07–14/01/2021

Encadrants : Simon Forest – simon.forest@liris.cnrs.fr
Rémy Chaput – remy.chaput@univ-lyon1.fr
Sujet et exécutables disponibles sur perso.liris.cnrs.fr/sforest/ibi/.

# Intelligence Bio-Inspirée
# TP : Algorithmes génétiques

## 1 Introduction

Vous cherchez à vous reconnecter à un vieux site internet que vous fréquentiez pendant votre enfance, mais vous ne vous souvenez plus de votre mot de passe ! Le site ne vous permet pas de réinitialiser votre compte, mais heureusement, le système d'authentification est mal conçu, et à l'aide d'un logiciel d'analyse, vous pouvez savoir à chaque tentative à quel point vous être proches du bon mot de passe.

La mesure de proximité entre vos tentatives et le mot de passe semble assez complexe. Vous allez donc utiliser un algorithme génétique pour retrouver votre mot de passe.

## 2 Mise en place

Ce travail est à faire seul ou en binômes. Vous aurez besoin de votre identifiant étudiant (commençant par `1`), il servira à vous attribuer un mot de passe unique. Les mots de passe ont été distribués aléatoirement, votre identifiant n'est pas utilisé dans le calcul de la mesure de similarité.

Vous avez à votre disposition un exécutable, qui vous permettra de comparer vos tentatives au vrai mot de passe. Le code lui-même ne vous est pas accessible[1]. L'exécutable prend au moins deux arguments : votre identifiant, et autant de chaînes de caractères que de tentatives simultanées.

Par exemple, sur Ubuntu :

```
./unlock 11716601 PASSWORD ALGOGEN
PASSWORD    0.294163
ALGOGEN     0.386632
```

et sur Windows (64 bits) :

```
unlock64.exe 11716601 PASSWORD ALGOGEN
PASSWORD    0.311828
ALGOGEN     0.350305
```

Le programme renvoie une valeur située entre 0 et 1, où 1 signifie que la chaîne de caractères donnée en entrée est exactement identique au mot de passe à trouver. Vous ne connaissez pas la manière dont les chaînes de caractères sont comparées. La mesure de similarité n'est pas idéale, et elle peut comporter des maxima locaux, ou vous envoyer localement dans la mauvaise direction.

Pour vous aider, un exemple de programme Python vous est donné en annexe. Il contient une fonction prenant en argument un identifiant et une liste de tentatives, et renvoie une liste de scores.

[1] Il vous est interdit de décompiler le fichier.

# 3 Objectifs

Vous devez utiliser un algorithme génétique pour trouver le mot de passe, ou au moins s'en approcher le plus possible. Vous en avez les informations suivantes :

- Il est composé uniquement de chiffres et de lettres majuscules.
- Il contient entre 12 et 18 caractères.

Un « individu » dans votre algorithme correspondra donc à une tentative de mot de passe, son **phénotype** étant une chaîne de 12 à 18 chiffres et lettres majuscules.

Vous ne serez pas évalués uniquement sur la découverte ou non du bon résultat, mais surtout sur la conception de l'algorithme et la justification de vos choix concernant :

- le codage du génotype ;
- la sélection (et la présence ou non d'élitisme) ;
- les mutations ;
- le cross-over ;
- les valeurs des hyper-paramètres (nombre d'individus, taux de mutation, etc.).

Vous aurez sûrement besoin de plusieurs itérations successives de votre algorithme, n'hésitez pas à observer son comportement pour l'améliorer[2].

# 4 Travail à rendre

Vous devrez envoyer, par mail à **simon.forest@liris.cnrs.fr**, votre livrable composé de :

- votre code source, commenté, compressé dans une archive ;
- votre rapport au format PDF **ou** *Jupyter Notebook*[3].

La date limite est fixée au 31 janvier 2021.

## 4.1 Code

Vous devez programmer un algorithme génétique, de préférence en Python. Il est recommandé de structurer proprement votre code, en utilisant plusieurs classes notamment. Mettez à part la définition des hyper-paramètres, pour qu'il soit plus facile de les retrouver et modifier (par exemple à l'aide d'un fichier de configuration).

Votre code doit pouvoir être lancé tel quel, sans modifier les paramètres, et donner en sortie le mot de passe trouvé (ou la solution la plus proche). Vérifiez bien que votre algorithme fonctionne en le relançant plusieurs fois, et que vous n'êtes pas tombés sur le bon résultat par un simple coup de chance.

## 4.2 Rapport

Vous devez expliquer et justifier vos choix pour l'implémentation de votre algorithme et le réglage des hyper-paramètres. Certains arguments seront assez empiriques, mais vous pouvez aussi mentionner des implémentations que vous avez essayées et qui n'ont pas fonctionné comme vous l'espériez.

[2] Observez si la *fitness* croît, stagne, ou même re-diminue. Observez aussi comment les phénotypes se rapprochent de la solution.

[3] Vous pouvez également intégrer tout votre code dans le *notebook*, mais il peut être préférable de segmenter votre code en plusieurs fichiers.

Vous devrez aussi analyser l'efficacité de votre algorithme. Vous pourrez prendre en compte la quantité d'opérations effectuée, le temps de convergence, etc. Vous pourrez vous appuyer sur des graphiques, par exemple :

- évolution de la meilleure *fitness* ou *fitness* moyenne au cours des générations ;
- *fitness* en fonction de la distance entre le meilleur phénotype et le mot de passe final ;
- influence d'un hyper-paramètre sur le temps de convergence ;
- etc.

## 5 Annexe

Ce code fonctionne avec Python 3.5 sous Ubuntu, des ajustements seront à prévoir selon votre système.

```
# coding : utf8
# !/usr/bin/env python

import subprocess

STUDENT_ID = 11716601 # A REMPLACER

def check(student_id, passwords) :
    proc = subprocess.Popen(["./unlock", str(student_id)] \
            + passwords, stdout = subprocess.PIPE)
    results = []
    while True :
        line = proc.stdout.readline()
        if not line :
            break
        results.append(float(str(line).split("\\t")[1]\
                .split("\\n")[0]))
    return results

print(check(STUDENT_ID, ["PASSWORD", "ALGOGEN"]))
```

Résultat :

```
$ python demo.py
[0.256673, 0.381885]
```