teaching sciences for innovation

Proposition de corrigé

Concours : Concours Centrale-Supélec

Année : 2017

Filière : MP - PC - PSI - TSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

https://www.upsti.fr/espace-etudiants/annales-de-concours

A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

Licence et Copyright

Informez-vous !

Retrouvez plus d'information sur les Sciences de l'Ingénieur, l'orientation, les Grandes Ecoles ainsi que sur les Olympiades de Sciences de l'Ingénieur et sur les Sciences de l'Ingénieur au Féminin sur notre site : www.upsti.fr

L'équipe UPSTI

Élément de corrigé - Informatique - Concours Centrale Supélec 2017

I Création d'une exploration et gestion des points d'intérêt

I.A Génération d'une exploration d'essai

I.A.1) Choix du points au hasard

```
def generer_PI(n, cmax):
    tab = []
    k = 0
    while k < n:
        x = random.randrange(0, cmax+1)
        y = random.randrange(0, cmax+1)
        if [x, y] not in tab:
            tab.append([x, y])
            k += 1
        PI = np.array(tab)
    return PI
```

I.A.2) Choix du points au hasard

On suppose que la zone d'étude considérée est représentée par une surface carrée de longueur $cmax + 1$. Pour avoir un résultat, il faut donc qu'il y ait au moins n points différents dans cette zone.

On peut en déduire que $(cmax + 1)^2 \geq n$.

I.A.3) Calcul des distances

```
def calculer_distances(PI):
    posR = np.array([position_robot()])
    piR = np.concatenate((PI, posR))
    n = len(piR)
    dist = np.zeros((n, n))
    for i in range(n):
        for j in range(i): # Matrice symétrique
            dist[i, j] = sqrt((piR[i,0] - piR[j,0])**2 + (piR[i,1] - piR[j,1])**2)
            dist[j, i] = dist[i, j]
    return dist
```

I.B Traitement d'image

I.B.1) Analyse d'une image

```
def F1(photo):
        n = photo.min()
        b = photo.max()
        h = np.zeros(b-n+1, np.int64)
        for p in photo.flat :
                h[p-n] += 1
        return h
```

La fonction F1 calcule l'histogramme d'une photographie c'est à dire le nombre de pixels de chaque intensité comprise entre l'intensité minimale **n** et maximale **b** de la photo considérée.

La fonction détermine dans un premier temps les valeurs minimale **n** et maximale d'intensité **b** de la photo avec les méthodes .min() et .max().

On initialise ensuite un vecteur de (**b-n+1**) zéro codé par des entiers sur 64 bits.

Pour finir, la fonction parcourt toute la photo et pour chaque pixel d'intensité p, elle incrémente de 1 la valeur comprise à l'indice $p-n$ de la liste h, ainsi h[i] contient le nombre de pixels d'intensité i+n.

I.B.2) Sélection de points d'intérêts

```
def selectionner_PI(photo, imin, imax):
    h, l = photo.shape
    PI = []
    for x in range(h):
        for y in range(l):
            if imin <= photo[x, y] <= imax:
                PI.append([x, y])
    return np.array(PI)
```

I.C Base de données

I.C.1) Numéro d'exploration

```
SELECT ex_num
FROM explo
WHERE ex_deb IS NOT NULL AND ex_fin IS NULL
```

I.C.2) Liste des points d'intérêts

On appelle **n** le numéro de l'exploration considérée.

```
SELECT pi_num, pi_x, pi_y
FROM pi
WHERE ex_num = n
```

I.C.3) Surface

```
SELECT ex_num, (MAX(pi_x) - MIN(pi_x)) * (MAX(pi_y) - MIN(pi_y)) * 1e-6
FROM explo JOIN pi ON (ex_num) WHERE ex_fin IS NOT NULL
GROUP BY ex_num
```

I.C.4) Surface maximale

Les coordonnées sont stockées dans des variables de type SQL integer qui représentent une distance en millimètres. Si n est le nombre de bits utilisés par le SGBD pour représenter le type entier, la largeur maximale de la zone est de $2^{n-1}-1$ millimètres et sa surface en kilomètres carrés de $(2^{n-1}-1)^2 * 10^{-12}$. Pour $n = 32$, nous obtenons $4.6 * 10^6$ km^2 à rapprocher des 44 km parcourus par Opportunity.

I.C.5) Utilisation instrument

```
SELECT
   in_nom,
   COUNT(*),
```

```
   SUM(it_dur)
FROM
   explo AS E JOIN
   analy AS A ON E.ex_num = A.ex_num JOIN
   intyp AS I ON A.ty_num = I.ty_num
WHERE
   ex_deb IS NOT NULL AND
   ex_fin IS NULL
GROUP BY
   in_nom
```

II Planification d'une exploration : première approche

II.A Quelques fonctions utilitaires

II.A.1) Longueur d'un chemin

```
def longueur_chemin(chemin, d):
    longueur = d[-1, chemin[0]]
    for k in range(len(chemin) - 1):
        longueur += d[chemin[k], chemin[k+1]]
    return longueur
```

II.A.2) Normalisation d'un chemin

```
def normaliser_chemin(chemin, n):
    res = []
    for p in chemin:
        if 0 <= p < n and p not in res:
            res.append(p)
    for i in range(n):
        if i not in res:
            res.append(i)
    return res
```

II.B Force brute

II.B.1)

Il y a n choix possibles pour le premier point du chemin, $n-1$ pour le deuxième, etc. Au total nous aurons donc $n!$ possibilités.

II.B.2)

Avec 20 point d'intérêts nous pouvons obtenir $20!$ (2^{18}) chemins différents.

Une fois que nous avons obtenu tous ces chemins, nous allons calculer la longueur de chaque chemin.

Si l'on suppose que le temps d'une opération est de l'ordre de la nanoseconde, il faut donc 2^9 secondes, soit plus de 80 ans pour trouver le plus court chemin !

Il n'est véritablement pas envisageable d'utiliser un tel algorithme pour ce problème.

II.C Algorithme du plus proche voisin

II.C.1) Fonction *plus_proche_voisin*

```
def plus_proche_voisin(d):
    n = len(d) - 1
    dMax = d.max()
    res = [n]
    for p in range(n):
        dMin = dMax
        dist = d[res[-1]]
        for j in range(n):
            if j not in res:
                if dist[j] <= dMin:
                    dMin = dist[j]
                    jMin = j
        res.append(jMin)
    return res[1:]
```

II.C.2)

Pour calculer la complexité de notre algorithme nous allons détailler les différentes étapes.

- initialisations (calculer_distances) : $\mathbf{O(n)}$;
- boucle externe effectuée n fois ;
- boucle interne effectuée n fois ;
- corps de boucle interne en $\mathbf{O(n)}$ (j not in res)

=> $\mathbf{O(n^3)}$

II.C.3)

Les PI proposés sont alignés, considérons donc le problème en dimension 1 en cherchant une position initiale du robot sur la ligne reliant les points. En remarquant que l'espacement entre les points va croissant, si on place le robot entre les points 0 et 1 plus près du point 1, l'algorithme du plus proche voisin va l'obliger à un aller-retour inutile.

Plaçons, par exemple, le robot en (0, 2000).

Algorithme du plus proche voisin : (0, 2000) -> (0, 3000) -> (0, 0) -> (0, 7000) : longueur 11000

Plus court chemin : (0, 2000) -> (0, 0) -> (0, 3000) -> (0, 7000) : longueur 9000

III Deuxième approche : algorithme génétique

III.A Initialisation et évaluation

III.A.1) Fonction *creer_population*

```
def creer_population(m, d):
    n = len(d) - 1
    popu = []
    for i in range(m):
        chemin = random.sample(range(n), n)
        popu.append((longueur_chemin(chemin, d), chemin))
    return popu
```

III.B Sélection

III.B.1) Fonction *reduire*

```
def reduire(p):
    p.sort()
    m = len(p)
    m = math.ceil(m/2)
```

```
    del p[m:]
```

III.C Mutation

III.C.1) Fonction *muter_chemin*

```
def muter_chemin(c):
    n = len(c)
        i , j = random.sample(range(n),2)
    c[i], c[j] = c[j], c[i]
```

III.C.2) Fonction *muter_une_population*

```
def muter_population(p, proba, d):
    n = len(p)
    for i in range(n):
        if random.random() <= proba:
            longueur, chemin = p[i]
            muter_chemin(chemin)
            p[i] = longueur_chemin(chemin, d), chemin
```

III.D Croisement

III.D.1) Fonction *croiser*

```
def croiser(c1, c2):
    n = len(c1)
    return normaliser_chemin(c1[:n // 2] + c2[n // 2:], n)
```

III.D.2) Fonction *nouvelle_generation*

```
def nouvelle_generation(p, d):
    m = len(p)
    for i in range(m):
        l1, c1 = p[i]
        l2, c2 = p[(i+1) % m] # Permet de récupérer le terme suivant i mais le premier élément lorsqu
        c = croiser(c1, c2) # i vaut la longueur de la liste -1
        p.append((longueur_chemin(c, d), c))
```

III.E Algorithme complet

III.E.1) Fonction *algo_genetique*

```
def algo_genetique(Pl, m, proba, g):
    d = calculer_distances(Pl)
    p = creer_population(m, d)
    for i in range(g):
        reduire(p)
        nouvelle_generation(p, d)
        muter_population(p, proba, d)
    return min(p)
```

III.E.2) Dégradation

Il est possible que la fonction muter_population fasse muter le meilleur individu vers un chemin plus long. De ce fait, rien n'assure avec l'implantation proposée que la génération n+1 ne soit pas plus mauvaise que la génération n.

Pour éviter cela, il suffit de s'assurer de ne pas muter le meilleur individu. n pourrait alors comparer la population muter avec la population précédente. Si la longueur du chemin est plus grande alors on récupère la population précédente.

III.E.3) Condition d'arrêt

En plus du nombre de génération, on peut imaginer d'utiliser :
- le temps écoulé
- un critère de convergence
- un mélange (convergence + temps maxi)

Par rapport au nombre de génération, le temps écoulé présente l'intérêt de pouvoir s'ajuster plus facilement aux paramètres opérationnels du robot.

Un critère de convergence n'est pas évident à mettre en place. Si on ne considère que le meilleur individu, on risque de s'arrêter sur un optimum local. D'autre part, ce type d'algorithme progresse généralement par saut, il faut donc éviter de prendre un plateau pour le résultat final. On peut par exemple considérer que s'il n'observe pas d'amélioration au bout d'un certain nombre de générations successives, le programme s'arrête.