Evolution artificielle et robotique

Fonction fitness, algorithme (1+1)-ES, robotique évolutionniste

Mise à jour : Mars 2018

Rendez-vous sur la page http://pages.isir.upmc.fr/~bredeche/ - onglet 3i025 pour télécharger le code source Python nécessaire pour faire les exercices ci-dessous (dépendances: Pygame, Matplotlib), ainsi que le script permettant de tracer vos résultats. Vous travaillerez à partir d'une copie du fichier *robot_randomsearch.py*

Le fichier *robot_randomsearch.py* contient une implémentation d'un algorithme d'optimisation naïf: la recherche aléatoire. L'environnement contient un unique robot dont la fonction de contrôle est une combinaison linéaire des entrées sensorielles avec des paramètres générés au hasard. La fonction d'évaluation favorise la distance parcourue. Lisez attentivement les commentaires au début du fichier avant de commencer.

Prise en main

1. Etudier le code fourni. Modifier le budget de la recherche aléatoire (variable *maxEvaluations*) et fixez-le à 100 évaluations. Exécutez le code et observez le comportement obtenu.

2. Modifiez la sortie console pour afficher à chaque évaluation sous format CSV (i.e. valeurs séparées par des virgules) les éléments suivants: numéro d'évaluation, fitness actuelle, meilleure fitness depuis le début. Vous sauvegarderez (par copié-collé) ces résultats dans un fichier. Vous utiliserez la fonction *updateFitness3(.)* pour calculer la fitness. Exemple de sortie console:

   0, 22555.3660952, 22555.3660952
   1, 199.618627164, 22555.3660952
   2, 80.8005718674, 22555.3660952
   3, 25642.2593266, 25642.2593266
   etc.

3. En utilisant le script python contenu dans le paquet *multiplotCSV.zip*, tracez l'évolution de la fitness au cours du temps. De même, tracez un second graphe suivant l'évolution de la meilleure fitness (i.e. la courbe ne redescend jamais).

   Syntaxe: python *plotCSV.py nomdufichier.csv <index_abscisse> <index_ordonnée>*

4. En utilisant le script python contenu dans le paquet plotCSV.zip, tracez l'évolution de la fitness au cours du temps en compilant les résultats de plusieurs runs. De même, tracez un second graphe suivant l'évolution de la meilleure fitness (i.e. la courbe ne redescend jamais).

   Syntaxe: python *plotCSV.py repertoire/*.csv <index_abscisse> <index_ordonnée>*
   Remarque: <repertoire> est optionnel, mais pratique.

5. Fixez maintenant le budget à 500 évaluations. Exécutez le code et observez le résultat. Tracez les courbes d'évolution de la fitness actuelle et de la meilleure fitness pour un seul run (cf. question 3)

Fonction de contrôle

La fonction de contrôle se trouve dans la méthode stepController(.). On utilisera un budget de 100 évaluations pour l'instant.

1. Modifiez le contrôleur pour prendre en compte les senseurs à +/-80° (en plus des 4 déjà pris en compte). Vous devrez aussi adapter le reste du code (vous avez besoin de quatre paramètres supplémentaires). Comparez vos résultats à l'expérience de contrôle réalisée à l'exercice précédent. Les comparaisons doivent toujours être faite en utilisant plusieurs runs de chaque algorithme (cf. exercice 1, question 4)

Algorithme d'optimisation sur des valeurs discrètes

1. Implémentez un algorithme génétique comme vu en cours, afin d'optimiser un génome contenant des valeurs discrètes: chaque gène a pour valeur soit -1, 0 ou +1. Chaque gène codera pour un paramètre de la fonction de contrôle (i.e. un lien est inhibiteur (-1), excitateur (+1) ou neutre (0)).

   - taille de population: 10
   - mutation: 1.0 / nb_de_gènes
   - sélection: tournoi avec k=8
   - budget total d'évaluation: 500 évaluations (ie. 50 générations pour une taille de population de 10)

2. Comparez vos résultats (en traçant les résultats sur une figure) à l'expérience de contrôle réalisée à l'exercice précédent. La fonction fitness à utiliser est dans *updateFitness3(.)*. Les comparaisons doivent toujours être faite en utilisant plusieurs runs de chaque algorithme (cf. exercice 1, question 4)

3. Faites varier les paramètres de l'algorithme ci-dessous pour améliorer les résultats (vitesse et qualité).

Fonction d'évaluation de la performance

La fonction d'évaluation de performance, ici appelée fonction *fitness*, est implémentée dans la méthode updateFitness(.). Vous trouverez dans le code trois exemples de fonctions fitness.

1. Modifiez le code pour compter le nombre de cases visitées par un robot (Attention: il ne s'agit pas d'une mesure de performance, en tout cas pour l'instant). Vous ajouterez cette information dans les données affichées dans la console en plus de celles déjà présentes. C'est-à-dire, étant donné la fonction fitness précédemment testée: *numéro d'évaluation, fitness actuelle, meilleure fitness depuis le début, score en nombre de cases visitées, score en nombre de cases visité obtenu par l'individu ayant obtenu la meilleure fitness* (i.e. ce n'est pas forcément le meilleur score en terme de nombre de cases visitées). Exemple:

   0, 22555.3660952, 22555.3660952, 54, 54
   1, 199.618627164, 22555.3660952, 43, 54
   2, 80.8005718674, 22555.3660952, 11, 54
   3, 25642.2593266, 25642.2593266, 84, 84
   4, 1642.45685512, 25642.2593266, 96, 84
   etc.

2. Tracez deux courbes, suivant respectivement l'évolution de la meilleure fitness et l'évolution du score en nombre de cases de l'individu ayant eu la meilleur fitness. Faîtes au moins 5 expériences et un budget de 100 évaluations.

3. Il y a plusieurs (trois) fonctions fitness déjà présentes dans le code. Comparer leur performance en prenant en compte le nombre de cases visitées, en faisant plusieurs (≥5) runs pour chaque fonction (et un budget d'évaluation de ≥100, à définir selon le temps qu'il vous reste).
4. Utilisez maintenant le score de nombre de cases visitées comme mesure de performance. Avec les mêmes paramètres d'expérience que précédemment (budget et répétitions), comparer le résultat avec ceux obtenus à la question précédente en prenant le nombre de cases visités comme élément de comparaison.

Algorithme d'optimisation sur des valeurs continues (optionnel)

1. Implémentez l'algorithme (1+1)-ES vu en cours, avec une valeur de *sigma* fixe que vous déterminerez (pour commencer, fixez une petite valeur, p.ex. $\leq 10^{-4}$). A noter que votre génome sera maintenant défini dans $\mathbb{R}^n$, ce qui permet un réglage plus fin des paramètres de contrôle.

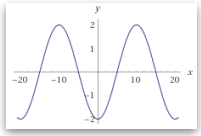
2. Comparez vos résultats (en traçant les résultats sur une figure) à ceux obtenus dans les exercices précédents. La fonction fitness à utiliser est dans *updateFitness3(.)*.

3. Implémentez la règle des 1/5e vue en cours pour la mise à jour du sigma. Comparez.

4. Clonez le fichier *robot_randomsearch.py*, et modifiez le nouveau fichier afin d'afficher le comportement d'un robot pour un génome. Le but est de pouvoir rejouer un génome déjà obtenu. Il s'agit là principalement de modifier la fonction main en fin de fichier pour ne garder (et modifier légèrement) que la partie permettant de tester le meilleur individu obtenu. Cette fonction vous sera très utile pour revoir les comportements obtenus et les tester sous différentes conditions. Il faudra aussi penser à sauver le meilleur génome de chaque run (p.ex. en l'affichant dans la console à la fin de l'optimisation).

Remarques:
- le script *plot.py* ignore les lignes commençant par un caractère dièse ("#").
- La fonction *gauss(0,1)* permet de tirer des nombres dans N(0,1). Rappel: $N(\mu,\sigma) = \mu + \sigma * N(0,1)$
- Vous devrez probablement borner les valeurs de vos paramètres afin qu'ils n'"explosent" pas.
  - Exemples de domaines de valeurs pour les paramètres:
    - valeurs de poids du contrôleur entre -10 et +10 (tirage au hasard à l'initialisation)
    - sigma entre 10^{-7} et 10^{-1} (valeurs exactes à définir)
  - Il n'est pas utile de borner les valeurs dans le génome. Il faut plutôt convertir la valeur lue avant son utilisation. Par exemple, pour borner l'interprétation d'une valeur dans [a,b], on peut utiliser la transformation suivante[1], en initialisant le génome dans [-10,+10] (cf. illustration dans la figure avec a=-2 et b=2):

$$a + \frac{1}{2}(b-a)\left(1 - \cos\left(\frac{\pi x}{10}\right)\right)$$



Si vous avez le temps...

1. Pour l'instant, chaque robot est évalué 4 fois, avec une condition initiale légèrement différente à chaque fois. Le but est d'obtenir des comportements généralistes, robustes à un changement dans l'environnement. Evaluez l'impact du nombre de réévaluations sur la performance. Vous utiliserez la fonction *updateFitness3(.)* et un nombre de réévaluations de 1, 4 ou 10. En plus de la comparaison entre les courbes obtenues, vous rejouerez 40 fois le meilleur individu obtenu par chacune des trois expériences pour compiler un score moyen. Comparez la performance et l'écart type.
2. Ecrivez une nouvelle fonction fitness avec comme objectif d'obtenir un meilleur résultat en terme de nombre de cases visitées.
3. Si vous avez le temps, re-faîtes toutes les expériences avec un budget de 1000 évaluations, et 11 réplications par expérience (plutôt que 5).
4. Modifiez le code pour mettre deux agents, dont l'un des deux est un wander (i.e. pas d'optimisation). Définissez une fonction fitness qui récompense la capacité à suivre un autre robot. Comparez le robot wander et le meilleur robot évolué sur le score défini pour le projet multirobot_teamwars. Les deux robots seront en compétition dans le même environnement.
5. Modifiez multirobot_teamwars.py pour pouvoir utiliser des contrôleurs obtenus par évolution (i.e. il s'agit de modifier uniquement la méthode stepController(.)). Testez votre équipe avec les meilleurs contrôleurs obtenus dans ce TME dans le cadre du projet.

Remarques par rapport au projet: *il est tout à fait possible d'entraîner, avec un (1+1)-ES (ou même avec une recherche aléatoire), des comportements dédiés à des situations particulières, puis de les organiser comme modules de comportements dans une architecture de contrôle (par exemple, subsomption ou arbres de comportements). Une architecture de contrôle peut ainsi très bien arbitrer entre des comportements écrits à la main et d'autres précédemment évolués.*

[1] http://cma.gforge.inria.fr/cmaes_sourcecode_page.html#practical