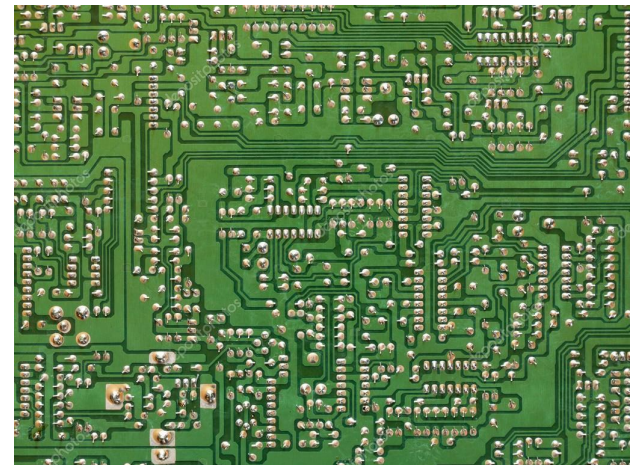
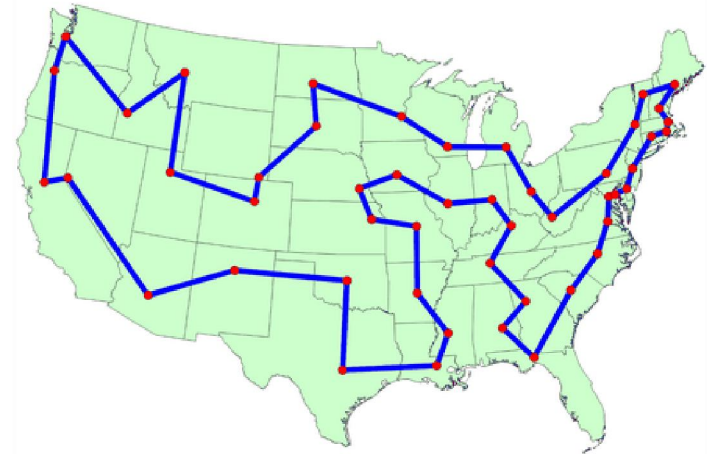


Parcours optimal d'un grand nombre de points

Problème du voyageur de commerce

Le problème



Problématique

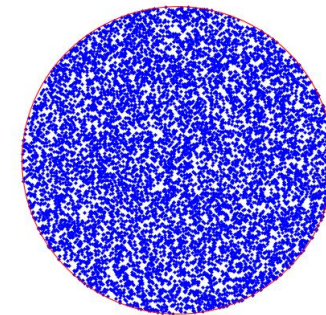
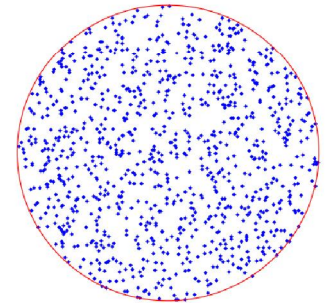
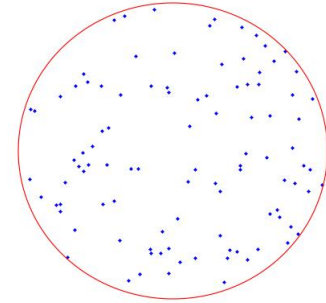
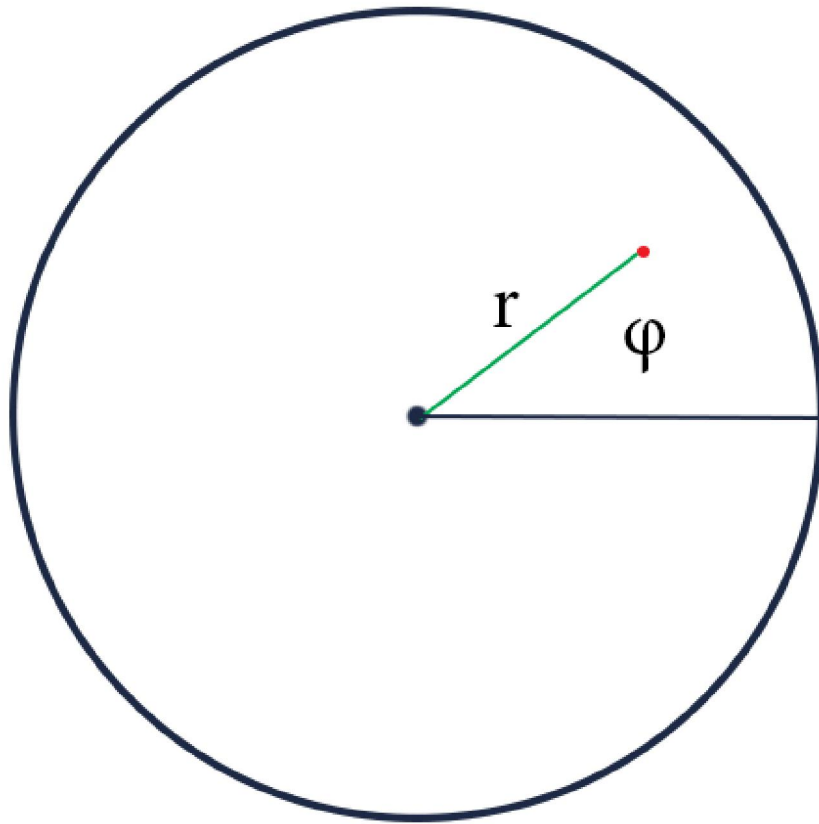
Naïf : $O(n!)$

Held & Karp : $O(2^n n^2)$

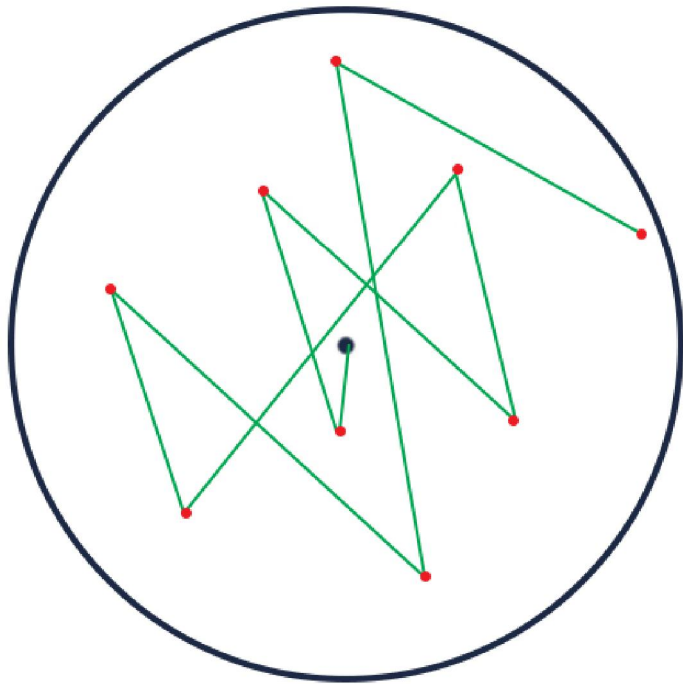
Plan

- Méthode impliquant un tri
 - Théorie
 - Expérience
- Méthode gloutonne
 - Théorie
 - Expérience
- Comparaison

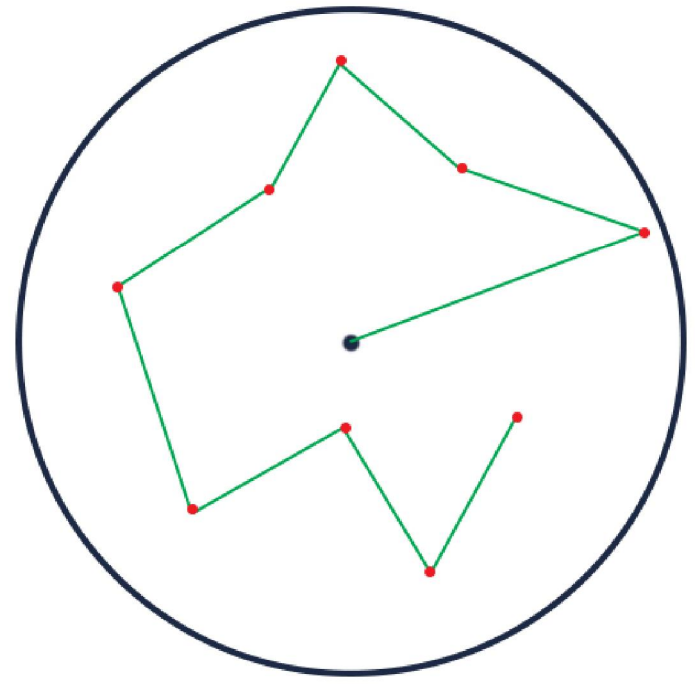
Modélisation



Tri selon le rayon, l'angle

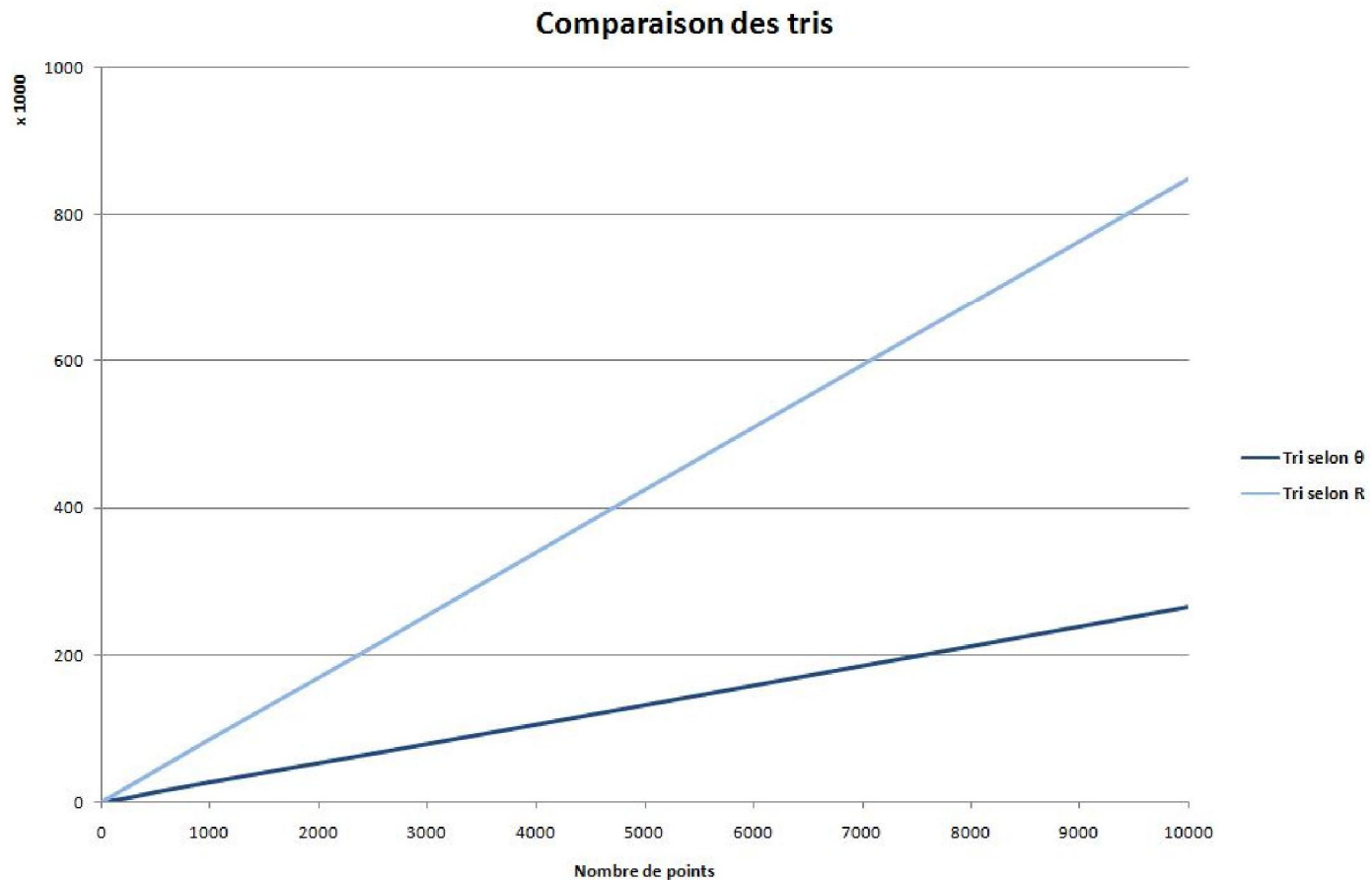


Rayon

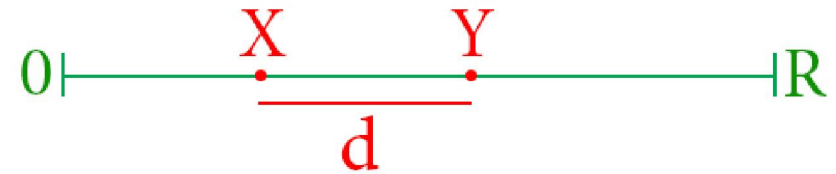
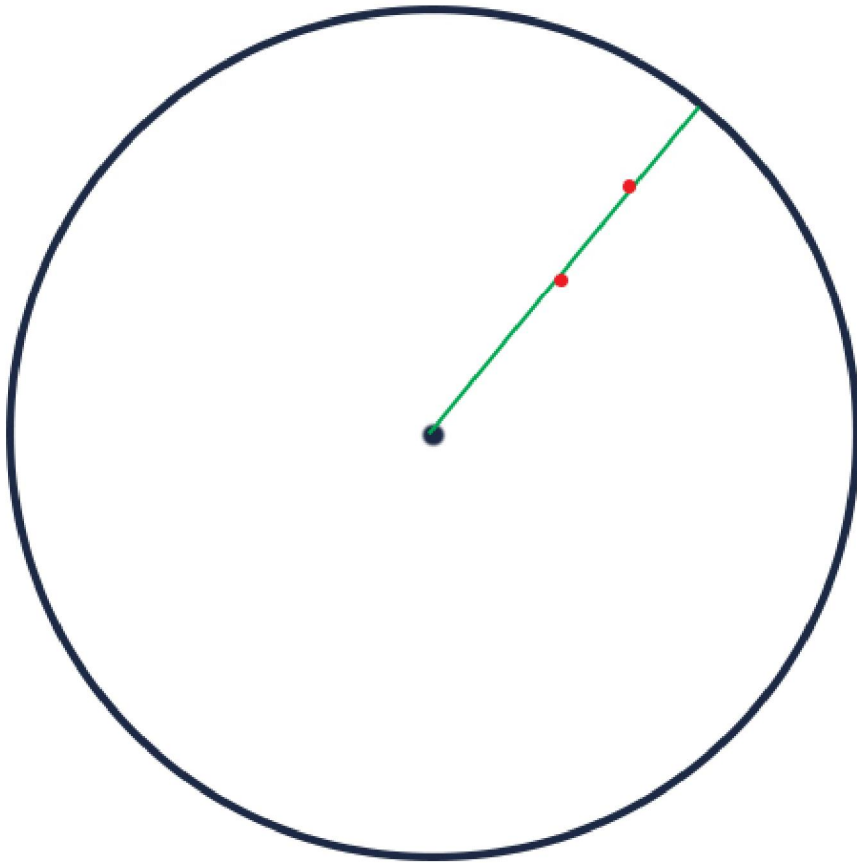


Angle

Comparaison des deux tris



Etude théorique du tri selon l'angle



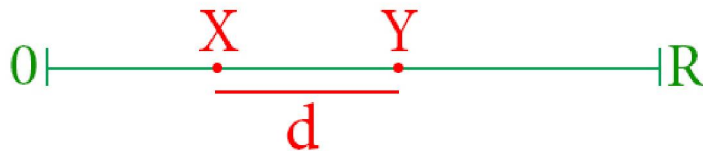
$$E(d) = \frac{4R}{15}$$

$$E(D) = \frac{4R}{15} \times n$$

Démonstration du tri selon l'angle

$$f(t) = \frac{2t}{R^2}$$

$$\forall t \in [0, R] : P(X = t) = P(Y = t) = f(t)$$



$$E(D) = \frac{4R}{15} \times n$$

$$E(d) = E(|X - Y|)$$

$$= \iint_0^R |x - y| \cdot f(x) \cdot f(y) \cdot dx \cdot dy$$

$$= \frac{4}{R^4} \int_0^R x \left[\int_0^R |x - y| \cdot y \cdot dy \right] dx$$

$$= \frac{4}{R^4} \int_0^R x \left[\int_0^x (x - y) \cdot y \cdot dy + \int_x^R (y - x) \cdot y \cdot dy \right] dx$$

$$= \frac{4R}{15}$$

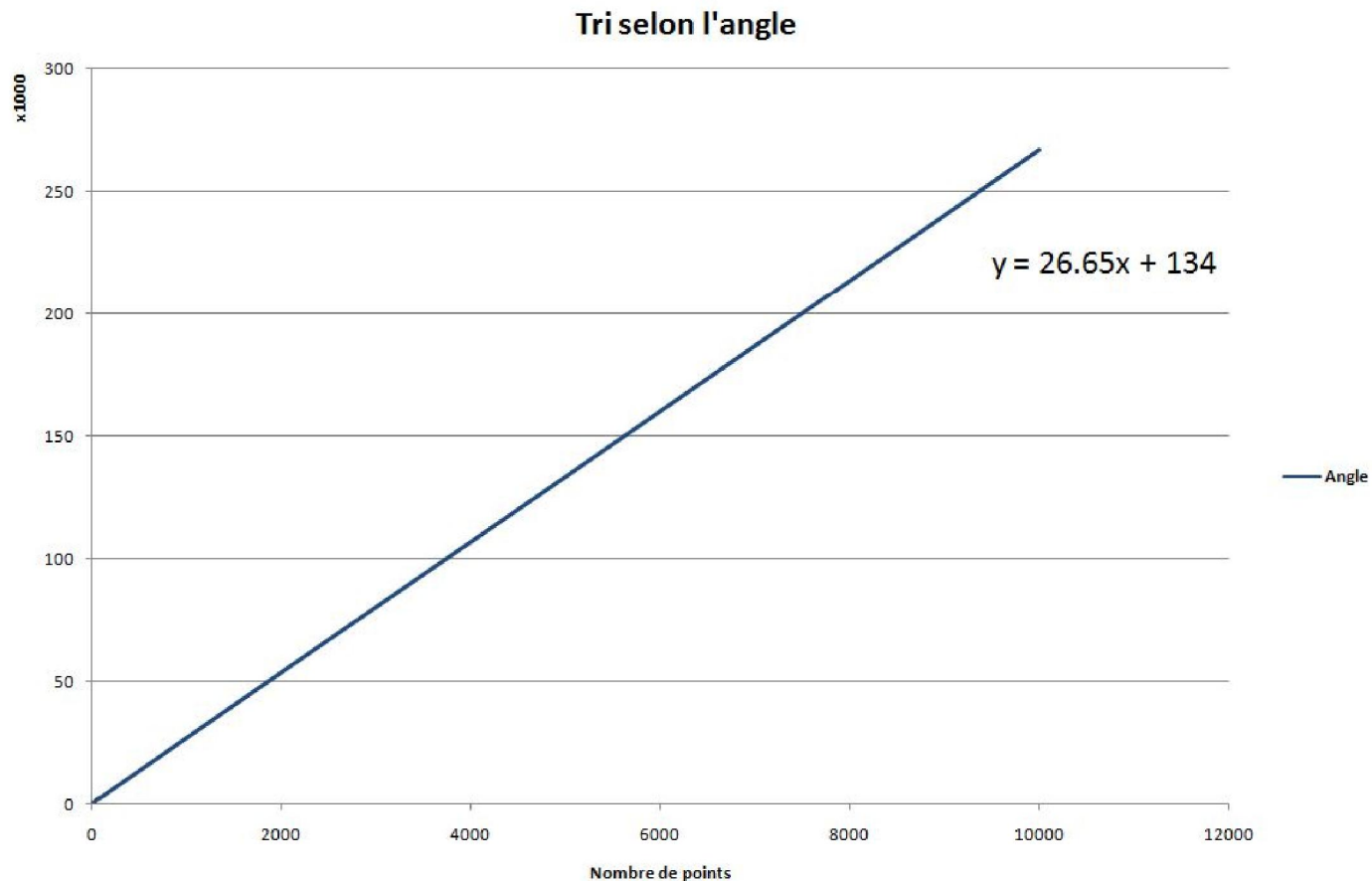
Tri fusion

```
def segmentation (L):  
    n=len(L)  
    if n==1 :  
        return(L)  
    return(L[:n//2],L[n//2:])  
  
def fusionT (A,B):  
    if A==[] :  
        return(B)  
    elif B==[] :  
        return(A)  
    else:  
        if A[0][1] > B[0][1]:  
            return([B[0]]+fusionT(A,B[1:]))  
        else :  
            return([A[0]]+fusionT(A[1:],B))  
  
def trifusionT(L):  
    if len(L)<=1:  
        return(L)  
    return(fusionT(trifusionT(segmentation(L)[0]),trifusionT(segmentation(L)[1])))
```

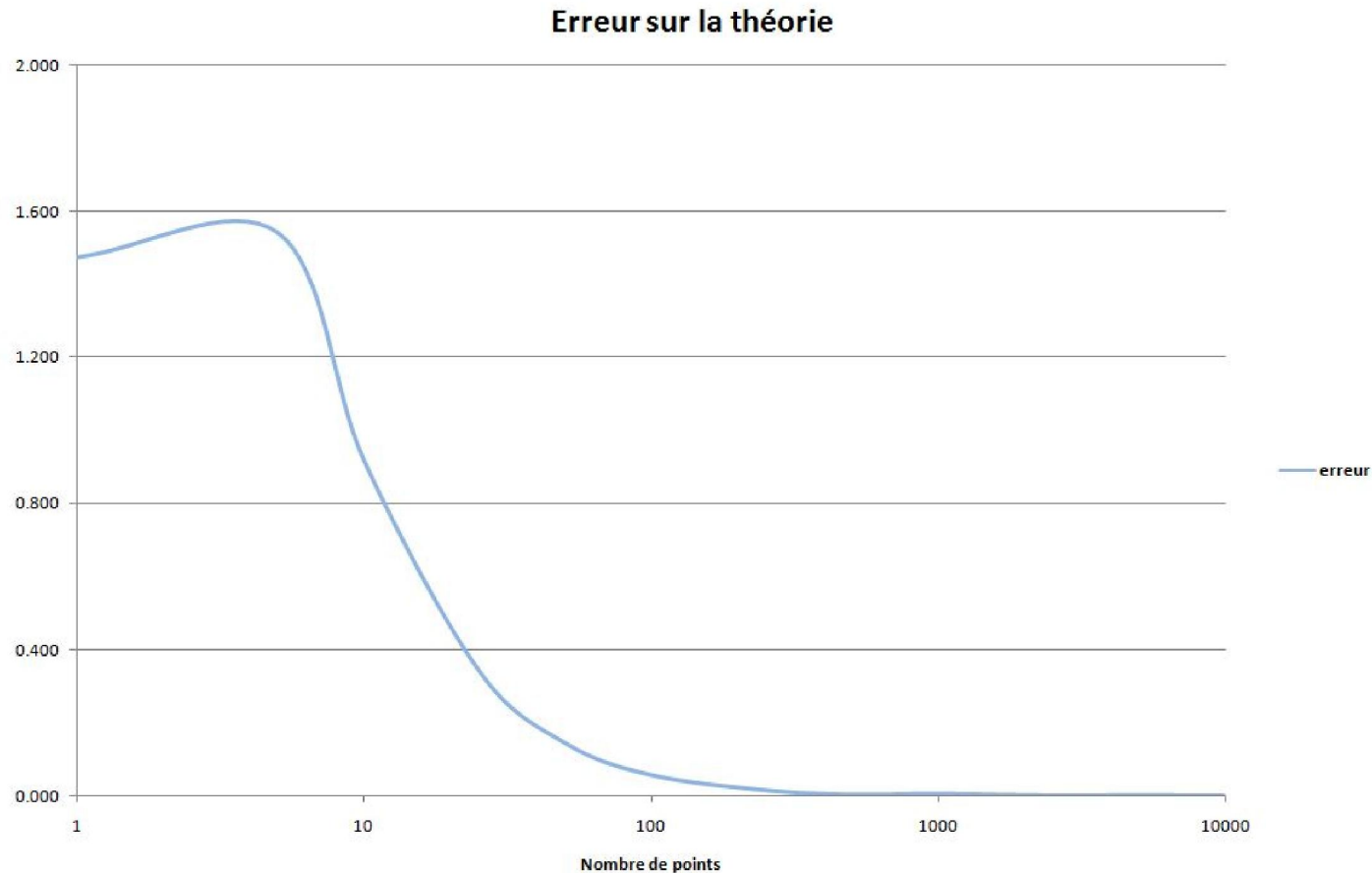
Calcul de distance

```
def distance(A):  
    """  
    Calcul de la longueur d'un chemin  
    """  
    n=len(A[0])  
    somme=0  
    if n!=0:  
        somme=A[0][0]  
        for i in range(0,n-1):  
            x1=A[0][i]*cos(A[1][i])  
            x2=A[0][i+1]*cos(A[1][i+1])  
            y1=A[0][i]*sin(A[1][i])  
            y2=A[0][i+1]*sin(A[1][i+1])  
            somme=somme+sqrt((x2-x1)**2+(y2-y1)**2)  
    return somme
```

Résultats expérimentaux



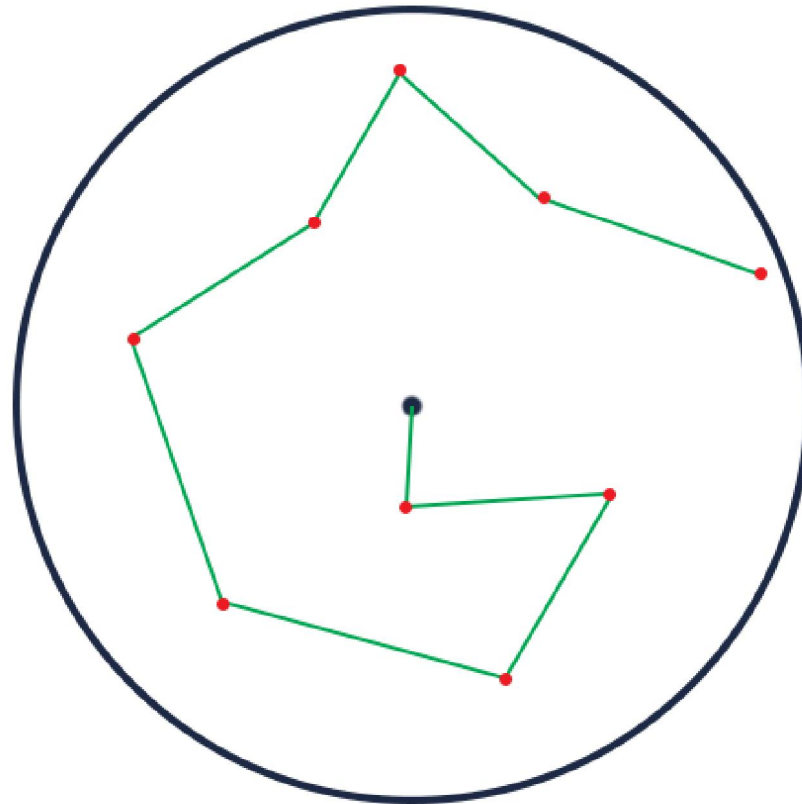
Comparaison avec la théorie



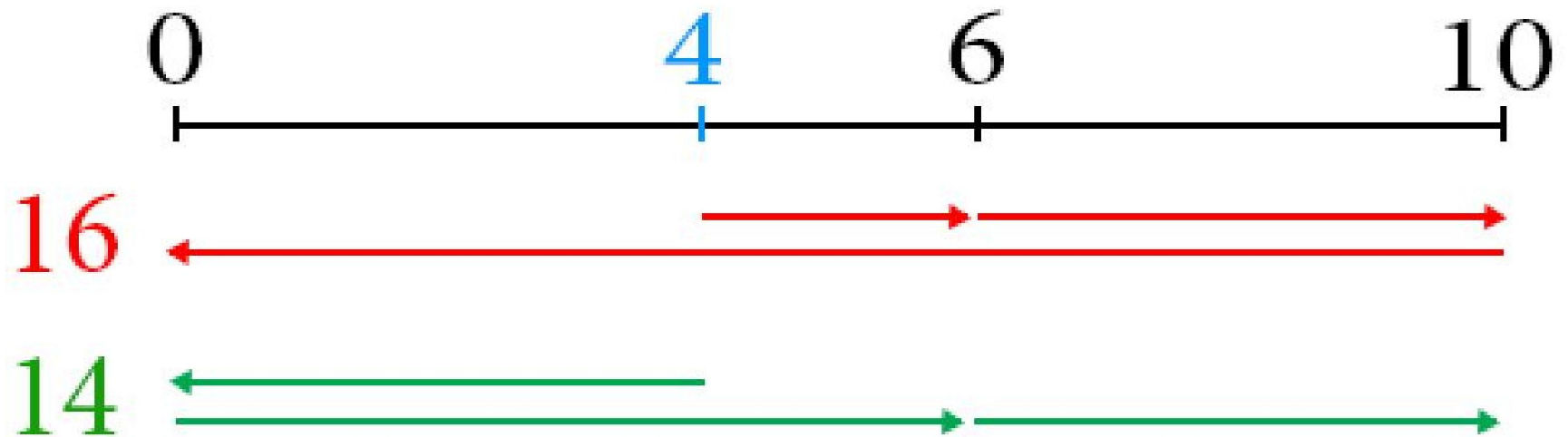
Bilan du tri selon l'angle

- Points positifs
 - Complexité en $O(n \cdot \ln(n))$
- Points négatifs
 - Trajet proportionnel à n

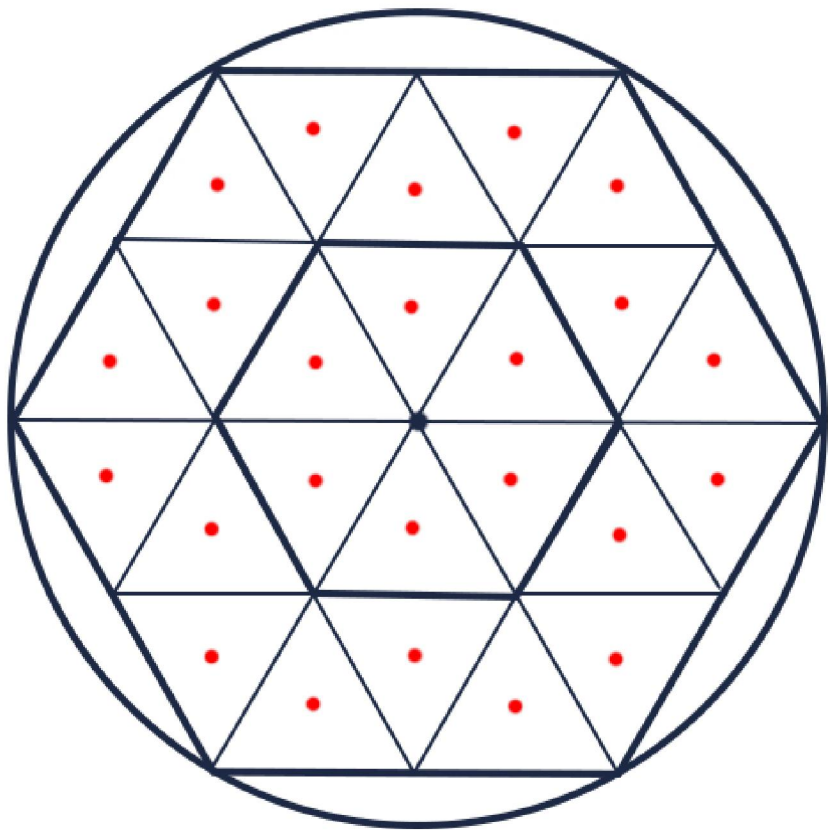
Méthode de parcours de proche en proche



Proche en proche imparfait



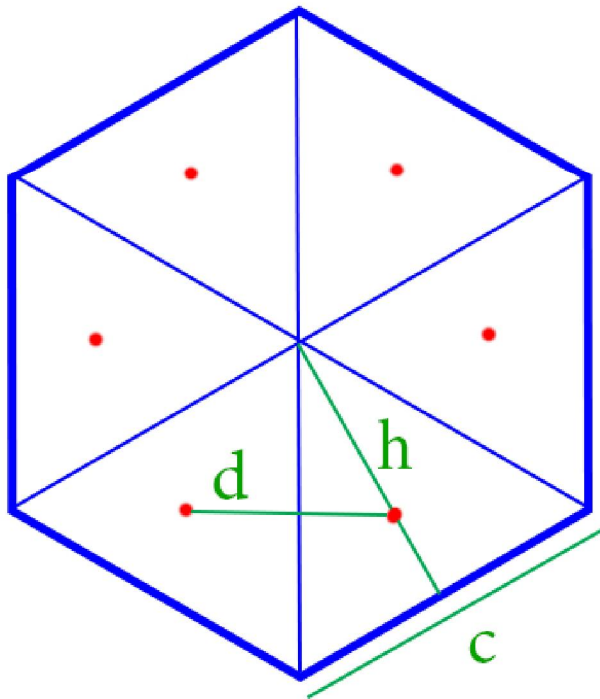
Etude théorique du proche en proche



$$d \approx \frac{1.555 \times R}{\sqrt{n}}$$

$$D \approx 1.555 \times R \times \sqrt{n}$$

Démonstration du proche en proche



$$h = \frac{\sqrt{3}}{2} \times c$$

$$A = \frac{\pi R^2}{n} = \frac{ch}{2} = \frac{\sqrt{3}}{4} \times c^2$$

$$c = 2 \sqrt{\frac{\pi}{n\sqrt{3}}} \times R$$

$$d = \frac{2}{3} \times h = \frac{c}{\sqrt{3}} = 2 \times 3^{-\frac{3}{4}} \times \sqrt{\frac{\pi}{n}} \times R$$

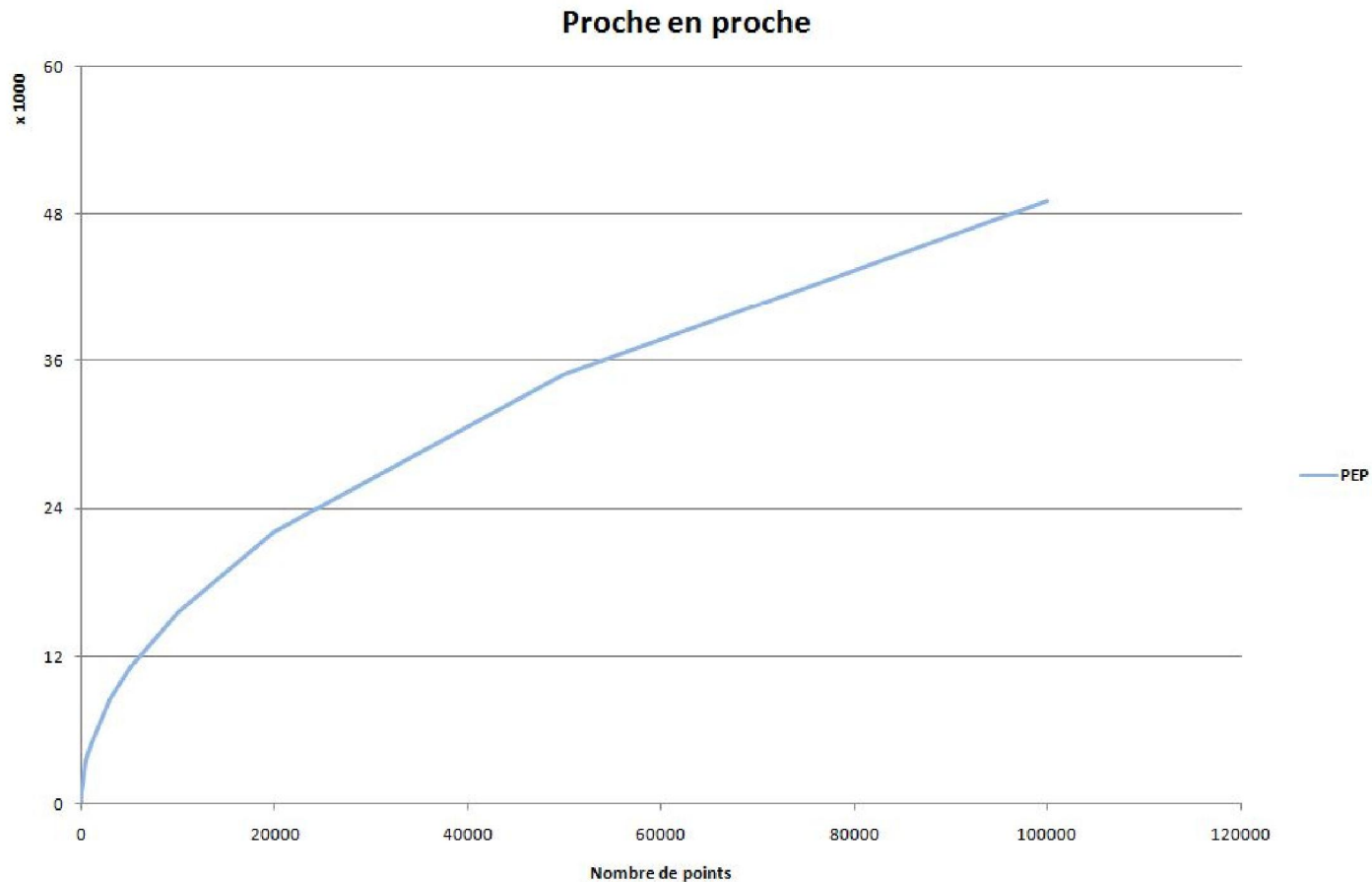
$$D = nd = 2 \times 3^{-\frac{3}{4}} \times \sqrt{\pi n} \times R$$

$$D \approx 1.555 \times R \times \sqrt{n}$$

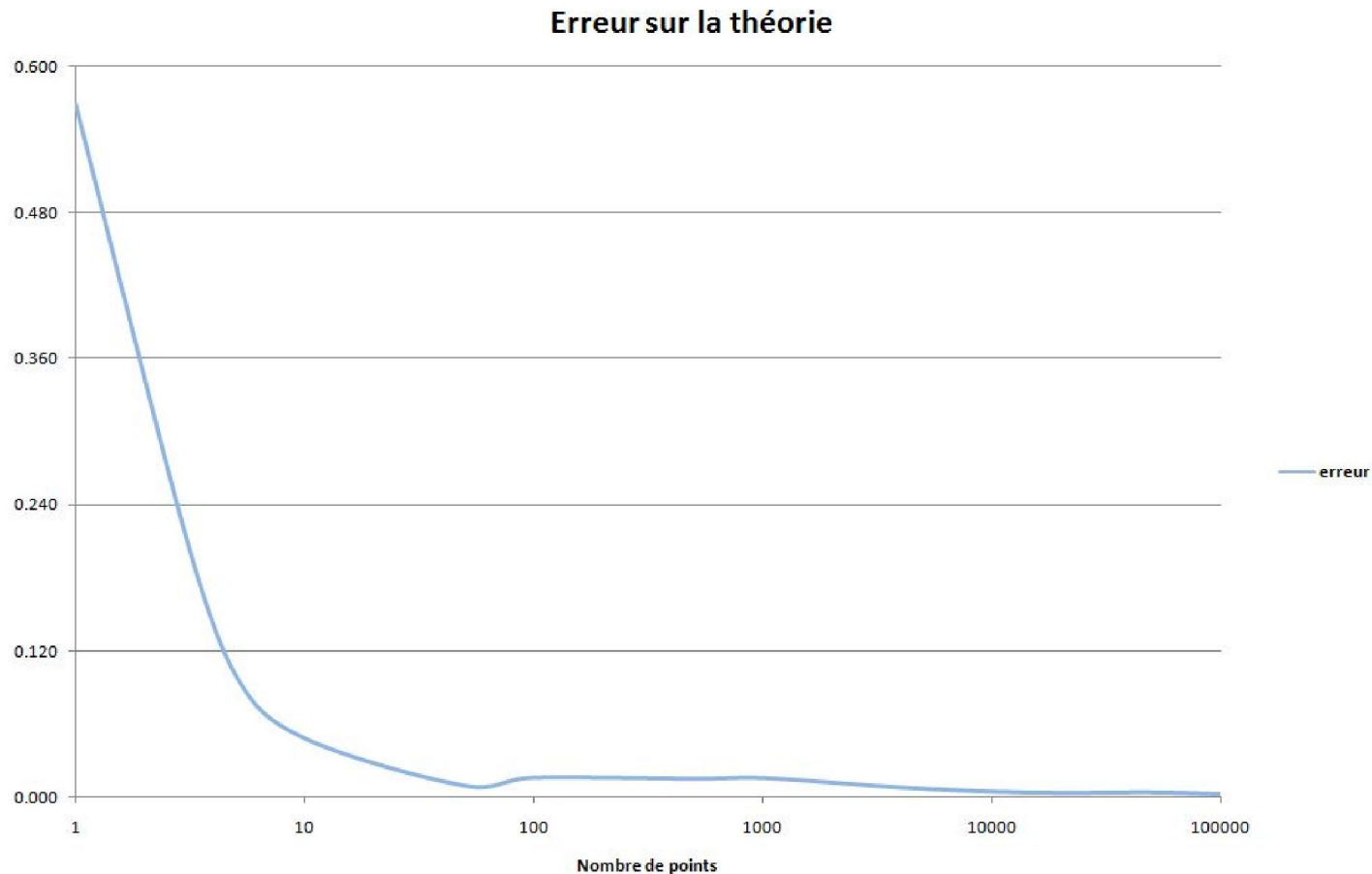
Proche en proche

```
def procheenproche(P):  
    """  
    Détermine la longueur du parcours de proche en proche  
    """  
    d=0  
    dernier=[0,0]  
    while len(P[0])>= 2:  
        bon=0  
        m=sqrt((P[0][0]*sin(P[1][0])-dernier[0]*sin(dernier[1]))**2+(P[0][0]*cos(P[1][0])-dernier[0]*cos(dernier[1]))**2)  
        for i in range(0,len(P[0])):  
            k=sqrt((P[0][i]*sin(P[1][i])-dernier[0]*sin(dernier[1]))**2+(P[0][i]*cos(P[1][i])-dernier[0]*cos(dernier[1]))**2)  
            if k<m:  
                m=k  
                bon=i  
        dernier=[P[0][bon],P[1][bon]]  
        P[1].remove(P[1][bon])  
        P[0].remove(P[0][bon])  
        d=d+m  
    d=sqrt((P[0][0]*sin(P[1][0])-dernier[0]*sin(dernier[1]))**2+(P[0][0]*cos(P[1][0])-dernier[0]*cos(dernier[1]))**2)  
    return (d)
```

Résultats expérimentaux



Comparaison avec la théorie



Bilan du parcours de proche en proche

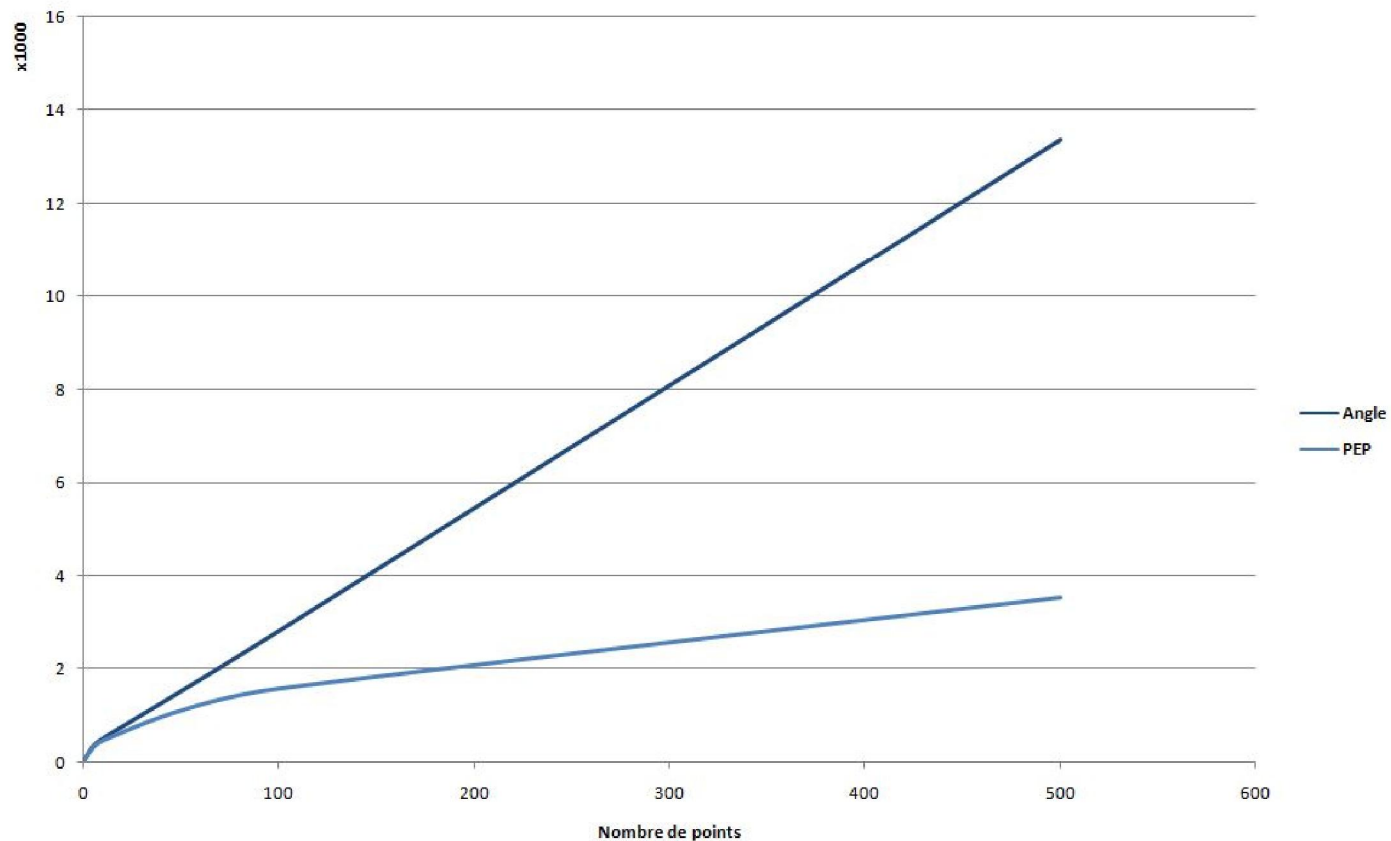
- Points positifs
 - Trajet proportionnel à $\sqrt{n}$
- Points négatifs
 - Complexité en $O(n^2)$

Comparaison des deux méthodes

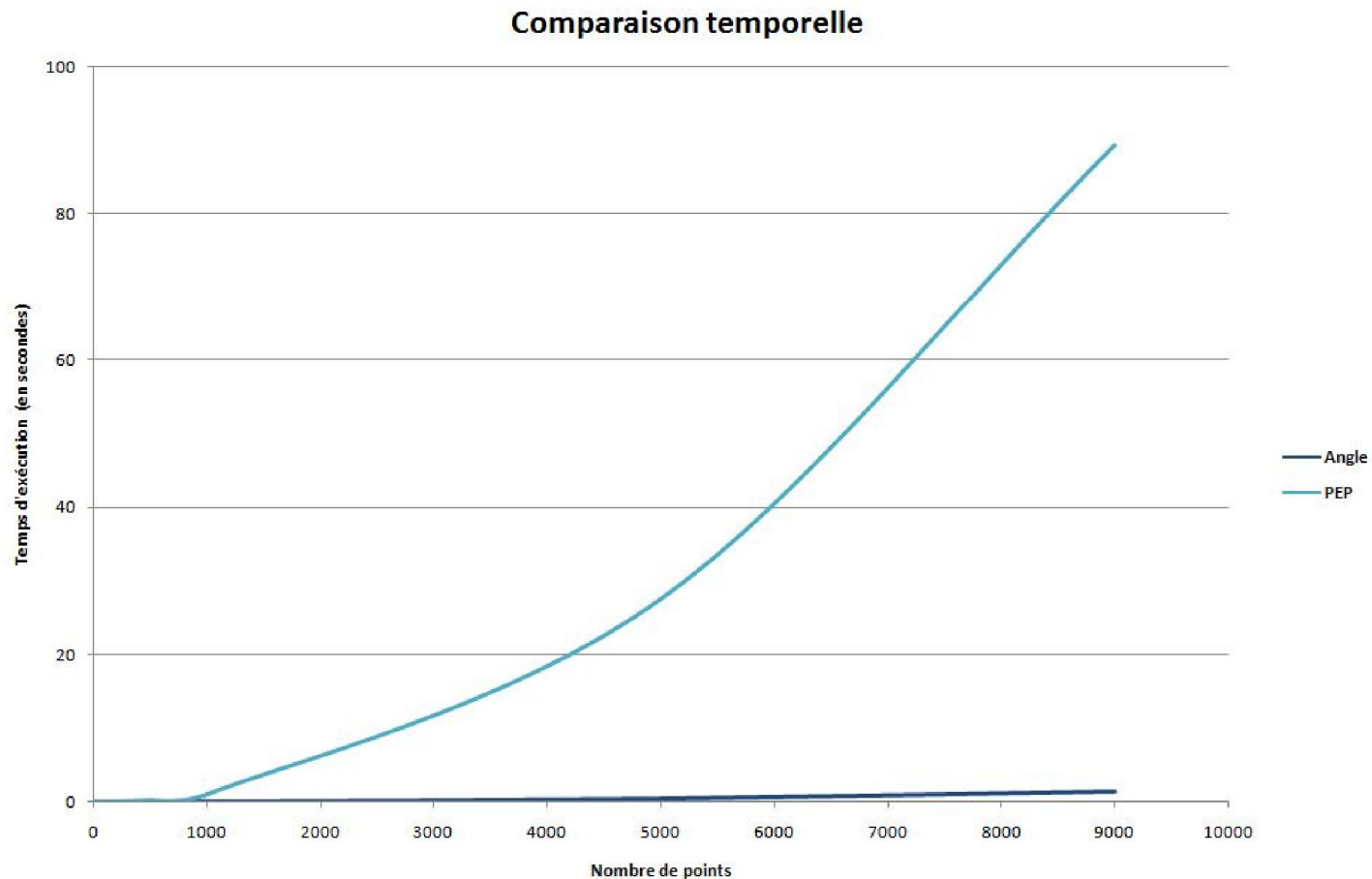
	Tri selon l'angle	Parcours de proche en proche
Proportionnalité de la longueur de parcours	n	$\sqrt{n}$
Complexité temporelle	$O(n \cdot \ln n)$	$O(n^2)$

Longueur de parcours

Comparaison des deux méthodes

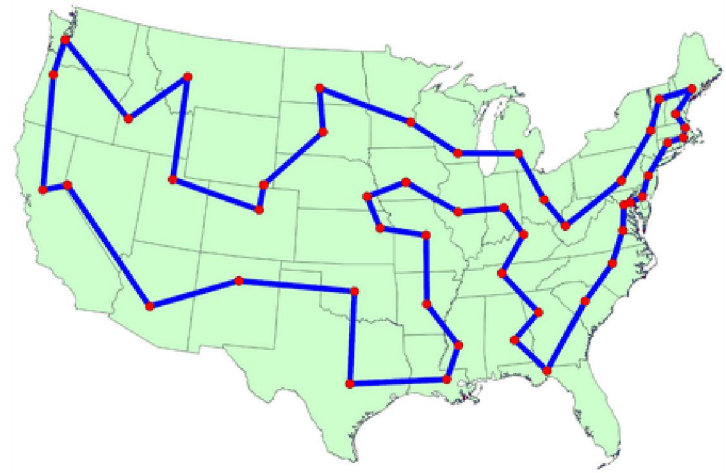
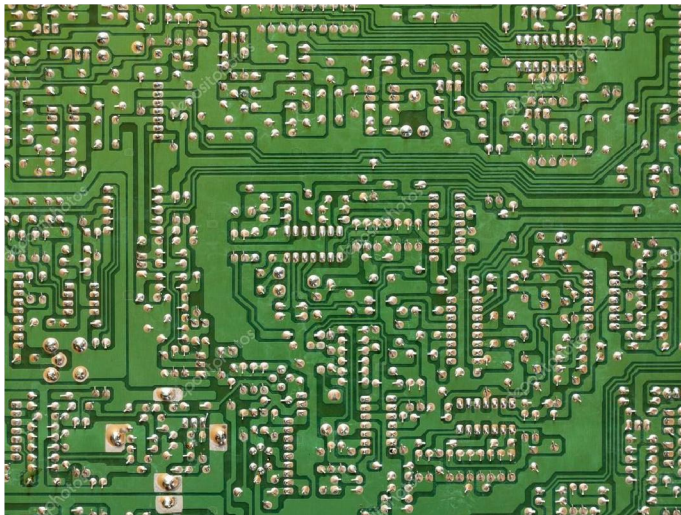


Temps d'exécution



Conclusion

- Qualité de la solution ou temps d'exécution ?



Annexe : parcours parfait naïf

```
def permutation(L):  
    """  
    Génère toutes les permutations possibles d'une liste de points  
    """  
    if len(L) == 0:  
        return [[]]  
    else:  
        return [[x] + ys for x in L for ys in permutation(delete(L, x))]  
  
def delete(L, item):  
    lc = L[:]  
    lc.remove(item)  
    return lc  
  
def parfait(P):  
    """  
    Retourne la longueur du chemin de longueur minimale  
    Prendre au maximum 10 points  
    """  
    A=permutation(P)  
    L=[]  
    for k in range (0,len(A)):  
        L.append(couples_to_liste(A[k]))  
    mini=distance(L[0])  
    for k in range(1,len(L)):  
        test=distance(L[k])  
        if test<mini:  
            mini=test  
    return(mini)
```