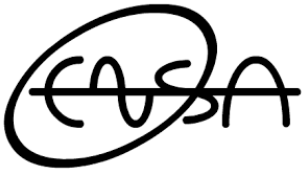


Étude des algorithmes évolutionnistes: Les algorithmes génétiques

SADOUNE Abdelkarim
RABHI Mohamed Amine

Université Mohammed Premier d'Oujda
École Nationale des Sciences Appliquées

encadré par : **Mr.BOUKSIM Mohcine**

13 septembre 2021

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Les algorithmes évolutionnistes

1.1.1 Définition

Les algorithmes évolutionnistes ou algorithmes évolutionnaires (evolutionary algorithms en anglais), sont une famille d'algorithmes dont le principe s'inspire de la théorie de l'évolution pour résoudre des problèmes divers. Ce sont donc des méthodes de calcul bioinspirées. L'idée est de faire évoluer un ensemble de solutions à un problème donné, dans l'optique de trouver les meilleurs résultats. Ce sont des algorithmes dits stochastiques, car ils utilisent itérativement des processus aléatoires.

La grande majorité de ces méthodes sont utilisées pour résoudre des problèmes d'optimisation, elles sont en cela des métaheuristiques, bien que le cadre général ne soit pas nécessairement dédié aux algorithmes d'optimisation au sens strict. On les classe également parmi les méthodes d'intelligence computationnelle.

1.1.2 Origines

Ces algorithmes manipulent des populations de solutions, l'arbre de la vie, tel que le représente Charles Darwin dans son ouvrage L'Origine des espèces, où il présente ses théories sur l'évolution des êtres vivants. Les algorithmes évolutionnaires s'inspirent de l'évolution des êtres vivants, en considérant que celle-ci tend à produire des organismes plus adaptés à leur environnement.

Selon la théorie de l'évolution, plusieurs mécanismes sont à l'œuvre pour ce faire. Schématiquement :

• Les caractéristiques d'un organisme sont en grande partie codées dans ses gènes,

• chaque population d'organismes est composée d'individus tous différents,

• les individus sont plus ou moins adaptés à leur environnement, les organismes transmettent une partie de leurs caractéristiques à leurs descendants,

• Les individus plus adaptés se reproduisent «plus efficacement», et leurs caractéristiques ont donc tendance à être plus répandues dans la population.

1.1.3 Terminologie

Tous les algorithmes évolutionnaires font évoluer un ensemble (une population) de solutions (les individus). Les individus sont représentés par leur génotype, qui s'exprime sous la forme d'un phénotype, auxquels on associe une qualité (la fonction fitness). Les algorithmes sont conçus de façon que plus la fitness d'un individu est élevée, plus il y a des chances pour transmettre son génotype au sein de la génération suivante.

À chaque étape de l'algorithme est associé un « opérateur », qui décrit la façon de manipuler les individus. On regroupe parfois les différents opérateurs sous des termes génériques :

• opérateurs de sélection pour la sélection et le remplacement,

• opérateurs de variation pour la mutation et le croisement.

1.1.4 Algorithme

Pour ce faire, on utilise l'algorithme général suivant :

---

construction et évaluation d'une ***population initiale***
Jusqu'à atteindre un ***critère d'arrêt*** *:*
sélection *d'une partie de la population*
reproduction *des individus sélectionnés*
mutation *de la descendance*
évaluation *du degré d'adaptation de chaque individu*
remplacement *de la population initiale par une nouvelle population.*

Après avoir initialisé une première population d'individus, on itère un nombre fini de fois, jusqu'à atteindre un critère d'arrêt (par exemple un nombre maximum de générations). La première étape de sélection permet de séparer les individus qui participeront à la reproduction de ceux qui n'y participeront pas. Les individus sélectionnés (les parents) se reproduisent (on dit aussi que l'on effectue des croisements), donnant un ensemble d'enfants partageant une partie des caractéristiques de leurs ascendants. Ces enfants subissent alors une étape de mutation, qui modifie aléatoirement leur génotype. Les nouveaux individus sont alors évalués (on met à jour leur valeur en faisant appel à la fonction objectif). Enfin, on choisit un nombre d'individus déterminé parmi l'ensemble (parents + enfants), pour former la génération suivante.

1.2 Principales familles

1.2.1 Algorithmes génétiques

Les algorithmes génétiques (AG) impliquent l'évolution d'une population de vecteurs de dimension fixe composés de caractères, généralement des bits. L'idée de AG est vaguement basé sur les chaînes d'ADN, qui composent tous les organismes vivants. Les AG sont utilisés pour découvrir une solution, généralement numérique, résoudre un problème donné, sans avoir aucune connaissance a priori sur l'espace de recherche. Seul un critère de qualité est nécessaire pour discriminer les différentes solutions. Dans la plupart des cas, ce critère est une mesure objective qui permet de quantifier la capacité de l'individu à résoudre le problème donné. Le processus de recherche se fait en appliquant itérativement à une population des solutions potentielles des opérations de variation génétique

(croisement et mutation), et des opérations de sélection naturelle biaisées en faveur des individus les plus forts. Utilisant ce processus darwinien, la population de solutions potentielles évolue avec le temps jusqu'à ce qu'un certain critère d'arrêt soit atteint. La figure 1.2 montre plus de détails sur le processus évolutif de l'AG. Dans un contexte d'optimisation des fonctions avec des paramètres réels, une variante importante de l'AG consiste à représenter les individus directement par des vecteurs de nombres réels.

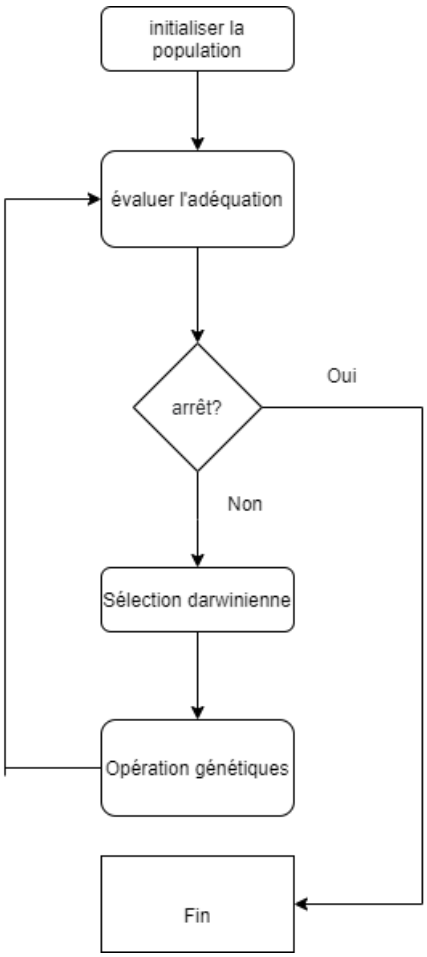


FIGURE 1.1 – Processus évolutionnaire classique des algorithmes génétiques.

1.2.2 Programmation génétique

La programmation génétique (PG) est un paradigme pour la programmation et l'automatisation informatique par heuristique basée sur les mêmes principes d'évolution que les AG : opérations de variation génétique, par croisements et mutations et opérations de sélection naturelle. La différence entre PG et AG réside avant tout dans la représentation des individus. En effet, la PG Consiste à évoluer des individus dont la structure est similaire à celle des programmes informatiques. PG a été formellement exprimé par Koza au début des années 1990 . La PG utilisé par Koza représente des individus sous forme d'arbres, c-à-d des graphes orientés et sans cycle, dans lesquels chaque nœud est associé à une opération Elémentaire par rapport au domaine du problème. Plusieurs autres représentations comme des programmes linéaires et des graphiques cycliques ont été utilisés depuis. la PG est particulièrement adapté à l'évolution de structures complexes de dimensions variables.

1.2.3 Programmation évolutionnaire

La programmation évolutive (PE) a été développée par L.J. Fogel dans le 1960 et repris par D.B. Fogel et autres dans les années 1990. la PE a été conçu à l'origine dans le but de mettre à niveau les machines d'état fini et a ensuite été étendu aux problèmes d'optimisation des paramètres. Cette approche met l'accent sur la relation entre les parents et leurs descendants plutôt que pour simuler des opérateurs génétiques d'inspiration naturelle. la PE n'utilise pas de représentation spécifique mais plutôt un modèle évolutif de haut niveau, associé à une représentation et à un opérateur de mutation directement appropriée au problème à résoudre. Pour réaliser la PE, une population de solutions potentielles au problème est d'abord généré au hasard. Chaque individu *i* de la population produit des descendants résultant de mutations. Une opération de sélection naturelle est alors appliquée afin de former une nouvelle population de μ individus. Le processus de mutation et de sélection est répétée itérativement jusqu'à ce qu'une solution acceptable soit trouvée.

1.2.4 Stratégies d'évolution

Les stratégies d'évolution (SE) sont développées par I.Rechenberg et H.-P. Schwefel à Berlin dans les années 1960 . Les SE représentent l'individus comme un ensemble de caractéristiques de la solution potentielle.

En général, cet ensemble prend la forme d'un vecteur de nombres réels de dimension fixe. la SE s'appliquent à une population de parents (de dimension μ) à partir de laquelle les individus sont sélectionnés au hasard afin de générer une population de descendants(de dimension μ). Ceux-ci sont ensuite modifiés par des mutations qui consistent à ajouter une valeur générée aléatoirement selon une fonction de densité de probabilité paramétré. Les paramètres de la fonction de densité de probabilité, appelés paramètres éléments de la stratégie, évoluent également dans le temps selon les mêmes principes que les paramètres caractérisant les individus. Pour former la nouvelle population, μ individus sont choisis (approche (μ,)) parmi les μ meilleurs individus des descendants, ou (approche (μ+)) parmi les μ meilleurs individus de μ parents et descendants. Les SE modernes peuvent également intégrer une approche multi-échelles où les stratégies de l'évolution sont imbriquées.

Chapitre 2

Algorithme génétique

Les algorithmes génétiques (AG) utilisent des concepts du principe de sélection naturelle pour traiter un large éventail de problèmes, en particulier l'optimisation. Ils sont robustes, générique et facilement adaptables. Inspiré par la façon dont le darwinisme explique le processus d'évolution des espèces, Les AG sont décomposés selon les étapes suivantes : initialisation, évaluation, sélection, croisement, mutation, mise à jour et l'achèvement.Fondamentalement, ce qu'un algorithme génétique fait, c'est créer une population de réponses possibles au problème à traité, puis mettre-la au processus d'évolution. Ensuite, sera décrit chaque étape :

- **Evaluation** : la capacité des solutions est évaluée (en individus de la population) au moyen d'une analyse afin de déterminer quels sont les individus les plus aptes au sein de la population (meilleure solution du problème)

- **Sélection** : les individus sont sélectionnés pour la reproduction. La probabilité qu'une solution donnée soit sélectionnée est proportionnel à sa fonction fitness

- **Crossover** : les caractéristiques des solutions choisies sont recombinés en générant de nouveaux individus

- **Mutation** : les caractéristiques des individus résultant du processus de reproduction sont modifiés, ainsi ajouter de la variété à la population

- **Mise à jour** : les individus créés dans cette génération sont inséré dans la population

- **Finition** : On vérifie si les conditions pour la fin de l'évolution ont été atteintes, en revenant à l'étape d'évaluation sinon, en terminant l'exécution.

2.1 Les avantages et les inconvénients

2.1.1 Les avantages

- **Parallélisme**, facilement modifiable et adaptable à différents problèmes

- **Facilement distribué**

- **Capacité de recherche** de l'espace de solution large et étendue

- **Processus d'optimisation** non fondé sur les connaissances utilisées pour une fonction d'aptitude à des fins d'évaluation

- **Capable d'optimisation multi-objectifs** peut retourner une suite de solutions potentielles

- **Bon choix** pour les problèmes d'optimisation à grande échelle / grande variété

2.1.2 Les inconvénients

- **Aucune garantie de trouver des maxima globaux**. Mais là encore, à part la force brute, il y a rarement une garantie pour des problèmes non triviaux. Mais la probabilité de rester coincé dans un maximum local au début est quelque chose à laquelle vous devrez peut-être faire face.

- **Temps de convergence**. Vous avez généralement besoin d'une population de taille décente et de nombreuses générations avant de voir de bons résultats. Et avec une simulation lourde, vous pouvez souvent attendre des jours pour une solution.

• **C'est un art noir**. Le réglage de tous les paramètres de l'AG, comme le taux de mutation, le pourcentage d'élitisme, les paramètres de croisement, les paramètres de normalisation et la sélection de la forme physique, etc., ne sont souvent que des essais et des erreurs.

• **Autres aspects complexes**. Outre les paramètres génétiques de l'AG, d'autres éléments tels que la fonction de fitness, le choix du codage génétique, la cartographie du génotype au phénotype, etc., sont également importants pour l'efficacité du système.

2.2 Domaines d'application

2.2.1 Applications ludiques

De par leur nature, les algorithmes génétiques peuvent être utilisés à des fins purement ludiques et répondre à des problématiques sur des jeux qui se jouent en solitaire. En effet, un travail d'apprentissage grâce à un système de scoring est plus qu'approprié au monde du jeu, étant donné que le score est un élément central de n'importe quel jeu pour pouvoir classer les joueurs entre eux. La fonction d'évaluation étant déjà calculée via le jeu, il est d'autant plus facile de développer des algorithmes génétiques.

On peut noter d'ailleurs quelques exemples intéressants d'application à des titres cultes du jeu vidéo :

- Flappy Bird
- Mario
- Pacman
- ...

2.2.2 Informatique décisionnelle

Les algorithmes génétiques sont mis en œuvre dans certains outils d'informatique décisionnelle ou de data mining par exemple pour recher-

cher une solution d'optimum à un problème par mutation des attributs (des variables) de la population étudiée.

Ils sont utilisés par exemple dans une étude d'optimisation d'un réseau de points de vente ou d'agences (banque, assurance...) pour tenter de répondre aux questions :

- quelles sont les variables (superficie, effectif...) qui expliquent la réussite commerciale de telle ou telle agence ?

- en modifiant telle variable (mutation) de telle agence améliore-t-on son résultat ?

2.2.3 Applications industrielles

Un premier exemple est une réalisation effectuée au sein de l'entreprise Motorola. Le problème pour lequel Motorola a utilisé les algorithmes génétiques concerne les tests des applications informatiques. En effet, lors de chaque changement apporté à une application, il convient de retester l'application afin de voir si les modifications apportées n'ont pas eu d'influence négative sur le reste de l'application. Pour cela, la méthode classique est de définir manuellement des plans de test permettant un passage dans toutes les fonctions de l'application. Mais ce type de test nécessite un important travail humain. Le but de Motorola a donc été d'automatiser cette phase de définition de plans de tests. Ils ont pour cela défini un algorithme où chaque individu correspond à un résultat d'exécution d'un programme (l'enchaînement des valeurs passées au programme) et où chaque individu reçoit une valeur qui correspond à son aptitude à passer dans un maximum de parties du code de l'application. Finalement, l'outil développé permet, à l'aide d'un algorithme génétique, de faire évoluer ces programmes de test pour maximiser les zones testées de façon que lors de modifications apportées à l'application on puisse soumettre celle-ci à des tests efficaces. D'autres domaines industriels utilisent aujourd'hui les algorithmes génétiques. On peut retenir entre autres l'aérodynamique où des optimisations sont mises au point à l'aide de ces outils, l'optimisation structurelle, qui consiste à minimiser le poids d'une structure en tenant compte des contraintes de tension admissibles pour les différents éléments, et la recherche d'itinéraires : ces algorithmes ont été utilisés par la NASA pour la mission d'exploration de Mars, dans la gestion des déplacements du robot Pathfinder.

2.2.4 L'optimisation combinatoire (le problème du sac à dos)

En algorithmique, le problème du sac à dos, (Knapsack Problem) est un problème d'optimisation combinatoire. Il modélise une situation analogue au remplissage d'un sac à dos, ne pouvant supporter plus d'un certain poids, avec tout ou partie d'un ensemble donné d'objets ayant chacun un poids et une valeur. Les objets mis dans le sac à dos doivent maximiser la valeur totale, sans dépasser le poids maximum.

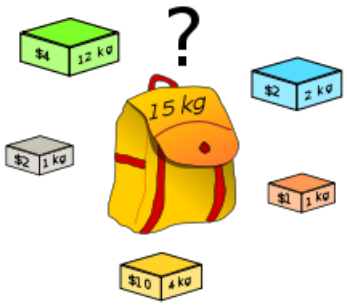


FIGURE 2.1 – Quelles boîtes à choisir afin de maximiser la somme emportée tout en ne dépassant pas les 15 kg autorisés ?

Disons que vous allez passer un mois dans la nature. La seule chose que vous transportez est le sac à dos qui peut supporter un poids maximum de 30 kg. Maintenant, vous avez différents objets de survie, chacun ayant ses propres «points de survie» (qui sont donnés pour chaque élément du tableau). Donc, votre objectif est de maximiser les points de survie.

Voici le tableau donnant des détails sur chaque élément.

OBJET	*POIDS*	*POINTS DE SURVIE*
SAC DE COUCHAGE	15	15
CORDE	3	7
COUTEAU DE POCHE	2	10
TORCHE	5	5
BOUTEILLE	9	8
GLUCOSE	20	17

l'initialisation

Pour résoudre ce problème en utilisant un algorithme génétique, notre première étape serait de définir notre population. Notre population contien-

dra donc des individus, chacun ayant son propre ensemble de chromosomes.

Nous savons que les chromosomes sont des chaînes binaires, où pour ce problème 1 signifierait que l'élément suivant est pris et 0 signifie qu'il est abandonné.

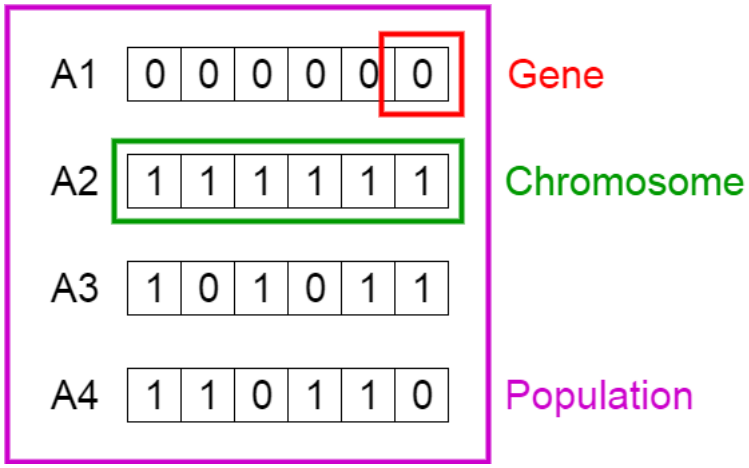


FIGURE 2.2 – Ensemble de chromosomes considéré comme notre population initiale.

La fonction fitness

Calculons les points de forme pour nos deux premiers chromosomes. Pour le chromosome A1 [100110],

OBJET	*POIDS*	*POINTS DE SURVIE*
SAC DE COUCHAGE	15	15
TORCHE	5	5
BOUTEILLE	9	8
TOTAL	29	**28**

De même pour le chromosome A2 [001110],

OBJET	*POIDS*	*POINTS DE SURVIE*
COUTEAU DE POCHE	2	10
TORCHE	5	5
BOUTEILLE	9	8
TOTAL	16	**23**

Donc, pour ce problème, notre chromosome sera considéré comme plus apte lorsqu'il contient plus de points de survie. ***Par conséquent, le chromosome 1 est plus adapté que le chromosome 2.***

Sélection

Maintenant, nous pouvons sélectionner des chromosomes convenables de notre population qui peuvent s'accoupler et créer leurs rejetons.

L'idée générale est que nous devrions sélectionner les chromosomes adaptés et leur permettre de produire des rejetons. Mais cela conduirait à des chromosomes plus proches les uns des autres dans quelques prochaines générations, et donc moins de diversité.

Par conséquent, nous utilisons généralement la méthode de sélection de la roulette.

FIGURE 2.3 – La roue de roulette.

Prenons une roue et divisons-la en m divisions, où m est le nombre de chromosomes dans nos populations. La zone occupée par chaque chromosome sera proportionnelle à sa valeur de fitness.

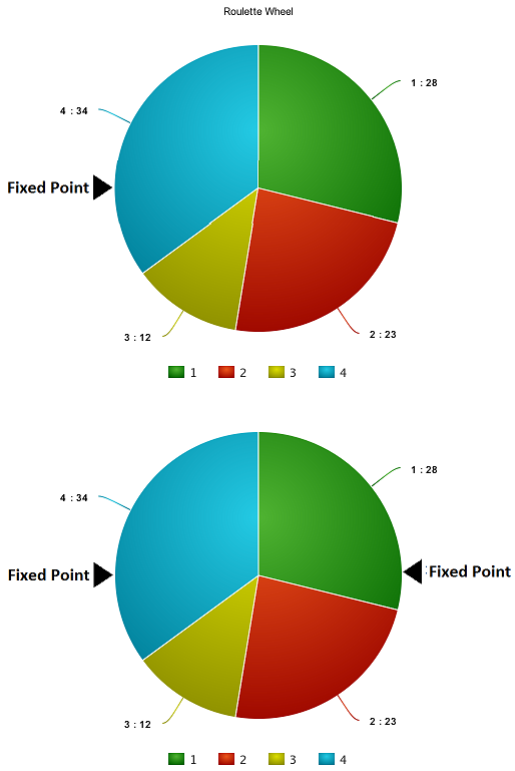
	POINTS DE SURVIE	*POURCENTAGE*
chromosome 1	28	28.9%
chromosome 2	23	23.7%
chromosome 3	12	12.4%
chromosome 4	34	35.1%

Sur la base de ces valeurs, créons notre roue de roulette.

Donc, maintenant cette roue est tournée et la région de roue qui vient devant le point fixe est choisie comme parent. Pour le deuxième parent, le même processus est répété.

Parfois, nous marquons deux points fixes comme le montre la figure ci-dessous.

Ainsi, dans cette méthode, nous pouvons obtenir nos deux parents en une seule fois.

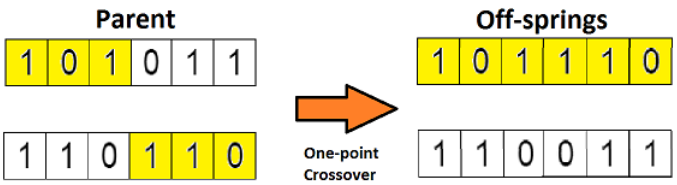


Cette méthode est connue sous le nom de méthode de **sélection universelle stochastique**.

Crossover

Donc, dans cette étape précédente, nous avons sélectionné des chromosomes parents qui produiront des rejetons.

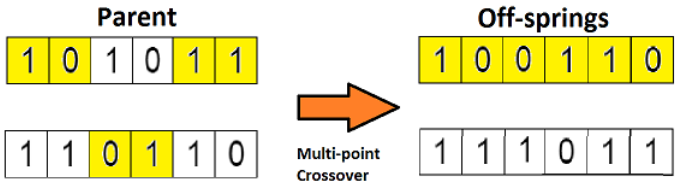
Donc, en termes biologiques, le croisement n'est rien d'autre que la reproduction.



Il s'agit de la forme la plus élémentaire de croisement, connue sous le nom de croisement à un point. Ici, nous sélectionnons un point de croise-

ment aléatoire et les queues des deux chromosomes sont permutées pour produire un nouveau rejet.

Si vous prenez deux points de croisement, alors il sera appelé en tant que croisement multipoint, comme indiqué ci-dessous.

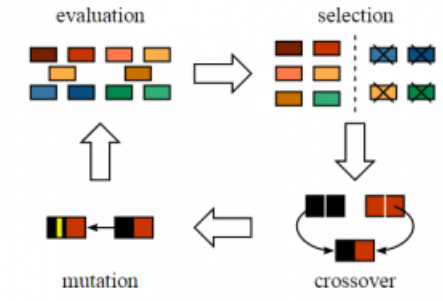


Mutation

Maintenant, si vous pensez au sens biologique, les enfants produits ont-ils les mêmes traits que leurs parents ? La réponse est non. Au cours de leur croissance, il y a un changement dans les gènes des enfants qui les rend différents de ses parents. Ce processus est connu sous le nom de mutation, qui peut être définie comme une modification aléatoire du chromosome, ce qui favorise également l'idée de diversité dans la population.

Une méthode simple de mutation est illustrée dans l'image ci-dessous.

L'ensemble du processus est donc résumé comme le montre la figure ci-dessous.



Les rejetons ainsi produits sont à nouveau validés en utilisant notre fonction de fitness, et s'ils sont considérés comme aptes, ils remplaceront les chromosomes les moins ajustés de la population.

Mais la question est de savoir comment nous saurons que nous avons atteint notre meilleure solution possible ?

Donc, fondamentalement, il existe différentes conditions de résiliation, qui sont énumérées ci-dessous :

1. Il n'y a pas d'amélioration de la population pendant plus de *x* itérations.

2. Nous avons déjà prédéfini un nombre absolu de génération pour notre algorithme.

3. Lorsque notre fonction fitness a atteint une valeur prédéfinie.

2.2.5 Modèles de prédiction (séries temporelles)

l'initialisation

Après avoir choisi une méthode de représentation, l'étape suivante Consistera à choisir la taille de la population et une méthode pour générer la population initiale. La population initiale devrait disposer d'un réservoir de gènes aussi grand que possible afin de pouvoir explorer l'ensemble de l'espace de recherche et, par conséquent, la population initiale a été générée en tenant compte cinq individus. Chaque individu est un modèle de prévision de séries chronologiques, c'est-à-dire qu'il possède un de ses gènes dont la valeur est égale à un et les autres gènes dont la valeur est égale à zéro.

Evaluation

L'évaluation est un processus d'attribution d'une valeur à chaque individu selon une fonction fitness. Cette valeur montre la bonté d'un individu. Dans ce cas, nous proposons une métrique nommée indice d'élasticité (EI) comme une fonction fitness. Le but est de maximiser la somme des EI (SEI) dans les prévisions précédentes. L'équation 2.1 montre la somme de l'EI (SEI). Nous pouvons voir dans l'équation 2.1 que l'IE peut supposer des valeurs comprises entre 0 et 1. Plus la valeur est proche de 1,meilleure est la solution. Le paramètre NE représente le nombre de prédictions passées. Ainsi, le meilleur individu (solution) est celui qui maximise la somme

d'Ei.La figure 2.4 illustre le processus d'évaluation.

$$SEI = \sum_{i=1}^{N} \frac{min(V_{exp,i}, V_{pred,i})}{max(V_{exp,i}, V_{pred,i})} \tag{2.1}$$

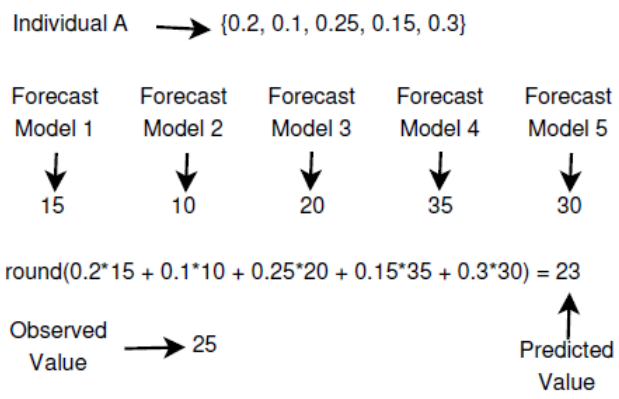


FIGURE 2.4 – Exemple d'evaluation

Dans la figure 2.4, nous pouvons voir comment les modèles de prévision sont combinés en utilisant les gènes individuels comme poids pour les valeurs. Après cela, nous avons la valeur prédite et nous pouvons utiliser l'équation 2.1 pour calculer l'EI. Ce processus est répété pour chaque individu dans toutes les prédictions précédentes, et chaque valeur d'EI est ajoutée aux valeurs précédentes.L'individu avec la valeur la plus élevée de la SEI est le mieux placé pour résoudre le problème.Nous voulons utiliser la métrique EI comme fonction Fitness car elle favorise le modèle qui a dans la plupart du temps les meilleures performances, c'est-à-dire qu'il est moins sensible aux valeurs aberrantes.

Selection

Le processus de sélection consiste à choisir deux parents parmi la population pour le croisement. Dans ce cadre, nous avons choisi les 50 meilleures individus classées à l'étape de l'évaluation.

Crossover

Il est fait pour explorer de nouvelles solutions. Cet opérateur modifie les parties définies de deux membres qui sont sélectionnées et obtient

différents membres qui donnent de nouveaux points dans l'espace de recherche. Dans ce cas, l'opérateur croisé choisi était la moyenne arithmétique des valeurs attribuées aux gènes de chaque participant individuel de l'intersection. Après que les parents ont été sélectionnés comme décrit ci-dessus, dans la phase de sélection, on aura la combinaison de tous les parents, deux par deux, ainsi le couple s'accouplera avec la probabilité de 0,90. La figure 2.5 montre une opération de croisement.

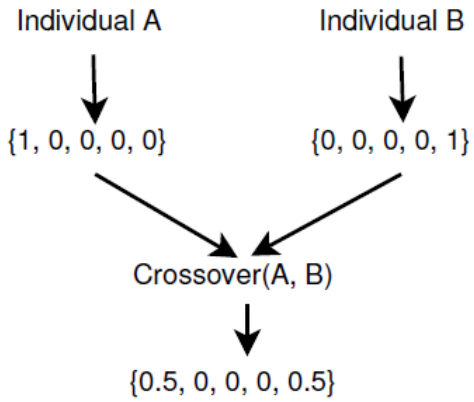


FIGURE 2.5 – Exemple de croisement

Mutation

L'opérateur de mutation est essentiel au succès d'AG car il détermine les directions de recherche et évite la convergence précoce. Cependant, contrairement au croisement, la mutation se fait généralement en modifiant les gènes d'un chromosome et sa probabilité est faible. ici, nous avons utilisé l'opérateur de swap comme opérateur de mutation avec une probabilité de 0.10. Dans l'opérateur d'échange, deux gènes sélectionnés aléatoirement échangeront leurs valeurs.

La figure 2.6 montre une opération de mutation.

FIGURE 2.6 – Exemple de mutation

Mettre à jour

À ce stade, les individus résultant du processus de croisement et de mutation sont entrés dans la population, selon la politique adoptée par l'AG, la population maintient une taille fixe et les individus sont créés en nombre égal à celui de leurs prédécesseurs et les remplacent complètement. Cependant, il existe des alternatives à cette approche, par exemple, tous les N meilleurs individus peuvent toujours être maintenus. Dans notre cas, nous voulons maintenir les individus originaux et leurs fils résultant du processus de croisement et de mutation.

Finition

La finition n'implique l'utilisation d'aucun opérateur génétique. Il est simplement composé d'un test qui donne l'ordre au processus d'évolution si l'AG a atteint un point d'arrêt prédéterminé. la règle d'arrêt est quand les individus cessent d'évoluer, c'est-à-dire quand la meilleure solution (individuelle) de la génération actuelle est la même que la génération précédente, ou lorsque l'algorithme atteint une centaine d'itérations.

2.2.6 Segmentation d'image

L'AG propose d'explorer l'espace des solutions, au sens que chaque pixel est groupé en d'autres pixels par une fonction de distance basée sur des segments déjà calculés à la fois locaux et globaux. Il y a deux problèmes fondamentaux à résoudre lors de la conception d'un AG pour la segmentation d'image, ce sont :

1) sélectionner une représentation appropriée pour chaque solution.

2) formuler des fonctions de fitness appropriées en termes du poids.

Presque tous les algorithmes de segmentation d'image contiennent des paramètres qui sont utilisés pour contrôler les résultats de la segmentation, le système génétique peut modifier dynamiquement les paramètres pour obtenir les meilleures performances. La segmentation d'image basée sur le processus de l'AG peut être résumée comme suit :

• Une fois la population initiale disponible, le cycle génétique commence comme le montre la figure 2.7 .

• La qualité des résultats segmentés pour chaque paramètre de l'ensemble est évalué. Si la qualité de segmentation maximale pour la population actuelle est supérieure à une valeur de seuil prédéfinie ou à un autre critère d'arrêt, alors le cycle se terminera. Alternativement, si les performances ne sont pas atteintes, les opérateurs de recombinaison génétique (croisement et mutation) sont appliqués aux individus à haute résistance dans la population, ce qui donne un nouvel ensemble de descendants avec des meilleures performances.

• La nouvelle population est renvoyée au processus de la segmentation d'image, et ainsi, le cycle recommence. Chaque passage dans la boucle (Segmentation, Fitness, Crossover et Mutation) est appelé génération. Le cycle illustré à la figure 2.7 se poursuit jusqu'à ce que les critères d'aptitude maximale ou de la fin soient atteints.

• La fonction fitness est liée à la variance interne de chaque segment d'image et a deux poids pour regrouper les pixels en sous-ensembles en fonction de la fonction de distance. Par conséquent, l'intensité des pixels et la position spatiale sont utilisées. ceci dit, la fonction de distance est basée

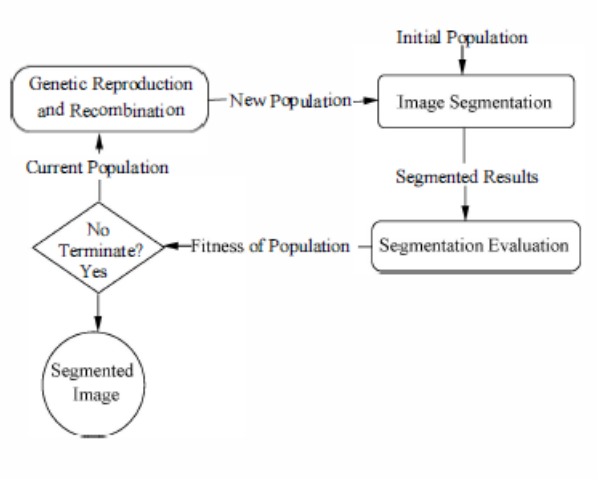


FIGURE 2.7 – Organigramme de la méthode de segmentation basée sur les AG

sur l'intensité des pixels et la position spatiale pour définir la similitude entre les pixels, comme le montre la figure 2.8 .

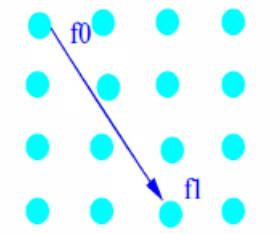


FIGURE 2.8 – Le graphique et la fonction distance

ALGORITHME DE SEGMENTATION

La segmentation d'une image numérique F peut être considérée comme un partitionnement naturel de ses pixels, basé sur une fonction de distance définie dans F. F de taille $(M \times Npixels)$ peut être décrit comme :

$$F = F(i,j), 1 \leq i \leq M, 1 \leq j \leq N, 0 \leq F(i,j) \leq C \quad (1)$$

où C est le nombre de prototypes de classe qui peuvent être sélectionnés librement, de sorte que, F est une collection de pixels où chaque valeur de pixel est proportionnelle au degré de similitude de région. Les

valeurs de pixel dans F sont normalisées dans la plage $[0, 1]$. Les pixels avec des valeurs proches de un sont de bons candidats pour être considérés comme des points de bord. Les régions où sous-ensembles de F sont désignés par : $S = (SI, ..., SK)$.

Prétraitement d'image La procédure de clustering, une étiquette est attribuée à chaque pixel qui prend une valeur entre 1 et K, telle que, chaque région possible de F peut être représenté par une matrice, et cela correspond à un pixel du sous-ensemble ou de la partition. F est associé au pondéré graphique G comme : $G \equiv< F, \delta >$. δ est la fonction de distance qui relie entre f_0 et f_1 comme le montre la figure 2.8 .

Partitionnement d'image L'algorithme de segmentation dépend de la partition des sommets du graphe pondéré $G = (V, E, 5)$ en K sous-ensembles. Un exemple de graphique pondéré de trois segments est présenté dans Figure 2.9 . Comme indiqué précédemment, $G = (V, E, 5)$, $S_i \cap S_j = \Phi$,

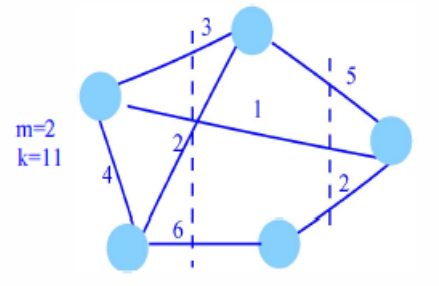


FIGURE 2.9 – Segmentation d'image et partitionnement du graphe

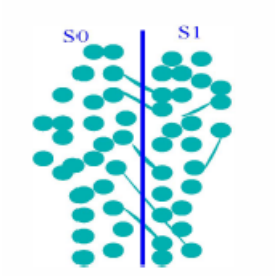


FIGURE 2.10 – Illustration du processus de bipartition entre deux sous-ensembles

et $S_i \cup S_j = F$, Par conséquent, nous voulons disjoindre S_i et S_j en supprimant les arêtes reliant les deux parties comme le montre la figure

2.10 . La coupe de S_i et S_j peut être définie comme indiqué dans l'équation 2 :

$$cut(S_i, S_j) = \sum_{i \in S_i j \in S_j} \delta(i,j) \qquad (2)$$

La partition optimale est celle qui minimise la fonction de distance (fonction de similarité). Un pixel $f \in F$ est représenté dans un espace de $3D$ avec (i_f, j_f, l_f). (i,j) représente la position du pixel et l représente son niveau de gris.

le regroupement des pixels en partitions est basé sur la fonction de distance calculée pour chaque partition comme défini dans l'équation 3.

$$\begin{aligned} \delta(i,j) &= w_0 \times \frac{|l_i - l_j|}{max(l_i, l_j)} \\ &+ w_1 \times \frac{1}{|N(i)|} \sum_{z \in N(i)} \frac{|l_z - l_j|}{max(l_z, l_j)} \end{aligned} \qquad (3)$$

$\omega_0, \omega_1 \geq 0$ et $\forall i,j \in F, i \sim j \iff \delta(i,j) \leq \Phi$

$N(i)$ est un voisinage du pixel i. ω_0 et ω_1 sont les poids attachés à sa fonction. Par conséquent, les plages de ω_0, ω_1 sont normalisées entre 0 et 1 et chaque caractéristique contribue également à la fonction de distance.

CONFIGURATION EXPÉRIMENTALE POUR LES AG

Pour une image d'entrée F de taille $N \times M$, $F(i,j)$ indique la valeur de niveau de gris du pixel situé à l'emplacement (i,j). La procédure proposée pour les AG basés sur la segmentation d'image comprend les étapes suivantes :

1. Codage des données Le chromosome génétique est codé par une chaîne binaire de 32 bits. Les 8 bits les moins significatifs identifient l'étiquette de pixel de la grappe à laquelle appartient le pixel après la procédure de regroupement. Alors que, la position de pixel (i,j) est codée dans les 24 bits les plus significatifs. Par conséquent, chaque chromosome peut être désigné par la paire d'ordre comme le montre la figure 2.11.

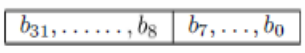


FIGURE 2.11 – Une représentation chromosomique

2. Fonction Fitness L'évaluation d'une technique de segmentation n'est pas une tâche facile, puisque le résultat de segmentation attendu dépend de l'application.

La fonction fitness de la qualité de segmentation varie d'une image à l'autre.

La fonction fitness a été définie sur la base de la fonction de distance calculée entre chaque chromosome et la région correspondante S. La fonction fitness a été définie avec deux poids caractérisés par ω_0 et ω_1. Dans cette étude, un indicateur de concordance entre l'humain et la classification automatique a été définie comme indiqué dans l'équation 4 :

$$\gamma = \frac{1}{K}\sum_{k=1}^{K}\beta_k \times \frac{\#agr_k}{P_k} \qquad (4)$$

où : $P_k = max(\mid Seg_k \mid, \mid S \mid)$, Seg_k désigne le k^{eme} segment récupéré par les humains, S désigne le k^{eme} segment récupéré par la machine, $\mid Seg_k \mid$ et $\mid S \mid$ désignent la taille correspondante et $\sharp agr_k$ est la plus grande intersection de pixels entre Seg_k et S.

3. Le Crossover et la Mutation Le croisement classique à point unique avec la probabilité de croisement P_c et La mutation binaire avec la probabilité P_m a évolué dans le système génétique proposé. Un exemple du croisement et la mutation binaire en point unique sont illustrés dans la figure 2.12 .

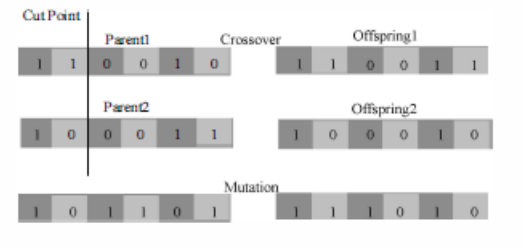


FIGURE 2.12 – Exemple d'opérations de croisement et de mutation

4. Processus de sélection Dans la population d'AG, $N \times M$ nombres de chromosomes sont générés à l'aide du nombre aléatoire, tels que :

$P(0)$ est la population au temps 0. le croisement et la mutation binaire en point unique sont appliqués à la population initiale pour obtenir une

$$P(0) = \{\nu(0), \nu(1), \ldots, \nu_{M \times N}(0) \qquad (5)$$

autre génération, $P(1)$, et ainsi de suite. La nouvelle population est déterminé par le processus de sélection de telle sorte que la nouvelle population est meilleure que l'ancienne en termes d'aptitude.

5. Condition de résiliation La procédure de l'étape 1 à l'étape 4 est répétée jusqu'à ce que la condition de convergence soit satisfaite ou que le nombre de générations soit atteint. Le processus de convergence est basé sur la variance totale de la région comme indiqué dans l'équation 6.

$$Var_t = \sum_{k=1}^{K} \sigma(k)^2 \qquad (6)$$

où, t est le numéro d'itération, σ^2 est la variance du niveau de gris de la région k, et K est le nombre total de régions. La boucle continuera jusqu'à $\mid Var_{t+1} - Var_t \mid \leq \epsilon$, où ϵ est une petit valeur. La méthode proposée de segmentation d'image basée sur l'AG est illustré dans la figure 2.13 .

for every $f \in F$
  if $\delta(f, l_{S_k})$=*min*$\{\delta(f, g_{S_i}),\ i = 1, 2, \ldots, K_{max}\}$
then,
  A single point crossover and bit mutation are applied to current population;
  A new population $P(t+1)$ is generated.
  assign (f, S_k)
  update $(\{\mu_k, \sigma_k\})$
  end
end
Increase iteration by one;
repeat until $(|Var_{t+1} - Var_t| \leq \epsilon)$

FIGURE 2.13 – Algorithme de segmentation d'image basé sur AG

Chapitre 3

Le voyageur de commerce

3.1 Généralisation

Le problème du voyageur de commerce : ce problème est un classique d'algorithmique. Son sujet concerne les trajets d'un voyageur de commerce. Celui-ci doit s'arrêter en plusieurs points, et le but de l'algorithme est d'optimiser le trajet de façon que celui-ci soit le plus court possible. Dans le cas où huit points d'arrêts existent, cela est encore possible par énumération (2520 possibilités : pour n arrêts, n supérieur ou égal à 3, il y a $\frac{(n - 1) !}{2}$ chemins possibles) mais ensuite, l'augmentation du nombre d'arrêts fait suivre au nombre de possibilités une croissance exponentielle.

Par le biais d'algorithmes génétiques, il est possible de trouver des chemins relativement corrects. De plus, ce type de problèmes est assez facile à coder sous forme d'algorithme génétique. L'idée de base est de prendre la longueur de chemin comme fonction d'évaluation. Pour effectuer le croisement de deux chemins :

- On recopie le premier chemin jusqu'à une « cassure ».

- On recopie ensuite les villes du second chemin. Si la ville est déjà utilisée, on passe à la ville suivante.

Chemin	Codage
A	***1 2 3 4*** : 5 6 7 8 9
B	***4 1*** 6 ***3*** : 9 8 ***2*** 5 7
fils	1 2 3 4 : 6 9 8 5 7

Soit un itinéraire qui contient 9 clients, on suppose que l'on croise les deux chemins suivants (un chiffre représente un client). On croise ces deux chemins après le locus 4 : on obtient le chemin fils.

3.2 Un algorithme génétique pour résoudre le problème du voyageur de commerce implémenté en Python 3

Nous utiliserons un AG pour trouver une solution au problème du voyageur de commerce. ce probleme est décrit comme suit : « Étant donné la liste des villes et les distances entre chaque paire de villes, quel est le plus court itinéraire possible pour visiter chaque ville et retourner à la ville d'origine ? »

À ce stade-là il y a deux règles importantes :

1. Chaque ville doit être visitée exactement une seule fois.

2. Nous devons retourner à la ville de départ, donc notre distance totale doit être calculée en conséquence.

et comme approche générale d'algorithme :

- **Gène :** une ville (représentée par les coordonnées (x, y))

- **Individu (ou « chromosome ») :** une seule voie répondant aux conditions ci-dessus

- **Population :** un ensemble de voies possibles (c.-à-d. un ensemble d'individus)

- **Parents :** deux itinéraires combinés pour créer un nouvel itinéraire

- **Bassin d'accouplement :** un ensemble de parents qui servent à créer notre prochaine population (créant ainsi la prochaine génération de routes)

- **Fitness :** une fonction qui nous dit à quel point chaque itinéraire est bon (dans notre cas, à quel point la distance est courte)

- **Mutation :** une façon d'introduire des variations dans notre population en échangeant au hasard deux villes dans un itinéraire

- **L'élitisme :** un moyen de porter les meilleurs individus dans la prochaine génération

On a déclaré les packages et les paramètres qu'on a besoin

```
import random
import copy
import os
import time
import math
import csv
import tkinter

list_of_cities = []
# probability that an individual Route will mutate
k_mut_prob = 0.4
# Number of generations to run for
k_n_generations = 100
# Population size of 1 generation (RoutePop)
k_population_size = 100
# Size of the tournament selection.
tournament_size = 7
# If elitism is True, the best from one generation will
   carried over to the next.
elitism = True
```

La classe **GA** pour réunir toutes les méthodes liées à l'algorithme génétique.

- $crossover()$: Retourne un itinéraire enfant de Route() après avoir reproduit les itinéraires des deux parents. Les itinéraires doivent être de même longueur.

- $mutate()$: Échange deux index aléatoires dans $route\_to\_mut.route$

- $tournament\_select()$: Sélectionner aléatoirement la taille du tournoi $tournament\_size$ une quantité de $Routes()$ à partir de la population d'entrée. Prend le plus fort du plus petit nombre de $Routes()$.

```
# Class GA
class GA(object):


    def crossover(self, parent1, parent2):


        child_rt = Route()

        for x in range(0, len(child_rt.route)):
            child_rt.route[x] = None

        start_pos = random.randint(0, len(parent1.route))
        end_pos = random.randint(0, len(parent1.route))

        if start_pos < end_pos:
            # do it in the start-->end order
            for x in range(start_pos, end_pos):
                child_rt.route[x] = parent1.route[x]  # set
    the values to eachother
        # if the start position is after the end:
        elif start_pos > end_pos:
            # do it in the end-->start order
            for i in range(end_pos, start_pos):
                child_rt.route[i] = parent1.route[i]  # set
    the values to eachother

        for i in range(len(parent2.route)):
            # if parent2 has a city that the child doesn't
    have yet:
            if not parent2.route[i] in child_rt.route:
                # it puts it in the first 'None' spot and
    breaks out of the loop.
                for x in range(len(child_rt.route)):
                    if child_rt.route[x] is None:
                        child_rt.route[x] = parent2.route[i]
                        break

        child_rt.recalc_rt_len()
        return child_rt

    def mutate(self, route_to_mut):

        # k_mut_prob %
        if random.random() < k_mut_prob:

            # two random indices:
            mut_pos1 = random.randint(0, len(route_to_mut.
    route) - 1)
```

```
45              mut_pos2 = random.randint(0, len(route_to_mut.
    route) - 1)
46
47              # if they're the same, skip to the chase
48              if mut_pos1 == mut_pos2:
49                  return route_to_mut
50
51              # Otherwise swap them:
52              city1 = route_to_mut.route[mut_pos1]
53              city2 = route_to_mut.route[mut_pos2]
54
55              route_to_mut.route[mut_pos2] = city1
56              route_to_mut.route[mut_pos1] = city2
57
58          # Recalculate the length of the route (updates it's .
    length)
59          route_to_mut.recalc_rt_len()
60
61          return route_to_mut
62
63
64      def tournament_select(self, population):
65
66          tournament_pop = RoutePop(size=tournament_size,
    initialise=False)
67
68          for _ in range(tournament_size - 1):
69              tournament_pop.rt_pop.append(random.choice(
    population.rt_pop))
70
71          # returns the fittest:
72          return tournament_pop.get_fittest()
```

Exemple

trouvons le plus court chemin passé par ces villes en utilisant l'application des AG.

ville	x	y
oujda	346	-19
berkane	348	-23
taourirt	344	-28
taza	342	-40
figuig	320	-12
nador	351	-29
jerada	343	-21
ain beni mathar	340	-20
zaio	350	-27
lamrija	340	-32

```
Finished evolving 100 generations.
Elapsed time was 3.5 seconds.

Initial best distance: 106.91
Final best distance:   95.26
The best route went via:
    figuig,ain beni mathar
,jerada,oujda,berkane ,zaio,nador,taourirt ,taza,lamrija
```

c’est les résultats que nous avons trouvé.

Conclusion

Dans le cadre de notre projet de fin d'année, nous avons étudié les algorithmes génétiques.

Dans un premier temps, nous avons commencé par définir les algorithmes évolutionnistes d'une façon générique et ses origines, ensuite nous avons défini quelques algorithmes de la même famille comme la programmation génétique, la programmation évolutionnaire et la stratégies d'évolution. dans un deuxième temps nous avons détaillé les algorithmes génétiques en citant quelques avantages et inconvénients de ces derniers et aussi quelques domaines d'application. À la fin, nous avons conçu une application en python qui sert à résoudre le problème du voyageur de commerce en utilisant ces algorithmes à savoir leur capabilité d'optimisation.

En guise de conclusion, les algorithmes génétiques sont un outil d'optimisation robuste, capable de résoudre des problèmes complexes et nous donne une vision sur la puissance de l'aléatoire.

Perspectives

Et parce qu'on ne peut jamais tout aborder, les perspectives de notre projet s'étendent incontestablement en large et en travers sur bons nombres de points on cite :

- Une étude sur l'optimisation multi-objectif/multicritère.
- Autres variantes des AG tel :
  - NPGA :Algorithme génétique de Niched Pareto.
  - NSGA II :Algorithme génétique de tri non donné.
- L'optimisation de Réseaux de Neurones en Utilisant les AG(Optimiser les poids des interconnections entre neurone dans le cas d'un réseau de neurones multicouches, cas d' un réseau de neurones convolutifs (Optimisation de l'architecture :Nombre de couche).

Références

- J. H. Holland, Adaptation In Natural And Artificial Systems, University of Michigan Press (1975)

- « La robotique évolutionniste » dans Pour la Science, no 284, p. 70 (juin 2001)

- (en) M. Mitchell, An Introduction to Genetic Algorithm, MIT Press (1996)

- « Comment l'ordinateur transforme les sciences » dans Les Cahiers de Science et Vie

- D. Goldberg, Algorithmes génétiques, Addison–Wesley France (1994)

- (en) R. Poli, W. B. Langdon et N. F. McPhee (trad. de l'anglais), A Field Guide to Genetic Programming

- http ://www.jade-cheng.com/au/coalhmm/optimization/

- https ://www.tutorialspoint.com/genetic_algorithms /genetic_algorithms_fundamentals.htm

- https ://www.researchgate.net/publication/286650215_Combining_time _series_prediction_models_using_genetic_algorithm_to_autoscaling_Web_applications _hosted_in_the_cloud_infrastructure

- https ://github.com/maoaiz/tsp-genetic-python