Rapport EI DEMAILLE COURNIL RABEUX REBOLA CAHITTE

Mathéo Cahitte, Clément Cournil-Rabeux, Melkior Demaille, Clément Rebola

January 2026

1 Introduction

Le sujet de cet EI est l'optimisation par les algorithmes génétiques, plus particulièrement dans le calcul de la trajectoire spatiale d'une séquence d'ADN. L'indication que nous avons est d'étudier des plasmides : sans connaître leur géométrie exacte on sait cependant qu'ils doivent boucler. De plus, la trajectoire donnée doit correspondre à celle engendrée par une table indiquant le décalage dans l'espace correspondant à chaque couple possible de dinucléotide - c'est à dire trois angles pour chacun des 16 dinucléotides, la translation étant fixée. C'est en partant d'une table de référence donnée par un modèle expérimental que nous devons itérer, afin de trouver une table proche mais plus adaptée.

Une recherche rapide sur les plasmides permet d'apprendre que le fait qu'ils bouclent ne dépend en réalité pas de leur géométrie de moindre contrainte. En effet, soit un plasmide est copié directement depuis un plasmide préexistant (donc cyclique dès le début), soit des enzymes "reconnaissent" les extrémités compatibles du plasmide sous forme linéique pour le recoller. Avec cette information, on peut réinterpréter le sujet comme étant de trouver une table modélisant un replis le meilleur possible étant donné un plasmide en argument, et non pas une "table universelle" comme celle du modèle initial. En effet, au vu de la réalité des plasmides, une telle table est au mieux inadaptée à la condition de bouclement. Une autre raison empêche d'entraîner une "table universelle" valable pour toute séquence ADN : une telle table devrait fournir un bon repliement pour chaque séquence. Or cela nécessiterait de longues séquences de référence dont on connaîtrait la position spatiale de chaque nucléotide, comme c'est le cas pour les protéines utilisées par AlphaFold. Ici la seule information donnée sur les entrées est que ce sont des plasmides. La seule chose que nous pouvons mesurer est donc à quel point ils "bouclent", sans autre prérequis envisageable sur la forme précise qu'ils occupent dans l'espace.

Dans ce rapport, nous avons choisi de commencer par présenter et motiver nos choix théoriques et méthodologiques, d'expliquer ensuite notre algorithme avec plus de formalisme, puis finalement — schémas à l'appui — d'expliquer comment nous l'avons mis à l'épreuve, et les conclusions que nous en avons tiré.

2 Méthodologie

2.1 Pourquoi un algorithme génétique ?

En jouant sur 32 paramètres (les deux premiers angles pour chaque dinucléotide), on a un espace des configurations trop grand pour envisager une résolution analytique (d'autant que le calcul devrait prendre en compte les positions de milliers de bases). De plus, il y a potentiellement de nombreux minimums locaux qu'une simple descente de gradient risque de ne pas éviter efficacement. Pour finir, on se doute que les trajectoires sont tout de même des fonctions lisses (même si très sensibles) des angles reportés dans les tables, un algorithme génétique est donc bien adapté pour que génération après génération, on se rapproche des extremums les plus prometteurs.

2.2 Quelle Génèse ?

Nous n'avons pas d'a priori sur la table optimale, et le modèle fournit seulement des bornes à respecter pour chacun des paramètres de la table de rotation. Pour éviter des biais de sélection, nous avons voulu commencer avec

la population la plus "aléatoire" possible. On veut donc maximiser l'entropie de Shannon :

$$H(X) = -\int_{\mathcal{D}} f(x)\ln(f(x))\,dx \tag{1}$$

où $f(x)$ représente la densité de probabilité de la variable aléatoire X sur le domaine $\mathcal{D}$.

C'est pourquoi nous avons décidé de suivre une loi uniforme (respectant les bornes d'incertitude données) pour le tirage initial (la génèse). La loi uniforme maximise en effet l'entropie sur un segment, sans connaissance supplémentaire.

2.3 Quelle fonction de fitness ?

Le sujet repose sur les plasmides. La compréhension que nous en avons est qu'il faut arriver pour chaque plasmide à une table qui donne la trajectoire la plus proche possible d'une chaîne qui boucle, dans la limite des écarts-types donnés par le modèle. D'un point de vue pragmatique, aux échelles de taille des plasmides, des effets des forces intermoléculaires ou de la température doivent courber d'identiques dinucléotides différemment, un effet qu'on ne pourra ici pas prendre en compte (cf. introduction).

Pour la fonction de fitness, la première idée qui vient est donc de mesurer la distance entre la première et la dernière base dans l'espace. Plus les extrémités sont proches, plus il est convainquant que la table corresponde à celle modélisant au mieux le chemin du plasmide. En réfléchissant un peu plus, le plasmide devant boucler, ce qui nous intéresse est la différence entre la position de la première base de la séquence (l'origine), et la position où elle se retrouverait en parcourant la boucle jusqu'au bout. On peut même adjoindre les k premières bases à la fin de notre séquence, et minimiser la distance entre la position de chacune d'entre elle en début et en bout de chaîne. Le fait de mesurer la distance sur au moins deux bases permet en outre de tenir compte de la table dans le rapprochement des extrémités. Il est plus souhaitable de boucler de façon compatible avec la table de rotation que de retrouver nos extrémités certes superposées mais avec un angle d'incidence quelconque - ceci peut s'apparenter à une condition de continuité C^k.

Nous avons eu une idée pour vérifier plus loin la compatibilité d'une table avec un plasmide, mais qui pointait vers les limites du modèle. La table permet de visualiser la circularisation du plasmide à partir d'une représentation linéique donnée ; en revanche, nous n'avons pour l'instant aucune garantie qu'elle reste valable si le client prend un autre gène comme début du plasmide. Autrement dit, il faudrait tester la qualité du bouclage du chemin obtenu avec un "début/fin" du plasmide différent. Ainsi nous avons ajouté en paramètre de notre fonction le nombre de coupes autres que celle fournie qu'il faudra tester (réparties de façon homogène). Évidemment, ceci multiplie le temps d'exécution de la fonction de fitness, et la rend bien plus exigeante : nous ne sommes même pas certains qu'il puisse exister de solution acceptable en général, avec le modèle fourni.

2.3.1 Formalisation de la fonction de fitness

La qualité d'un individu (une table de rotation) est définie par sa capacité à refermer convenablement la séquence nucléotidique du plasmide sur elle-même. Pour assurer une continuité d'ordre supérieur à la simple fermeture topologique, nous imposons un recouvrement de k bases.

Soit une trajectoire composée de points $\mathbf{P}_i \in \mathbb{R}^3$. La distance de fermeture pour une configuration donnée est définie par :

$$d(\mathcal{P}, k) = \sqrt{\sum_{i=0}^{k-1} \|\mathbf{P}_i - \mathbf{P}_{N+i}\|^2}. \tag{2}$$

Pour s'affranchir de la dépendance au point de coupure initial du plasmide, nous évaluons la somme quadratique du score sur m points de départ différents, répartis uniformément le long de la séquence :

$$\text{Fitness} = \sqrt{\sum_{j=0}^{m} d_j^2}, \tag{3}$$

où d_j est le score obtenu pour la j-ème coupure. Cette métrique pénalise non seulement l'écart spatial entre les extrémités, mais aussi les cassures de pente à la jonction, agissant ainsi comme une condition de régularité géométrique.

Compromis entre minimisation et invariance

Le choix du nombre de coupes m (`nbcuts`) définit la nature de l'optimisation :
— **Optimisation locale** ($m = 0$) : On cherche à minimiser $\mathcal{F} = d_0$. La convergence est rapide car l'espace des solutions admissibles est vaste.
— **Optimisation structurelle** ($m \geq 1$) : On cherche à minimiser $\mathcal{F} = \sqrt{\sum_{j=0}^{m} d_j^2}$.

2.4 Quelle sélection ?

Pour passer d'une génération à la suivante, nous avons essayé plusieurs méthodes de sélection, pour les comparer et déterminer celle qui convergeait le mieux, le plus vite. Nous avons voulu laisser en argument la proportion des individus gardés d'une génération à l'autre.

La sélection élitiste - sélectionnant les k individus au meilleur score parmi les n d'une génération - est un bon départ. Certes on abandonne une partie de l'aléatoire qui fait l'efficacité des algorithmes génétiques, mais tant qu'une bonne façon de gérer cet aléatoire - bien adaptée à la sensibilité des trajectoires, et ne perdant pas son intérêt en considérant des mutations - n'a pas été trouvée, cette sélection reste valide. C'est une bonne référence, un point de repère pour savoir si une autre méthode est intéressante ou non.

Pour adapter la sélection par tournoi à une proportion quelconque de survivants, nous avons choisi d'autoriser un individu à être tiré plusieurs fois. Ainsi ceux qui ont un bien meilleur score auront une meilleure espérance de représentation. La motivation étant qu'un "outsider" aura toujours une chance d'être pris, et un très bon candidat aura moins de chance de se faire oublier. Cette option étant souvent peu efficace une fois combinée avec le reste de l'algorithme, on l'a progressivement modifiée pour qu'elle garde forcément un pourcentage des meilleurs.

Nous avons aussi introduit la sélection roulette. Le problème de cette méthode est qu'elle est dépendante non-plus seulement de l'ordre des solutions, mais également de la valeur numérique de leur score. Ainsi, à priori, composer ce score avec toute fonction croissante nous donne une façon valide de faire tourner la roulette. Pour rester simple nous avons dans un premier temps directement utilisé le résultat de la fonction de fitness (distance euclidienne).

Pour raffiner la sélection roulette, nous nous sommes inspirés de la recuite simulée. En effet, on peut voir la pression évolutive comme un "ressort" qui doit pousser les extrémités du plasmide à se rejoindre. En ce sens un potentiel en x^2 (où x se trouve être la fonction de fitness) semble adapté. Pour le facteur de Boltzmann il reste à fixer une température pour favoriser les bonnes solutions sans directement écraser celles qui s'en rapprochent : c'est la sélection "roulette exponentielle". Pour avoir des poids qui ne soient pas interprétés comme nuls par python, nous avons choisi de les diviser par le poids minimal (sorte de normalisation, la fonction random.choose nécessitant simplement des poids positifs, pas nécessairement de somme unitaire). La fonction de poids est donnée par :

$$W_i = exp\left(\frac{S_{min}^2 - S_i^2}{T_0}\right).$$

En s'inspirant plus encore du recuit, on peut vouloir diminuer la température avec les générations pour affiner la convergence. Il faut cependant tâcher de ne pas être trop brusque pour ne pas tomber directement sur un extremum local trop pauvre. À l'inverse, ne pas diminuer la température assez rapidement fait perdre l'intérêt de ce paramètre. L'ensemble des options étant trop larges nous l'avons réduit à une décroissance affine minorée de la température, choisissant des paramètres convenables expérimentalement. La fonction de poids à la génération k est donnée par :

$$W_i(k) = exp\left(\frac{S_{min}^2 - S_i^2}{\max(T_0 - kC, T_{min})}\right).$$

Une dernière version de la roulette-recuit est la sélection "Recuit normalisé". Ici, pour adapter la descente de température, on se base sur le meilleur score à une génération donnée : plus ce score est bas, plus la température

est froide, donc plus les coupes seront importantes. En effet, si pour un individu i on a $S_i = S_{min} + \Delta S_i$, alors $S_i^2 \sim_0 S_{min}^2 + 2S_{min}\Delta S_i$. On a alors pour la fonction de poids :

$$W_i = exp\left(\frac{S_{min}^2 - S_i^2}{2 \times S_{min}}\right) \sim_0 exp(-\Delta S_i),$$

et :

$$W_i = exp\left(\frac{S_{min}^2 - S_i^2}{2 \times S_{min}}\right) \sim_\infty exp\left(\frac{-S_i^2}{2S_{min}}\right).$$

On opère alors une sélection qui ne coupe pas trop rapidement proche du meilleur, mais qui laisse peu de chance aux très mauvais cas de continuer. Cette méthode est théoriquement très élégante, mais en pratique nous n'avons pas obtenu de très bons résultats avec notre algorithme. Elle ne sera donc pas beaucoup plus développée.

Ensuite, nous avons implémenté la roulette-rang. Ici on oublie même la valeur de la fonction de fitness pour ne retenir que l'ordre qu'elle donne - on a cependant toujours la même problématique de composition possible avec une fonction croissante quelconque. On commence par tirer avec une probabilité proportionnelle au complémentaire du rang ($W_i = n - rang(i)$).

Dans une variante de la sélection roulette-rang, on utilise une distribution géométrique des probabilités. On prend comme raison une pression de sélection $0 < q < 1$, et on ajuste le rapport des probabilités de choix du k-ième sur le k+1-ième. Ceci permet une répartition un peu plus contrôlée des "outsiders". Il semble cependant toujours dommage d'"oublier" une part de l'information donnée par fitness mais moins guider l'aléatoire peut le rendre plus puissant. Ici :

$$W_i = q(1-q)^{n-rang(i)-1}.$$

Pour la roulette, on a un algorithme de la sorte :

```
SelectionRoulette(Population P, Taux τ, Mode, Temp T) :

        N_sel ← ⌊taille(P) × τ⌋
        S_min ← min_{ind∈P}(ind.score)
        S_total ← Σ_{ind∈P}(ind.score)
        ListePoids ← []

        Pour chaque individu ind dans P :
              S ← ind.score

              Si Mode == Classique :
                    W ← 1 − (S/S_total)
              Sinon Si Mode == Exponentiel :
                    W ← exp((S_min^2 − S^2)/T)
              Sinon Si Mode == Exp_Normalise :
                    W ← exp((S_min^2 − S^2)/S_min)

              Ajouter W à ListePoids

        // Tirage pondéré avec remise (random.choices)
        Geniteurs ← Tirage(Population, Poids=ListePoids, k=N_sel)

        Retourner Geniteurs
```

2.5 Quelle reproduction ?

La question qui se pose ici est de savoir si les individus mieux classés doivent se reproduire d'avantage que les moins classés, y compris après l'étape de sélection. En effet, tester plusieurs variations du meilleur candidat d'une génération à l'autre peut être intéressant. Nous pouvions jouer sur plusieurs paramètres : le nombre d'occasions qu'a un survivant de se reproduire, et le poids et le type de son impact sur les gènes de ses descendants. Le premier aspect

est en partie pris en compte par l'étape de sélection, nous l'avons donc ajouté de manière optionnelle (paramètre "poisson" de l'algorithme). En ce qui concerne le second aspect, voici les méthodes successives que nous avons choisies.

La reproduction entre deux individus I_1 et I_2, de scores respectifs f_1 et f_2, génère un descendant dont les gènes sont une combinaison linéaire de ceux des parents.

Pour chaque gène x_i, la nouvelle valeur x'_i est calculée par :

$$x'_i = \alpha \cdot x_{i,1} + (1 - \alpha) \cdot x_{i,2}, \tag{4}$$

où le coefficient de pondération α est déterminé par l'importance relative des scores :

$$\alpha = \frac{f_2}{f_1 + f_2}. \tag{5}$$

Si $f_1 + f_2 = 0$, on définit par défaut $\alpha = 0.5$ pour obtenir un mélange équitable. Cette méthode permet de créer un nouvel individu situé sur le segment reliant les deux parents dans l'espace des solutions, avec un biais en faveur du parent ayant le score le plus élevé (sous réserve de la définition du score).

Une autre méthode pour la création des nouveaux gènes est d'associer à x'_i $x_{i,1}$ avec la probabilité α ou $x_{i,2}$ avec la probabilité $(1 - \alpha)$. On évite ainsi de mélanger des gènes importants et d'obtenir trop enfants "malformés".

Cependant, on se rend compte que cette méthode peut s'avérer instable. On décide alors de la combiner avec la précédente, en rajoutant un paramètre de couplage β de sorte que :

$$\mathbb{P}(x'_i = x_{i,1} * (1 - \beta) + (\alpha \cdot x_{i,1} + (1 - \alpha) \cdot x_{i,2}) * \beta) = \alpha \tag{6}$$

$$\mathbb{P}(x'_i = x_{i,2} * (1 - \beta) + (\alpha \cdot x_{i,1} + (1 - \alpha) \cdot x_{i,2}) * \beta) = 1 - \alpha. \tag{7}$$

Nous verrons dans la partie suivante quelles valeurs de ces paramètres ont donné les meilleurs résultats.

2.6 Quelle mutation ?

Choisir la fréquence et l'amplitude des mutations est primordial. Une trop haute fréquence de mutation et on perd l'intérêt d'avoir un héritage, trop faible et on retrouve les écueils de la descente de gradient - surtout si les génomes sont mélangés à chaque nouvel individu, avec une tendance à l'uniformisation. Nous avons d'abord choisi de répartir l'amplitude des mutations autour des valeurs initiales avec une loi normale (coupée pour éviter de dépasser les bornes du modèle), et d'avoir une probabilité égale pour chaque gène de muter. De plus, nous avons joué sur la largeur de la normale et la probabilité des mutations au fil des générations pour affiner les solutions, une fois un bon candidat trouvé. Plus tard, nous avons ajouté une faible probabilité de grande mutation : la simple loi normale permet d'aller vers un minimum local, il fallait rajouter un aspect "fort imprévu" pour s'assurer qu'on n'orbite pas indéfiniment autour d'une solution médiocre. Voici la formalisation.

Pour chaque gène, si un tirage aléatoire $u \sim \mathcal{U}(0, 1)$ est inférieur au taux de mutation m, la nouvelle valeur v'_i est initialement calculée comme suit :

$$v'_i = \max\left(\mu_i - \sigma_i, \min\left(\mu_i + \sigma_i, v_i + \delta\right)\right) \tag{8}$$

Où :

— $\delta \sim \mathcal{N}(0, (\sigma_{mut} \cdot \sigma_i)^2)$ représente le saut de mutation.

— μ_i et σ_i sont la moyenne et l'écart-type de référence (issus de `Rot_data`).

— v_i est la valeur avant mutation et v'_i la valeur après mutation.

Pour modéliser les mutations plus rares et plus importantes on ajoute un second taux de mutation $m' << m$ et un écart type $\sigma' > \sigma_{mut}$. On applique ensuite le même processus, les gènes des enfants passent donc à travers deux étapes de mutation différentes.

3 L'algorithme génétique

3.1 Les paramètres de l'algorithme

Le comportement de l'algorithme génétique est régi par un ensemble d'hyper-paramètres définissant la stratégie d'évolution.

- **Taille de la population (`nb_individus`)** :
- **Nombre d'itérations (`nb_iterations`)** : Le nombre de générations.
- **Fonction de Fitness** (`fitness()`) : Fonction d'évaluation à minimiser.
- **Taux de sélection** (`taux_selec`) : pourcentage de la population subsistant d'une génération à l'autre
- **Méthode de sélection** (`select(taux_selec, fitness)`) : Dépend du taux de sélection et de la fonction de fitness. Dans cet exemple les stratégies suivantes sont implémentées (cf. partie 1) :
  - *Élitiste*
  - *Tournoi*
  - *Roulette (avec variantes)* :
    - Roulette Classique (`selection_roulette`)
    - Roulette Exponentielle (`selection_roulette_exp`)
    - Roulette Exp. Normalisée (`selection_roulette_exp_normal`)
- **Méthode de croisement** : Définit comment deux parents génèrent un enfant. Dans notre implémentation, il s'agit d'un **croisement barycentrique pondéré** : chaque paramètre de l'enfant est une moyenne pondérée des parents, où le parent ayant le meilleur score a plus d'influence (poids plus fort) sur le résultat final (Voir 2.4).
- **Méthode de génération** : Permet de générer la population initiale, selon une loi uniforme.

3.2 Pseudo-code général

```
P ← GenererPopulation(nb_individus)

Pour k allant de 1 à nb_iterations :

      // 1. Évaluation et Sélection
      scores ← fitness(P)
      Geniteurs ← select(P, taux_selec, scores)

      // 2. Reproduction
      P_new ← Geniteurs

      Tant que taille(P_new) < nb_individus :
            p1, p2 ← choix_aleatoire(Geniteurs)
            Enfant ← croisement(p1, p2)
            Ajouter Enfant à P_new

      P ← P_new

Retourner MeilleurIndividu(P)
```

3.3 Calcul de complexité

Soit n le nombre d'individus dans la population, G le nombre de générations, et τ le taux de sélection ($0 < \tau < 1$). On note C_{fit} le coût temporel de l'évaluation de la fonction de fitness pour un individu (calcul de la trajectoire 3D).

3.3.1 Complexité de l'étape de Reproduction (Croisement et Mutation)

À chaque itération, nous avons une population de n individus. On a donc un nombre d'enfants à générer qui est de $n_{enfants} = \lfloor n(1-\tau) \rfloor$

La création d'un enfant (croisement de deux vecteurs de taille fixe + mutation) est une opération élémentaire de coût constant $\mathcal{O}(1)$ par rapport à n. On a donc la complexité pour une génération :

$$C_{repro} = \mathcal{O}(n(1-\tau))$$

τ étant fixé en tant qu'hyper-paramètre, on a

$$\boxed{C_{repro} = \mathcal{O}(n)}$$

3.3.2 Complexité de l'étape de Sélection

— **Sélection par Tournoi** : On tire n fois aléatoirement une paire d'individus et on les compare en temps $2C_{fit} = \mathcal{O}(1)$.
Pour chaque génération, on a la complexité :

$$C_{select} = \mathcal{O}(n)$$

— **Sélection Élitiste** :
On doit d'abord trier les individus , en $C_{fit}\mathcal{O}(n \log n)$, c'est-à-dire en $\mathcal{O}(n \log n)$. Ensuite, on récupère les $\tau \cdot n$ premiers, en $\mathcal{O}(n)$.
Pour chaque génération, on a la complexité :

$$C_{select} = \mathcal{O}(n \log(n))$$

— **Sélection par Rang** (Réel ou Géométrique) :
On trie d'abord la population pour trouver les rangs : $\mathcal{O}(n \log n)$.
On assigne ensuite un poids à chaque individu selon le tri ($\mathcal{O}(n)$) .
On doit ensuite tirer les géniteurs selon ces probabilités. Comme le tirage pondéré nécessite une recherche dichotomique ($\mathcal{O}(\log n)$ par individu), cette phase coûte $n \cdot \tau \times \mathcal{O}(\log n) = \mathcal{O}(n \log n)$.

Pour chaque génération, la complexité totale est donc

$$C_{select} = \mathcal{O}(n \log n)$$

— **Sélection par Roulette** (Classique ou Exponentielle) :
On parcourt la population pour calculer les poids (somme linéaire ou exponentielle) et la distribution cumulée en $\mathcal{O}(n)$. On effectue ensuite le tirage pondéré des géniteurs en $n \cdot \tau \times \mathcal{O}(\log n)$.
Pour chaque génération, on a la complexité :

$$C_{select} = \mathcal{O}(n \log n)$$

Seule la méthode par Tournoi est en $\mathcal{O}(n)$ car sa sélection aléatoire est uniforme (pour un tirage : $\mathcal{O}(1)$ si uniforme vs $\mathcal{O}(\log(n))$ si pondéré,).
Toutes les autres méthodes (Élitiste, Rang, Roulette) incluent soit une étape de tri, soit un tirage pondéré, menant à une complexité quasi-linéaire $\mathcal{O}(n \log n)$.

3.3.3 Complexité Globale

La complexité totale sur G générations somme les coûts d'évaluation, de sélection et de reproduction. On distingue deux cas :

1. **Avec la Sélection par Tournoi** :
   La sélection est linéaire ($\mathcal{O}(n)$). Le coût total est :

$$C_{total} = G \times (\underbrace{n \cdot C_{fit}}_{\text{Évaluation}} + \underbrace{\mathcal{O}(n)}_{\text{Sélection}} + \underbrace{\mathcal{O}(n)}_{Croisement+mutation})$$

   Ici, les termes sont du même ordre. Comme le coût unitaire $C_{fit} \geq 1$, le terme d'évaluation domine mathématiquement :

$$\boxed{C_{total} = \mathcal{O}(G \cdot n \cdot C_{fit})}$$

2. **Avec les autres sélections** (Élitiste, Rang, Roulette) :
   La sélection se fait en ($\mathcal{O}(n \log n)$) et on a donc

$$\boxed{C_{total} = G \cdot (n \cdot C_{fit} + (n \log n))}$$

   Dans notre contexte , le coût de calcul physique C_{fit} peut devenir très élevé. Tant que la population n reste raisonnable ($C_{fit} \gg \log n$), le temps de calcul est dominé par la physique plutôt que par le tri. On retrouve alors le même ordre de grandeur pratique :

$$\boxed{C_{total} \approx \mathcal{O}(G \cdot n \cdot C_{fit})}$$

   En pratique, $C_{fit} \approx k \cdot n_{cut} \cdot Plasmid_{size}$ où $k \approx \frac{1}{120000}(secondes)$, ainsi, pour une population de 1000 individus, 25 Générations et un plasmide de taille 8000, le temps de calcul est d'environ 0h30-1h

4 Résultats et choix du modèle

Afin de sélectionner le meilleur modèle, nous avons décidé de

4.1 Choix de la fitness

On a comparé différentes versions de fitness :

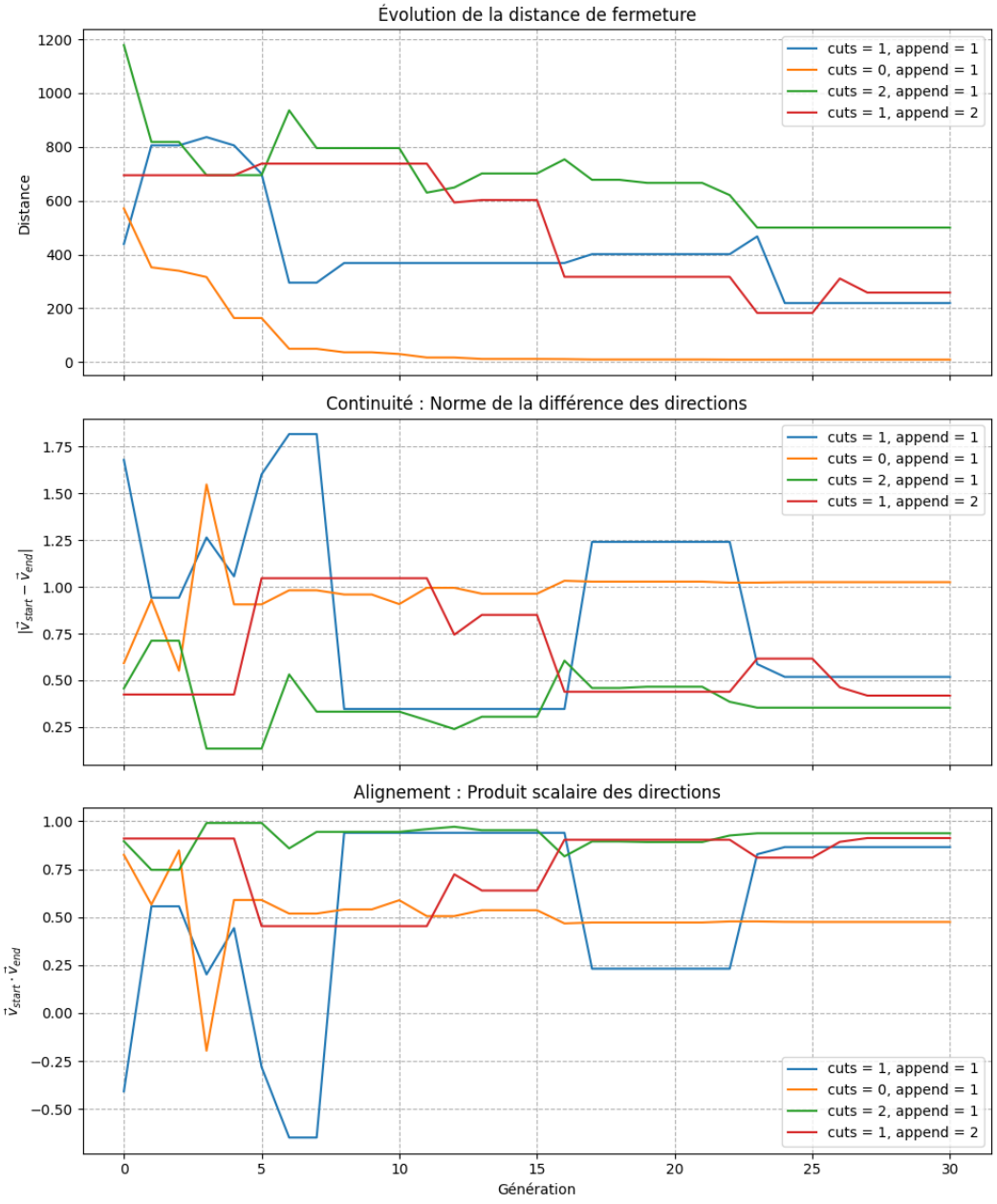

FIGURE 1 – Comparaison des fonctions de fitness selon le nombre de coupes. Paramètres fixés : `nb_individus = 150`, `nb_generations = 24`, `taux_selec=0.3`, `selection_type = "elitiste"`

On observe sans surprise qu'une sélection uniquement basée sur la superposition des deux extrémités du plasmide obtient le meilleur résultat pour cette tâche. Pour l'alignement et la continuité, les autres fonctions de fitness (`nb_cuts >= 1`) réussissent mieux .
Pour `nb_cuts=2` et `append=1`, la performance est meilleure en continuité et en alignement que sur tout les autres, bien que `nb_cuts=1` et `append 2` s'en rapproche.
On peut en conclure que, pour avoir un résultat satisfaisant selon d'autres indicateurs que la seule distance des extrémités, une fonction de fitness avancée avec plusieurs bases de recollement et plusieurs coupes est une meilleure solution.

Cependant, les fonctions de fitness plus avancées sont coûteuses en temps de calcul, le choix devra donc dépendre du but et des contraintes :
Si l'on veut un résultat rapide ou qu'on ne se préoccupe que de la proximité des extrémités, on préférera la fonction de fitness basique.
Sinon, on voudra ajouter des coupes et des points de recollement supplémentaires.

4.2 Choix de la sélection

Nous avons d'abord fait une présélection des 4 méthodes qui nous paraissaient les plus performantes. En effet, les méthodes comme rang réel ou roulette classique nous ont donnés des mauvais résultats assez tôt lors des premières simulations. Elles n'ont donc pas été considérées au-delà.

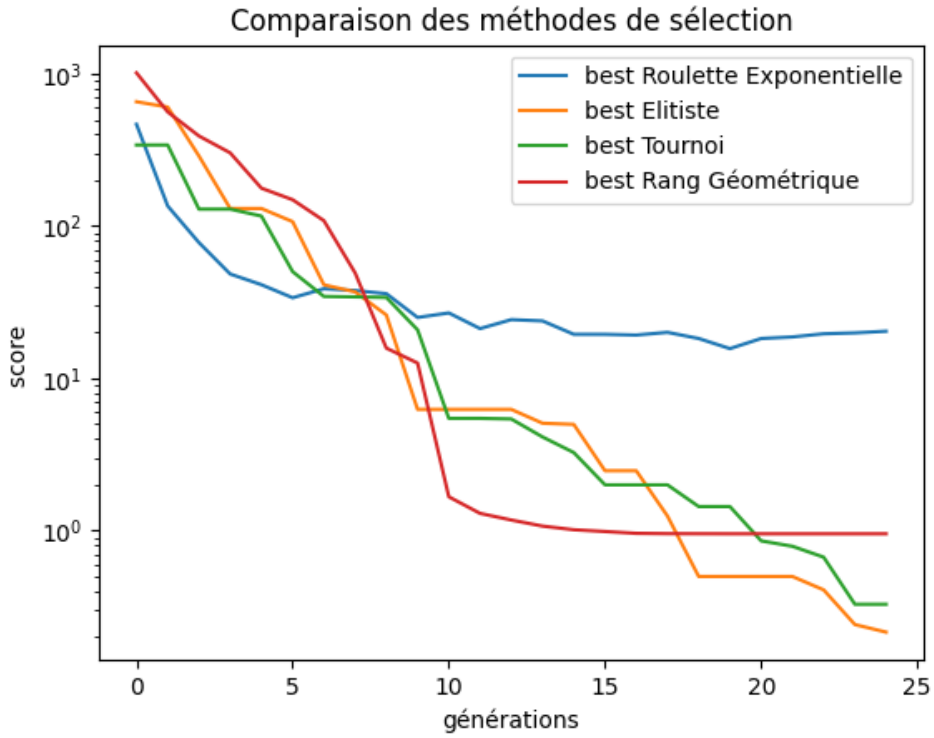

FIGURE 2 – Comparaison des performances selon les différentes méthodes de sélection. Paramètres fixés : `nb_individus = 150`, `nb_generations = 24`, `taux_selec=0.3`, et fitness 'basique' (0,1)

On remarque que si `roulette_exp` converge plus rapidement que les autres au début, elle stagne (d'où l'intérêt de varier en température). `elitiste` reste la méthode donnant le meilleur résultat in fine, c'est donc encore notre choix par défaut. Les méthodes `tournament` (Tournoi) et `rang_geo` (rang géométrique) donnent aussi des résultats satisfaisants — peut-être parce qu'ils proposent un choix des survivants proche de celui de `elitiste`.
Comme `tournament` est un peu moins coûteux en complexité que d'autres méthodes stochastiques, il reste un choix pertinent si l'on veut économiser du temps de calcul, avoir un bon résultat, et garder une part d'aléatoire. Il est par exemple intéressant si l'on veut faire une simulation avec une population particulièrement large.

4.3 Choix des paramètres du modèle

4.3.1 Choix du taux de sélection

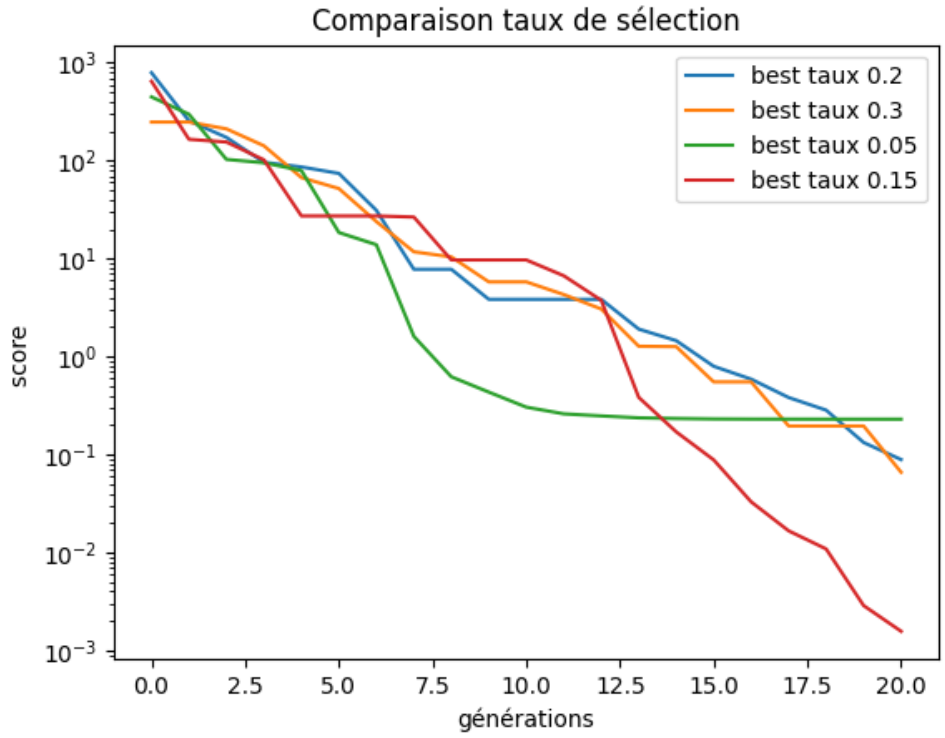

FIGURE 3 – Comparaison des performances selon les différents taux de sélection. Paramètres fixés : `nb_individus = 150`, `nb_generations = 20`,
selection `elitiste`, et fitness 'basique' (0 cuts et 1 append)

4.3.2 Choix du paramètres β pour la mutation

β	**Fitness sans coupure**	**Fitness pour une coupure**
0 (sélection stochastique)	3.11	35.48
0.3	0.03	31.75
0.7	0.15	4.32
1 (combinaison linéaire)	1.85	34.5

TABLE 1 – Valeurs de fitness en fonction du paramètre β (les optima sont encadrés)

Via ces résultats, on a donc décidé de prendre $\beta = 0.3$ pour une fitness basique, et $\beta = 0.7$ pour une fitness avec coupure.

5 Résultats obtenus

5.1 Fitness Basique vs Avec coupes

Nous avons d'abord fait une simulation avec (`nb_individu = 500`) et (`nb_generations = 30`) , le temps de calcul étant long, on a décidé de prendre la fonction de fitness classique avec (`nb_cuts = 0` et `append = 1`). On a pris la meilleure méthode testée auparavant, (`elitiste`).

On l'a comparé avec une simulation ayant (`nb_individu = 500`) et (`nb_generations = 30`), aussi avec la méthode de sélection (`elitiste`). Nous avons pris pour cette deuxième simulation, `nb_cuts = 1` et `nb_append = 1`

Voici ce qu'on obtient

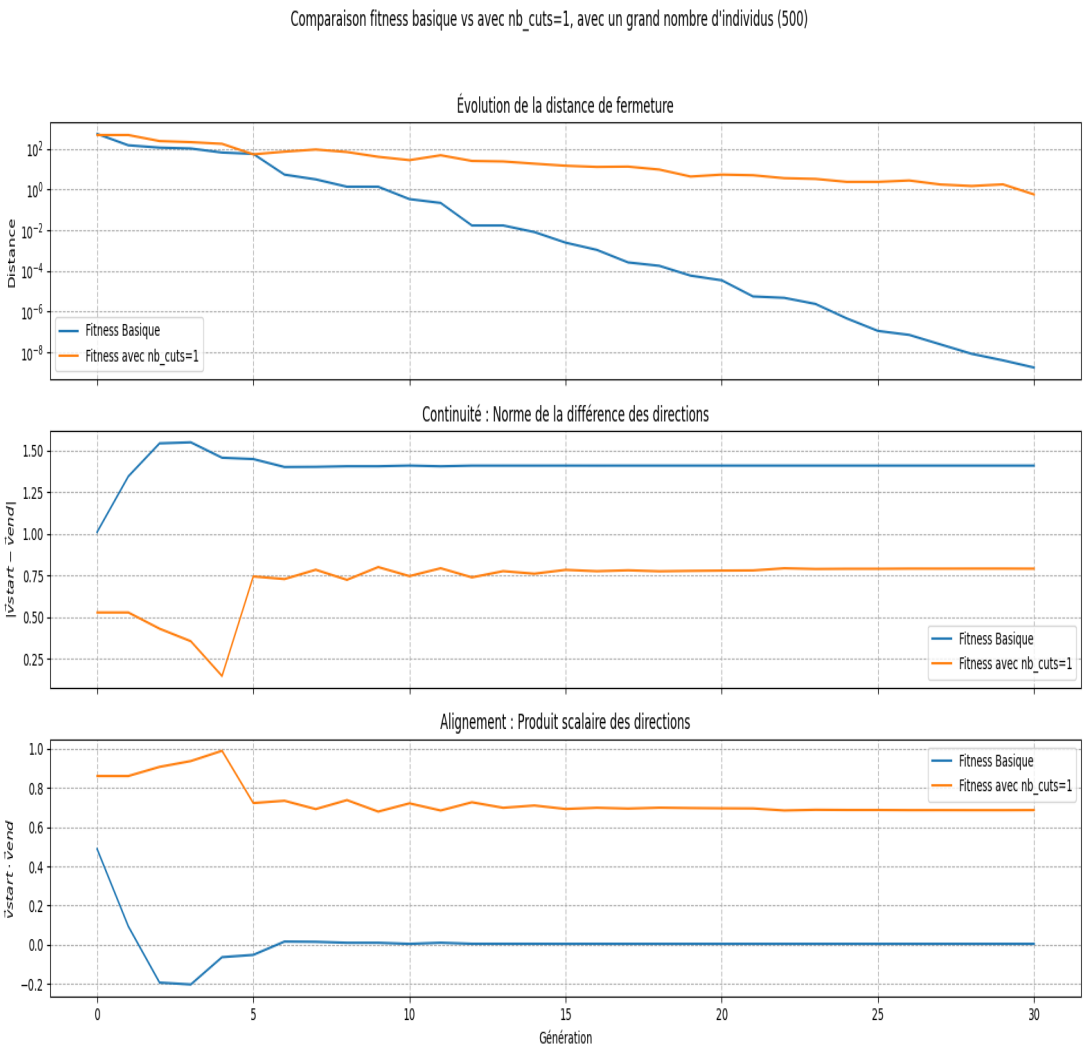

FIGURE 4 – Résultats obtenus sur 500 individus

On observe que fitness basique permet effectivement de bien minimiser la distance, mais que comme attendu, il n'y a pas d'alignement ni de continuité observée : les angles des vecteurs incidents ne sont pas proches

5.2 Recuit Simulé

Voici un exemple donné pour la roulette exponentielle avec recuit affine, avec une coupure supplémentaire et un test de recollement sur deux bases.

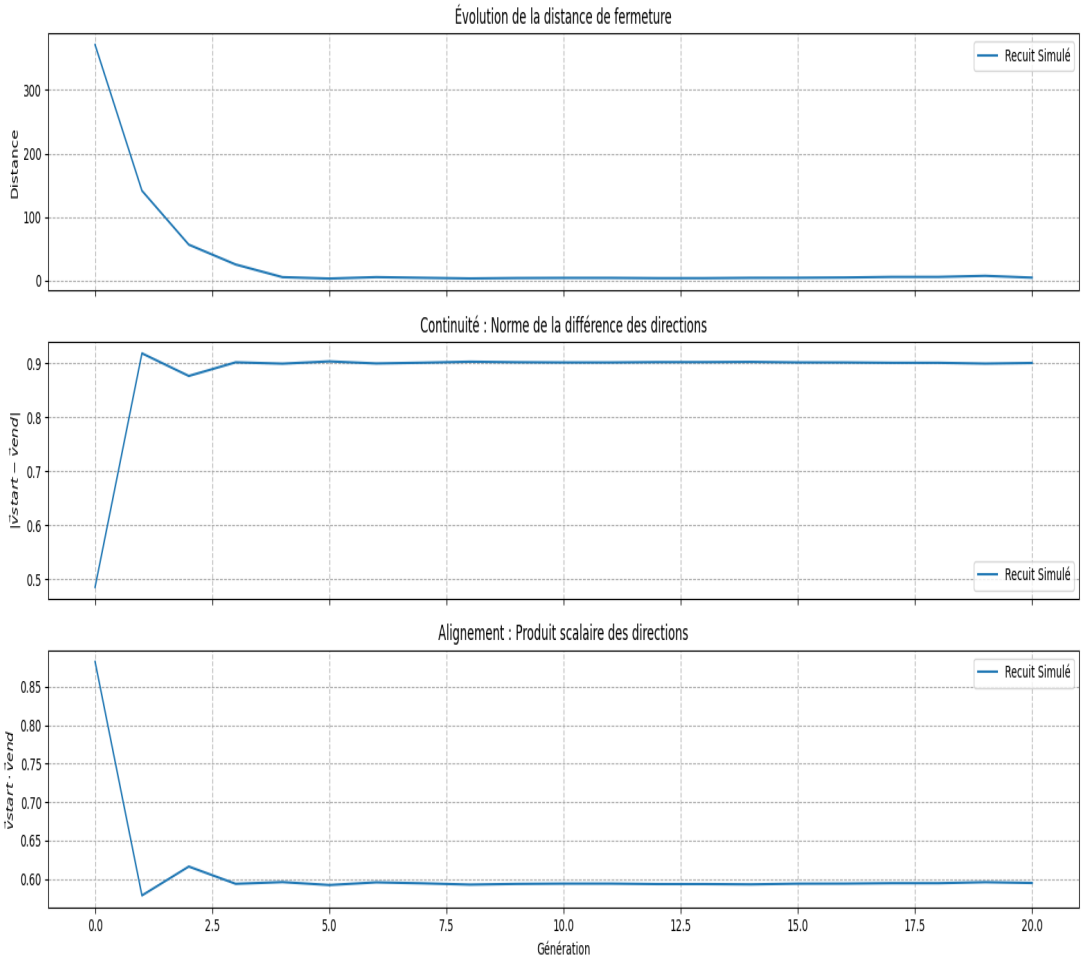

FIGURE 5 – Résultats obtenus via méthode du recuit simulé

On remarque que si la distance entre les extrémités est bonne, l'angle est mauvais. De plus, la méthode converge très rapidement vers un point fixe, ce qui indique une température trop rapidement trop basse. Il resterait un travail à faire sur cette fonction de sélection. Les benchmarks prenant parfois des heures à tourner, nous n'en avons pas eu l'occasion.

6 Conclusion

En conclusion, l'algorithme génétique donne de très bonnes solutions - si tant est que la taille de la population et d'autres hyperparamètres sont adaptés au problème envisagé. Le flou derrière ce qui fait la qualité d'une solution nous a poussé à modulariser, concevoir des benchmarks multiples, des représentations et des analyses variées de nos résultats. Le fichier (dont l'apport théorique est maigre) `executeur_comparaison_algos.py` est conçu dans l'optique de rendre accessible et "user-friendly" la majorité des options qui pourraient être intéressantes - pour un examinateur comme pour un éventuel utilisateur du programme. Le fichier `README.md` présente plus précisément les outils que nous avons construits et utilisés pour ce projet.

Merci pour votre attention.