Comment sauver Tatouine ?

1. Quelle est la taille de l'espace de recherche (utiliser une notation scientifique)?

On a le système d'équations suivant :

$$\begin{cases} x(t) = p_1 \times \sin(p_2 \times t + p_3) \\ y(t) = p_4 \times \sin(p_5 \times t + p_6) \end{cases}$$

Où p_i , $i \in [1; 6]$ sont les paramètres du système.

On sait que ces paramètres sont à valeurs dans $[-100; 100]$.

On a donc : taille = $(Card([-100; 100]))^n$, où n nombre de paramètres.

Si on considère l'espace de recherche comme continu alors sa taille sera infinie.

On le considère donc comme discret.

Sur l'espace entier entre $[-100; 100]$ on a : $(Card([[-100; 100]]))^6 = 200^6$

Pour chaque pas décimal ajouter il faut multiplier le $Card([-100; 100])$ par 10.

Pour 3 nombres après la virgule on a : $(Card([-100; 100]) \times 10^3)^6 = (200 \times 10^3)^6$

La taille de l'espace de recherche est donc de $(200 \times 10^3)^6$ soit environ 64 000 milliards de milliards de milliards de possibilités.

2. Quelle est votre fonction fitness?

La fonction Fitness utilisée somme les distances de Manhattan entre les valeurs réelles et les valeurs approchées pour les x et les y confondus.

```python
def Fitness(individual, known_values):
    score = 0
    for t, real_values in known_values.items():
        x_estimate = individual[0] * sin(individual[1]*t + individual[2])
        y_estimate = individual[3] * sin(individual[4]*t + individual[5])
        x_real = real_values[0]
        y_real = real_values[1]
        score += abs(x_real - x_estimate) + abs(y_real - y_estimate)
    return score
```

3. Décrivez les opérateurs mis en œuvre (mutation, croissement)?

Les deux opérateurs mis en œuvre sont la fonction XOSubject() qui assure le croisement entre deux individus et la fonction MutateSubject() qui fait muter un individu.

- La fonction de croisement XOSubject() :
  Elle assure un croisement entre deux individus pris en entrée. Elle existe en deux alternatives, un croisement simple qui va ajouter la moitié d'un premier individu et la moitié du deuxième, et un croisement double qui va ajouter le premier tier d'un individu, le deuxième tier du second et enfin le dernier tier du premier.
  Elle est dotée d'un paramètre strat qui permet soit de sélectionner le croisement simple : '1XO', soit le double : '2XO', soit d'utiliser les deux : 'Mix'.
  Afin de randomiser la fonction, il y a également tirage au sort de l'individu qui sera premier et de celui qui sera second.

- La fonction de mutation MutateSubject() :
  Elle assure la mutation d'un individu pris en entrée.
  Elle tire au hasard deux pivots dans la liste de paramètres de l'individu.
  Elle existe en trois alternatives : une mutation par insertion du second pivot après le premier, une mutation par inversion des deux pivots, une mutation par inversion de la sous-liste de paramètres entre les deux pivots inclus, une mutation par insertion sur le premier pivot d'une valeur aléatoire.
  Elle est dotée d'un paramètre strat qui permet soit de sélectionner la mutation par insertion : 'Insertion', soit la mutation par inversion :

‘Swap’, soit la mutation par inversion de la sous-liste : ‘Reversion’, soit l’insertion d’une nouvelle valeur au hasard : ‘Random’, ou bien d’utiliser les quatre : ‘Mix’.

4. Décrivez votre processus de sélection.

La fonction de sélection opère selon le principe d’une roue biaisée :
On divise la population en 5 parts : la première contient les 10% des meilleurs individus, la deuxième les 20% suivants, la troisième les 30% comme la quatrième et la cinquième les 10 moins bons %. On associe à chacune une probabilité différente d’être tiré au sort : 30% pour la première catégorie, 25% pour la deuxième, 20% pour la troisième, 15% pour la quatrième et 10% pour la dernière.
Une fois la catégorie tirée selon ces pourcentage de probabilités, on tire un individu de la catégorie de manière équiprobable au sein de celle-ci.
On tire ensuite la méthode de transmission des caractères là encore de manière biaisée : 2% de transmettre directement ses caractéristiques sans altération, 60% de le faire avec croisement avec un autre individu tiré au hasard dans la même catégorie, 38% de le faire avec une mutation.
On réitère l’ensemble des opérations autant de fois qu’il y a d’individus dans la population.

5. Quel est la taille de votre population, combien de générations sont nécessaires avant de converger vers une solution stable ?

La taille de population initiale est de 10000 individus, en général l’algorithme converge vers une solution après 150 générations

6. Combien de temps votre programme prend en moyenne (sur plusieurs runs) ?

La création d’une génération prend en moyenne 1,1s, en le faisant tourner plusieurs fois sur 200 générations on obtient une moyenne de 3min41s

7. Si vous avez testé différentes solutions qui ont moins bien fonctionnées, décrivez-les et discutez-les.

Pour le processus de sélection, j'ai testé deux autres méthodes qui donnaient des résultats moins satisfaisants :

- Une méthode de « filtre linéaire » : qui faisait survivre systématiquement les meilleurs 10% des individus, faisait se croiser les 70% suivants et muter les derniers 20%. Cette méthode ne permettait pas de converger vers une solution, et la valeur initiale de la meilleur fitness était trop peu modifiée. Surement car elle induisait un élitisme trop fort, sans introduire assez d'aléatoire pour pouvoir renverser cette élite.
- Une méthode « Elitiste » : qui ne faisait survivre et se reproduire que les meilleurs 10% des individus. Cette méthode était meilleure que la première mais là encore elle ne permettait pas de diminuer de manière satisfaisante la fitness du meilleur individu. En plus de son élitisme voir de son eugénisme, son travers était probablement un manque de diversité des paramètres des parents qui empêchait l'émergence d'assez d'aléatoire pour modifier cette élite.

J'ai donc finalement adopté la méthode de la roue biaisée.

Au niveau des croisements et des mutations, j'ai adopté les systèmes mixtes sur ces deux opérateurs pour augmenter la diversité des paramètres testés, cela a induit une convergence moins rapide mais plus efficace que des méthodes de croisements et de mutations uniformes.

J'ai également modifié les différents paramètres qui régisse le nombre d'individu qui passe d'une génération à l'autre sans changement, le nombre d'individus issus de croisement et le nombre d'individus issus de mutations.

Meilleure solution :

Fitness = 20,58 soit 0,343 lorsqu'on la normalise par le nombre d'opérations d'additions.

Points : -13.693;-21.113;-20.862;21.831;41.102;22.874