Algorithme génétique

Groupe 4

Romain Ageron, Tom De Coninck, Paul Dufresne, Thibaut Pellerin

1 février 2021

CentraleSupélec

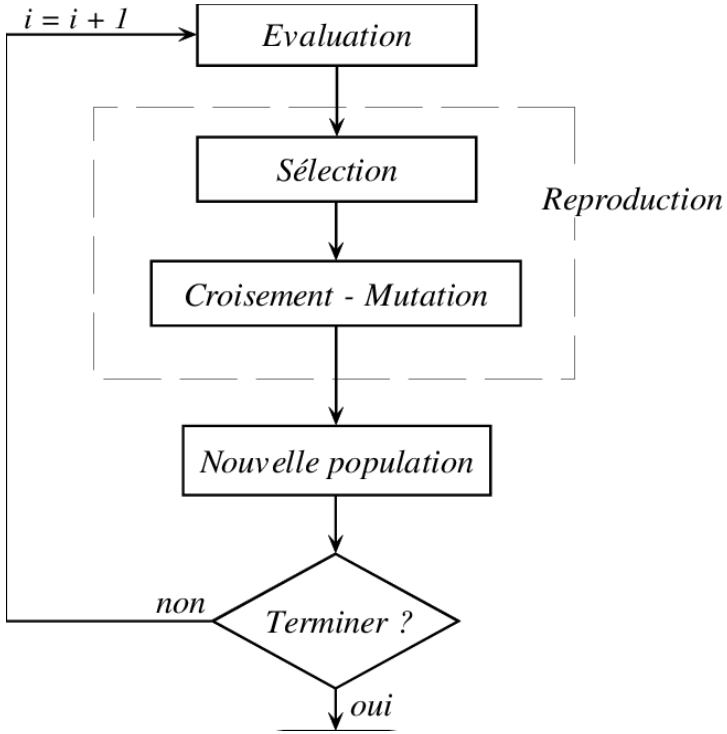
On dispose d'une chaîne de nucléotides (A,T,C,G).

- chaque dinucléotide (ex : "AT") doit avoir une certaine orientation dans l'espace
- on veut trouver les orientations qui rendent la trajectoire associée à la séquence **circulaire**
- nos variables sont les orientations, les individus seront donc des tables d'orientations
- on dispose de mesures expérimentales, les orientations doivent rester dans un intervalle autour de ces mesures

Paires/orientations	Twist	Wedge	Direction
AA	35.62	7.2	-154
AC	34.4	1.1	143
AG	27.7	8.4	2
...	...	...	...

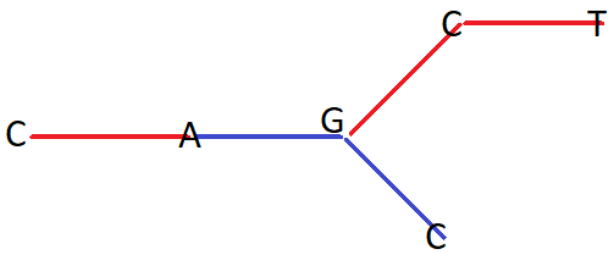
On peut réduire le nombre de variables en exploitant les symétries : $AG^{-1} = CT$ et AG et CT ont même Twist, Wedge et leurs Directions sont opposées : on a finalement **20 variables**.

Une algorithme génétique est une méthode d'**optimisation** qui s'inspire de la sélection naturelle.

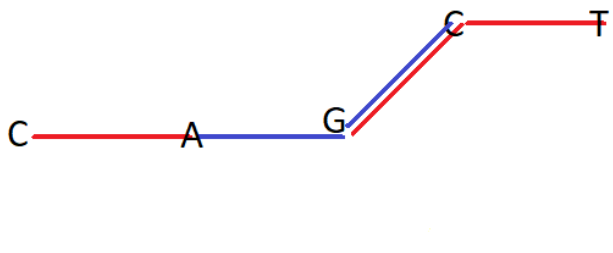


Les différentes étapes d'un algorithme génétique sont :

- **évaluation** : attribuer un score à un individu selon l'objectif du problème
- **sélection** : conserver certains individus via différentes méthodes (classement, tournoi...)
- **croisement** : créer de nouveaux individus à partir de ceux sélectionnés (permet de creuser un minimum)
- **mutation** : modifier aléatoirement certains individus (permet notamment de sortir d'un minimum local)



(a) Bonne distance mais mauvaise orientation



(b) Bonne distance et bonne orientation

Jonction pour une chaîne $GCT...CA$

Pour contrôler la qualité de la trajectoire, on ajoute (en bleu) les deux premiers nucléotides à la fin de la chaîne. Ici, G, le premier nucléotide, est bien placé mais l'orientation de GC n'est pas forcément valide.

AG	Entity	RotTable
population pop_number	param fitness	OriginalRotTable interval
tournament reproduction	fitness_calculation mutation	getTwist rot_to_entity

Les deux premières classes sont génériques, et `RotTable` est propre au problème du plasmide circulaire. `AG` et `RotTable` font appel à `Entity`.

Voici les fonctions qui correspondent aux différentes étapes de l'algorithme :

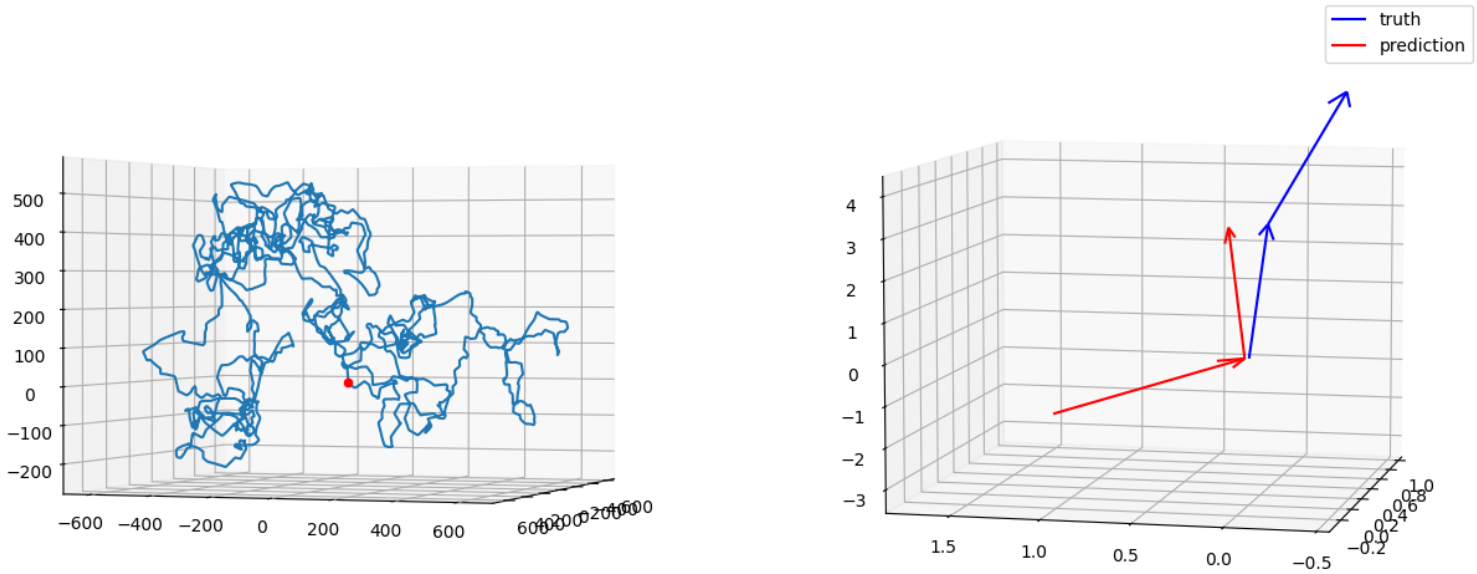
- la méthode `fitness_calculation` estime l'erreur de distance d et d'orientation o, et renvoie $(1 + d) \times (1 + o)$
- pour la sélection, nous utilisons un tournoi probabiliste
- la reproduction est assez simple : dans un premier temps nous avons fait la moyenne de deux tables
- `mutation` modifie une des variables d'un individu
- enfin, `global_mutation` fais muter une partie de la population

L'étape de mutation s'est révélée très importante :

- elle permet d'explorer l'espace du problème
- des mutations trop limitées bloquent l'algorithme dans des minimums locaux
- des mutations excessives risquent de gâcher des individus prometteurs

Nous avons choisi de faire muter une part importante de la population, mais pas les individus ayant les meilleurs scores. La mutation modifie une des variables, aléatoirement dans l'intervalle associé.

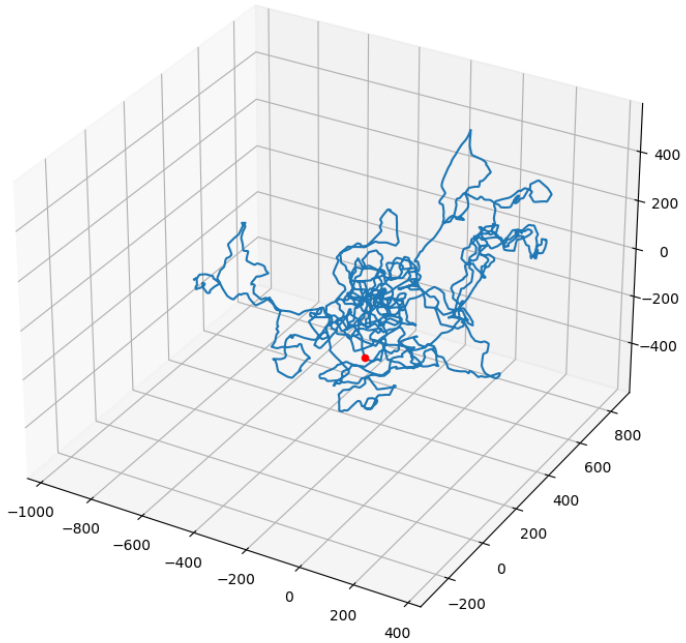
La trajectoire totale est représentée à gauche, et la jonction à droite (un vecteur = un dinucléotide).

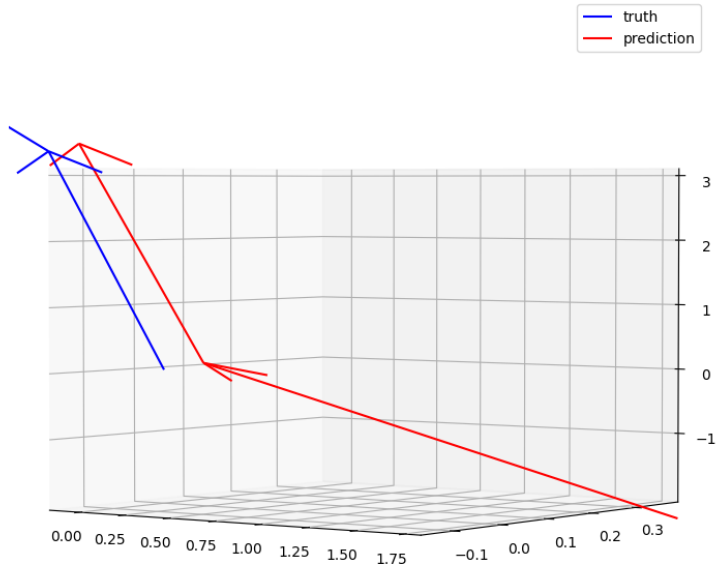


Une chaîne de longueur 8000, avec 200 générations et une population de 1000

Ici, l'erreur de position est faible mais celle d'orientation non négligeable.

On peut modifier légèrement la fonction `fitness_calculation` afin de pénaliser davantage l'erreur sur l'angle.

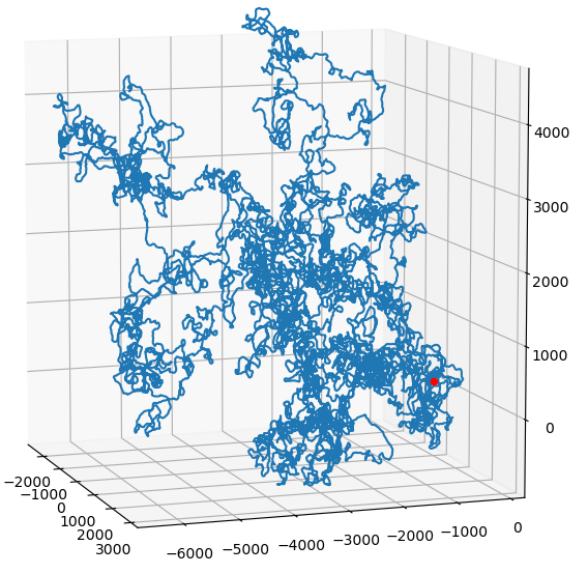


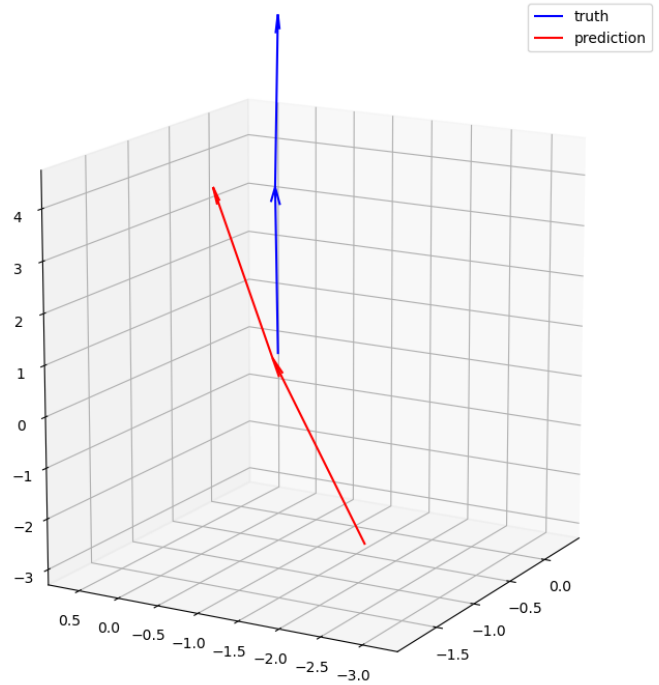


Une chaîne de longueur 8000, avec 250 générations et une population de 200

Les dinucléotides réel et fictif sont ici parallèles, mais la distance est plus importante.

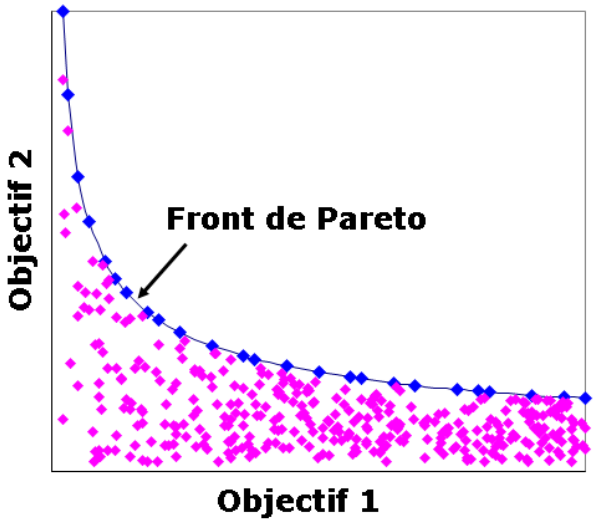
Après une bonne heure de calcul, on obtient ceci pour une chaîne de 180000 nucléotides.





- Le temps d'exécution est d'environ 3 minutes pour une chaîne de longueur 8000, une population de 100 et une centaine de générations
- L'accumulation d'erreurs numériques rend les résultats pour la chaîne de 180k peu fiables
- Difficile d'équilibrer la fonction `fitness_calculation` afin d'avoir une bonne distance et une bonne orientation

Ici, l'optimisation est **multi-critère**. Nous nous sommes ramené à une fonction `fitness_calculation` scalaire, mais il peut être plus efficace de mesurer indépendamment les deux erreurs.
On se prive alors de l'ordre sur $\mathbb{R}$, mais on peut pallier ce problème en utilisant par exemple des *fronts de Pareto*.



Une autre idée serait de faire une fonction de mutation dynamique (par exemple plus de mutation lorsque l'algorithme stagne).