Cahier des Charges

Edouard Fouassier - Maxime Gonthier - Benjamin Guillot
Laureline Martin - Rémi Navarro - Lydia Rodrigez de la Nava

Algorithme Génétique

12 mars 2018

Table des matières

1 Introduction

Le nom algorithme génétique provient des ressemblances avec le monde du vivant. On utilise des notions telles que la sélection naturelle, les générations ou bien encore la sélection naturelle, dans le but de trouver la meilleure population possible. Cet algorithme a été étudié par John Holland de 1960 à 1975 dans son ouvrage Adaptation in Natural and Artificial System inspiré entre autres par la "learning machine" de Turing.

L'algorithme génétique s'applique à une grande variété de domaines, tels que la génétique pour étudier l'évolution des gènes d'une espèce donnée sur plusieurs générations. On peut aussi l'appliquer dans le cas d'apprentissage pour par exemple apprendre à un robot à se déplacer en fonction des obstacles, ou bien on peut l'utiliser dans l'optique de trouver un résultat optimisé. On peut citer dans cette dernière catégorie l'optimisation d'un portefeuille d'actions en fonction de leur risques, l'optimisation de la ventilation dans le cas d'incendie dans un espace confiné ou bien simplement l'optimisation d'une fonction.

L'algorithme génétique est particulièrement utile lorsque l'utilisateur étudie une population de valeurs et qu'il recherche une solution approchée parmi ces valeurs. Il se démarque des autres algorithmes car il est très facilement adaptable à différents problèmes. De plus, on utilise des règles de transition probabiliste, ce qui est pertinent pour des problèmes où les résultats sont des valeurs approchées.

2 Fondement du projet

2.1 But du projet

L'objectif du projet est de réaliser un programme utilisant un algorithme génétique permettant de délivrer une solution ou un ensemble de solutions optimales à problème donné.

2.2 Personnes et organismes impliqués dans les enjeux du projet

Le projet a pour client principal le Professeur Leila Kloul.

2.3 Utilisateurs du produit

Le produit se veut employable par n'importe qui cherchant une réponse adaptée à un problème d'optimisation lié à une fonction.

3 Contraintes sur le projet

3.1 Contraintes non négociables

Le logiciel doit permettre à l'utilisateur d'entrer les données de son problème dans une interface textuelle ou de sélectioner un fichier texte contenant déjà celles-ci. De plus, l'algorithme génétique doit être générique afin de fournir une utilisation indépendante du problème à traiter. Enfin, l'utilisateur doit pouvoir obtenir en sortie un fichier montrant l'évolution de l'algorithme durant l'opération. Il doit pouvoir choisir entre les formats suivants : Xfig et/ou LaTeX et/ou PostScript.

3.2 Glossaire et conventions de dénomination

Nous utiliserons dans la suite du cahier des charges les termes suivants pour parler de l'algorithme génétique :

- Une **population d'individus** est un ensemble de solutions potentielles. C'est l'ensemble de données sur lequel l'application sera utilisée afin d'en tirer le meilleur résultat possible.
- Un **individu** ou un **chromosome** est une solution potentielle.
- La **taille** d'un individu s'exprime en puissance de 2. C'est la taille de la solution potentielle au problème (elle peut s'exprimer sous la forme d'un tableau, d'une chaîne,...).
- Un **gène** est une partie de l'individu.
- L'**alphabet** est l'ensemble de caractères que peut prendre un gène (ex : 0,1 si on utilise des individus codés en binaire)
- Une **génération** est une itération de l'algorithme génétique utilisé dans l'application. Elle correspond à la création d'une nouvelle population.
- La **fonction fitness** est la fonction que l'on cherche à optimiser. Elle sert également à évaluer la qualité des individus d'une génération.
- **L'algorithme génétique** est composé de trois étapes :
  - La **reproduction** correspond à la sélection de plusieurs individus de l'ensemble de base pour former de nouveaux individus, potentiellement meilleurs. Ici nous avons choisi une sélection par roulette, où les parents sont sélectionnés selon leurs scores : plus le score d'un individu est élevé, plus il a de chance d'être sélectionné. L'avantage de cette sélection est qu'elle est préservative, cela signifie que même les individus les moins performants conservent une chance d'être sélectionnés et leur évite une disparition prématurée.
  - La **mutation** correspond à une modification ponctuelle d'un gène d'un individu. Nous avons choisi une mutation basée sur une probabilité préférablement faible, c'est-à-dire qu'un gène a une certaine probabilité d'être modifié. Cette méthode nous semble être adéquate pour un algorithme génétique générique, car puisqu'elle est équiprobable, elle est applicable à un grand nombre de domaines d'application et une probabilité très faible évite de fausser les résultats.
  - Le **crossover** est le croisement d'individus pour en créer de nouveaux. Pour notre logiciel, nous avons choisi d'appliquer un cross-over en un point. Cette méthode consiste à séparer deux individus parents en deux morceaux, puis de les recombiner pour former deux enfants. Cette méthode nous semble être le meilleur choix pour un algorithme génétique générique, car elle imite la génétique humaine et évite une convergence prématurée.

3.3 Faits et hypothèses utiles

Il est impossible de produire un programme totalement générique, c'est pourquoi nous limitons ici l'utilisation du logiciel aux problèmes d'optimisation et de prédiction. Pour l'utiliser dans des problèmes d'apprentissage, il faudrait le coupler avec un autre programme comme un système de classifieur, ou encore un réseau de neurones, ce qui n'est pas le but de ce projet.

4 Exigences fonctionnelles

4.1 Portée du produit

Le programme est générique, et donc applicable dans toutes sortes de situations.

4.2 Exigences fonctionnelles et exigences sur les données

On doit proposer une interface graphique permettant de saisir les données du problème. De plus, il faut trouver un ensemble de solutions à ce problème et les afficher dans un fichier sortie avec une représentation de l'évolution des générations.

5 Exigences non fonctionnelles

5.1 Ergonomie et convivialité du produit

Le produit devra avoir une interface intuitive et lisible pour faciliter son utilisation et les fichiers de sortie devront être simple à comprendre.

5.2 Facilité d'utilisation et facteurs humains

L'utilisation du logiciel ne doit nécessiter aucun pré-requis à l'utilisateur autre que la connaissance du problème auquel il cherche une solution. Le logiciel doit ainsi procéder à une vérification de tous les champs remplis.

5.3 Maintenance, support, portabilité, installation du produit

Le logiciel se veut simple à installer sous Linux. Il sera aussi fait en sorte que l'on puisse ajouter de nouveaux modules au programme simplement.

6 Organigramme

Voir Organigramme page annexe (Page 7)
Les données initiales sont : taux de mutation, identifiant plus tableau pour le test, fonction fitness, nombre de génération, score à atteindre sur la fonction, taille individu, critère d'évaluation, nombre de critères, alphabet, taille population, probabilité de crossover, nom de fichier.

Interface :
- Affichage des différents champs de saisie puis des résultats
- Réception des valeurs (Données initiales)
  - par fichier
  - par saisie des champs de l'interface
- Option d'arrêt pendant l'exécution de l'AG : on peut arrêter le programme à tout moment et récupérer les résultats jusqu'à l'arrêt

Gestion de la validité des données :
- Test des valeurs reçues
- Enregistrement des valeurs dans un fichier

Module d'initialisation :
— Transformation de valeurs en paramètres
— Creation de la population initiale
— Lecture du fichier des données

Evaluation de la population :
— Evaluation de chaque individu et attribution d'un score

Algorithme Evaluation : paramètre : une population
    Si le nombre de critères == 1
        Si l'option d'évaluation du critère == '>'
            Tri de la population puis attribution du score en fontion du rang après le tri
    Sinon Pour chaque individu de la population faire :
            Utilisation de la fonction fitness, le score correspond au résultat renvoyé

Complexité dans le pire des cas : $O(n^2)$ (n : taille de la population)

Tests d'arrets :
— Test de convergence de la fonction fitness
— Test du nombre de générations

Génération de la nouvelle population :
— Sélection par roulette
— Cross-over/mutation
— Création de la nouvelle population
— Génération de nombres aléatoires

Algorithme Roulette : paramètre : les individus
    si le nombre d'individus est pair
        limite = population/2
    sinon
        limite = population-1/2
    scoremax = $\sum score$
    Pour chaque individu on calcule sa valeur (en pourcentage) en fonction du scoremax
    Tant qu'on a pas assez d'individu sélectionné
        Si le pourcentage du premier individu est supérieur à une valeur aléatoire
            il est sélectionné
        Pour chaque individu sauf le premier faire :
            si (le pourcentage de l'individu > à une valeur aléatoire
            ET que le pourcentage de celui qui le précède est inférieur à cette valeur)
                cet individu est sélectionné.
    On retourne les individus sélectionnés

Complexité dans le pire des cas : $O(n^2)$

Gestion d'entrées sorties :
— Calcul des statistiques de l'éxécution : les données d'entrées et les résultats
— Ecriture des résultats dans latex

— Ecriture des résultats dans PostScript
— Ecriture des résultats dans Xfig
— Lecture d'un fichier d'entrée avec les données
— Ecriture des valeurs dans un fichier

7 Autres aspects du projet

7.1 Estimation des coûts du projet

Module	Nombre de lignes	Temps	Affectation
Interface	250-300	5-6 heures	Rémi et Edouard
Gestion de la validité des données	50-60	1-2 heures	Maxime et Laureline
Initialisation du programme	transformation des valeurs en paramètres : 30 création de la population initiale : 40 lecture du fichier de données : 40	2-3 heures	Edouard et Rémi
Evaluation de la population	25-30	1-2 heures	Lydia et Ben
Tests d'arrets	par itération ou forcé : 40 par convergence : 70	2-3 heures	Laureline et Maxime
Génération de la nouvelle population	40-50	2-3 heures	Ben et Lydia
Gestion d'entrées sorties	200	4-5 heures	tout le monde
Coût Total	**745-820**	**14-24 heures**	

7.2 Manuel utilisateur et formations

Un manuel d'utilisation sera fourni pour conseiller à l'utilisateur les données les plus performantes pour avoir des solutions plus efficaces.

8 Conclusion

Pour résoudre les exigences du client, nous avons élaboré un cahier des charges qui décrit les différents dispositifs nécessaires pour conceptualiser un algorithme génétique générique. Nous y proposons un programme qui permettra au client de simuler des problèmes variés, tels que des problèmes génétiques, mathématiques, d'optimisation, etc. pour trouver une solution la plus

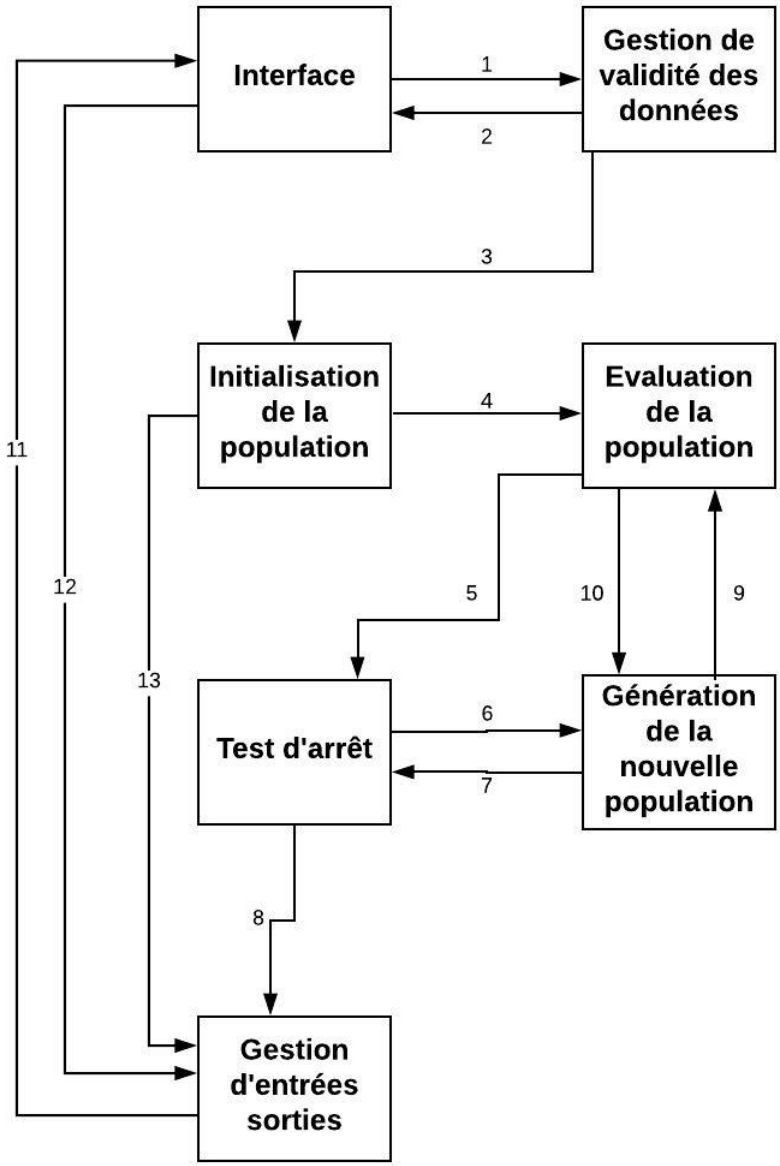
approchée possible. Nous avons conclu de ce cahier des charges que le langage le plus adapté pour notre logiciel est le langage C++ car il est hybride. En effet, nous aurons d'une part besoin d'un langage objet pour implémenter les individus qui seront un objet caractérisé par plusieurs champs. Les individus formeront une population qui sera elle aussi un objet. D'autre part, nous aurons besoin d'un langage procédural afin d'effectuer des calculs sur les individus. Le C++ nous semble également avantageux pour la fonction fitness qui nécessite l'utilisation d'un parser. Cependant, le travail à fournir pour produire notre propre parser nous semble fastidieux, long en terme de temps et de ligne, et qui ne lira au final qu'une seule fonction. Nous avons donc fait le choix d'utiliser un parser déjà existant et de mettre à profit le temps gagné pour se concentrer complètement sur l'application.

9 Bibliographie

— Présentation des algorithmes génétiques et de leurs applications en économie :
http ://deptinfo.unice.fr/twiki/pub/Minfo04/IaDecision0405/
Prsentationdesalgorithmesgntiquesetdeleursapplicationsenconomie_P.pdf
— La page wikipedia sur l'algorithme génétique :
https ://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique
— Tutoriel Python sur les algorithmes génétiques :
https ://skyduino.wordpress.com/2015/07/16/tutorielpython-les-algorithmes-genetiques-garantis-sans-ogm/
— Etude sur les algorithmes génétiques :
http ://souqueta.free.fr/Project/files/TE_AG.pdf
— Exemple d'utilisation sur un jeu vidéo :
https ://www.youtube.com/watch ?v=9tCySY6TLk8
— Vidéos d'explications de l'algorthme génétique :
https ://www.youtube.com/watch ?v=VOIPosVDkEE
https ://www.youtube.com/watch ?v=bLdrDnZfpPo
https ://youtu.be/mAybMmf0veY
— L'application des algorithmes génétiques dans la science des données :
https ://www.analyticsvidhya.com/blog/2017/07/introduction-to-genetic-algorithm/
— Explications de l'aglorithme générique :
http ://khayyam.developpez.com/articles/algo/genetic/
http ://informatique.coursgratuits.net/methodes-numeriques/algorithmes-genetiques.php
— Etude sur l'inertage de feux :
http ://www.ligeron.com/telecharger.php ?id=11
— Comparaison de l'algorithme génétique et du réseau de neuronnes :
https ://stackoverflow.com/questions/1402370/when-to-use-genetic-algorithms-vs-when-to-use-neural-networks
— Livre : Algorithmes génériques : exploration, optimisation et apprentissage automatique de David E. Goldberg

10 Annexe

Organigramme :



Légende :

1. Nom fichier
2. Message d'erreur
3. Nom fichier
4. Population initiale
5. Population initiale
6. Population n- 1
7. Nouvelle population générée
8. Population
9. Population sans score
10. Population avec score
11. Nom du fichier
12. Nom du fichier
13. Nom du fichier