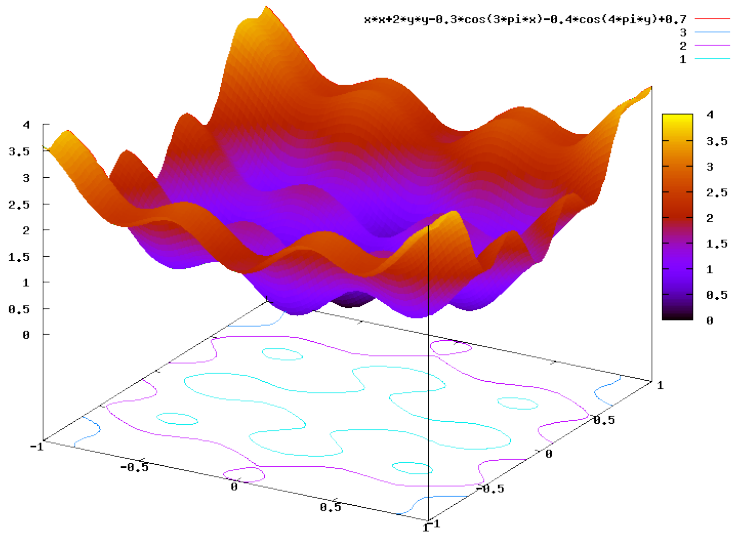
Projet

Les Algorithmes « Génétiques »
- Application au jeu
Ultra Master Mind (UMM) -

pfm-2023

Les algorithmes génétiques sont exploités pour rechercher des optimums dans des fonctions de coût complexes



Algorithmes génétiques

- S'inspirent de la théorie de l'évolution (Charles Darwin, XIX ème siècle).
  - Une population d'individus évolue grâce au mécanisme de la reproduction sexuée.
  - Les individus les plus adaptés à leur milieu se reproduisent plus que les autres (sélection naturelle).
  - Parfois, des mutations peuvent survenir, qui introduisent de la variété dans la population. On retrouve donc les notions suivantes :

Algorithmes génétiques

Principales notions :

Théorie de l'évolution	**Algorithmes génétiques**	**Problème du master mind**
Individu/chromosome	Une solution possible au problème	Une séquence
Population	L'ensemble des solutions étudiées	Les séquences à évaluer
Reproduction	Croisement de deux solutions pour en produire une nouvelle	Nouvelle séquence obtenue par combinaison de deux autres
Mutation	Modification aléatoire d'une solution	Permutation de certaines gènes dans une séquence
Sélection	Élimination des solutions les moins adaptées	Élimination des séquences les moins proches
Gène	Élément d'une solution	Un élément de la séquence

Principe de fonctionnement d'un algorithme génétique

- **Genèse** : création d'une population initiale
- **Évolution** au cours de plusieurs générations :
  - Évaluation des individus de la population
  - Sélection d'une partie de la population
  - Reproduction par croisement de certains individus
  - Mutation de certains individus
- **Sélection** du meilleur individu de la dernière génération

Pour en savoir plus

- Sur les algorithmes génétiques
  - http://fr.wikipedia.org/wiki/
  - http://www-igm.univ-mlv.fr/~dr/XPOSE2013/tleroux_genetic_algorithm/fonctionnement.html
- Exemple AG appliqués au PVC
  - http://www.rennard.org/iva/agapll.html

Projet

- Partie I : Ultra Master Mind
- Partie II : Ultra Master Mind++

Problème de “l’Ultra” Master Mind

- On considère une chaîne de caractères de longueur **L** quelconque (phrase mystère)
- Les caractères sont des codes ASCII codés sur un octet de **0** à **255** (alphabet de **256** caractères)
- Le jeu consiste à découvrir la phrase mystère. Le joueur soumet des phrases de longueur **L**, et le système répond en indiquant simplement le nombre de caractères en correspondance (*match*) et le nombre de caractères mal placés (*miss placed*).
- N*ote : pour **L=100**, la phrase mystère est cachée dans une botte de foin de **256**100** soit environ **2**800 ~ 10**80** soit environ le nombre d’atomes dans l’univers*

Problème de “l’Ultra” Master Mind

- On souhaite résoudre ce problème en exploitant un algorithme génétique.
- Pour cet algorithme, les chromosomes sont des chaînes de caractères de taille **L**
- Il s’agit donc d’implanter les fonction de fitness, de genèse, de sélection, de croisement et de mutation et d’orchestrer la dynamique de population (produire les générations successives)

Problème de “l’Ultra” Master Mind

- On utilisera le package ***random*** de python pour générer les tirages aléatoires requis (en particulier la fonction ***randint***)
- On introduira les paramètres suivants :
  - **NG** : le nombre de générations
  - **L** : longueur des chromosomes (et de la phrase mystère)
  - **N** : la taille de la population (nombre d’individus)
  - **TS** : le taux de sélection (ou de reproduction)
  - **TM** : le taux de mutation

Problème de “l’Ultra” Master Mind

- Fonction ***Fitness(C, PM)*** : **C** est un chromosome, **PM** est la phrase mystère.
- (cf. diapo suivante)
- Fonction de sélection : tous les individus de la population possèdent un chromosome. Les individus sont classés selon la valeur fournie par la fonction Fitness. Les **TS x N** meilleurs individus sont sélectionnés pour la reproduction. Les autres sont supprimés, ils seront remplacés par les descendants créés par reproduction.
- Fonction reproduction : deux individus **P** et **M** sont tirés au hasard parmis les individus sélectionnés. On tire au hasard un point de **cut** entre **L/3** et **2L/3**. On constitue un nouveau chromosome (individus) CM de la manière suivante
- **CM = concaténation(P[:cut], M[cut:])**
- Le nouvel individus est ajouté à la population : on itère la procédure de manière à retrouver une population de **N** individus.

Problème de “l’Ultra” Master Mind

- Fonction ***Fitness(C, PM)***
- On testera les fonctions suivantes :
  - ***Fitness(C, PM) = -Σ(\|C[i]-PM[i]\|), i=1..L***
  - ***Fitness(C, PM) = #match+alpha.#Missed_placed***
  - ***Fitness(C,PM) = - LevenshteinDistance(C,PM)***

(*) https://pypi.org/project/python-Levenshtein/

Problème de “l’Ultra” Master Mind

- Fonction de mutation : on sélectionne au hasard **TM x N** individus sur lesquels la mutation d’un gène va porter.
  - Pour chaque individus sélectionné, on tire au hasard un gène (un caractère) parmi L et on change aléatoirement sa valeur

Problème de “l’Ultra” Master Mind

- **Travail demandé**

- Programmer le jeu UMM en Pythons 3.x
  - On demandera à l’utilisateur de choisir une phrase mystère
  - Puis le programme s’exécutera en listant le meilleur élément courant de la population
  - On tracera également la courbe Fitness en fonction de la génération
  - On étudiera l’algorithme en faisant varier les paramètre (L,N,TS,TM)
- Le code sera documenté avec soin (définition des entrées et des sorties, + code de test unitaire pour chaque fonction).
- Un programme principal sera fourni avec un exemple de “run”.

Livrables

- Le code documenté avec programme principal et jeu de tests
- Un document PDF contenant :
  - Titre, auteur, résumé
  - Spécification, conception de votre solution en insistant sur ses spécificités
  - Organisation du code (fonctions, flux)
  - Procédure d’installation et d’utilisation
  - Analyse comportementale (courbes, variations des paramètres, convergence, complexité empirique, etc.)
  - Conclusion (rappel des résultats obtenus, discussion, perspective)

Ultra Master Mind++

- Pour cette version du jeu on considère que la longueur **L** de la phrase mystère reste inconnue pour l’algorithme génétique.
  - - Modifier votre code précédent pour qu’il puisse découvrir la phrase mystère en gérant des génomes de taille variable et adapter la fonction de *Fitness* en conséquence.
  - - Tester votre nouveau code comme vous l’avez fait pour le précédent en faisant varier les paramètres de votre algorithme.

Password Guessing Attack (PGA)

- On suppose que l'algorithme de vérification de mots de passe installé sur une station de travail consomme plus ou moins d'énergie selon que la chaîne de caractères testée est plus ou moins proche du mot de passe que l'on cherche à craquer.
- Plus précisément on observe que l'énergie consommée E suit grosso modo l'équation suivante:
  - E(CC,PWD) = A.D(CC,PWD)
  - où :
    - - CC est la chaîne de caractères testée
    - - PWD est le mot de passe recherché
    - - A est une constante de proportionnalité
    - - D(CC,PWD) est la longueur du préfixe commun entre CC et PWD

- **Q1** : pouvez-vous modifier votre programme UMM++ pour tenter de réaliser une PGA sur ce type d'algorithme ?
- **Q2** : proposer un algorithme de vérification de mots de passe à ce type d'attaque.

That's all folks!

À vous de jouer