Programmation évolutive et applications

Plan :

- Introduction : présentation de l'idée

- Présentation du programme en se basant sur l'exemple du nombre premier

- Interprétation et concrétisation

- Complexité, améliorations, autres applications (à faire tourner)

I) Introduction : motivation

- ⇨ 2001 L'Odyssée de l'espace : HAL 9000 à l'œil rouge
- ⇨ Donner à la machine la capacité d'évoluer, c'est lui donner la possibilité de nous dépasser
- ⇨ Un problème qu'on ne sait pas résoudre ?
- ⇨ Un problème long et fastidieux à résoudre ?
- ⇨ Il suffit de rentrer ce qu'on veut et la machine fait en sorte de s'en approcher

- ⇨ L'aléatoire utilisé pour converger : Dichotomie, jeu du + ou -

II) Présentation du programme en se basant sur l'exemple du nombre premier

- ⇨ Programme en lui-même : ne fait que gérer des sous-programmes, des fonctions
- ⇨ Ces fonctions sont elles-mêmes des compositions de fonctions élémentaires, simples, établies par l'utilisateur
- ⇨ A la génération N, 200 fonctions existent.
- ⇨ Le programme utilise une loi (établie par l'utilisateur) afin d'affecter un score à chaque fonctions.
- ⇨ Le programme tue alors les 100 plus mauvais scores et copie les 100 meilleurs à la génération N+1
- ⇨ On en copie alors une nouvelle fois les 10 en leur appliquant une mutation
- ⇨ On copie les 90 restantes en appliquant un échange (un cross-over en génétique)
- ⇨ Exemple imagé avec arbre de mutation
- ⇨ Exemple imagé avec arbre de cross-over

- ⇨ Réexplication plus imagée avec l'exemple
- ⇨ Fonctions élémentaires : Portes ET-OU-NON
- ⇨ Loi : Comparer la fonction à isprime() sur les entiers de 2 à 255
  Pour cela, on donne donc la liste des nombres premiers

- ⇨ On sait que, comme une solution au problème existe, au bout de X générations, X assez grand, on aboutira à une solution
- ⇨ Il s'agit ensuite de limiter la taille de la fonction solution en limitant le nombre de fonctions élémentaires qui la composent (petite modification de la loi)

III) Interprétation et concrétisation

- ⇨ String totalement illisible
- ⇨ Conversion sous forme de bel arbre (circuit logique) (avec l'aide de graphviz)
- ⇨ Mappage adapté aux composants (AND-OR-NOT)
- ⇨ Celui-ci a été fait manuellement au fil de la soudure mais aurait pu (dû) se faire par l'intermédiaire de mon programme (vu l'agglomérat de fils)

- ⇨ Soudure
- ⇨ Alimentation USB (5V pour les composants TTL (CMOS aussi))
- ⇨ Câble qui gênait : prise jack en guise de port
- ⇨ Soucis dû à un début de soudure avec un mauvais étain (trop ou pas assez de plomb)

IV) Complexité, améliorations, applications

- ⇨ Un peu plus de 100h pour le 256 (amélioration progressive)
- ⇨ Ecriture/lecture pour la sauvegarde => Long/peu utile (mieux vaut garder en mémoire et vider)
- ⇨ Programmation en thread : dual core puis 8 core
- ⇨ Choix des fonctions à garder, au-delà du score : plus courtes fonctions/ dernières prioritaires
- ⇨ Empilement-dépilement peut-être pas le mieux adapté

- ⇨ Evaluer la complexité : faire une batterie de test nb premier à 32 ou 64 (ou autre application)
- ⇨ Parler des applications : lentilles, RLC, portes logiques, plus court chemin, intelligence artificielle (très long),