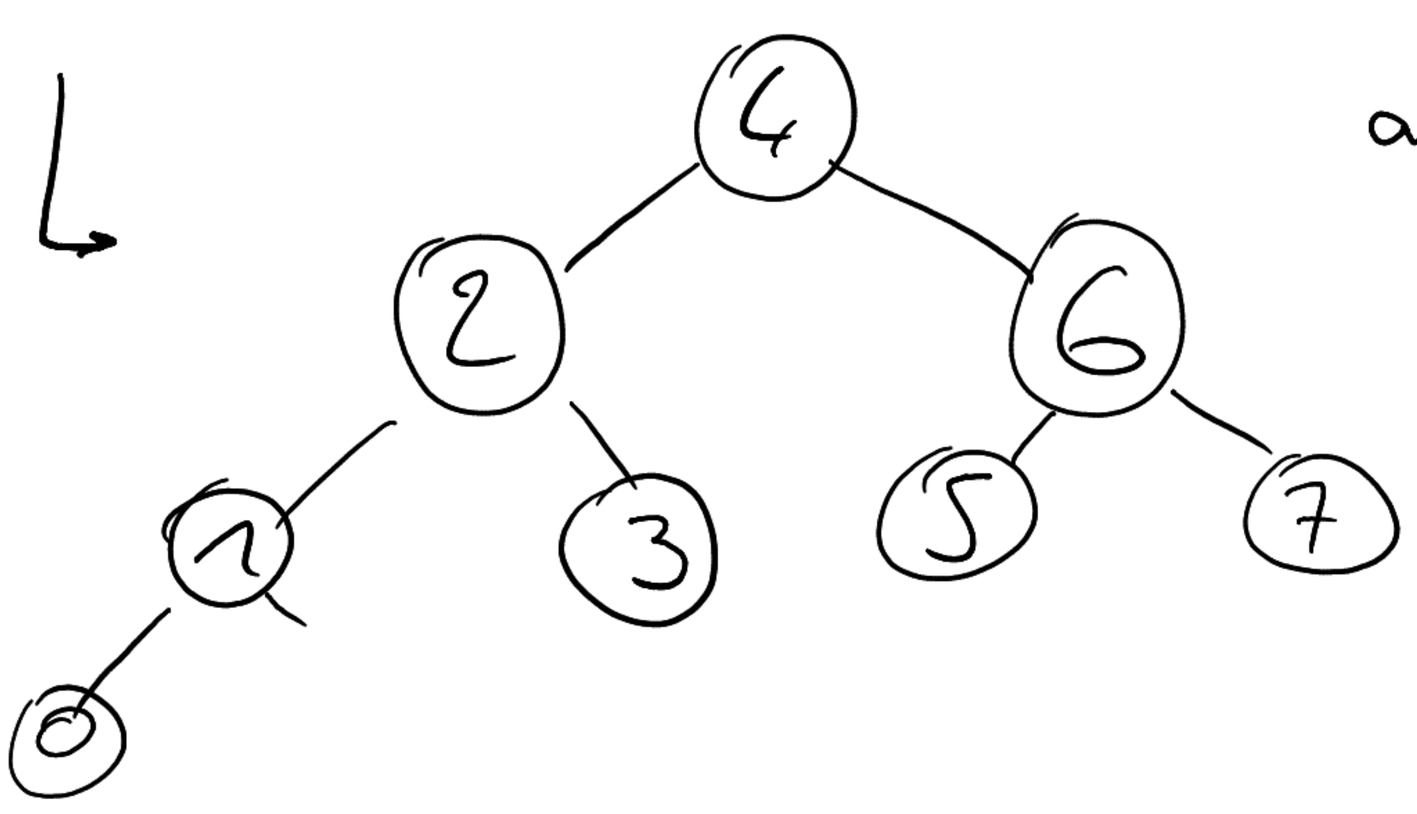
- Tas min contenant les k plus proches voisins
  ↳ $O(n \log(k))$ en temps et $O(k)$ en mémoire
- Tas min contenant toutes les distances
  ↳ $O(n + u \log(u))$ en temps et $O(u)$ en mémoire
  ↳ Peu efficace pour beaucoup de requêtes.

Idée : Utiliser une structure d'arbre binaire pour diviser l'espace de recherche

Définition: Un arbre k-dimensionnel est un arbre binaire de recherche équilibré où les éléments sont des éléments de $\mathbb{N}^d$ comparés à chaque profondeur dans une dimension différente.

Exemple :





arbre 2d

```
Fonction construction_arbre_rd(points)
    | Retourner construction(0, points, 0, |points|)

Fonction construction(i, points, bas, haut)
    | Si bas = haut alors
        | Retourner Vide
    | Sinon
        | mid := (bas + haut) / 2
        | idx_med := Indice du médian des points pour la ie coordonnée
        | Echanger points[idx_med] et points[mid]
        | Placer dans points[bas: mid-1] les éléments ≤ à ceux de
        | points[idx_med], et dans points[mid+1. haut] les éléments ≥ à.
        | Retourner Noeud(construction((i+1)%.dim, points, bas, mid-1),
                        points[mid],
                        construction((i+1)%.dim, points, mid+1, haut)
                    )
```

Privilégier la recherche pour le développement.
Pas le temps de faire les deux algorithmes.

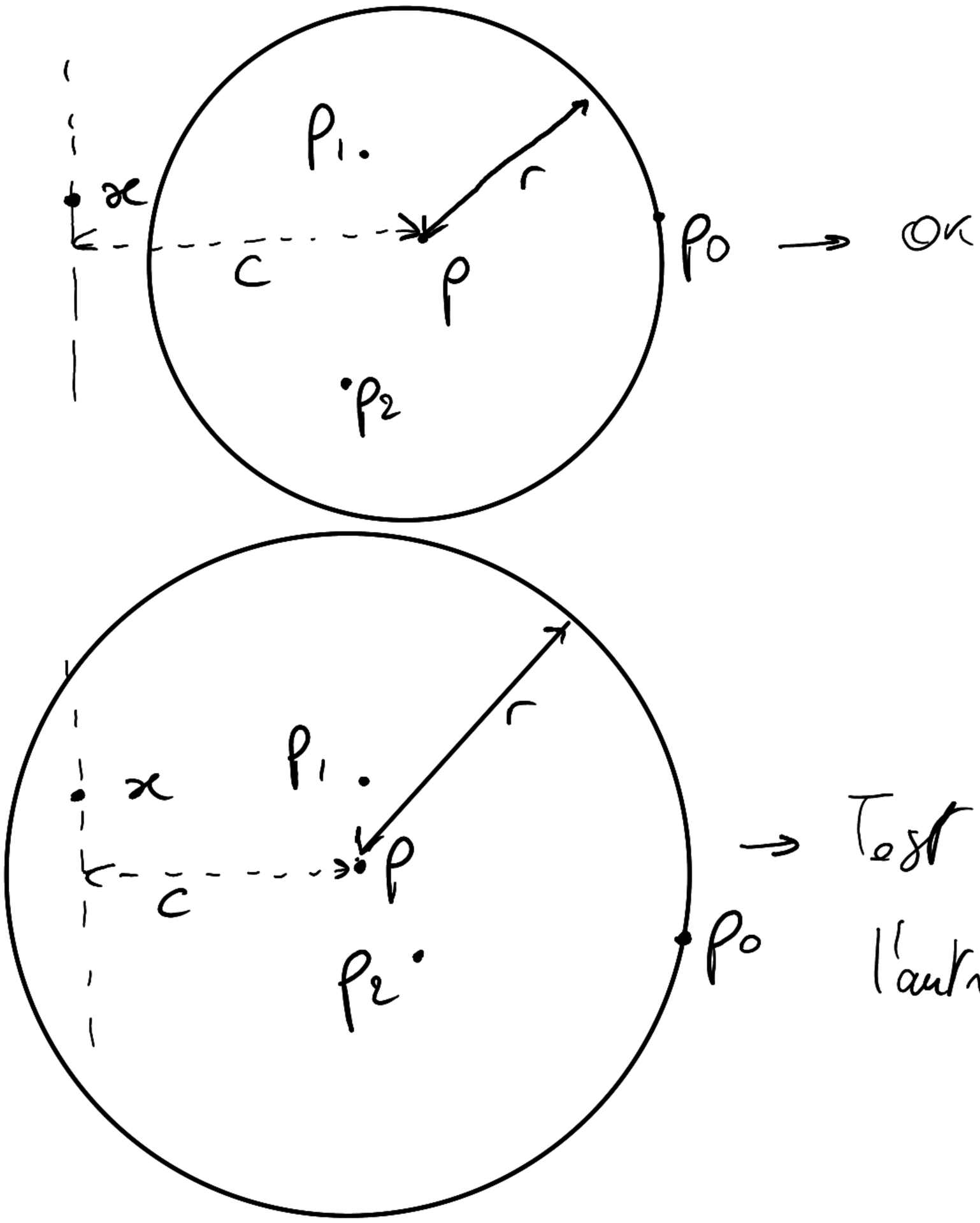
```
Fonction ppv (k, point0, kd tree)
    voisins := creer_tas_max()

    Fonction visiter(arbre, i)
        Si arbre = Vide alors skip
        Sinon arbre = Noeud(l, p, r) alors
            t1, t2 := Si point0.i ≤ p.i alors (l,r) sinon (r, l)
        visiter(t1, (i+1) %. dim)
        Si |voisins| < k ou
            max(voisins).distance > |p.i - point0.i| alors   (|p.i - point0.i| : c)
            enfiler(voisins, {point = p; distance = d(p, point0)})
            Si |voisins| > k alors supprimer_max(voisins)
            visiter(t2)

    visiter(kdtree)
    Retourner voisins
```

Complexité :

$O(n \log_2(n))$ pour la construction de l'arbre

$O(k \log(n))$ (en moyenne) par appel à ppv (admis)

Exemple :



→ Test dans l'autre sous-arbre