République Algérienne Démocratique et Populaire

Ministère de L'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie

Houari Boumediene

FACULTÉ D'INFORMATIQUE

Mini-projet

PROBLÈME DU SAC À DOS MULTIPLE

PARTIE II

Elaboré par :

- MEDRISS Amina 202031081328
- BELLILI Idir 202031049355

Section : SII

Groupe; 1

- Mai 2024 -

Sommaire

List of Algorithms

Liste des Figures

1 Introduction

Dans la première partie de ce projet, nous avons abordé la résolution du problème du sac à dos multiple (Multiple Knapsack Problem) à l'aide de méthodes de recherche aveugles et informées. Plus précisément, nous avons implémenté les algorithmes de parcours en largeur (BFS), de parcours en profondeur (DFS) et l'algorithme A* exploitant une heuristique. Si ces méthodes se sont avérées efficaces pour des instances de petite taille, leur temps d'exécution explose rapidement avec l'augmentation de la complexité du problème.

Pour traiter des instances de plus grande taille, il est nécessaire de recourir à des approches plus performantes. C'est dans ce contexte que s'inscrit cette deuxième partie du projet, qui vise à résoudre le problème du sac à dos multiple à l'aide de métaheuristiques. Les métaheuristiques sont des techniques d'optimisation approximatives qui permettent d'obtenir des solutions de bonne qualité dans un temps raisonnable, même pour des problèmes de grande taille.

Dans cette partie, nous allons explorer deux métaheuristiques distinctes: les algorithmes génétiques (GA) et l'optimisation par essaim d'abeilles (BSO). Ces méthodes s'inspirent respectivement des principes de l'évolution naturelle et du comportement des abeilles à la recherche de nourriture.

Après avoir décrit le fonctionnement détaillé de chacune de ces métaheuristiques, nous procéderons à une série d'expérimentations visant à déterminer les réglages optimaux de leurs paramètres. Nous évaluerons également leurs performances en fonction de la taille du problème à résoudre. Enfin, nous comparerons les résultats obtenus par ces deux approches et les mettrons en perspective avec ceux obtenus lors de la première partie du projet.

2 Optimisation du Problème de Sac à Dos Multiple par Metaheuristique : Approche et Résolution

2.1 Algorithme génétique (AG)

L'algorithme génétique (AG) est une méthode d'optimisation inspirée du processus de sélection naturelle. Il fonctionne en évoluant progressivement une population de solutions candidates vers des solutions de meilleure qualité.

Dans notre contexte, l'objectif de l'AG est de trouver une répartition optimale des objets entre les différents sacs tout en respectant les contraintes de poids de chaque sac. Pour ce faire, nous représentons le problème sous forme de chromosomes, où chaque chromosome correspond à une configuration potentielle des objets dans les sacs.

L'AG opère en itérant à travers des générations de populations de chromosomes. À chaque génération, une sélection est effectuée parmi les chromosomes en fonction de leur aptitude (fitness) par rapport à la fonction objectif. Les chromosomes les mieux adaptés ont une probabilité plus élevée d'être sélectionnés pour la reproduction.

La reproduction implique la combinaison de paires de chromosomes pour créer de nouveaux individus, simulant ainsi le croisement génétique. Ces nouveaux individus peuvent également subir des mutations aléatoires, introduisant ainsi une diversité génétique dans la population.

En résumé, l'AG explore de manière itérative l'espace des solutions en utilisant des opérateurs de sélection, de croisement et de mutation pour évoluer vers des solutions optimales pour le problème du sac à dos multiple.

Algorithm 1 GA

Data: *but*: Noeud, *sacs*: tableau de sac, *objets*: tableau d'objet, $POPULATION\_SIZE$, $POPULATION\_SIZE$, *popList*, $MAX\_GENERATIONS$

Result: *but*: Noeud avec solution optimal

$popList \leftarrow GénérationAléatoirePopulation(N)$

$Evaluation(popList)$

for $i = 1; i <= MAX\_GENERATIONS$ **do**

  $Parents \leftarrow Selection(popList)$

  $EnfantsC \leftarrow Croisement(Parents)$

  $EnfantsM \leftarrow Mutation(EnfantsC)$

  $Evaluation(EnfantsC)$

  $Evaluation(EnfantsM)$

  $popList \leftarrow Remplacement(popList, EnfantsC, EnfantsM)$

end

return $MeilleurIndividu(popList)$

Les opérateur de l'algorithme génétique

- Le croisement

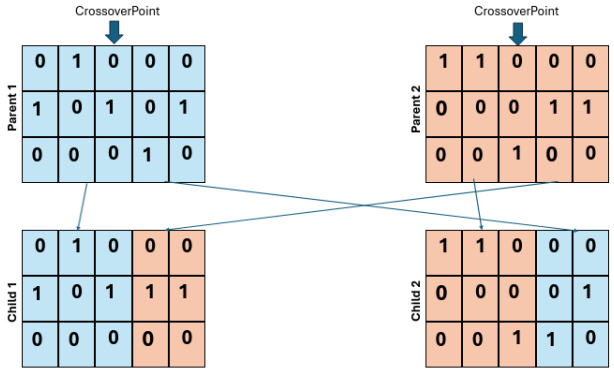


Figure 1: Le croisement

- La mutation

Figure 2: La mutation

Choix des paramètres empiriques:

Le choix des paramètres empiriques dans un algorithme génétique (AG) est crucial pour son efficacité.

- **Taille de la population**: Détermine le nombre d'individus dans chaque génération. Une population plus grande peut conduire à une exploration plus exhaustive de l'espace des solutions, mais elle augmente également le temps de calcul.
- **Taux de croisement:** Détermine la probabilité qu'une paire de chromosomes subisse un croisement génétique lors de la reproduction. Une valeur élevée favorise l'exploration de nouvelles régions de l'espace de recherche, tandis qu'une valeur faible peut maintenir la convergence vers des solutions locales.
- **Taux de mutation:** Détermine la probabilité qu'un gène dans un chromosome subisse une mutation. La mutation aide à introduire une diversité génétique dans la population, explorant ainsi de nouvelles solutions potentiellement meilleures. Une valeur trop élevée peut cependant perturber la convergence vers des solutions optimales.

- **Nombre de générations:** Détermine le nombre d'itérations ou de générations que l'AG effectue avant de terminer. Un nombre insuffisant de générations peut entraîner une convergence prématurée vers des solutions suboptimales, tandis qu'un nombre excessif peut entraîner un temps de calcul plus long sans amélioration significative des solutions.

- **Fonction de fitness:** la fonction d'évaluation de la fitness attribue un score à chaque individu de la population, représentant à quel point cette solution est performante ou adéquate par rapport aux objectifs du problème.

Dans notre cas, nous avons défini la fonction de fitness comme étant l'inverse de la somme des valeurs des objets non insérés, ajoutée à la somme des poids des objets insérés. Cette approche vise à maximiser la valeur totale des objets insérés tout en minimisant la somme des valeurs des objets non insérés, ce qui correspond à l'objectif de maximisation de la valeur et de minimisation du poids dans le problème du sac à dos multiple.

$$f = \frac{1}{\sum_{i=1}^{n} V_i - V_{\text{utilisé}}} + w_{\text{utilisé}}$$

- Pour le choix des autres paramètres de l'algorithme génétique, une approche empirique a été adoptée. Des tests ont été menés sur différentes instances du problème du sac à dos multiple, en faisant varier les paramètres restants de l'algorithme.

- L'expérimentation a été conduite en ajustant individuellement chaque paramètre, tout en maintenant les autres paramètres constants.

- Les performances de l'algorithme ont été évaluées en termes de qualité des solutions obtenues et de la convergence de l'algorithme.

- Les résultats de ces tests empiriques ont été analysés pour déterminer les combinaisons de paramètres les plus efficaces pour résoudre efficacement le problème du sac à dos multiple.

Analyse des tests

Nous avons mené plusieurs séries de tests sur des instances variées, en ajustant les paramètres empiriques, sur une période de cinq jours consécutifs.

Materiels

	PC
Processeur	11th Gen Intel i5-1135G7 (8) @ 4.200GHz
RAM	8GB 3200MHz
SE	Windows 10 Pro

Table 1: Materiels

Vous trouverez les resultats en format csv sur ce lien:ici

- **Taille de population :** 50
- **Nombre de générations :** 100
- **Taux de mutation :** 0.5
- **Taux de croisement :** 0.2

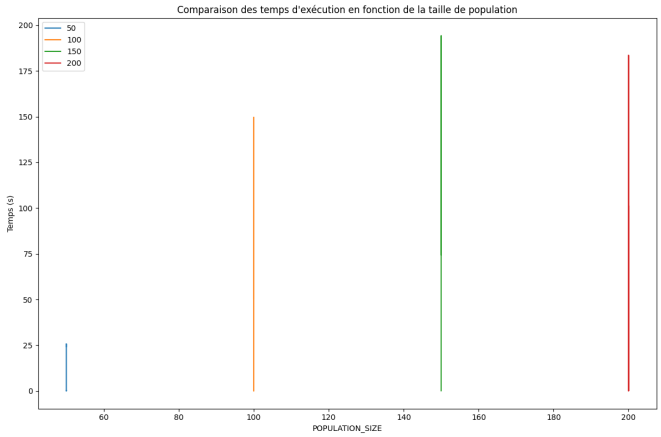


Figure 3: Comparaison des temps d'exécution en fonction de la taille de population

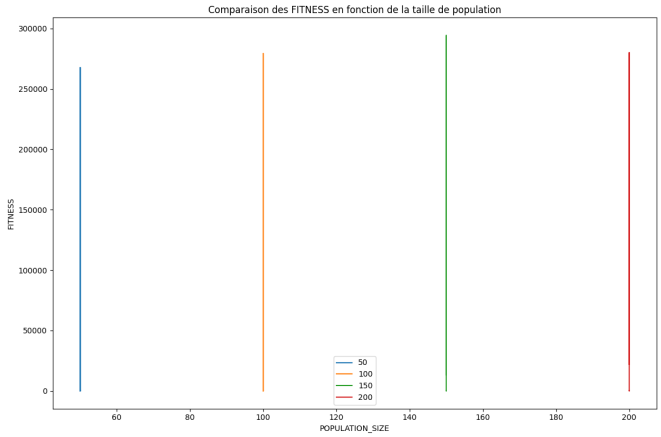


Figure 4: Comparaison des fitness en fonction de la taille de population

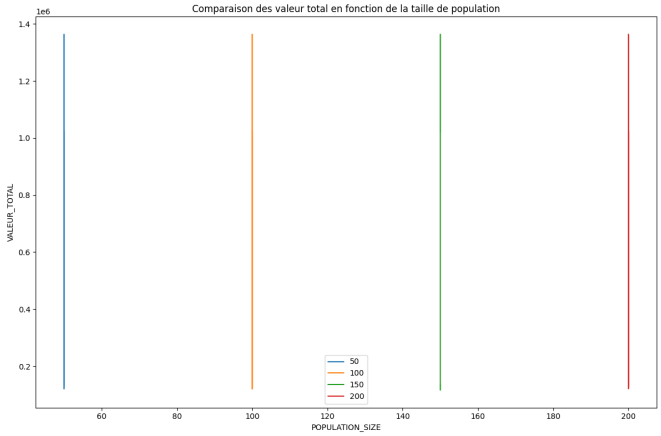


Figure 5: Comparaison des valeurs totales d'exécution en fonction de la taille de population

En analysant les différents graphiques, il est observé qu'en augmentant la taille de la population, le temps d'exécution augmente également. Les variations en termes de fitness et de valeurs totales ne sont pas significatives. Par conséquent, nous sélectionnerons une taille de population égale à 50, où les meilleurs résultats en termes de temps d'exécution ont été obtenus.

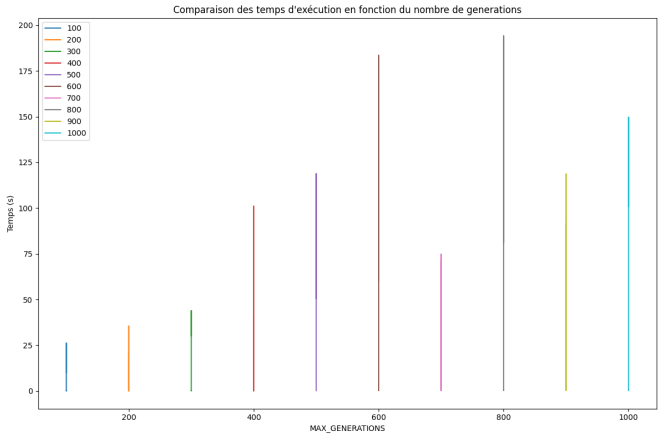


Figure 6: Comparaison des temps d'exécution en fonction du nombre de générations

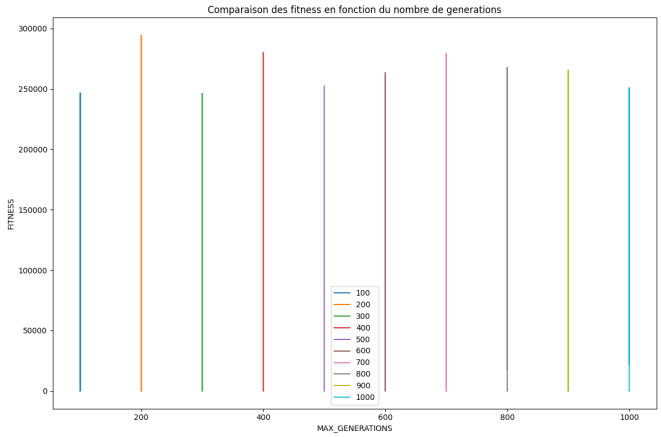


Figure 7: Comparaison des fitness en fonction du nombre de générations

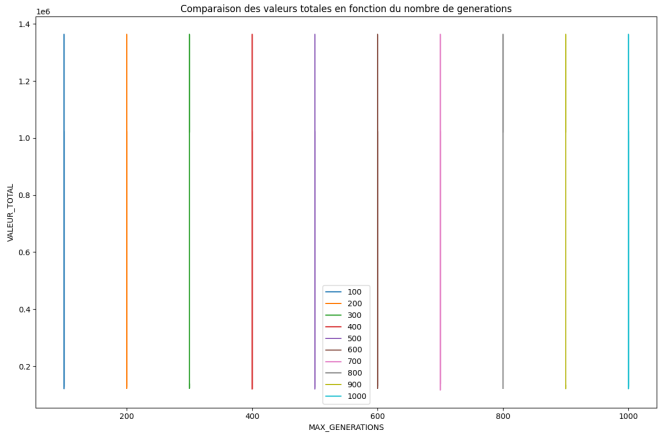


Figure 8: Comparaison des valeurs totales d'exécution en fonction du nombre de générations

En examinant les divers graphiques, il est remarqué que l'augmentation du nombre de générations entraîne également une augmentation du temps d'exécution. Les fluctuations en termes de fitness et de valeurs totales ne sont pas considérables. Par conséquent, nous opterons pour un nombre de générations de 100, où les performances les plus favorables en matière de temps d'exécution ont été constatées.

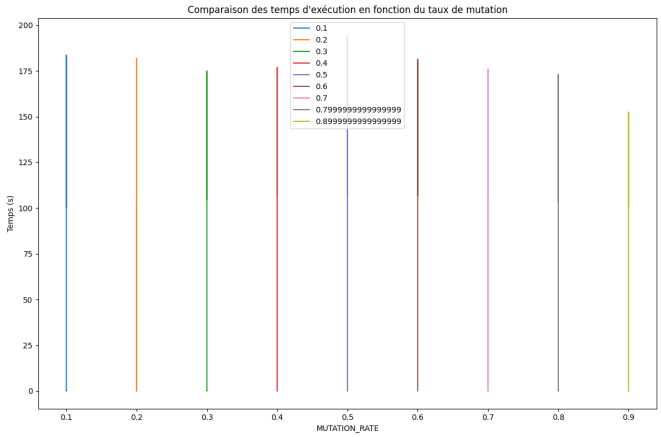


Figure 9: Comparaison des temps d'exécution en fonction du taux de mutation

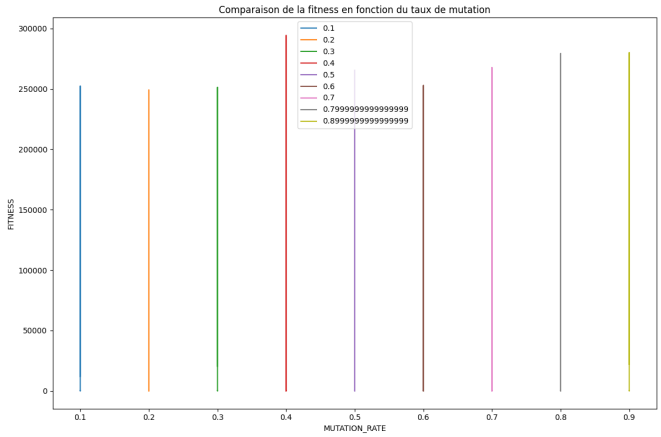


Figure 10: Comparaison des fitness en fonction du taux de mutation

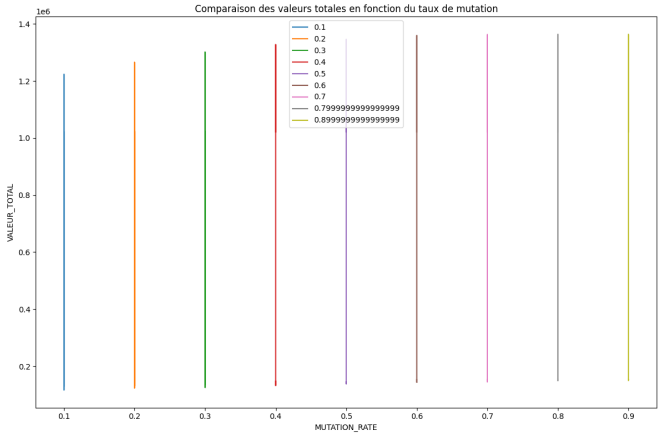


Figure 11: Comparaison des valeurs totales d'exécution en fonction du taux de mutation

En examinant attentivement les différents graphiques, il est constaté que l'analyse du temps et de la fitness est entravée par leur caractère variable et instable. En revanche, en ce qui concerne la valeur totale, une évolution exponentielle est observée, se stabilisant à un taux de mutation de 0.5.

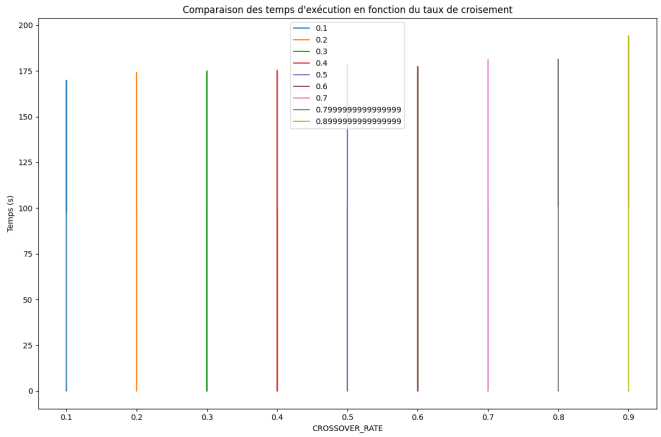


Figure 12: Comparaison des temps d'exécution en fonction du taux de croisement

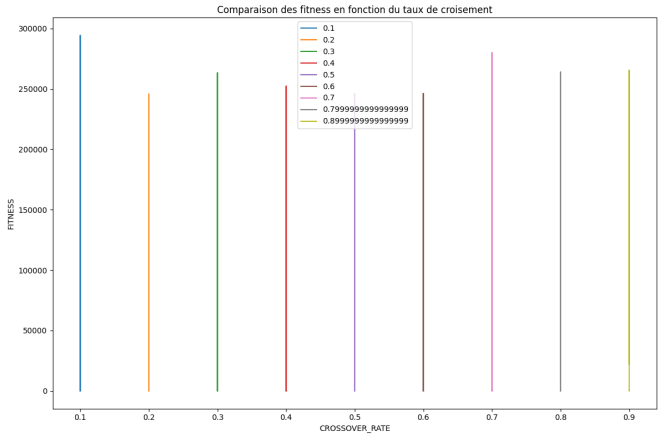


Figure 13: Comparaison des fitness en fonction du taux de croissement

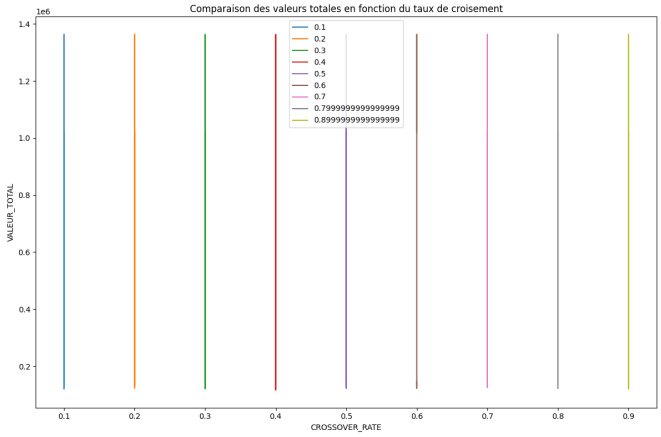


Figure 14: Comparaison des valeurs totales d'exécution en fonction du taux de croisement

Après une analyse approfondie des différents graphiques, il est observé que le taux de croisement de 0.2 produit des résultats supérieurs dans tous les aspects étudiés.

2.2 Bee swarm optimization algorithm(BSO)

L'Optimisation par Essaim d'Abeilles (BSO) est une métaheuristique d'optimisation bio-inspirée qui imite le comportement collectif des abeilles à la recherche de ressources alimentaires. Dans notre contexte, le BSO vise à trouver une répartition optimale des objets entre les différents sacs, tout en respectant les contraintes de poids de chaque sac.

Le processus BSO s'articule autour d'une colonie d'abeilles constituée de trois groupes principaux : les abeilles ouvrières, les abeilles spectatrices et les abeilles scouts. Chaque abeille représente une solution candidate au problème du sac à dos multiple.

Les abeilles ouvrières exploitent les sources de nourriture (solutions) existantes en recherchant des solutions de meilleure qualité dans leur voisinage. Les abeilles spectatrices, quant à elles, choisissent de suivre les abeilles ouvrières les plus performantes en fonction de leur aptitude (fitness) par rapport à la fonction objectif.

Périodiquement, des abeilles scouts sont envoyées pour explorer de nouvelles régions de l'espace de recherche, introduisant ainsi de la diversité dans la population de solutions.

Le processus itératif du BSO consiste à faire évoluer la colonie d'abeilles en alternant entre l'exploitation des meilleures solutions actuelles par les abeilles ouvrières et l'exploration de nouvelles régions prometteuses par les abeilles scouts. Cette dynamique permet à l'algorithme de converger progressivement vers des solutions optimales pour le problème du sac à dos multiple.

Algorithm 2 BSO

Data: NBR_BEE, $MAX\_CHANCES$, $LIST\_DANSE$, $LIST\_TABOO$, $Size\_Problem$, $MAX\_ITERATION$

Result: $Best\_SOL$: Noeud avec solution optimal

$SRef \leftarrow BeeInit(Size\_Problem)$

for $i = 1; i <= MAX\_ITERATION$ **do**

  $List\_TABOO.add(SRef)$

  $SEARCH\_AREA \leftarrow GenerateSearchPoint(SRef, FLIP)$

  for $k = 1; k <= NBR\_BEE$ **do**

    $LIST\_DANSE \leftarrow LOCALSEARCH(SEARCH\_Point)$

  end

  $Best\_SOL \leftarrow SelectSOL(SRef, Sbest)$

end

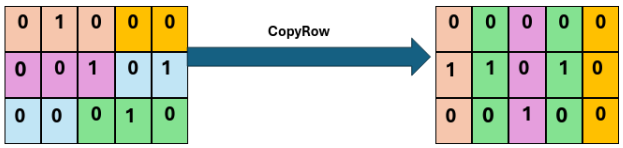
return $Best\_SOL$

Les Opérateurs

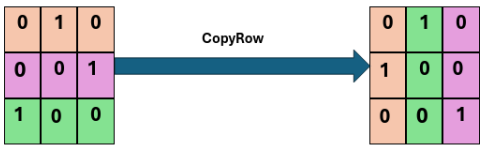
- La fonction CopyRow:

  dans cette fonction nous avons trois cas

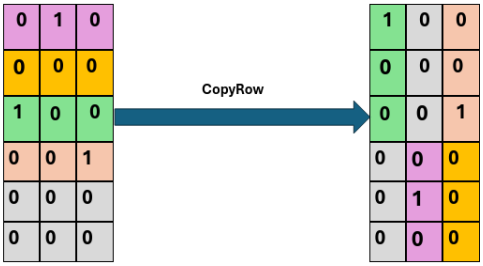
  - Cas: Nombre d'objets superieur au nombre de sacs



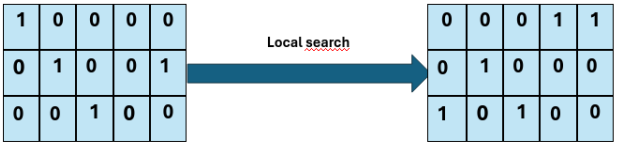
  - Cas: Nombre de sac égale le nombre d'objets



  - Cas: Nombre de sac supérieur au nombre d'objets



- Local search



Choix des paramètres empiriques:

Analyse des tests

Nous avons mené plusieurs séries de tests sur des instances variées, en ajustant les paramètres empiriques, sur une période de cinq jours consécutifs.

Materiels

	PC
Processeur	9th Gen Intel i3-9100f @ 3.60GHz
RAM	8GB 2600MHz
SE	Windows 10 Pro

Table 2: Materiels

Vous trouverez les resultats en format csv sur ce lien:ici

- **Nombre itérations :** 10
- **Nombre d'abeilles :** 10
- **FLIP :** 1
- **Nombre chances :** 3

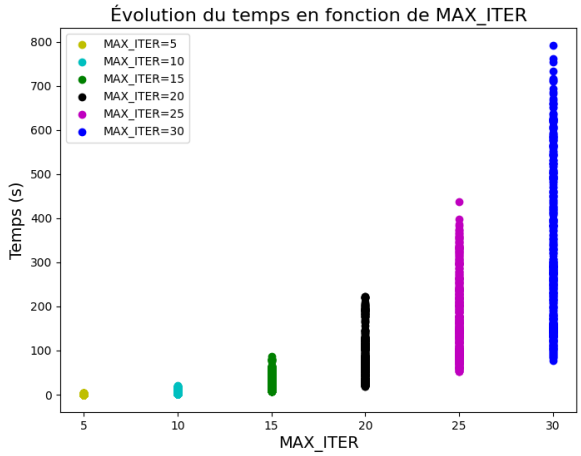


Figure 15: Comparaison des temps d'exécution en fonction du nombre d'itterations

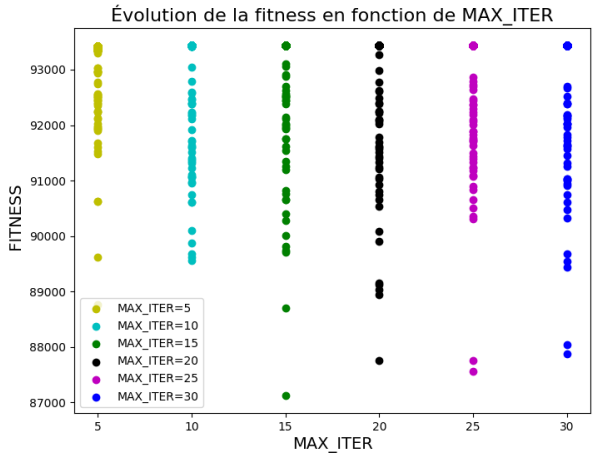


Figure 16: Comparaison des fitness en fonction du nombre d'itterations

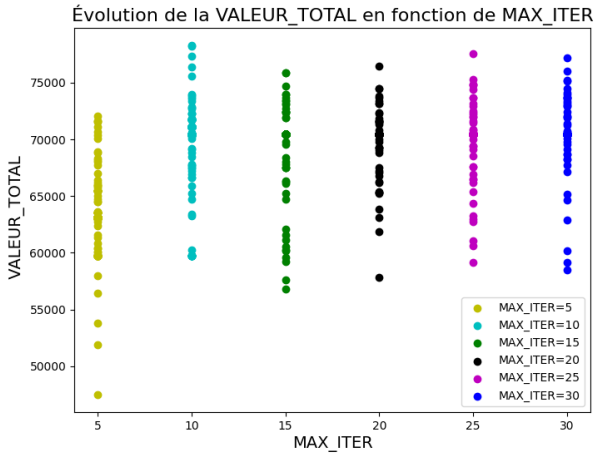


Figure 17: Comparaison des valeurs totales en fonction du nombre d'itterations

En analysant les différents graphiques, il est observé qu'un nombre d'itérations de dix offre de meilleurs résultats en termes de temps d'exécution, de fitness et de valeurs totales.

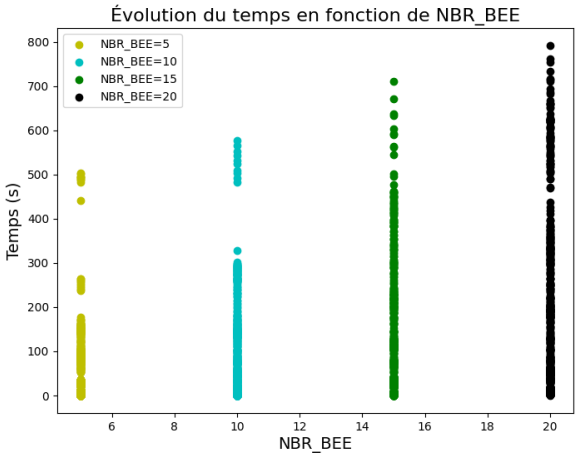


Figure 18: Comparaison des temps d'exécution en fonction du nombre d'abeilles

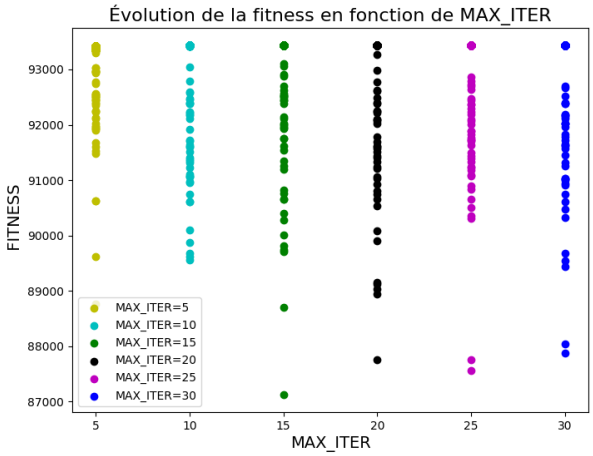


Figure 19: Comparaison des fitness en fonction du nombre d'abeilles

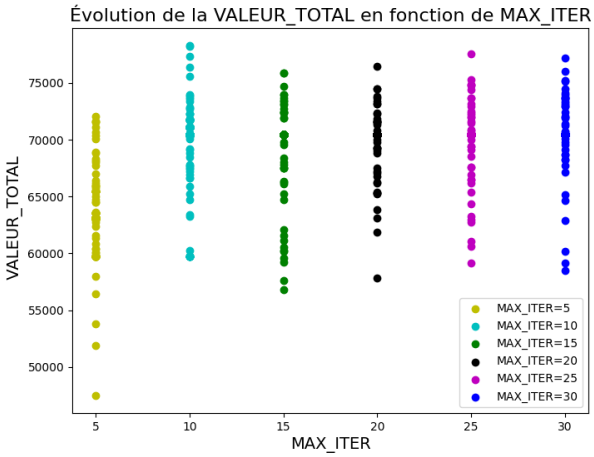


Figure 20: Comparaison des valeurs totales en fonction du nombre d'abeilles

À la suite de l'analyse des différents graphiques, il est constaté qu'un nombre de dix abeilles produit des résultats optimaux en termes de qualité et de temps d'exécution.

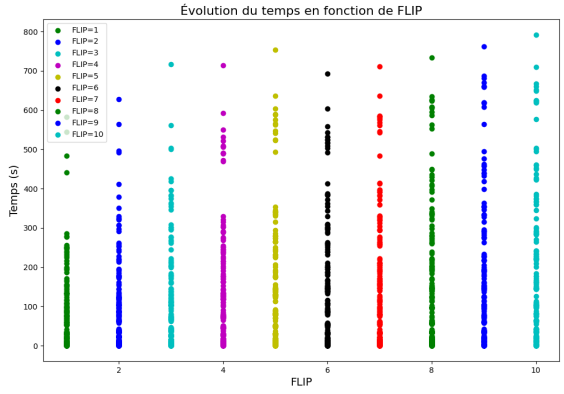


Figure 21: Comparaison des temps d'exécution en fonction du nombre de flip

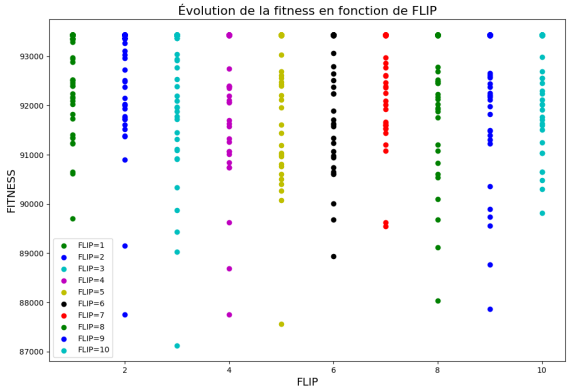


Figure 22: Comparaison des fitness en fonction du nombre de flip

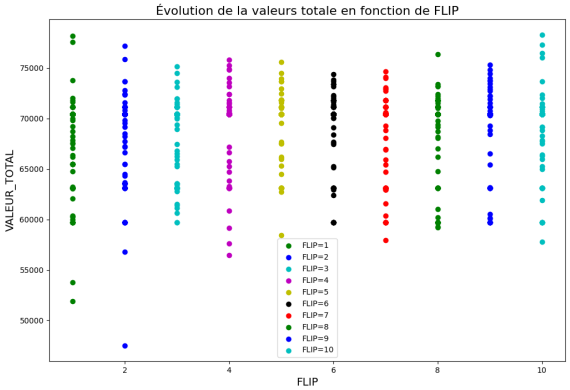


Figure 23: Comparaison des valeurs totales en fonction du nombre de flip

Suite à l'analyse des différents graphiques, il est constaté qu'un nombre de FLIP égale à 1 produit des résultats optimaux en termes de qualité et de temps d'exécution.

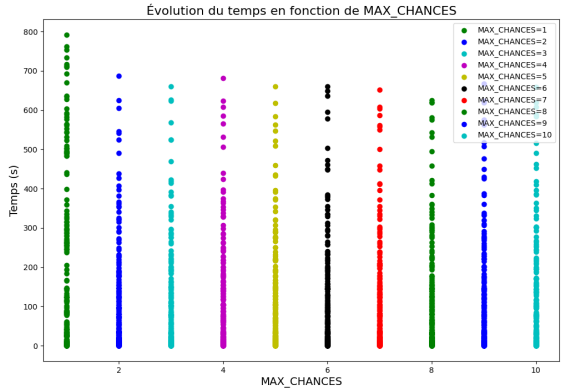


Figure 24: Comparaison des temps d'exécution en fonction du nombre de chances

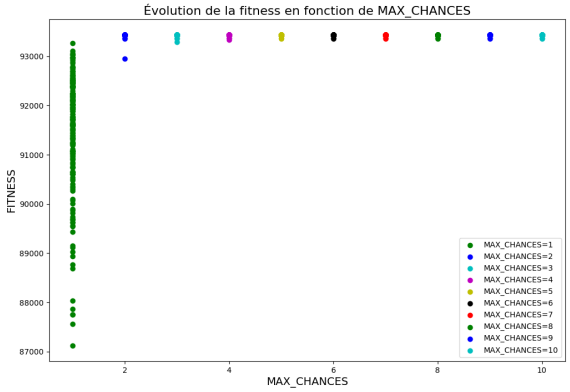


Figure 25: Comparaison des fitness en fonction du nombre de chances

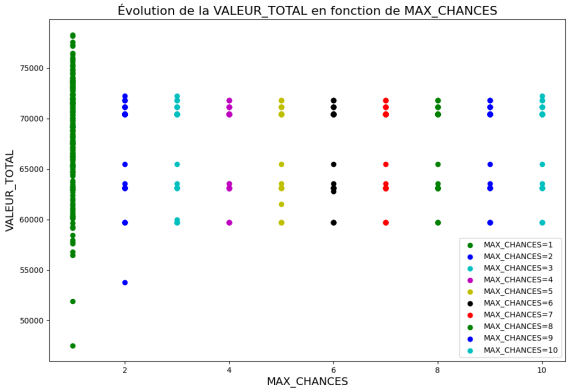


Figure 26: Comparaison des valeurs totales en fonction du nombre de chances

D'après les graphiques, on pourrait conclure qu'un nombre de tentatives de un était satisfaisant. Cependant, l'analyse des données CSV révèle le contraire, car cette configuration n'était pas stable. En revanche, un nombre de trois tentatives produit de meilleurs résultats.

2.3 BSO VS GA

Nous avons initié une série de tests en variant différentes instances du problème afin d'évaluer les performances des deux algorithmes.

Materiels

	PC
Processeur	9th Gen Intel i3-9100f @ 3.60GHz
RAM	8GB 2600MHz
SE	Windows 10 Pro

Table 3: Materiels

Vous trouverez les resultats en format csv sur ce lien:ici

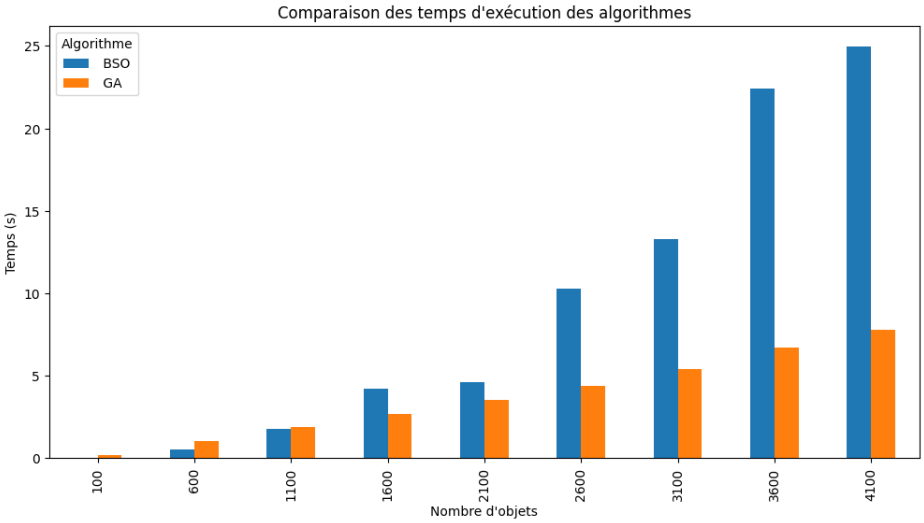


Figure 27: Comparaison des temps d'exécution

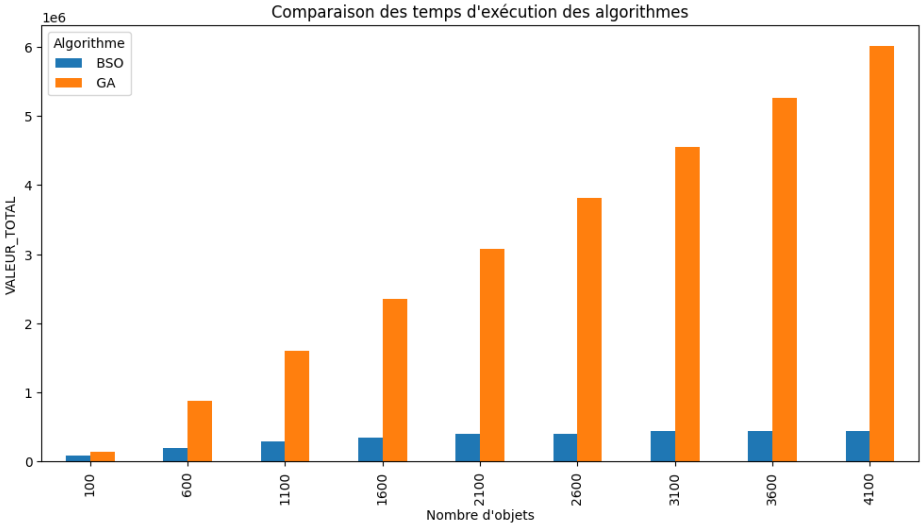


Figure 28: Comparaison des valeurs totales

Après une analyse rigoureuse comparant les performances des métaheuristiques d'optimisation que sont l'algorithme génétique (GA) et l'optimisation par essaim particulaire (BSO), les résultats empiriques démontrent indubitablement la supériorité du GA. En effet, le GA se révèle plus efficace tant au niveau du temps d'exécution que de la qualité des solutions obtenues, se traduisant par une valeur totale optimisée significativement plus élevée.

D'une part, la convergence du GA vers un optimum global est nettement plus rapide que celle du BSO, résultant en des temps de calcul considérablement réduits. D'autre part, les solutions générées par le GA atteignent systématiquement des valeurs totales supérieures à celles obtenues par le BSO, témoignant ainsi de la capacité du GA à explorer plus efficacement l'espace de recherche.

2.4 Interface graphique

Nous avons utilisé la même interface graphique en y ajoutant seulement nos deux métaheuristiques.

3 Conclusion

Dans cette deuxième partie du projet, nous avons abordé la résolution du problème du sac à dos multiple à l'aide de deux métaheuristiques distinctes : les algorithmes génétiques (GA) et l'optimisation par essaim d'abeilles (BSO). Ces approches s'inspirent respectivement des principes de l'évolution naturelle et du comportement des abeilles à la recherche de nourriture.

Après avoir décrit en détail les concepts théoriques sous-jacents à chacune de ces métaheuristiques, nous avons procédé à une série d'expérimentations approfondies visant à déterminer les réglages optimaux de leurs paramètres respectifs. Cette phase d'ajustement empirique s'est avérée cruciale pour garantir l'efficacité de ces algorithmes dans la résolution du problème du sac à dos multiple.

Les résultats obtenus ont démontré que les deux métaheuristiques, GA et BSO, sont capables de fournir des solutions de bonne qualité pour des instances de grande taille du problème, dans un temps raisonnable. Cependant, leur efficacité relative varie en fonction de la complexité de l'instance à résoudre.

En comparant leurs performances, il est raisonnable de conclure que l'algorithme génétique constitue l'approche la plus performante et la plus à même de résoudre de manière optimale les problèmes d'optimisation complexes, comparativement à l'optimisation par essaim d'abeilles.

Cette étude nous a permis de découvrir les métaheuristiques, une classe d'algorithmes d'optimisation puissants et flexibles, capables de traiter des problèmes complexes de manière efficace. Ces approches se sont révélées nettement plus performantes que les méthodes aveugles et heuristiques explorées dans la première partie du projet, ouvrant ainsi de nouvelles perspectives pour la résolution de problèmes d'optimisation combinatoire de grande taille.