



Master 1 MIAGE

Projet d'Intelligence Artificielle

Réalisé par :

HOUMEL Idir

MESSALI Nassim

Sous la direction de :

Mr AIRIAU Stéphane

09 décembre 2018

I). Explication du code réalisé :

Dans le but, d'obtenir un code lisible, bien structuré et évolutif nous avons décidé de créer un ensemble de classe utilisable par les différents algorithmes implémentés pour résoudre notre problème :

Classe StateForImage : Cette classe nous permet de récupérer les différentes caractéristiques de notre image cible (à savoir Mona Lisa) dont le tableau des couleurs de ses pixels ainsi que sa largeur et sa hauteur.

Classe State : Un State est une représentation de l'état courant de l'avancée de l'algorithme. Un State ne peut représenter qu'un seul état (un ensemble de Convexpolygon avec des caractéristiques bien définies), donc dès que l'on change une caractéristique, il faut créer un nouveau State.

Classe Technique : Dans le but, de factoriser notre code, nous avons créer cette classe dont toutes les techniques implémentées héritent. En effet, tous les algorithmes implémentés, utilisent une méthode (changeState()) qui permet de modifier aléatoirement un State donné en paramètre, ainsi qu'une méthode abstraite execute() qui est le cœur de nos algorithmes.

Classe Generator : Elle contient deux méthodes essentielles que sont :

- getInitialState () : qui génère un State composé dans notre cas de ConvexPolygon (appelée par l'algorithme HillClimbing).
- getInitialPopulation(n) : qui génère une liste de States dont le nombre est passé en paramètre (appelée par l'algorithme GenetiqueDist).

Algorithme HillClimbing :

Cette méthode consiste simplement :

1/ à générer à partir d'un State initial un ensemble de voisins (States) qui correspondent à une modification aléatoire d'une caractéristique (à savoir la position d'un sommet, la transparence et la couleur) d'un polygone du State initial (ceci réalisé grâce à la fonction changeState()).

Pour pouvoir contrôler l'avancement de ces modifications, nous avons mis en place des ratios pour que ces dernières soient légères. Ainsi le paramètre *deltaratio* définit l'incrément maximal à utiliser pour modifier un sommet. On définit cet incrément maximal comme un ratio de la taille maximale de l'image. Le paramètre *deltaratio01* définit quant à lui l'incrément maximal à utiliser pour modifier une grandeur dont la valeur est dans l'intervalle [0,1] (à savoir la transparence et le R, G, B).

2/Une fois le meilleur voisin choisi (grâce à la méthode successor ()) on compare sa distance par rapport à l'image cible avec la distance du State initial. Si elle est inférieure à celle du State initial, on recommence l'algorithme avec le voisin comme état courant. Dans le cas contraire, pour éviter que l'algorithme s'arrête et rende un State qui n'est pas satisfaisant, nous avons utilisé des redémarrages poussant l'algorithme à générer d'autres successeurs jusqu'à un seuil que l'on fixe nous-même (threshold).







Algorithme GénétiqueDist :

Cette méthode consiste quant à elle à :







- 1/ Générer une population aléatoirement en faisant appel à `getInitialPopulation(n)`.
- 2/ Trier cette population par ordre de distance croissante pour faciliter le choix des meilleurs individus.
- 3/ Générer un nombre d'enfants égal à la taille de la population initiale. Ceci en réalisant des croisements sur deux parents, choisis aléatoirement parmi une sélection des meilleures individus de notre population (grâce à la fonction `randomSelection()`).
- 4/ Selon une probabilité passée en paramètre, nous allons effectuer des mutations sur les enfants (cette probabilité est fixée de préférence à une valeur élevée car c'est ce qui permet de faire évoluer la population de manière significative).
- 5/ Ajouter les enfants, à la nouvelle population correspondant à une liste déjà créée, pour éviter d'écraser la population des parents.
- 6/ Une fois tous les enfants générés, notre nouvelle population devient notre population courante.
- 7/Récupérer le meilleur élément de notre population grâce à la méthode `getBestStateOfPopulation()` et tester si sa distance est inférieure à celle souhaitée (fixée par le seuil passé en paramètre). Dans ce cas on arrête et on rend ce State, sinon on réitère l'opération avec la nouvelle population.

II). Présentation des résultats obtenus :

Algorithme HillClimbing :

Paramètres	Distance/temps d'exécution	Image de départ	Image obtenue
Nombre Polys : 50 Deltaratio = 7/10 Deltaratio01 = 7/10 Threshold = 10 Nombre voisins = 50	Distance = 14,37 Temps = 1h30 (Taille image : 100x149)		
Nombre Polys : 500 Deltaratio = 7/10 Deltaratio01 = 7/10 Threshold = 10 Nombre voisins = 50	Distance = 9.99 Temps = 20h (Taille image : 100x149)		
Nombre Polys : 50 Deltaratio = 7/10 Deltaratio01 = 7/10 Threshold = 10 Nombre voisins = 50	Distance = 32,01 Temps = 8h (Taille image : 200x298)		

Algorithme GénétiqueDist :

Paramètres	Distance/temps d'exécution	Image de départ	Image obtenue
Nombre Polys : 50 Deltaratio = 7/10 Deltaratio01 = 7/10 Threshold = 10 Taille population = 50	Distance = 17,66 Temps = 6h (Taille image : 100x149)		
Nombre Polys : 500 Deltaratio = 7/10 Deltaratio01 = 7/10 Threshold = 10 Taille population = 50	Distance = 13,82 Temps = 21h		
Nombre Polys : 50 Deltaratio = 7/10 Deltaratio01 = 7/10 Threshold = 10 Taille population = 50	Distance = 40,01 Temps = 10h (Taille image : 200x298)		

Ainsi, d'après les résultats obtenus, nous pouvons remarquer que pour les mêmes paramètres nous obtenons de meilleurs résultats et plus rapidement avec l'algorithme HillClimbing, qu'avec l'algorithme génétique. Cela peut s'expliquer notamment du fait qu'au niveau de HillClimbing, notre état courant est forcément meilleur que son prédécesseur. Ceci, étant l'essence même de cet algorithme, qui teste après chaque nouvelle génération d'un voisin, si ce dernier est meilleur que l'état courant. Ce qui n'est pas forcément le cas avec l'algorithme génétique qui se base plus sur de l'aléatoire (par exemple le meilleur des pères peut être mieux que le meilleur des fils, malgré cela on continue notre algorithme avec les fils).

III). Discussion sur le projet et les difficultés rencontrées :

La première difficulté rencontrée réside dans le choix de l'algorithme à implémenter pour essayer de résoudre le problème posé. Pour cela de nombreuses recherches sur le net, ainsi que dans les livres ont été menées, pour essayer de cerner le problème en analysant et en comprenant les algorithmes susceptibles de satisfaire notre besoin.





Après cela, une autre difficulté résidait dans le fait de comprendre les erreurs générées par JavaFx notamment lors de l'import des fonctionnalités de JavaFx sous Eclipse.

Parmi les problèmes rencontrés et qui demeurent pour le moment non résolus on trouve :

→ Au niveau de la méthode `changeState()`, lorsqu'on modifie la position d'un sommet d'un polygone, nous n'arrivons pas à limiter cette modification de telle sorte que l'on évite de se retrouver avec des polygones non convexes lors de la génération d'un nouveau State. C'est pour cela que nous utilisons uniquement des triangles (qui sont nécessairement toujours convexe)

→ Au niveau de l'algorithme génétique nous avons été obligés d'écrire une variante. En effet dans cette dernière on ne choisit pas les parents de l'ensemble de la population, mais d'une sélection des meilleurs individus. Cela étant dû au fait que nous n'arrivons pas à trouver la définition précise de la fonction qui pour chaque distance nous retourne la probabilité associée.

Résultats complémentaires :

Paramètres	Distance/temps d'exécution	Image originale	Image obtenue
Nombre Polys : 500 Deltaratio = 7/10 Deltaratio01 = 7/10 Threshold = 10 Nombre voisins = 50	Distance = 54,23 Temps = 24h (Taille image : 300x149)		
Nombre Polys : 50 Deltaratio = 6/10 Deltaratio01 = 8/10 Threshold = 10 Nombre voisins = 50	Distance = 16,70 Temps = 40min		

Concernant la 1ere image on a utilisé d'autres polygones (pas nécessairement convexes). Cependant on a dû l'arrêter faute de temps.