CentraleSupélec

université PARIS-SACLAY

Rapport enseignement d'intégration

Théorie des jeux : Jeux évolutionnaires

Rapport de
KALLA Mehdi, KEMICHE Ghiles, PILORGET Maxime, SUBRAMANIAM Abiram

2 février 2024

Table des matières

Introduction

L'étude des molécules d'ADN est essentielle pour comprendre la vie sur Terre. Ces molécules sont constitués d'une succession de nucléotides (A, T, C, G), on parle alors de **séquence d'ADN**. Une manière pertinente de les étudier est de construire leur **trajectoire tri-dimensionnelle**. C'est ainsi qu'en 1993 des chercheurs ont proposé un modèle de conformation 3D permettant de transformer toute séquence d'ADN en une trajectoire tri-dimensionnelle.
Le problème ici étant que ce modèle est principalement adapté aux courtes séquences d'ADN. Notamment lorsque l'on applique ce modèle à des séquences circulaires (comme les plasmides) la trajectoire associée n'est pas circulaire, or on souhaiterait conserver cette propriété.

Notre **objectif** est donc de modifier ce modèle de conformation 3D, autrement dit **modifier** la **table de rotation** associée, afin de **rendre toute trajectoire de plasmide circulaire**.
On présente pour cela deux algorithmes stochastiques : le **recuit simulé** et **l'algorithme génétique**. On détaillera le principe des deux algorithmes, comment est-ce qu'on est parvenu à les coder et quels ont été les résultats de nos simulations. Enfin, on comparera les deux solutions et on discutera d'éventuelles perspectives et améliorations.

1- Recuit simulé

I - Introduction

Le recuit simulé est une technique d'optimisation heuristique qui s'inspire du processus de recuit dans la métallurgie. Dans notre cas, nous l'utilisons pour optimiser la conformation 3D d'une séquence d'ADN, en prenant en compte les contraintes de circularité et d'alignement, ainsi que la complémentarité des séquences.

II - Formulation du Problème

Notre objectif est de trouver la conformation optimale d'une table de rotation pour une séquence d'ADN, en s'assurant que la séquence est circulaire, bien alignée et que la séquence complémentaire suit le même twist et wedge mais dans la direction opposée. Cela implique la minimisation d'une fonction de coût qui prend en compte à la fois la distance entre les extrémités de la séquence et l'alignement de ces points. On cherchera donc à minimiser une certaine fonction objectif.

II.a - Fonction Objectif

La fonction de coût pour une table de rotation sur une séquence d'ADN donnée est calculée comme suit:

$w_d \times \text{DistanceCost} + w_a \times \text{AlignmentCost} + w_t \times \text{TwistComplementCost} + w_w \times \text{WedgeComplementCost}$

où :

- w_d est le poids associé à la distance entre le début et la fin de la trajectoire 3D.
- w_a est le poids associé à l'alignement des directions au début et à la fin de la trajectoire.
- w_t est le poids associé à la somme des différences absolues des twists des paires complémentaires.
- w_w est le poids associé à la somme des différences absolues des wedges des paires complémentaires.
- DistanceCost est la distance euclidienne entre le premier et le dernier point de la trajectoire 3D. elle est définie par la formule :

$$\text{DistanceCost} = \sqrt{(P_{fin,x} - P_{dbut,x})^2 + (P_{fin,y} - P_{dbut,y})^2 + (P_{fin,z} - P_{dbut,z})^2}$$

- AlignmentCost Le coût d'alignement (`AlignmentCost`) est calculé comme l'angle entre les directions initiale et finale de la trajectoire. Cela représente à quel point le début et la fin de la trajectoire sont alignés. Il est calculé comme suit :

$$\text{AlignmentCost} = \arccos\left(\frac{\vec{D}_{dbut} \cdot \vec{D}_{fin}}{\|\vec{D}_{dbut}\|\|\vec{D}_{fin}\|}\right)$$

où $\vec{D}_{dbut}$ et $\vec{D}_{fin}$ sont les vecteurs directionnels au début et à la fin de la trajectoire, respectivement.

- TwistComplementCost est le coût lié au twist des paires complémentaires est la somme des différences absolues des valeurs de twist pour chaque paire de nucléotides complémentaires. Il est défini comme :

$$\text{TwistComplementCost} = \sum_{\text{paires complémentaires}} |\text{Twist}_{n_1} - \text{Twist}_{n_2}|$$

  où n_1 et n_2 sont des nucléotides complémentaires.

- WedgeComplementCost est de même, le coût lié au wedge des paires complémentaires est la somme des différences absolues des valeurs de wedge pour chaque paire de nucléotides complémentaires :

$$\text{WedgeComplementCost} = \sum_{\text{paires complémentaires}} |\text{Wedge}_{n_1} - \text{Wedge}_{n_2}|$$

III - Algorithme

L'algorithme de recuit simulé optimise la table de rotations en minimisant la fonction de coût. Il suit les étapes suivantes:

1. Initialisation de la table de rotations, de la température et des paramètres de l'algorithme.
2. À chaque itération, une nouvelle table de rotations est générée en perturbant légèrement l'état courant en restant dans les intervalles autorisés.
3. La fonction de coût est calculée pour la nouvelle table de rotation.
4. La nouvelle table de rotation est acceptée selon la probabilité définie par la différence de coût et la température actuelle du système.
5. La température décroît.
6. L'algorithme se termine lorsque le nombre maximal d'itérations est atteint ou que la fonction de coût est inférieure à un seuil minimal.

III.a - Probabilité de transition

La probabilité de passer d'un état à un autre est déterminée par la différence de coût entre ces états et la température actuelle du système. La fonction de probabilité est définie comme $\exp(-\Delta E/T)$, où ΔE est la différence de coût et T est la température.

III.b - Température

La température initiale et la fonction décroissante sont des paramètres cruciaux de l'algorithme. Une température initiale élevée permet une exploration plus vaste de l'espace des solutions, tandis qu'un décroissement progressif aide à converger vers une solution optimale. Dans notre cas on a choisi une décroissance linéaire.

IV - Paramètres de l'algorithme

Les paramètres de l'algorithme, tels que le poids des termes de la fonction de coût, la température initiale, le nombre maximal d'itérations et le seuil minimal de coût, jouent un rôle crucial dans la performance et les résultats de l'optimisation. Un bon équilibre entre ces paramètres est essentiel pour obtenir une solution optimale.

IV.a - Poids de la Fonction de Coût

Les poids attribués à chaque terme de la fonction de coût, tels que w_d pour la distance et w_a pour l'alignement, déterminent l'importance relative de ces facteurs dans le coût total. Une valeur élevée de w_d donnera plus d'importance à la minimisation de la distance entre les extrémités de la trajectoire, tandis qu'une valeur élevée de w_a mettra l'accent sur l'alignement des extrémités de la trajectoire.

IV.b - Température Initiale (T_0)

La température initiale influence la capacité de l'algorithme à explorer l'espace des solutions. Une température élevée permet une plus grande exploration, ce qui peut être bénéfique pour éviter les minima locaux. Cependant, une température trop élevée peut également conduire à des sauts trop importants, rendant l'algorithme moins efficace pour affiner une solution proche de l'optimum.

IV.c - Nombre Maximal d'Itérations (k_{max})

Le nombre maximal d'itérations détermine combien de fois l'algorithme essaiera de trouver une meilleure solution. Un nombre plus élevé permet une exploration plus approfondie de l'espace des solutions, mais augmente également le temps de calcul.

IV.d - Seuil Minimal de Coût (e_{min})

Le seuil minimal de coût est la valeur en dessous de laquelle l'algorithme s'arrête, considérant qu'une solution suffisamment bonne a été trouvée. Un seuil plus bas peut conduire à une meilleure solution finale, mais peut également augmenter le temps de calcul si la solution est difficile à atteindre.

IV.e - Ordre de grandeur du coût et influence sur ΔE

L'ordre de grandeur du coût influence directement les variations de coût (ΔE) entre les états successifs. Si les termes de la fonction de coût sont trop élevés, même de petites améliorations peuvent entraîner de grandes diminutions de ΔE, ce qui rend la probabilité de transition vers ces états proche de 1 et réduit l'exploration de l'algorithme. À l'inverse, si l'ordre de grandeur est trop faible, les différences de coût entre les états peuvent être négligeables, conduisant à une exploration excessive et à un manque de convergence.

IV.f - Choix de la température initiale et de la fonction décroissante

La température initiale (T_0) et la fonction de décroissement déterminent la manière dont la température diminue à chaque itération. Une température initiale trop élevée peut conduire à une exploration excessive sans convergence, tandis qu'une température trop basse peut piéger l'algorithme dans un minimum local. La fonction décroissante doit permettre à la température de diminuer progressivement, donnant à l'algorithme le temps d'explorer l'espace des solutions tout en affinant progressivement la meilleure solution trouvée.

IV.g - Valeurs Suggérées pour les paramètres

Pour une séquence d'ADN de 8k:

- **Poids Distance** : 1.2
- **Poids Alignement** : 50
- **Température Initiale** : 1.0

- **Nombre Maximal d'Itérations** : 15000
- **Coût Minimal** : 50

Pour une séquence d'ADN de 180k:

- **Poids Distance** : 0.12
- **Poids Alignement** : 10
- **Température Initiale** : 50.0
- **Nombre Maximal d'Itérations** : 10000
- **Coût Minimal** : 50

V - Résultats de Simulations

Dans cette section, nous présentons les résultats obtenus après l'exécution de notre algorithme sur deux ensembles de données différents, une séquence d'ADN de 8k puis de 180k. Les paramètres ont été ajustés en fonction de la longueur de la séquence d'ADN pour optimiser la performance de l'algorithme.

V.a - Séquence d'ADN de 8k

Pour la séquence d'ADN de 8 000 nucléotides. Les résultats obtenus montrent comment l'algorithme converge vers une solution optimale, en minimisant la fonction de coût tout en respectant les contraintes de circularité et d'alignement.

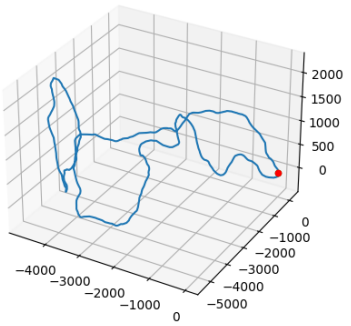

Figure 1.1: Tracé de la trajectoire avec la table optimale

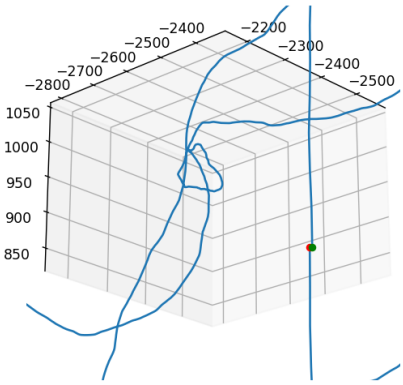

Figure 1.2: Zoom sur les points de début et de fin

On peut observer l'évolution du coût :

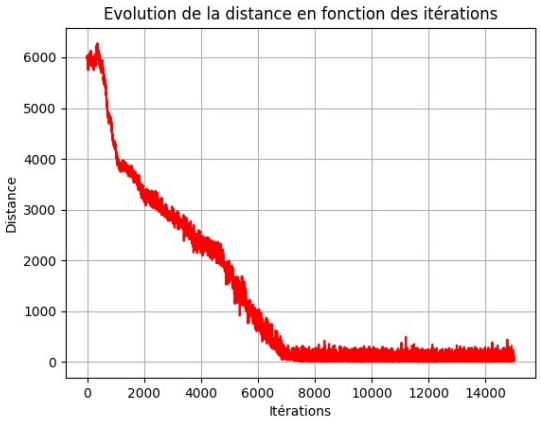

Figure 1.3: Evolution du coût

V.b - Séquence d'ADN de 180k

Pour la séquence d'ADN plus longue de 180 000 nucléotides, les paramètres ont été ajustés pour tenir compte de la complexité accrue de l'espace des solutions.

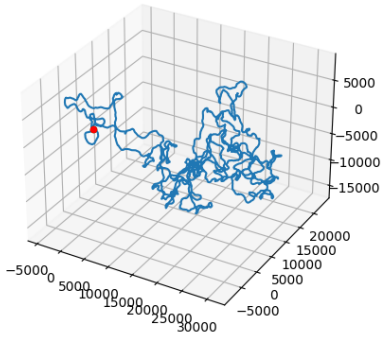

Figure 1.4: Tracé de la trajectoire avec la table optimale

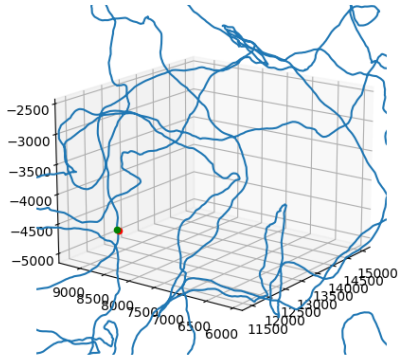

Figure 1.5: Zoom sur les points de début et de fin

2- Algorithme génétique

I - Principe de l'algorithme

L'algorithme génétique est un algorithme d'optimisation stochastique. Il consiste à appliquer la notion de sélection naturelle et la théorie de l'évolution de Darwin sur l'ensemble des solutions admissibles au problème afin de construire itérativement de nouvelles solutions potentiellement meilleures.
Plus précisément, voici chaque étape de l'algorithme :

- **1. Génération initiale :** On génère aléatoirement une population initiale (c'est-à-dire une liste d'individus).
- **2. Évaluation des individus :** On utilise une fonction objectif servant à évaluer un individu (une table de rotation donc). On applique cette fonction objectif à chacun des individus et on trie la liste obtenue des meilleurs individus (score le plus faible) aux pires individus.
- **3. Sélection :** On sélectionne ensuite les meilleurs individus. On a choisi pour cela la sélection par tournoi. Cette technique utilise la sélection proportionnelle sur des paires d'individus, puis choisit parmi ces paires l'individu qui a le meilleur score d'adaptation.
- **4. Encodage des gènes :** On a codé les gènes de chaque chromosome sur 16 bits afin de faciliter les opérations de croisement et de mutation que l'on va effectuer par la suite.
- **5. Croisement :** On produit de nouveaux individus en croisant des paires d'individus parmi ceux sélectionnés jusqu'à arriver à la même quantité d'individus qu'initialement (de sorte à garder une taille de population constante). On a utilisé pour cela le croisement à un point.
- **6. Mutation des gènes :** On modifie aléatoirement (avec une probabilité assez faible) certains bits sur chaque gène afin de favoriser une diversité génétique ce qui nous permettra de sortir d'éventuels optima locaux mais non-globaux.

II - Organisation du code

Nous avons décidé d'utiliser une approche orientée objet afin d'implémenter l'algorithme. Pour cela nous avons mis au point deux classes :

II.a - Classe individu

La classe individu permet d'encoder toutes les opérations relatives à la manipulation des gènes :

- **Initialisation** : La classe est initialisée en attribuant à un objet une table de rotation (générée aléatoirement comme un voisin de la table initiale), et deux paires de chromosomes : un chromosome de données et un de probabilités pour le "twist", et de même pour le "wedge".
- **Encodage des chromosomes** : La méthode *encode_chromosome* permet de construire les chromosomes de données de la manière suivante : chaque "twist" (resp. "wedge") d'un dinucléotide est converti en binaire, et ensuite inséré dans le chromosome correspondant.
- **Encodage probabilités** : La méthode *encode_probas* permet de construire les chromosomes de probabilités de la manière suivante : pour chaque gène du chromosome de données correspondant, est créé un gène de probabilités. Celui-ci est défini comme une suite aléatoirement générée de 0 et de 1.

- **Application des probabilités** : La méthode *mutate* permet d'appliquer les probabilités trouvées construites précédemment : chaque gène de chaque chromosome de données est comparé au gène correspondant pour le chromosome de probabilités, lorsqu'un 1 est rencontré sur le gène de probabilité, le bit correspondant est alors changé sur le gène de données.

- **Création de table de rotation** : La méthode extract_rotTable permet d'attribuer à la table de rotation de l'individu les valeurs (décimales) des chromosomes de données.

Soulignons que les structures de données utilisées pour définir les chromosomes sont des dictionnaires. Ce choix est justifié par le fait qu'il est alors plus simple d'attribuer les valeurs trouvées à la table de rotation (elle-même un dictionnaire de mêmes clés). L'utilisation du dictionnaire, et la conversion binaire-décimal a également induit la création de certains outils spécifiques, définis dans le fichier ./3dna/utilitaires.py.

II.b - Classe genetic

La classe genetic permet d'implémenter les différentes étapes de l'algorithme génétique :

- **Initialisation** : Un objet de genetic est initialisé avec une population (liste d'individus). Un attribut est initialisé pour chaque étape de l'algorithme, sous forme de liste ou de dictionnaire.

- **Fonction fitness** : La fonction fitness évalue le score d'un individu donné à partir de la fonction objectif que l'on a choisie. Cependant, en raison des différences d'implémentation entre l'algorithme recuit et l'algorithme génétique, il n'a pas été possible de pondérer de la même manière l'alignement, et d'implémenter la contrainte du complémentaire.

- **Evaluation** : La méthode do _evaluation évalue chaque individu selon leur "fitness", et les classe en ordre décroissant.

- **Sélection** : Nous avions d'abord opté pour une sélection par tournoi, mais nous avons fini par utiliser une sélection par élitisme, qui était plus simple à implémenter. La méthode do _selection renvoie une liste de la meilleure moitié des individus.

- **Croisement** : La méthode do _croisement permet d'effectuer des croisement en un point des chromosomes.

- **Mutation** : La méthode do _mutation applique les mutations sur les chromosomes (sauf ceux du premier, car les mutations sont en générales délétères, on parvient ainsi à assurer une meilleure convergence).

- **Algorithme** : Enfin la méthode algo _gen applique successivement les méthodes précédentes.

Notons que pour la classe genetic, nous avons privilégié l'utilisation de listes, plus faciles à manipuler pour le tri et la sélection des indices.

III - Résultats de simulation

Nous allons lancer l'algorithme pour $k = 800$ générations, avec une population initiale de $n = 24$ individus (cf. figures 2.1 et 2.2). Nous l'exécutons pour une chaîne d'ADN de 8 000 caractères. L'algorithme est exécuté en 3 minutes en moyenne, et converge vers une distance pondérée par un poids d'alignement de l'ordre de 100 (la distance seule atteint un minimum de 20). Nous discuterons dans la partie "Conclusion" des moyens éventuels pour améliorer la qualité du résultat de l'algorithme génétique.

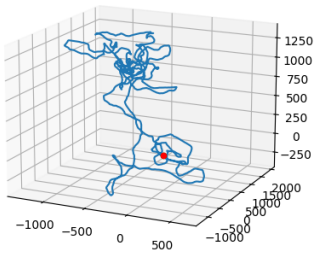

Figure 2.1 : Repliement de l'ADN pour plasmid 8k

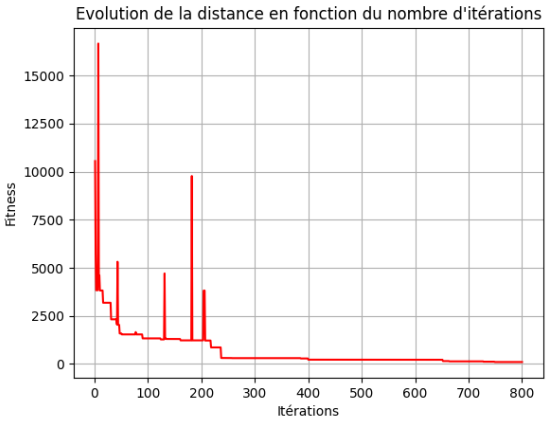

Figure 2.2 : Convergence pour plasmid 8k

3- Comparaison et conclusion

I - Comparaison des algorithmes

L'algorithme recuit prend 4 min pour obtenir une "bonne" convergence, avec des coûts négligeables. L'algorithme génétique, quant à lui converge en 3 minutes vers un coût de l'ordre de 100. Ce coût, bien que pondéré différemment que celui de l'algorithme recuit, caractérise un résultat de moindre qualité, car les pondérations du recuit sont plus élevées. Ceci est dû à la complexité temporelle élevée de l'algorithme génétique, qu'il serait possible de réduire davantage.

II - Conclusion

En conclusion, nous avons pu au cours de cet enseignement d'intégration, mettre en place deux algorithmes permettant de rendre la trajectoire de plasmides circulaire. D'une part nous avons implémenté l'algorithme recuit, qui a pu assurer une bonne convergence. D'autre part, l'algorithme génétique, en théorie plus performantes, a pu assurer une convergence, mais moins efficacement.
Des améliorations de l'algorithme génétique sont possibles : on peut par exemple réduire la complexité temporelle de certaines fonctions, optimiser les probabilités choisies, paralléliser une partie du code afin d'accélérer la vitesse d'exécution, changer le choix du voisinage ou encore la méthode de sélection.
De plus, une perspective intéressante serait d'optimiser les différents paramètres que l'on a eu à choisir dans les algorithmes (comme les poids, la température initiale, les probabilités d'encodage, etc...). Pour cela on pourrait utiliser un réseau de neurone ou un recuit simulé avec comme fonction de coût notre fonction objectif et comme variables nos paramètres que l'on souhaite optimiser. On exécuterait un tel programme une seule fois de sorte à obtenir ces paramètres optimaux et ainsi pouvoir grandement améliorer l'efficacité de nos deux algorithmes.

Bibliographie

[1] Joan HERISSON, *Algorithmiique génétique*

[2] P. J. M. VAN LAARHOVEN, E. H. L. AARTS *Simulated Annealing: Theory and Applications* : https://www.researchgate.net/profile/Demsew-Teferra-2/publication/338294434_Simulated_Annealing_Theory_and_Applications/links/5e0c8b65299bf10bc3878943/Simulated-Annealing-Theory-and-Applications.pdf