# Métaheuristiques
# Planification de rencontres sportives


Thomas Landais

# Sommaire

# Introduction:

Les problèmes dits **NP-complets** posent des défis particuliers en raison de la complexité de leur résolution par des algorithmes classiques. En effet, pour ces problèmes, il est souvent impossible de trouver une solution optimale dans un temps raisonnable, surtout lorsque la taille du problème augmente. Une approche consiste donc à se tourner vers des méthodes heuristiques ou métaheuristiques, qui permettent d'obtenir des solutions approximatives mais satisfaisantes en un temps réduit. Dans notre cas, le **problème de planification sportive** illustre bien cette difficulté.

Pour notre étude, nous considérons les spécifications suivantes du tournoi sportif :

- Le tournoi regroupe un ensemble $TT$T d'équipes, dont le nombre est pair ($/T/$|T| pair).
- Chaque équipe doit rencontrer toutes les autres équipes **exactement une fois**. Nous ne prenons pas en compte la notion de matchs aller-retour.
- Le tournoi se déroule sur $/T/-1/T/$ - $1$|T|−1 semaines, chaque équipe jouant **un seul match par semaine**.
- Les matchs se déroulent à une date précise et sur un terrain spécifique, au cours d'une période donnée, avec un total de $/T/2/T//2$|T|/2 matchs par période.

L'objectif de ce projet est d'explorer plusieurs algorithmes de résolution métaheuristiques pour aborder ce problème de planification sportive, notamment les algorithmes suivants :

- **Algorithme glouton** : une approche simple consistant à choisir localement la meilleure solution possible à chaque étape.
- **Recherche locale** : une méthode qui explore les solutions voisines d'une solution courante pour tenter d'améliorer l'optimisation.
- **Recuit simulé** : une technique qui simule le processus de refroidissement des métaux pour échapper aux optima locaux et converger vers une solution globale.
- **Recherche taboue** : une approche qui utilise la mémoire pour éviter de revenir sur des solutions déjà explorées.
- **Algorithme génétique** : une approche inspirée de la biologie qui simule la sélection naturelle et le croisement génétique.

Ce projet permet de :

- Comprendre le fonctionnement de différents algorithmes de résolution métaheuristique.
- Implémenter et adapter ces algorithmes pour un problème concret.
- Analyser les performances et les résultats obtenus avec chaque méthode.

Les tests ont été effectués pour des tournois de **8, 10, et 12 équipes** afin de comparer la qualité des solutions et les temps de calcul pour chaque taille d'effectif. Ces expérimentations permettent d'évaluer l'impact de la taille de l'effectif sur la performance des algorithmes et d'identifier ceux qui produisent les meilleurs compromis entre précision et temps d'exécution.

## Fonction d'évaluation fitness :

La fonction d'évaluation, ou **fonction de fitness**, attribue un score à chaque solution de planification en fonction du nombre de règles violées. Elle prend en entrée une programmation des matchs et retourne un entier : plus ce score est élevé, plus la solution est pénalisée, ce qui reflète un nombre élevé d'erreurs dans la planification.

Voici les principales pénalités appliquées selon les types d'erreurs détectées dans la planification :

- **Match répété** : attribue une pénalité de 10 points lorsqu'une même rencontre entre deux équipes se retrouve programmée plusieurs fois dans le tournoi.
- **Multiples matchs par semaine** : chaque équipe ne doit jouer qu'une seule fois par semaine ; une pénalité de 5 points est appliquée chaque fois qu'une équipe joue plusieurs matchs en une semaine.
- **Dépassement de la limite de périodes** : chaque période contient un nombre limité de matchs, et toute infraction à cette limite est sanctionnée de 5 points.
- **Auto-rencontre** : une équipe ne peut pas jouer contre elle-même ; chaque rencontre de ce type reçoit une pénalité de 25 points.
- **Match manquant** : chaque paire d'équipes doit se rencontrer exactement une fois ; les paires d'équipes sans match sont pénalisées de 15 points.
- **Match non défini** : les espaces vides dans la programmation où un match est absent reçoivent une pénalité de 25 points.

En mode **verbose**, la fonction fournit un retour détaillé en identifiant précisément où se situent les infractions dans la planification, avec un message indiquant le type d'erreur associé.

## Fonction de test :

La fonction `mean_score` a été conçue pour évaluer la performance d'un algorithme de planification en répétant plusieurs itérations. Elle calcule des statistiques clés, telles que le score de fitness, le temps d'exécution, et les scores maximum et minimum obtenus lors de l'exécution de l'algorithme.

Dans cette fonction, chaque itération commence par mesurer le temps d'exécution. Un planning est généré à l'aide de l'algorithme spécifié, et le score de fitness de ce planning est calculé. Ce score est ensuite additionné à une somme totale pour permettre le calcul d'une moyenne à la fin des itérations.

La fonction suit également le temps d'exécution global et identifie les scores de fitness maximum et minimum. À la fin de toutes les itérations, elle calcule la moyenne des scores de fitness et du temps d'exécution, fournissant ainsi une vue d'ensemble des performances de l'algorithme testé.

## Glouton

Un **algorithme glouton** est une méthode de résolution de problèmes où chaque décision est prise en choisissant la meilleure option disponible à chaque étape, sans se soucier des conséquences à long terme. Cette approche vise à obtenir une solution acceptable rapidement, mais elle n'est pas garantie d'être optimale. Elle est souvent utilisée dans des problèmes où une solution raisonnablement bonne peut être obtenue par une série de choix locaux optimaux.

Dans notre algorithme, les matchs sont programmés semaine par semaine en s'assurant que les règles de base ne sont pas enfreintes. Pour chaque semaine, le programme suit les équipes déjà programmées pour éviter les doublons. Les équipes sont ensuite appariées pour les matchs dans des "périodes" (créneaux) en vérifiant que :

1. Les équipes sélectionnées ne sont pas déjà engagées dans un autre match cette semaine.

2. Chaque équipe n'apparaît pas plus de deux fois dans la période actuelle.

Si ces conditions sont remplies, le match est ajouté à l'horaire de la semaine ; sinon, le créneau reste vide. Une rotation des équipes est ensuite effectuée en conservant la première équipe fixe et en réorganisant les autres, assurant ainsi une répartition des confrontations. Cette approche gloutonne produit une programmation des matchs sans conflits majeurs et satisfait les règles principales.

| Nombre d'équipes | Temps moyen d'exécution (secondes) | Score moyen de fitness |
|---|---|---|
| 8 | 0.00003 | 400.0 |
| 10 | 0.00006 | 680.0 |
| 12 | 0.00005 | 960.0 |

L'algorithme glouton, bien qu'extrêmement rapide et simple à implémenter, se révèle peu adapté pour le problème de planification sportive. Son approche basée sur des choix locaux sans analyse globale conduit à un score de fitness médiocre, particulièrement pour des tournois de taille croissante. En raison de l'augmentation des violations de contraintes, l'algorithme glouton ne parvient pas à optimiser efficacement la planification, ce qui le rend peu intéressant pour ce type de problème où des solutions de meilleure qualité sont essentielles.

## Recherche locale :

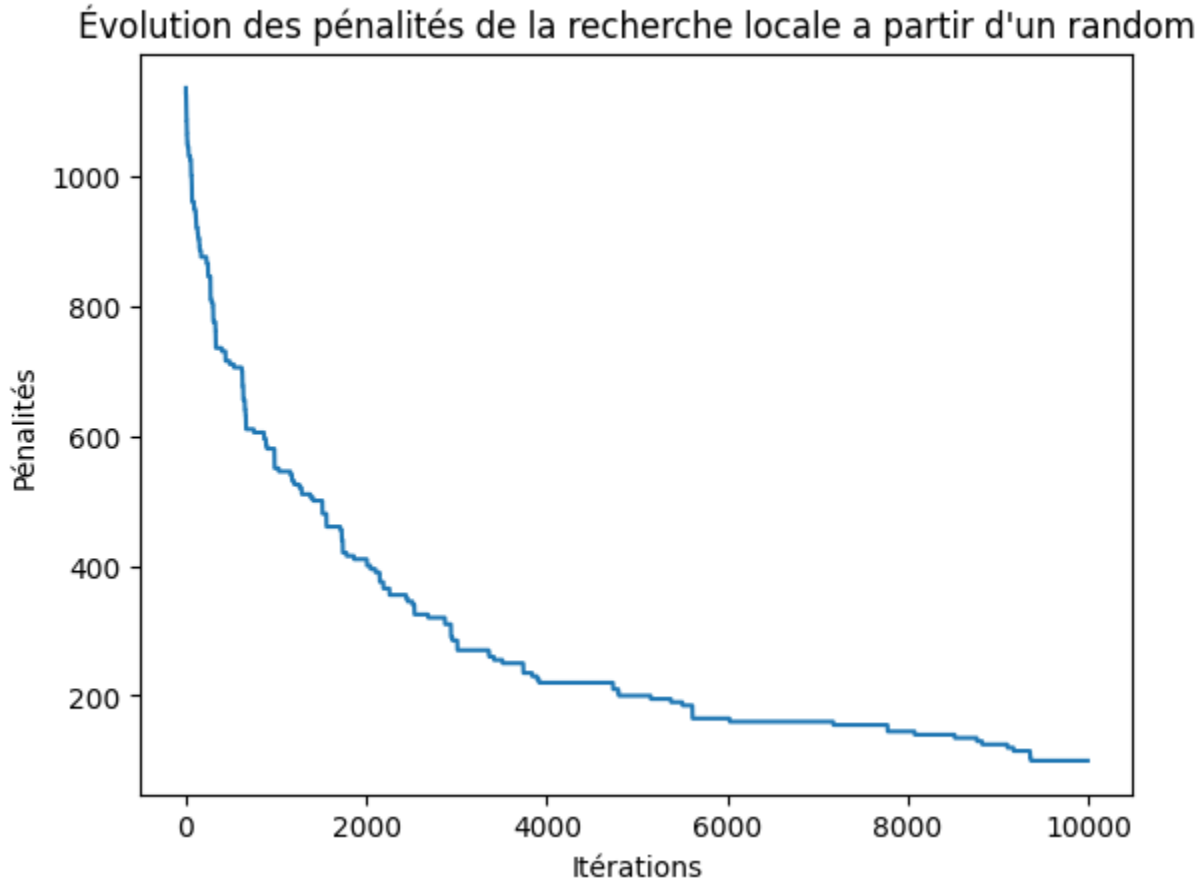
L'algorithme de **recherche locale** améliore une solution initiale en explorant ses voisins, c'est-à-dire des solutions obtenues par de petites modifications. Il est efficace pour atteindre rapidement une solution correcte, mais il peut se retrouver coincé dans un **optimum local**.

À chaque itération, l'algorithme effectue les étapes suivantes :

1. **Vérification de l'optimalité** : Si la solution actuelle n'a aucune violation de contraintes (`current_penalty == 0`), l'algorithme s'arrête.
2. **Génération d'un voisin** : L'algorithme sélectionne aléatoirement deux matchs dans le planning. Si l'un des matchs est vide, il est remplacé par un match aléatoire.
3. **Modification des matchs** : Avec une probabilité de 90 %, les deux matchs sont échangés. Sinon, une des équipes d'un matchs est remplacé par un match aléatoire.

4. **Évaluation** : Le planning modifié est évalué. L'algorithme répète cette procédure jusqu'à obtenir une solution satisfaisante ou atteindre un nombre maximal d'itérations.

| Nombre d'équipes | Temps moyen d'exécution (s) | Score moyen de fitness | Score maximal de fitness | Score minimal de fitness |
|---|---|---|---|---|
| 12 | 1.10714 | 130.60000 | 195.00000 | 80.00000 |
| 10 | 0.82983 | 66.40000 | 125.00000 | 20.00000 |
| 8 | 0.48572 | 25.90000 | 55.00000 | 0.00000 |




*Résultat sur une programmation de 12 équipes*

Les observations des résultats montrent que le score de fitness diminue rapidement au cours des premières 8000 itérations, ce qui indique une amélioration significative des

solutions initiales. Cependant, après cette phase, le score semble se stabiliser autour de la 8000ème itération, suggérant que l'algorithme se retrouve coincé dans un optimum local.

Ce phénomène est courant dans les algorithmes d'optimisation, où les améliorations initiales peuvent conduire à des solutions prometteuses, mais par la suite, l'algorithme a du mal à sortir de cet optimum local pour explorer de meilleures alternatives. Cette stagnation souligne la nécessité d'intégrer des mécanismes permettant d'échapper à ces pièges, tels que des stratégies de diversification ou des perturbations dans la recherche, afin d'améliorer la capacité de l'algorithme à explorer l'espace de recherche et à trouver des solutions optimales.

Les résultats obtenus à partir des évaluations montrent qu'aucune solution parfaite n'a été trouvée pour les algorithmes testés. Toutefois, une moyenne de score de fitness correcte a été atteinte, accompagnée d'un score maximal qui, bien que satisfaisant, reste relativement modeste.

Il est également important de noter que le temps d'exécution des algorithmes augmente de manière exponentielle en fonction du nombre d'équipes impliquées dans la planification.

## Recuit simulé :

Le recuit simulé est une méthode d'optimisation qui permet d'éviter les optima locaux. Il accepte temporairement des solutions moins bonnes pour explorer davantage l'espace de recherche. Bien qu'efficace, il nécessite un grand nombre d'itérations, ce qui peut rendre la recherche d'un optimum global incertaine.

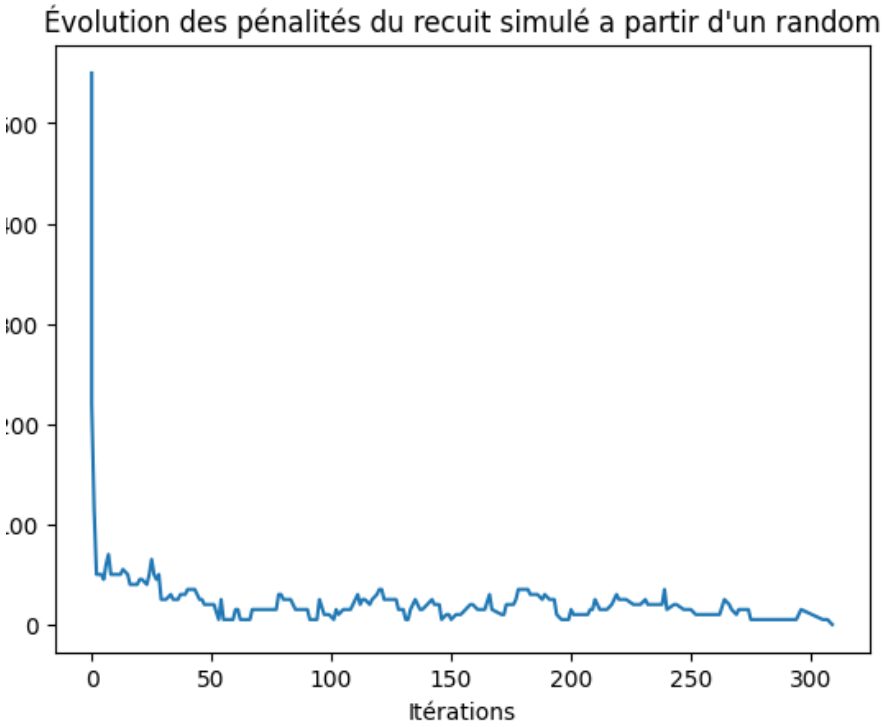
1. **Vérification de la pénalité** : Si la pénalité actuelle est nulle, le processus s'arrête.
2. **Génération d'un voisin** : Un nouveau calendrier est créé en échangeant deux matchs au hasard.
3. **Échange ou remplacement** : Avec une probabilité de 90 %, les matchs sont échangés ; sinon, un match aléatoire est ajouté.
4. **Évaluation de la pénalité** : La nouvelle pénalité est calculée.
5. **Critère d'acceptation** : La solution est acceptée si sa pénalité est meilleure ou selon une probabilité liée à la température.
6. **Mise à jour de la meilleure solution** : Si la nouvelle solution est meilleure ou que la température le permet, elle remplace l'ancienne, puis une recherche locale est appliqué dessus.

| Nombre d'équipes | Temps moyen d'exécution (s) | Score moyen de fitness | Score maximal de fitness | Score minimal de fitness | Température initiale | Taux de refroidissement |
|---|---|---|---|---|---|---|
| 12 | 7.20146 | 57.16667 | 145.00000 | 20.00000 | 230.32 | 0.94 |
| 10 | 8.48138 | 27.33333 | 60.00000 | 0.00000 | 227.82 | 0.97 |
| 8 | 4.42993 | 7.83333 | 25.00000 | 0.00000 | 126.53 | 0.89 |

*Résultat du recuit sur 8,10 et 12 équipes*

| | Période 1 | Période 2 | Période 3 | Période 4 |
|---|---|---|---|---|
| Semaine 1 | 6 vs 4 | 2 vs 3 | 7 vs 0 | 5 vs 1 |
| Semaine 2 | 3 vs 5 | 6 vs 2 | 0 vs 1 | 7 vs 4 |
| Semaine 3 | 7 vs 2 | 0 vs 6 | 5 vs 4 | 1 vs 3 |
| Semaine 4 | 1 vs 6 | 5 vs 7 | 4 vs 2 | 0 vs 3 |
| Semaine 5 | 7 vs 3 | 4 vs 1 | 5 vs 6 | 2 vs 0 |
| Semaine 6 | 0 vs 4 | 1 vs 7 | 6 vs 3 | 2 vs 5 |
| Semaine 7 | 0 vs 5 | 4 vs 3 | 2 vs 1 | 7 vs 6 |

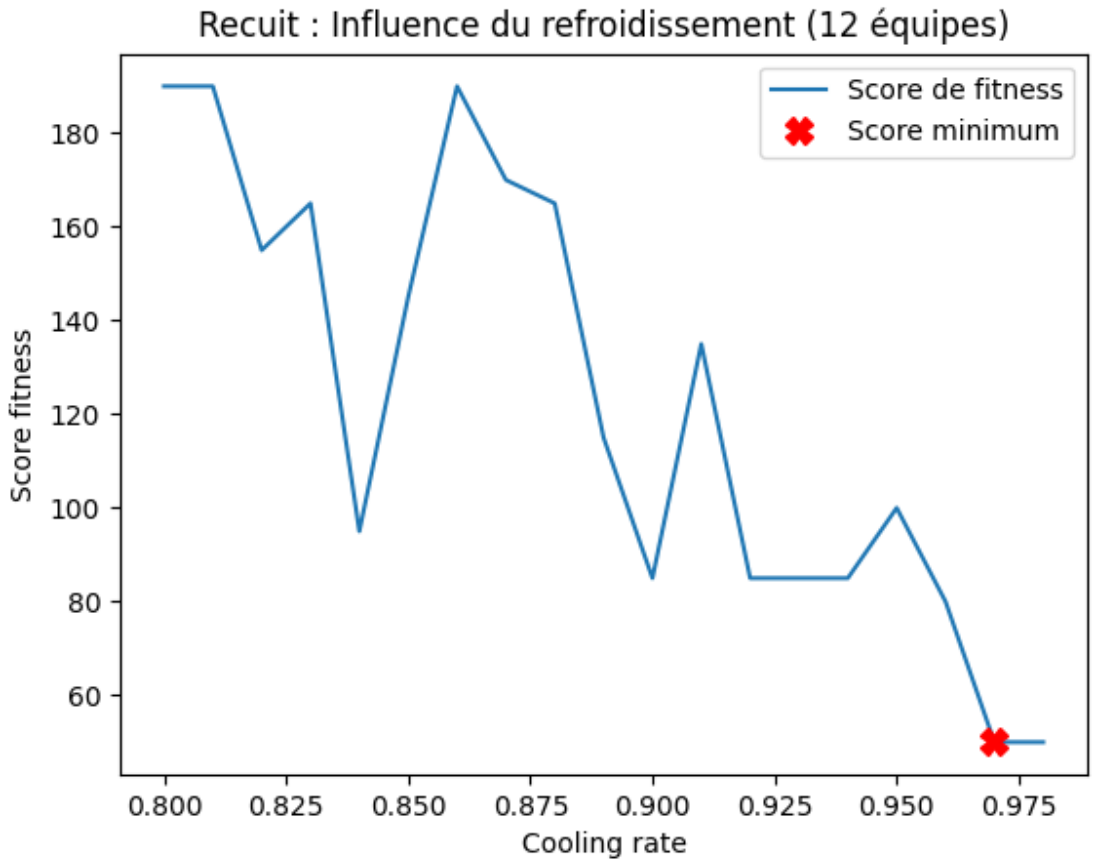
*Solution satisfaisante sur le recuit simulé sur 8 équipes*



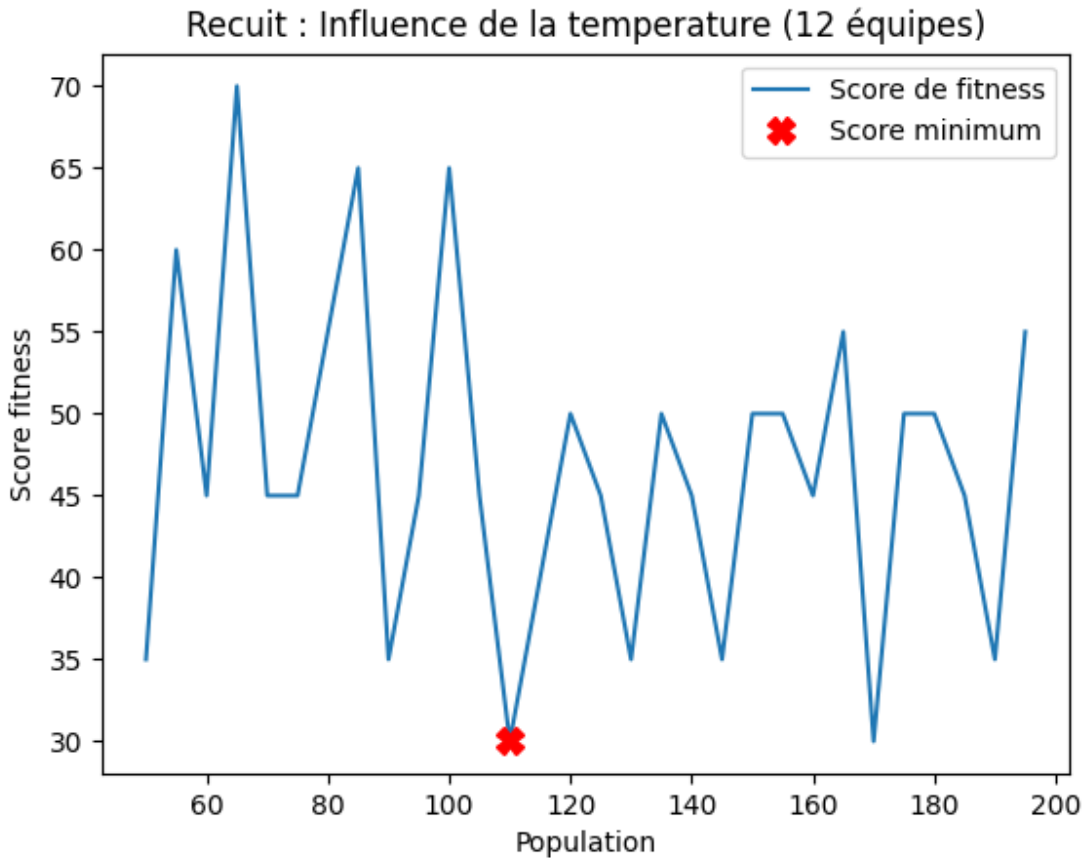
*Evolution du fitness score sur 8 équipes*

L'algorithme montre une descente rapide du score de fitness lors des premières itérations, avant d'explorer différents optima locaux. Une fois la température refroidie, il semble stagner sur le même score de fitness, atteignant une solution pour 8 équipes après environ 300 itérations.

Le recuit arrive a trouver une solution valable pour 10 équipes.




L'évolution du taux de refroidissement montre qu'un taux plus élevé est associé à un score de fitness plus faible. Cela indique que, dans ce cas, un taux de refroidissement élevé permet de visiter davantage d'optima locaux.




L'évolution du score de fitness en fonction de la température initiale indique que cette température de départ n'a pas d'impact significatif sur le score de fitness final atteint. Cela suggère que, quel que soit le niveau de température initial choisi, le résultat converge vers des scores de fitness similaires à la fin du processus d'optimisation.

## Recherche taboue :

La recherche tabou est une méthode d'optimisation combinatoire qui vise à éviter les optima locaux en interdisant temporairement certains mouvements ou solutions dans l'espace de recherche. Cela se fait à l'aide d'une "liste tabou", qui mémorise les mouvements récents pour les empêcher d'être répétés.

L'algorithme est construit de la façon suivante :

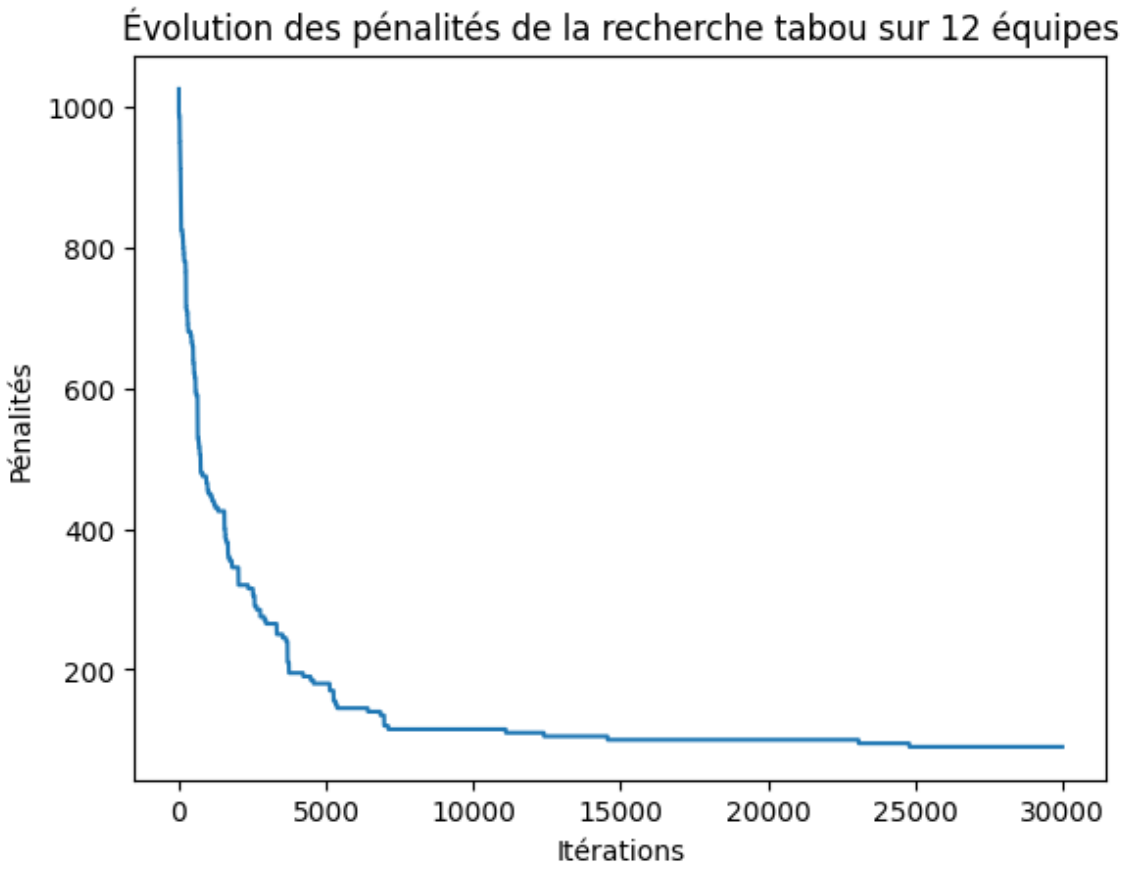
1. **Boucle d'itérations** : On exécute des itérations jusqu'à atteindre un nombre maximum spécifié (`max_iterations`).
2. **Vérification de la pénalité** : Si la pénalité actuelle est nulle, le processus s'arrête.

3. **Génération d'un voisin** :
    a. Avec une probabilité de 10 %, un changement aléatoire est effectué, sinon, un échange de deux matchs est réalisé.
    b. Chaque mouvement (changement ou échange) est vérifié par rapport à la liste tabou.
4. **Application du changement** : Si le mouvement n'est pas tabou, il est appliqué à la solution, et la pénalité de la nouvelle solution est calculée.
5. **Critère d'acceptation** : Si la nouvelle solution est meilleure (moins de pénalité) que l'actuelle, elle est acceptée, et le mouvement est ajouté à la liste taboue.
6. **Gestion de la liste tabou** : La liste tabou est mise à jour pour ne pas dépasser une taille définie (`tabu_tenure`), en supprimant les mouvements les plus anciens si nécessaire.

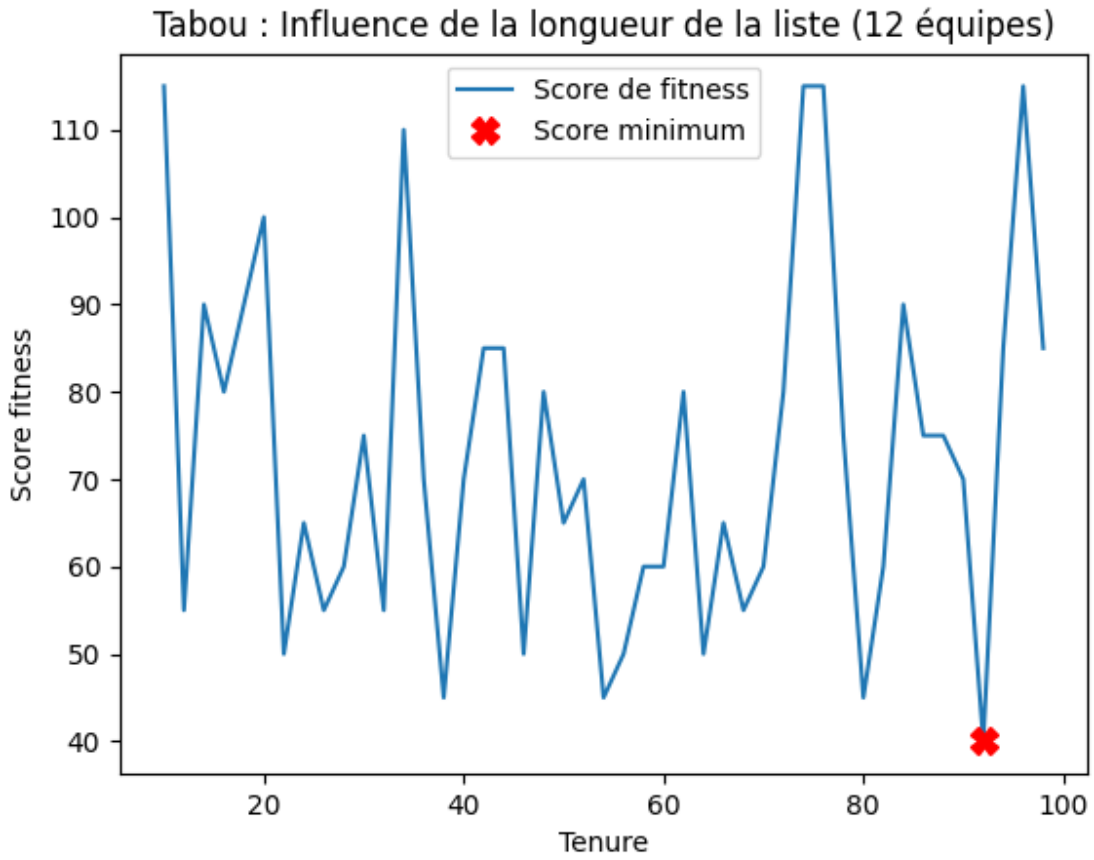
| Nombre d'Équipes | Temps Moyen d'Exécution (s) | Score Moyen de Fitness | Score Maximal de Fitness | Score Minimal de Fitness |
|---|---|---|---|---|
| 8 | 1.65272 | 18.20000 | 65.00000 | 5.00000 |
| 10 | 2.54654 | 38.00000 | 70.00000 | 15.00000 |
| 12 | 3.69062 | 70.00000 | 120.00000 | 35.00000 |

*Résultat du tabou sur 8,10 et 12 équipes*

Les résultats obtenus avec l'algorithme de recherche tabou montrent des performances intéressantes, même si aucune solution parfaite n'a été trouvée. Il est à noter que le temps d'exécution augmente avec le nombre d'équipes, ce qui est prévisible en raison de la complexité croissante du problème




L'évolution des performances de l'algorithme montre une décroissance rapide des scores de fitness jusqu'à atteindre environ 10 000 itérations. Après ce point, les scores semblent se stabiliser autour de 15 000 itérations. Cela indique que, bien que l'algorithme soit capable de trouver rapidement de meilleures solutions initiales, il atteint un plateau où les améliorations deviennent moins significatives. Cette stagnation peut suggérer que l'algorithme a exploré de manière exhaustive les solutions proches, mais qu'il peine à sortir de cet optimum local pour découvrir des solutions potentiellement meilleures.




L'évolution du score de fitness en fonction de la longueur de la liste taboue indique qu'il n'y a pas d'impact significatif sur le score final. Cela suggère que la taille de la liste taboue, bien qu'elle puisse influencer temporairement les mouvements explorés par l'algorithme, n'affecte pas substantiellement la qualité de la solution finale trouvée.

## Algorithme de recherche génétique simple :

Un **algorithme génétique (AG)** est une méthode d'optimisation inspirée des principes de la sélection naturelle et de l'évolution biologique. Il opère sur une population de solutions potentielles et utilise des mécanismes tels que la sélection, le croisement et la mutation pour générer de nouvelles solutions.

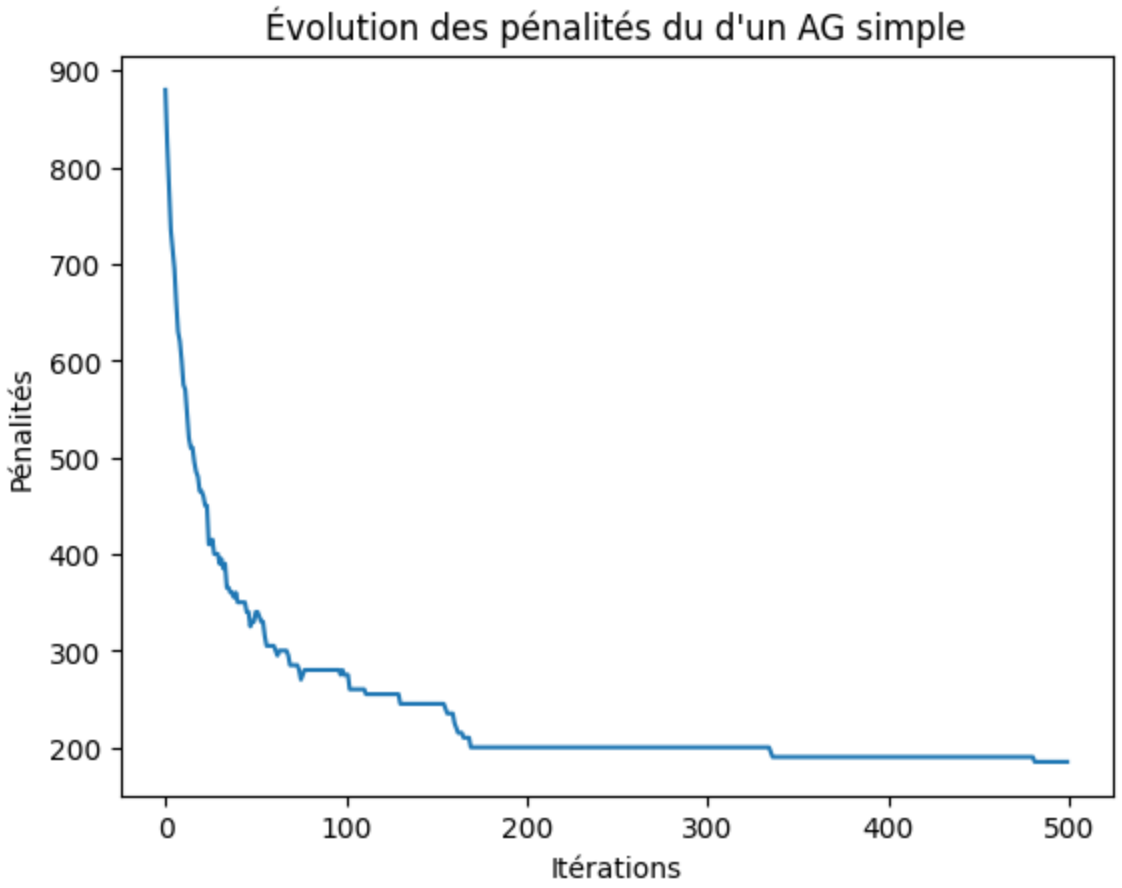
L'algorithme génétique que nous avons conçu suit une méthode structurée pour générer de nouvelles solutions à partir d'une population existante :

1. **Sélection des parents** : Nous choisissons aléatoirement des parents à partir de la population actuelle pour former une nouvelle génération.

2. **Croisement** : À partir de ces parents, nous appliquons un croisement pour créer des enfants sur ¼ de la population parente. Cette étape produit une partie de la nouvelle génération.
3. **Mutation** : Pour diversifier la population, nous sélectionnons aléatoirement d'autres parents et leur appliquons une mutation. Cela permet d'introduire des variations dans les solutions, ce qui aide à explorer de nouvelles pistes dans l'espace de recherche.
4. **Combinaison** : Nous continuons à combiner les enfants issus du croisement et de la mutation jusqu'à atteindre la taille souhaitée pour la nouvelle génération.

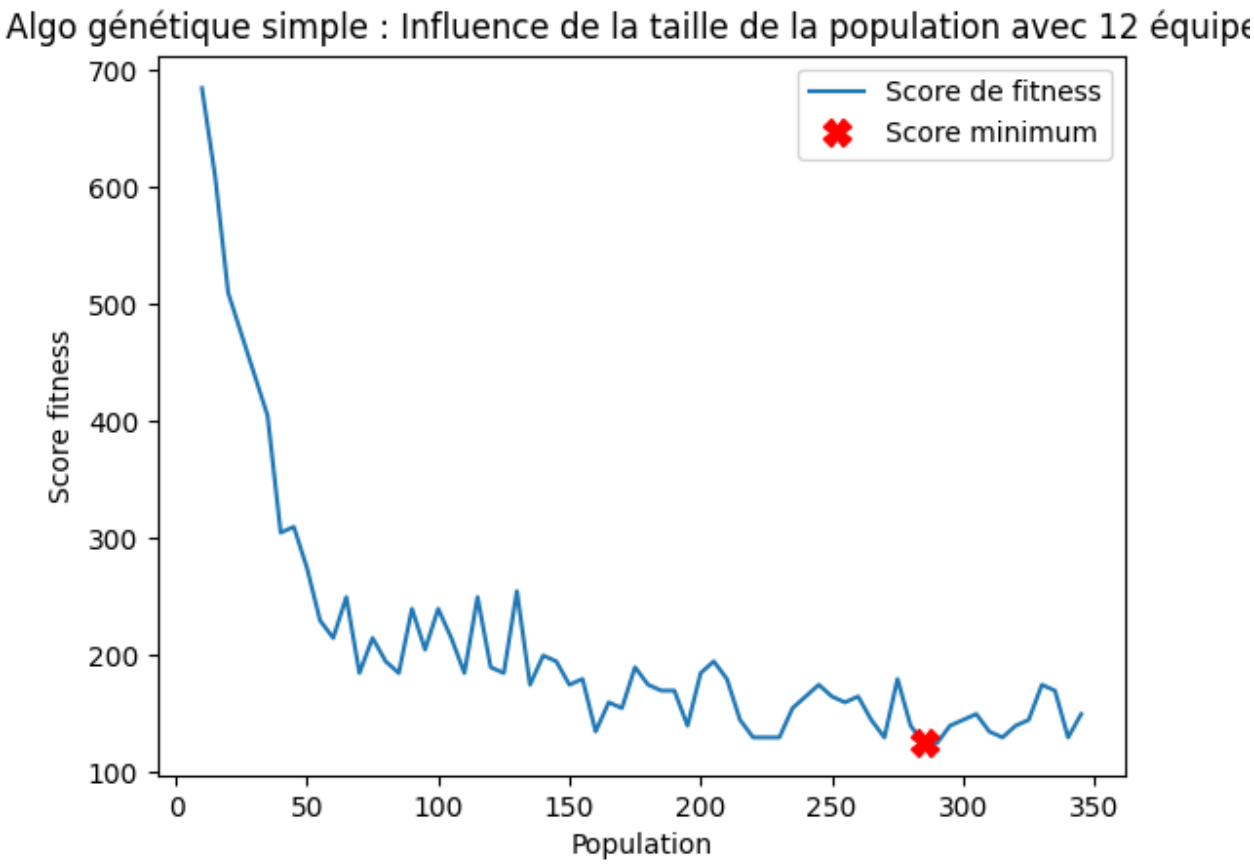
Ainsi, cet algorithme génétique utilise des techniques de sélection, de croisement et de mutation pour évoluer des solutions au fil des générations.

| Nombre d'équipes | Temps moyen d'exécution (s) | Score moyen de fitness | Score maximal de fitness | Score minimal de fitness |
|---|---|---|---|---|
| 12 | 3.59508 | 201.33333 | 245.00000 | 165.00000 |
| 10 | 2.53486 | 66.33333 | 105.00000 | 45.00000 |
| 8 | 1.62813 | 13.50000 | 30.00000 | 0.00000 |



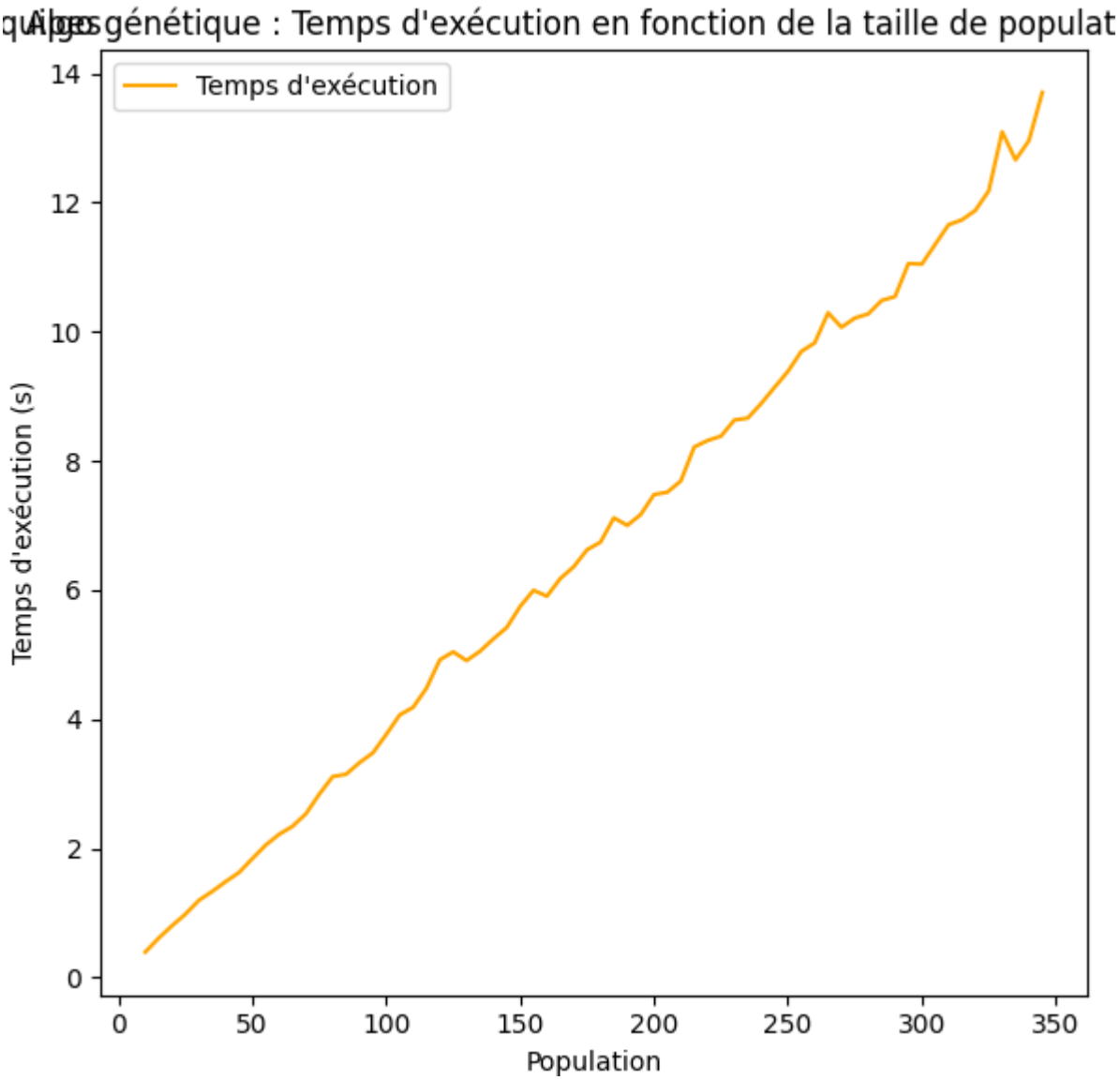

*Evolution du fitness score par itération sur 12 équipes*

La courbe montre une diminution rapide des scores de fitness durant les 100 premières itérations, indiquant une amélioration significative des solutions générées. Après cette phase initiale, le score semble se stabiliser autour de la 400ème itération, suggérant que l'algorithme a atteint un plateau et n'explore plus de nouvelles solutions améliorées de manière significative. Ce comportement est courant dans les algorithmes d'optimisation, où une phase d'exploration rapide est suivie d'une phase de consolidation.




Ce graphique illustre l'évolution du score de fitness en fonction de la taille de la population dans l'algorithme génétique. On observe que la taille de la population a un impact significatif sur le score de fitness : une population supérieure à 100 individus est cruciale pour atteindre de bons résultats. Cependant, après avoir dépassé une taille de population de 250, l'augmentation de la population n'apporte pas d'amélioration significative du score de fitness, car la courbe se stabilise.

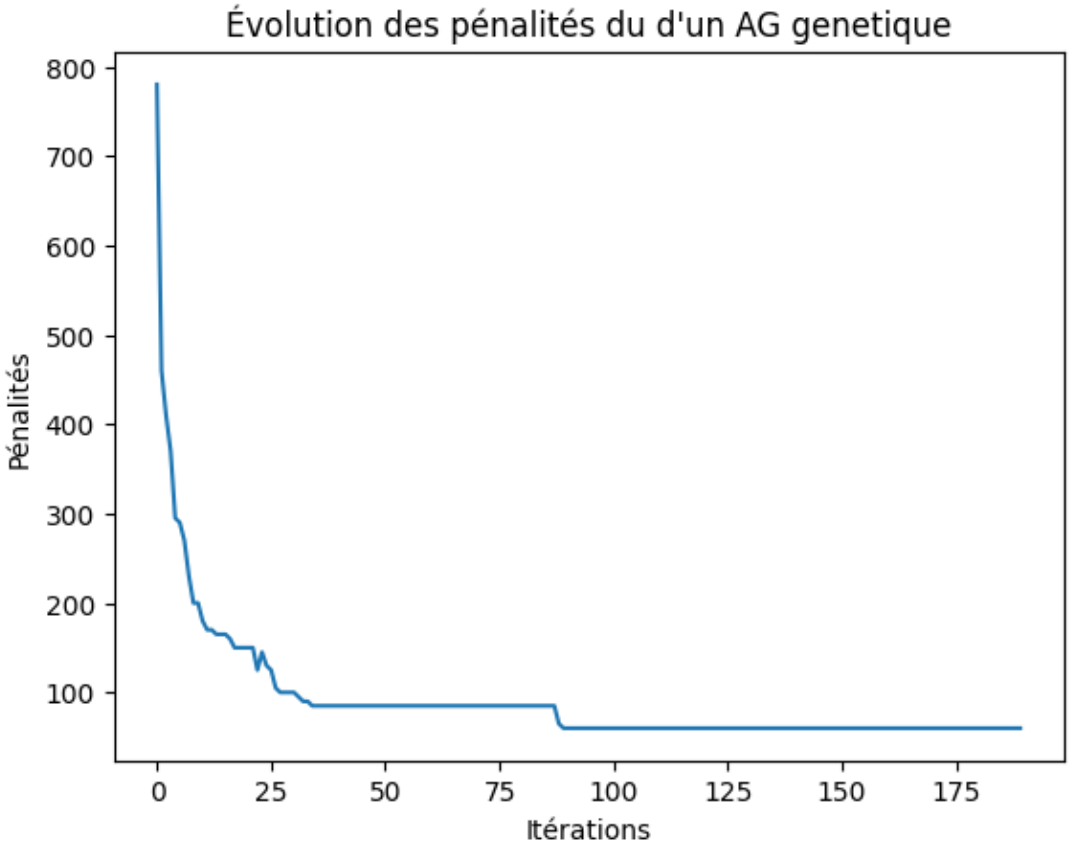
De plus, il est important de noter que l'augmentation de la taille de la population entraîne une augmentation considérable du temps d'exécution, ce qui soulève des questions sur l'efficacité des ressources et le compromis entre performance et temps de calcul.




L’Augmentation du temps de calcul en fonction de la population est linéaire.

# Algorithme de recherche génétique hybride :

Cette approche ressemble à un algorithme génétique simple, enrichi par l'utilisation de la recherche locale sur certains éléments de la population, ce qui augmente les chances de trouver des solutions optimales tout en conservant la diversité génétique nécessaire à l'algorithme.

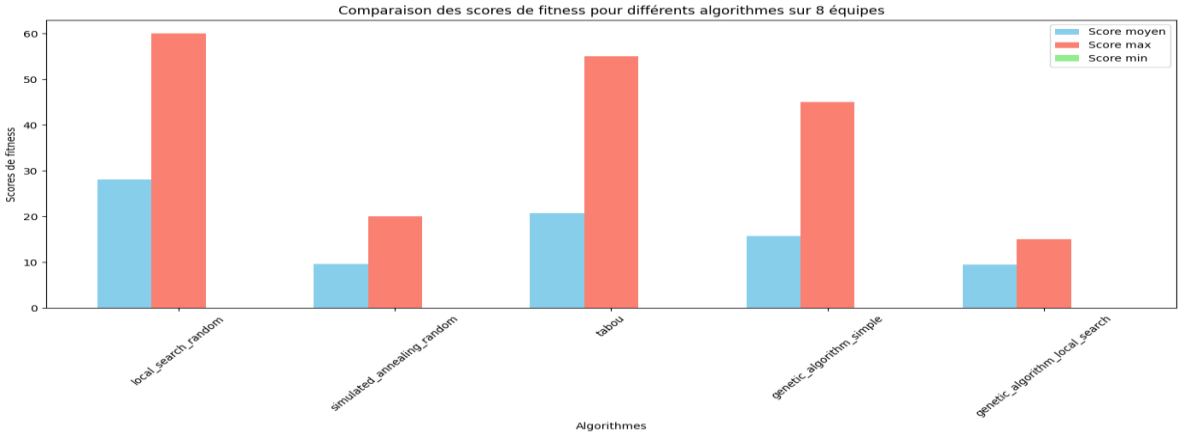


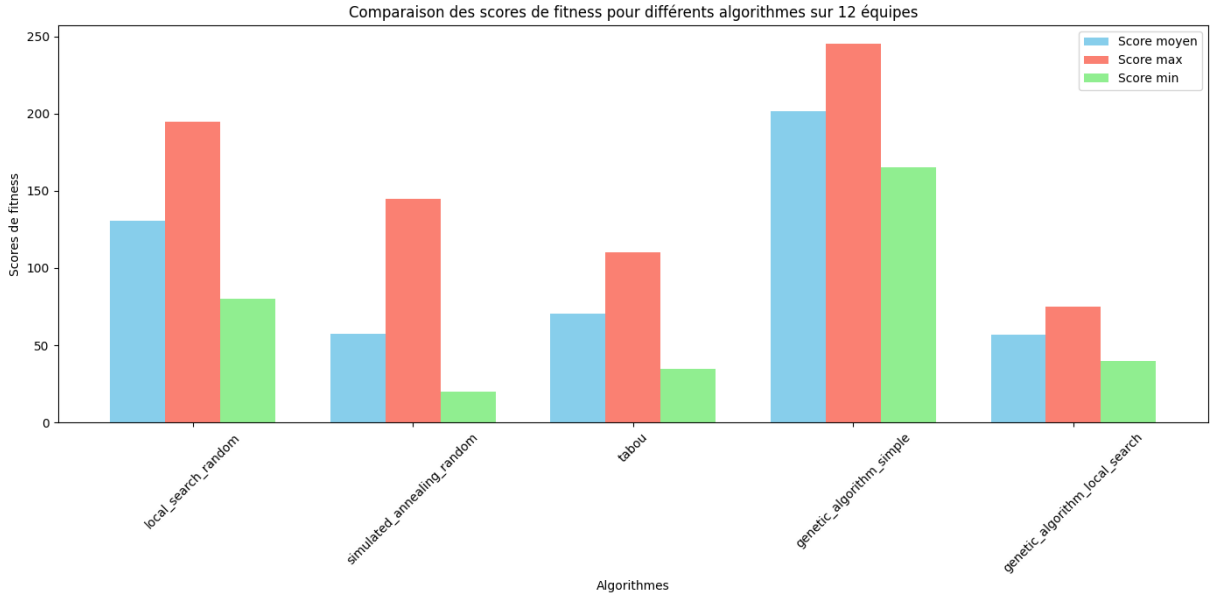

*Evolution du fitness score par itération sur 12 équipes*

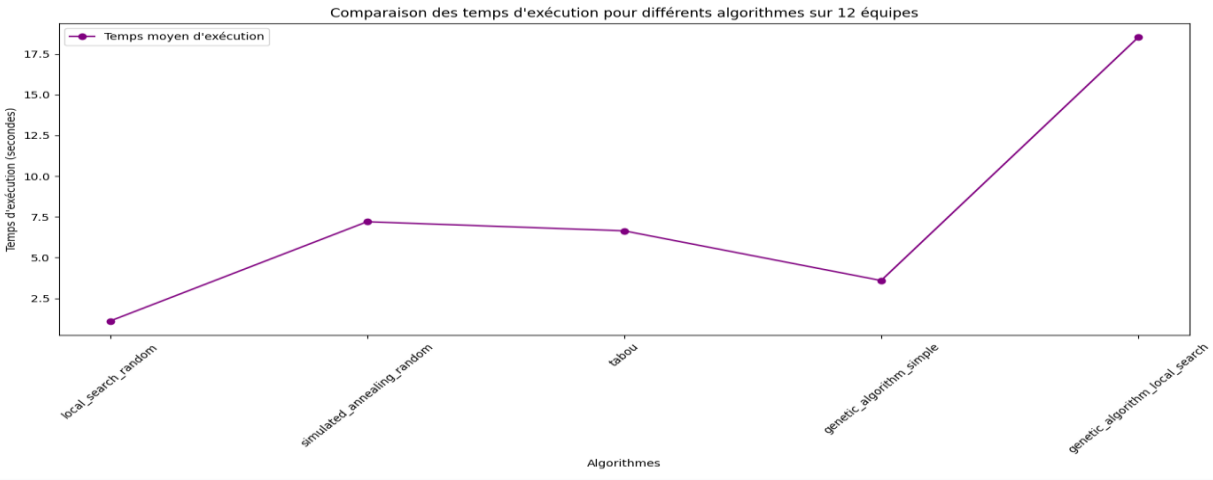
Voici un résumé de vos observations sur l'algorithme génétique avec recherche locale :

- **Comportement de la courbe** : La courbe de performance descend rapidement au début, puis se stabilise autour du même score après environ 25 itérations.
- **Performance** : Les scores obtenus sont excellents, mais le temps d'exécution est significativement plus long que pour les autres algorithmes. Cela est principalement dû à la recherche locale appliquée sur 5 éléments de la population à chaque itération.
- **Robustesse des résultats** : Les résultats sont solides avec des scores moyens, maximaux et minimaux qui montrent une performance constante et fiable.

## Conclusion :




## Récapitulatif des Résultats des Algorithmes :

Tous les algorithmes parviennent à trouver une solution satisfaisante pour 8 équipes. Cependant, seuls le recuit simulé y parviennent pour 10 équipes, et aucun ne réussit pleinement pour 12 équipes.

Analyse des algorithmes :

***Recherche locale*** : Rapide et efficace, mais tend à se figer dans des optima locaux, ce qui limite parfois sa performance.

***Recuit simulé*** : Moins rapide, mais il équilibre bien exploitation et exploration. Il obtient de bons scores de fitness minimum, même si son score maximum reste inférieur à celui d'autres méthodes, et il est globalement moins stable que l'AG hybride.

***Algorithme génétique simple*** : Il génère des scores modestes, mais son temps de calcul est relativement faible.

***Algorithme Tabou*** : Stable et régulier, il représente un bon compromis entre régularité des résultats et temps de calcul.

***Algorithme génétique hybride*** : Il produit les meilleurs scores, mais son temps de calcul est au moins deux fois plus long que celui des autres algorithmes. De plus, sa performance se dégrade avec l'augmentation du nombre d'équipes.

Chaque algorithme présente donc des avantages et des inconvénients selon les objectifs et contraintes.

Il est important de tester chaque algorithme en faisant varier les paramètres, car cela permet d'identifier les configurations optimales pour chaque situation. Chaque algorithme a des comportements différents selon les paramètres choisis, tels que les taux de mutation, la taille de la population, la température initiale, ou encore le taux de refroidissement.

Tester différentes valeurs pour ces paramètres permet de :

- **Optimiser les performances** : Trouver les meilleures configurations qui maximisent les résultats (par exemple, le score de fitness maximum ou la rapidité de convergence).
- **Éviter les pièges des optima locaux** : Certains paramètres peuvent influencer la capacité de l'algorithme à explorer de manière efficace l'espace de recherche.
- **Comprendre les compromis** : Chaque algorithme peut être plus adapté à des contraintes spécifiques (temps de calcul, qualité de la solution, etc.), et tester les paramètres aide à évaluer ces compromis.

- **Adapter aux variations des données** : Selon le nombre d'équipes ou la complexité du problème, les paramètres peuvent devoir être ajustés pour obtenir des résultats satisfaisants.

Ainsi, la variation des paramètres est cruciale pour tirer pleinement parti des avantages de chaque algorithme tout en minimisant leurs inconvénients.