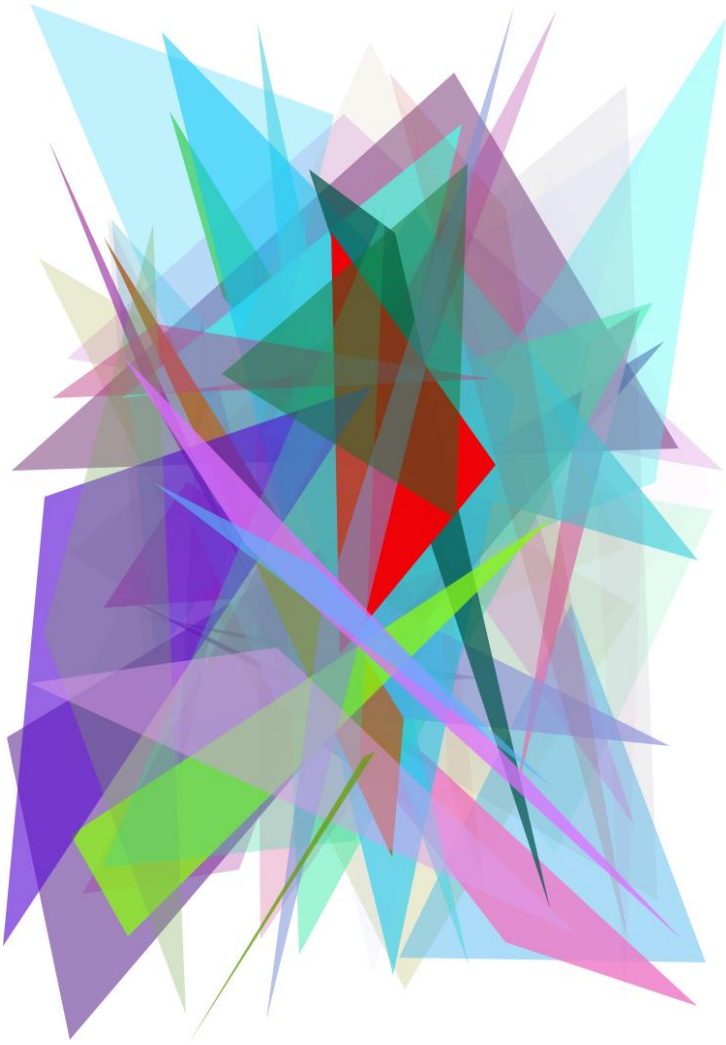
PROJET D'INTELLIGENCE ARTIFICIELLE

Dominika BANKIEWICZ & Aymeric MISLAH

2018 - 2019

APPROXIMER UNE IMAGE



MIDO
Mathématiques et Informatique
de la décision et des organisations

DAUPHINE UNIVERSITÉ PARIS | PSL

1. Description de l'algorithme

Notre algorithme se base sur le principe de l'algorithme glouton (greedy algorithm). A chaque étape, nous faisons un choix localement optimal dans l'espoir que ce choix nous mènera vers une solution globalement optimale.

Au départ, nous générons aléatoirement un individu composé de 50 polygones de k côtés. Ensuite, nous mutons cet individu afin d'en générer un nouveau. Si l'erreur de ce dernier est inférieure à celle de l'individu de base, nous le conservons. Sinon, nous recommençons. Et nous appliquons cela indéfiniment.

La notation:
g : valeur de l'erreur désirée
Img : image composée de 50 polygones
T : image à approximer

a. La fonction de l'erreur

La fonction de l'erreur est donnée par la formule suivante :

$$\text{erreur} = (\Sigma\Sigma\ (r(Img\_i\_j) - r(T\_i\_j))^2 + (g(Img\_i\_j) - g(T\_i\_j))^2 + (b(Img\_i\_j) - b(T\_i\_j))^2\ )^{0.5}$$

La somme suit la longueur et la largeur de l'image avec :

Img_i_j - pixel (i,j) de Img
T_i_j - pixel (i,j) de T
r(pixel): fonction qui retourne la valeur de couleur rouge
g(pixel): fonction qui retourne la valeur de couleur vert
b(pixel): fonction qui retourne la valeur de couleur bleue

b. Le pseudo-code pour approximer l'image Img:

```
function APPROXIMER(IMG, T, g)
returns new image
while(erreur(Img) > g)
{
     mut <- mutation(Img)
     if erreur(mut) < erreur(Img)
        Img <- muté
}
returns Img
```

c. La fonction de mutation

La fonction de mutation sélectionne au hasard un des 50 polygones. Ensuite, elle applique de manière aléatoire l'un des 4 types de mutation possible :

- changement d'opacité
- changement de valeur de la couleur rouge
- changement de valeur de la couleur verte
- changement de valeur de la couleur bleue
- changement d'un des sommets du polygone

En ce qui concerne la mutation qui modifie la position d'un des sommets du polygone, deux cas sont possibles :

- si le polygone n'a que trois côtés, alors il est forcément convexe ce qui nous permet de déplacer l'un de ses sommets librement.
- si le polygone a quatre côtés ou plus, alors il peut ne pas être convexe, donc au lieu de déplacer l'un de ses sommets, nous le remplaçons par un nouveau polygone de même couleur dont le nombre de côtés est réattribué aléatoirement.

d. Les raisons qui font que l'algorithme fonctionne

A chaque génération, nous ne progressons que si nous obtenons une erreur inférieure à celle de la génération en cours. Nous naviguons donc d'un minimum local à un autre en nous nous rapprochant de plus en plus du minimum global. Toutefois, en agissant de la sorte, on prend le risque d'être rapidement piégé dans un minimum local. C'est pour cette raison que nous avons décidé de ne pas muter complètement le polygone désigné. Au contraire, nous ne modifions qu'un seul de ses paramètres tout au long du processus, nous permettant ainsi de progresser très rapidement jusqu'à l'optimum.

2. Les résultats de l'algorithme

a. Performance de l'algorithme

Le tableau ci-dessous indique le temps approximatif nécessaire pour trouver une solution à un niveau d'erreur donnée pour l'image « **monaLisa-100.jpg »** :

Niveau d'erreur	Durée en minutes
21	~1
16	~2
13	~5

Et pour « **monaLisa-200.jpg** » :

Niveau d'erreur	Durée en minutes
65	~1
32	~5
30	~10

3. Les difficultés rencontrées

Au départ, nous avons essayé d'utiliser un algorithme génétique classique. Nous pensions qu'en ayant une population donnée d'images, nous pourrions progresser de manière significative par le biais de croisements et de mutations. Toutefois, nous nous sommes rendu compte que cette méthode nous menait dans une impasse. Nous pensions alors que la faute incombée à nos fonctions car ces dernières étaient rudimentaires. Pour faire court, nous sélectionnions les meilleurs parents de manière élitiste, puis nous les croisions par rapport à un pivot. Ensuite, nous avions un taux de mutation plus ou moins élevé qui déterminait combien d'enfants verraient l'un de ses polygones remplacés par un autre. Nous avons alors œuvré chacun de notre côté pour complexifier les fonctions et amener davantage de diversité dans nos populations. Malheureusement, aucune de nos méthodes n'a été concluante. Quoi que nous fassions, nous nous retrouvions toujours bloqué dans un minimum local impossible à franchir.

Après mûres réflexions, nous avons remarqué que l'algorithme génétique n'était pas parfaitement adapté à notre problème si nous le prenions tel quel car le gène utilisé n'est qu'une somme abstraite de polygones transparents. En d'autres termes, il n'y a aucun d'ordonnancement ce qui rend les croisements obsolètes. Afin d'illustrer cela, considérons deux images dont le niveau d'erreur est équivalent. Si nous nous attardons sur leur gène respectif, il y a de grandes chances que nous constations une dissemblance. Ainsi, nous pourrions les croiser dans tous les sens qu'à l'arrivée, nous n'aurions aucun enfant sensiblement meilleur. Ayant compris cela, nous avons alors remis en question l'utilisation d'une population de grande taille, car si les croisements deviennent inutiles, faire concourir plusieurs images l'est également.

C'est pour cette raison que nous nous sommes résolus à n'utiliser qu'un seul individu qui mute jusqu'à l'optimum. Ce changement de cap nous a permis de faire diminuer notre erreur minimum de 37 à 30 mais nous n'étions toujours pas satisfaits de la vitesse de l'algorithme. Le problème, c'était que nous remplacions le polygone désigné par un autre qui était généré de manière aléatoire. Par conséquent, quand nous arrivions à un minimum local, les transformations effectuées étaient bien trop violentes pour espérer une diminution de l'erreur. Dès lors, nous nous sommes mis à chercher comment est-ce que nous pourrions modifier que très légèrement les paramètres du polygone. C'est véritablement cela qui nous a permis d'atteindre notre objectif, à savoir implémenter un algorithme aussi puissant que rapide.

4. Annexes

a. Les résultats pour l'image « monaLisa-200.jpg » :

Erreur = 80

Erreur = 70

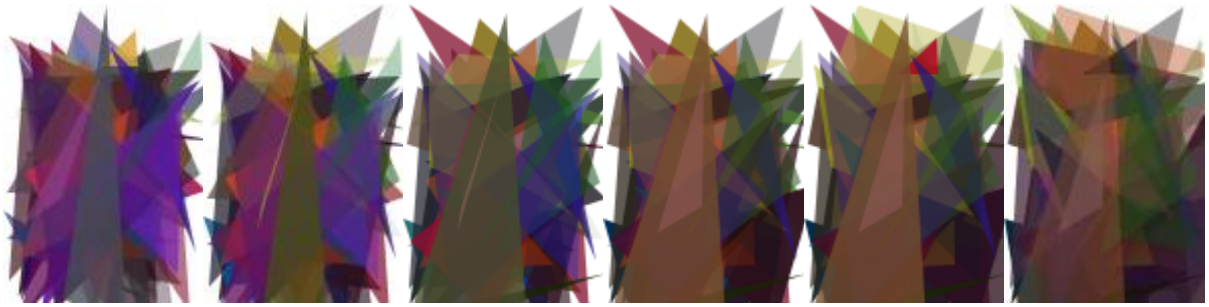
Erreur = 60

Erreur = 50

Erreur = 40

Erreur = 30

b. Les résultats pour l'image « monaLisa-100.jpg » :



Erreur = 80 **Erreur = 75** **Erreur = 70** **Erreur = 65** **Erreur = 60** **Erreur = 55**

Erreur = 50 **Erreur = 45** **Erreur = 40** **Erreur = 35** **Erreur = 30** **Erreur = 25**

Erreur = 20 **Erreur = 18** **Erreur = 16** **Erreur = 14**