


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'Electronique et d'informatique
Département d'informatique

Projet Partie 2

Meta heuristique

Recherche basée espace des solutions
Le problème des Sacs à Dos Multiples (Multiple knapsack problem)

Réalisé par :
OUCHAOU Chafaa
TAOUCI Kenza

Systèmes Informatiques Intelligents

Année scolaire : 2023/2024

Table des matières

CHAPITRE 1

Introduction

Les métaheuristiques sont des techniques de résolution de problèmes d'optimisation qui offrent des solutions approximatives dans un délai raisonnable. Elles sont devenues essentielles dans divers domaines, notamment en informatique, en ingénierie, en logistique et en sciences économiques pour résoudre des problèmes complexes où les méthodes traditionnelles atteignent leurs limites en raison de la taille ou de la complexité du problème.

Parmi les problèmes d'optimisation les plus étudiés se trouve le problème des sacs à dos multiples (Multiple Knapsack Problem, MKP). Ce problème présente un défi considérable en raison de sa complexité combinatoire, ce qui en fait un terrain propice pour l'application des métaheuristiques.

Avant de plonger dans l'étude des métaheuristiques pour résoudre le MKP, il est essentiel de comprendre la distinction entre les heuristiques et les métaheuristiques. Les heuristiques sont des techniques de recherche de solutions qui fournissent rapidement des résultats satisfaisants, mais sans garantie d'optimalité. Elles reposent souvent sur des règles simples et des connaissances spécifiques au problème. En revanche, les métaheuristiques sont des approches plus générales et flexibles qui guident la recherche d'une solution à travers un espace de recherche complexe en utilisant des stratégies d'exploration et d'exploitation. Contrairement aux heuristiques, les métaheuristiques ne sont pas spécifiquement conçues pour un problème donné, mais peuvent être adaptées à une large gamme de problèmes d'optimisation.

Dans ce rapport, nous explorerons différentes métaheuristiques appliquées au problème des sacs à dos multiples, telles que l'algorithmes génétiques, l'algorithmes d'intelligence en essaim (Bee Swarm Optimization). Nous expérimenterons par rapport au paramètre de chaque métaheuristique et avec différentes tailles du

problème, et nous évaluerons leur efficacité et leur performance par rapport aux méthodes heuristiques déjà vues dans la première partie une de ce projet.

CHAPITRE 2

Résolution du problème des Sacs à Dos Multiples

2.1 Algorithme génétique (AG)

L'algorithme génétique (AG) [2] est une métaheuristique inspirée du processus biologique de l'évolution naturelle. Cette méthode s'inspire de la théorie de l'évolution de Darwin, où les individus les mieux adaptés à leur environnement ont plus de chances de survivre et de transmettre leurs caractéristiques à la génération suivante.

L'inspiration principale derrière les algorithmes génétiques réside dans la façon dont les populations d'individus évoluent et s'adaptent au fil du temps pour résoudre des problèmes complexes d'optimisation. L'idée fondamentale est de représenter les solutions potentielles sous forme de "chromosomes" où chaque chromosome représente une solution possible et chaque gène représente une caractéristique ou une composante de la solution.

2.1.1 Étapes de déroulement de l'algorithme génétique

1. **Initialisation de la population :** Une population initiale d'individus est générée de manière aléatoire ou semi-aléatoire.
2. **Évaluation de la population :** Chaque individu de la population est évalué en fonction de sa qualité par rapport à la fonction objectif du problème d'optimisation.
3. **Sélection :** Les individus les plus performants, ou les plus "adaptés", sont sélectionnés pour survivre et reproduire, tandis que les individus moins performants ont moins de chance d'être sélectionnés.
4. **Reproduction :** Les individus sélectionnés se reproduisent pour créer une

nouvelle génération d'individus. Cela peut se faire par des opérateurs génétiques tels que la recombinaison (croisement) et la mutation, qui simulent les mécanismes de reproduction biologique.

5. **Remplacement :** La nouvelle génération remplace l'ancienne, et le processus se répète jusqu'à ce qu'une condition d'arrêt prédéfinie soit atteinte, telle qu'un nombre maximum d'itérations ou l'obtention d'une solution suffisamment proche de l'optimalité.

2.1.2 Algorithme

Algorithme 1 : Algorithme Génétique

Input : Taille_population : nombre de solutions dans chaque génération;
Taux_mutation : probabilité de mutation pour chaque gène;
Nombre_générations : nombre maximum d'itération à exécuter;
Fonction_Fitness : fonction qui évalue la qualité d'une solution;
Autres paramètres spécifiques au problème
Output : Meilleure_solution_trouvée

```
Initialiser une population aléatoire de solutions (population initiale);
foreach solution dans la population initiale do
    Calculer sa valeur de fitness en utilisant la fonction fitness;
while nombre maximum de générations n'est pas atteint do
    Sélectionner des solutions parentales pour la reproduction basée sur
     leur fitness;
    Créer de nouvelles solutions en combinant les solutions parentales par
     crossover et en appliquant des mutations;
    foreach nouvelle solution do
        Calculer sa valeur de fitness en utilisant la fonction fitness;
    Sélectionner les meilleures solutions pour former la prochaine
     génération;
return Meilleure solution trouvée dans la population finale;
```

2.2 Bee Swarm Optimization (BSO)

L'algorithme d'intelligence en essaim [1], également connu sous le nom de Bee Swarm Optimization (BSO), est une métaheuristique inspirée du comportement social des colonies d'abeilles lors de la recherche de sources de nourriture. Cette approche s'inspire de l'organisation complexe et de la coopération observées au sein des essaims d'abeilles pour développer des techniques d'optimisation.

L'inspiration principale derrière l'algorithme BSO réside dans la façon dont les abeilles communiquent et coordonnent leurs efforts pour trouver les meilleures sources de nourriture dans leur environnement. Les abeilles effectuent des danses et communiquent des informations sur la direction et la qualité des sources de nourriture, ce qui leur permet de collecter efficacement la nourriture tout en optimisant leurs trajets.

L'algorithme maintient une population de solutions candidates, représentées par des "abeilles". Les abeilles explorent l'espace de recherche en exploitant les solutions connues (exploitation) et en explorant de nouvelles régions (exploration) grâce à des stratégies de recherche locale et globale.

2.2.1 Étapes de déroulement de l'algorithme Bee Swarm Optimization

1. **Initialisation :** Initialisez une population d'abeilles, généralement réparties de manière aléatoire dans l'espace de recherche.
2. **Phase des abeilles employées :** Chaque abeille employée explore une solution dans son voisinage local et évalue sa qualité (fitness).
3. **Phase des abeilles observatrices :** Les abeilles communiquent des informations sur les bonnes solutions à leurs voisines. Les abeilles observatrices sélectionnent probabilistiquement des solutions en fonction des informations reçues et les évaluent.
4. **Phase des abeilles éclaireuses :** Les abeilles éclaireuses recherchent de nouvelles solutions en explorant les régions non visitées de l'espace de re-

cherche ou en remplaçant les solutions qui n'ont pas été améliorées pendant un certain nombre d'itérations.

5. **Phase de mise à jour :** Mettez à jour la population d'abeilles en fonction de la qualité des solutions trouvées. Les abeilles employées partagent des informations avec les abeilles observatrices, et les abeilles éclaireuses introduisent de la diversité dans la population.
6. **Terminaison :** Répétez le processus pour un nombre prédéfini d'itérations ou jusqu'à ce qu'un critère de terminaison soit atteint (par exemple, la convergence).

2.2.2 Algorithme

Algorithme 2 : Bee Swarm Optimization

Input : Nombre_sites_source : nombre de sites pour la récolte de nourriture;
Nombre_iterations : nombre maximum d'itérations à exécuter;
RHO : probabilité de devenir une abeille éclaireuse;
Fonction_Fitness : fonction qui évalue la qualité de chaque solution;
Autres paramètres spécifiques au problème
Output : Meilleure_solution_trouvée

```
Initialiser une population initiale de sites source générés aléatoirement;
foreach site dans la population initiale do
    Évaluer sa qualité en utilisant la fonction fitness;
while nombre maximum d'itérations n'est pas atteint do
    Les abeilles explorent les sites voisins et évaluent leur qualité;
    Les abeilles partagent les informations sur les sites voisins;
    foreach site do
        Avec une probabilité RHO et selon des condition d'inactivité le site
         sera changer en un autre site aléatoire;
        Sinon, les abeilles continuront à butiner ce sites et l'améliorer;
    Mettre à jour les positions des abeilles en fonction des informations
     partagées;
return Meilleure solution trouvée;
```

2.3 Expérimentation selon les paramètres des méta heuristique

Dans cette section, nous allons nous concentrer sur la manière fixer les paramètres des métaheuristiques de la meilleure manière possible, de manière à optimiser la fitness de la solution et à réduire le temps d'exécution, et cela pour les deux algorithmes, l'algorithme génétiques et le BSO.

2.3.1 Taille de la population (Population Size)

Pour effectuer ce test, on fixera le taux de croisement à 0.5, le taux de mutation à 0.1, le nombre de sacs à 70, le nombre d'items à 200 et le nombre de générations à 10.

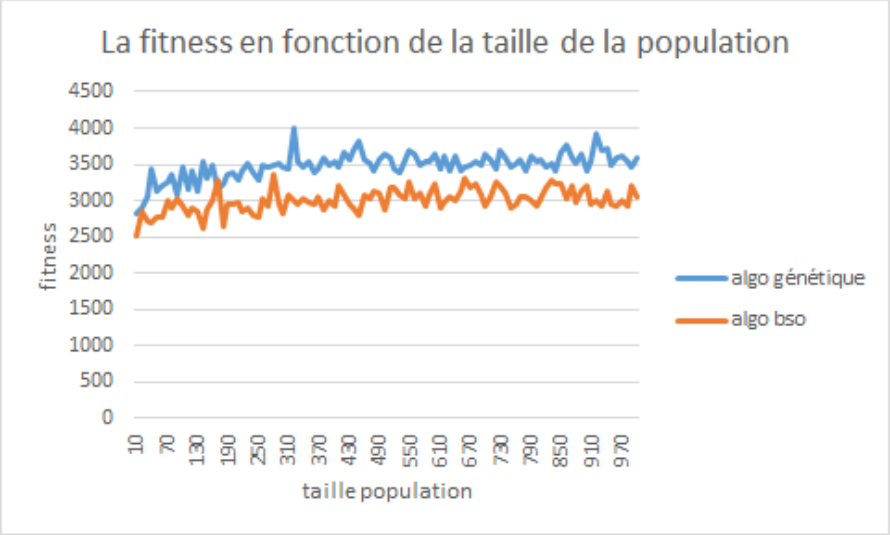

FIGURE 2.1 – la fitness en fonction de la taille de la population

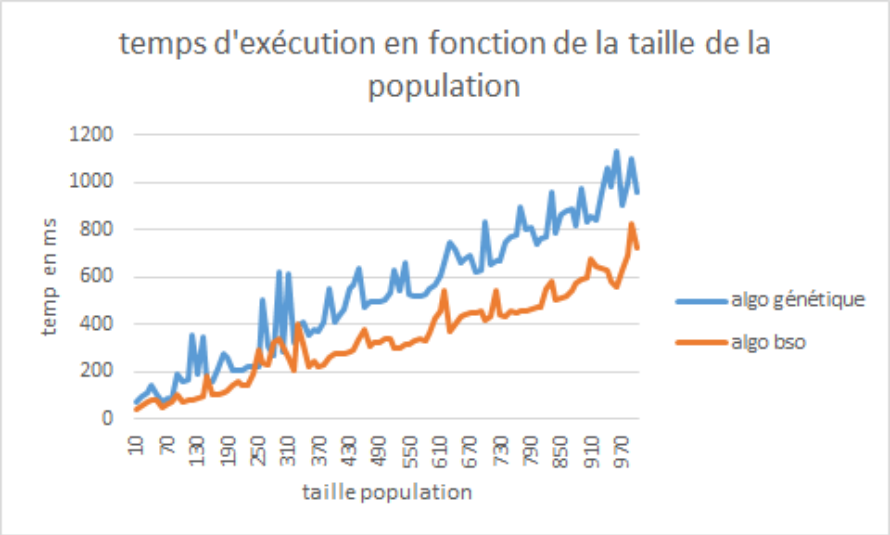

FIGURE 2.2 – temps d'exécution en fonction de la taille de la population

Après avoir réalisé les tests sur la taille de la population, nous avons pu dessiner les deux graphiques 2.1 et 2.2.

Après analyse des deux graphiques, nous avons constaté que le temps d'exécution croît de manière linéaire à chaque fois que l'on augmente la population, mais la fitness stagne après une certaine taille de population qui est environ de **170-250 individus**, et cela pour les deux algorithmes.

2.3.2 Nombre de générations (Number of Generations)

Pour effectuer ce test, on fixera le taux de croisement à 0.5, le taux de mutation à 0.1, le nombre de sacs à 70, le nombre d'items à 200 et la taille de la population à 180.

Après avoir réalisé les tests sur le nombre de générations, nous avons pu dessiner les deux graphiques 2.3 et 2.4.

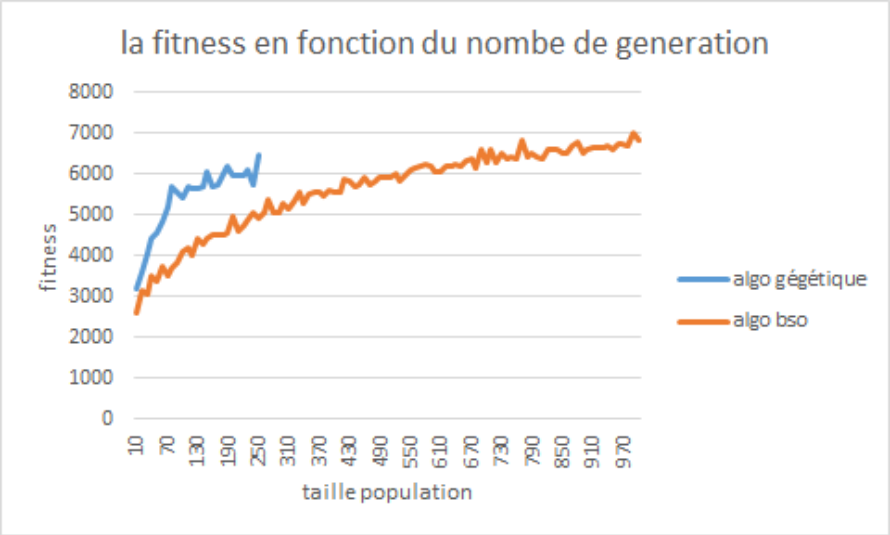

FIGURE 2.3 – la fitness en fonction du nombre de génération

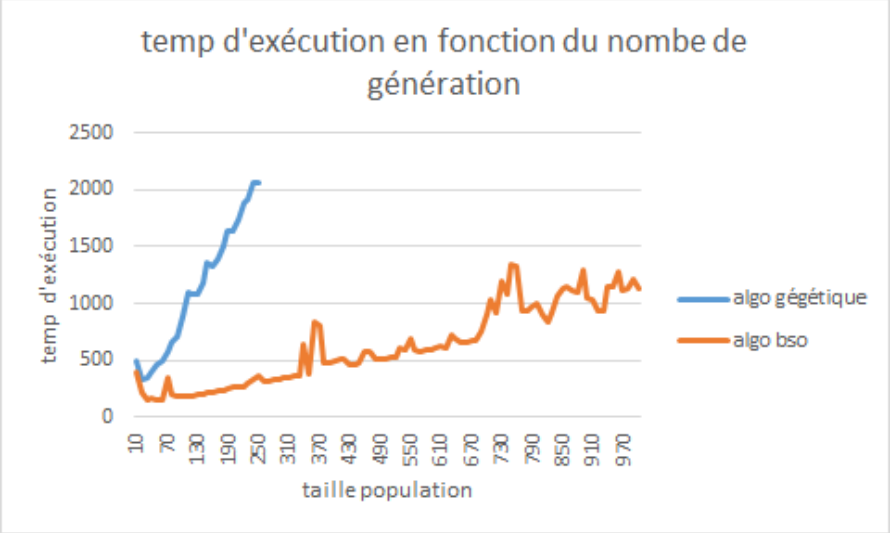

FIGURE 2.4 – temps d'exécution en fonction du nombre de génération

Après analyse des deux graphiques, nous avons constaté que contrairement au premier test, il y a une différence de comportement entre les algorithmes :

1. On constate que l'algorithme génétique commence à prendre plus de temps d'exécution rapidement avec l'augmentation du nombre de générations. Après avoir atteint 250 générations, le temps de calcul était de 6 secondes, ce qui nous a fait arrêter les tests. Mais en ce qui concerne la fitness, elle s'améliore avec chaque augmentation du nombre de générations.
2. En ce qui concerne l'algorithme BSO, on constate que la fitness s'améliore continuellement avec l'augmentation du nombre de générations sans pour autant consommer trop de temps.

En conclusion, le nombre de générations adéquat pour l'algorithme génétique serait d'environ **100 à 200 générations**,(Choisir une valeur de plus de 200 générations peut être considéré s'il n'y a pas trop de consommation de temps à cause des autres paramètres comme la taille du problème, par exemple.) tandis que pour l'algorithme BSO, on choisit un nombre de générations de **600 à 800 générations**.

2.3.3 Taux de mutation (Mutation Rate)

Pour effectuer ce test sur l'algorithme génétique, on fixera le taux de croisement à 0.5, le nombre de génération a 200 , le nombre de sacs à 70, le nombre d'items à 200 et la taille de la population à 180.

Après avoir réalisé les tests sur le taux de mutation , nous avons pu dessiner les deux graphiques 2.5 et 2.6.

Après analyse des deux graphiques, nous avons constaté que plus le taux de mutation est grand, moins la qualité de la solution est bonne, et cela entraîne aussi une augmentation du temps d'exécution. Pour cela, nous avons décidé de prendre **0.1** comme probabilité de mutation.

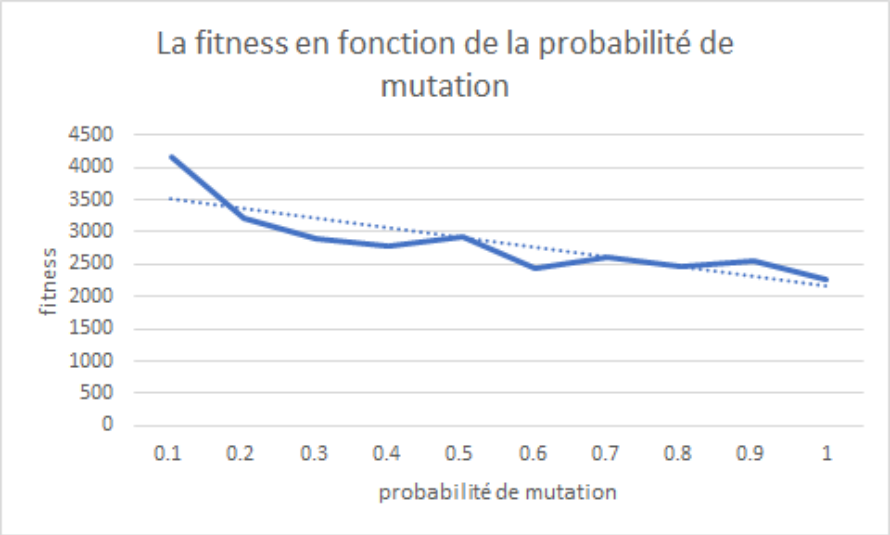

FIGURE 2.5 – La fitness en fonction de la probabilité de mutation

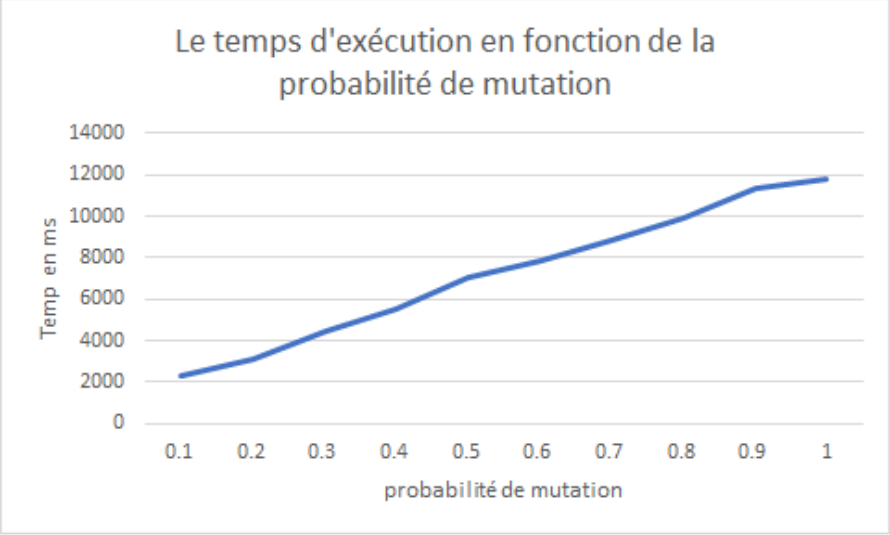

FIGURE 2.6 – Le temps d'exécution en fonction de la probabilité de mutation

2.3.4 Taux de croisement (crossover Rate)

Pour effectuer ce test sur l'algorithme génétique, on fixera le taux de mutation à 0.1, le nombre de génération a 200 , le nombre de sacs à 70, le nombre d'items à 200 et la taille de la population à 180.

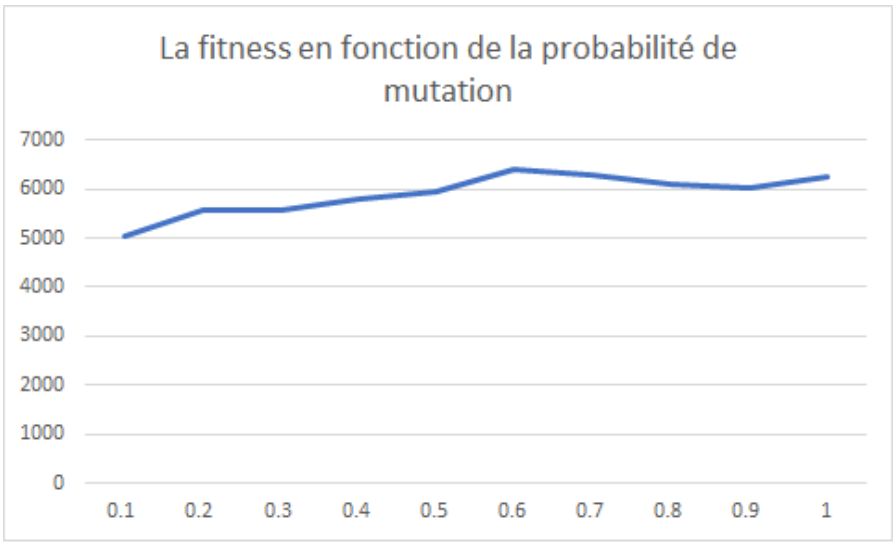

FIGURE 2.7 – La fitness en fonction de la probabilité de croisement

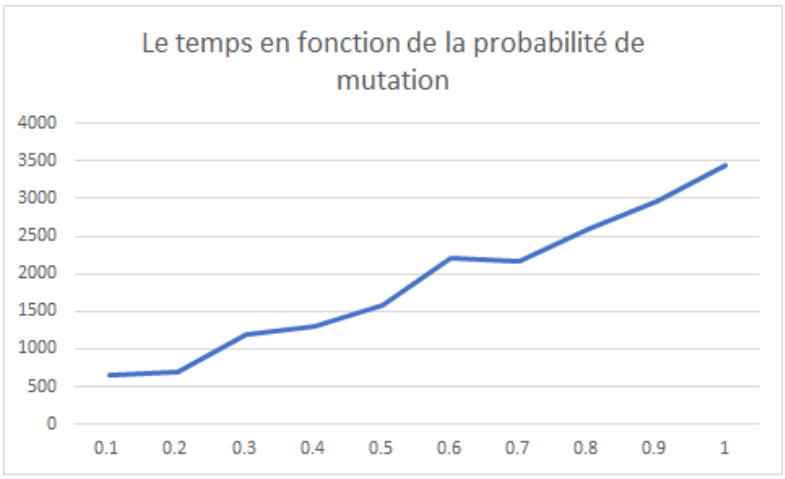

FIGURE 2.8 – Le temps en fonction de la probabilité de croisement

Après avoir réalisé les tests sur le taux de croisement , nous avons pu dessiner les deux graphiques 2.7 et 2.8.

Après analyse des deux graphiques, nous avons constaté que le graphique de la fitness atteint un pic avec une probabilité de croisement égale à 0.6 sans trop affecter le temps d'exécution. Pour cela, nous avons décidé de choisir **0.6** comme probabilité de croisement.

2.3.5 Probabilité de scout (Scout Rate) algoritheme BSO

Pour effectuer ce test sur l'algorithme BSO, on fixera le nombre de génération a 600 , le nombre de sacs à 70, le nombre d'items à 200 et la taille de la population à 180.

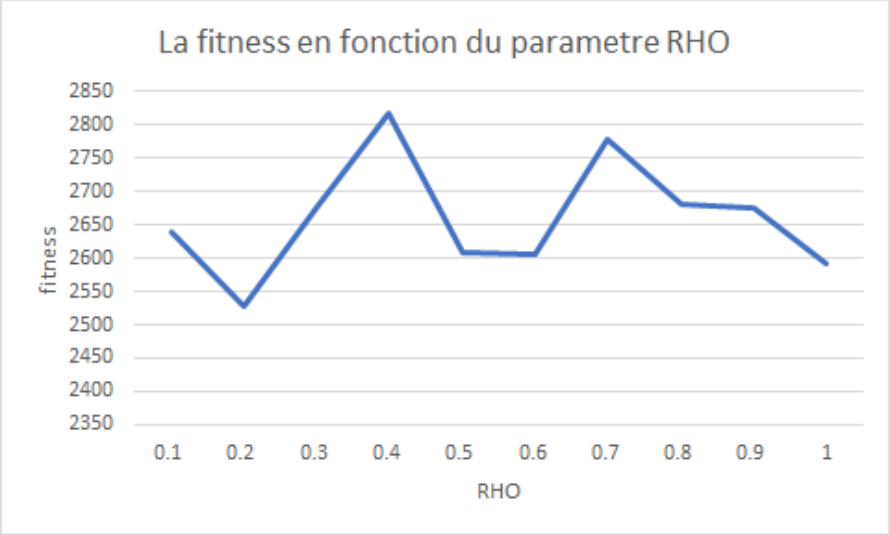

FIGURE 2.9 – La fitness en fonction de la Probabilité de scout

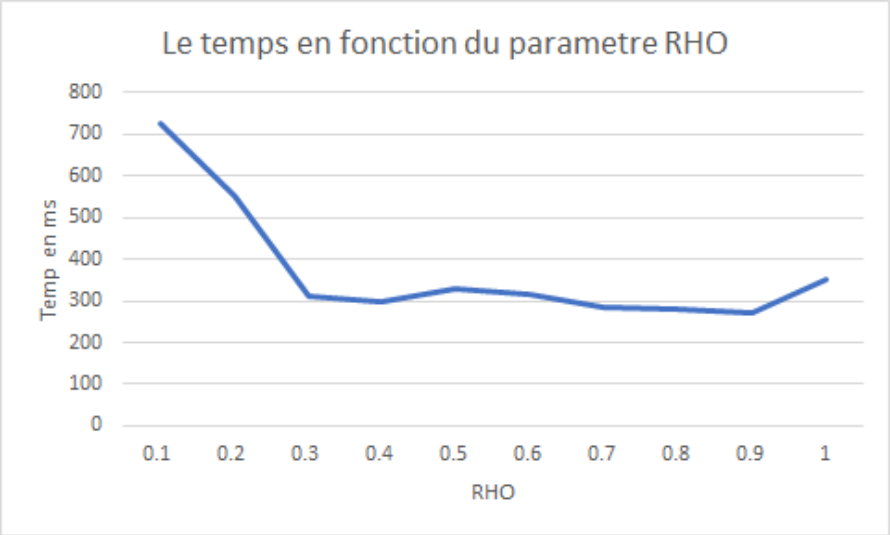

FIGURE 2.10 – Le temps en fonction de la Probabilité de scout

Après avoir réalisé plusieurs tests sur le taux de croisement , nous avons pu dessiner les deux graphiques 2.9 et 2.10.

Après analyse des deux graphiques, nous avons constaté que la probabilité de scout qui maximise la fitness de la solution est soit **0.4**, soit **0.75**.

2.4 Expérimentation sur la taille du problème pour chaque méta heuristique

Maintenant que nous avons fixé tous les paramètres des méta heuristiques de manière à optimiser la fitness de la solution, nous allons procéder aux tests en variant la taille du problème (le nombre de sacs et d'items).

Pour les tests, nous allons mettre en œuvre trois stratégies différentes

1. Augmenter à la fois le nombre de sacs et d'articles de manière équivalente.
2. Augmenter le nombre d'articles deux fois plus rapidement que le nombre de sacs.
3. Augmenter le nombre d'articles quatre fois plus rapidement que le nombre de sacs.

2.4.1 expérimentation numéro 1

Dans cette partie du test on fait augmenter à la fois le nombre de sacs et d'articles de manière équivalente.

Après avoir réalisé plusieurs tests sur la taille du problème , nous avons pu dessiner les deux graphiques 2.11 et 2.12.

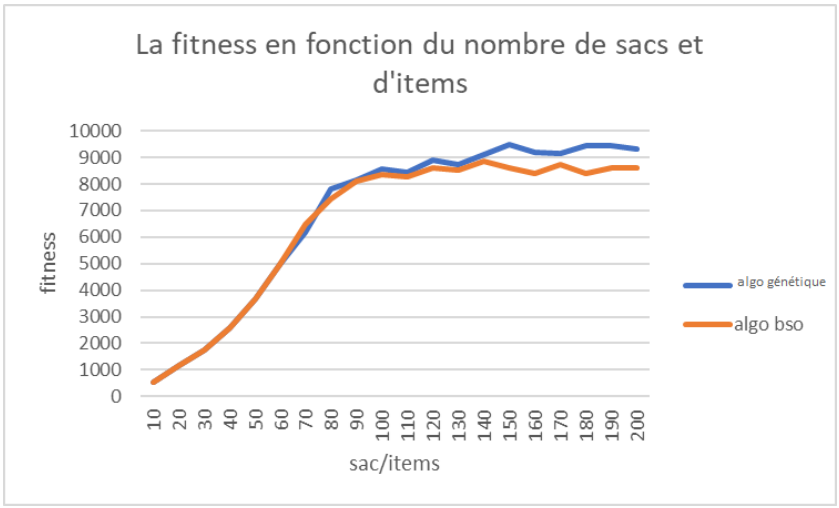

FIGURE 2.11 – La fitness en fonction du nombre de sacs et d'items

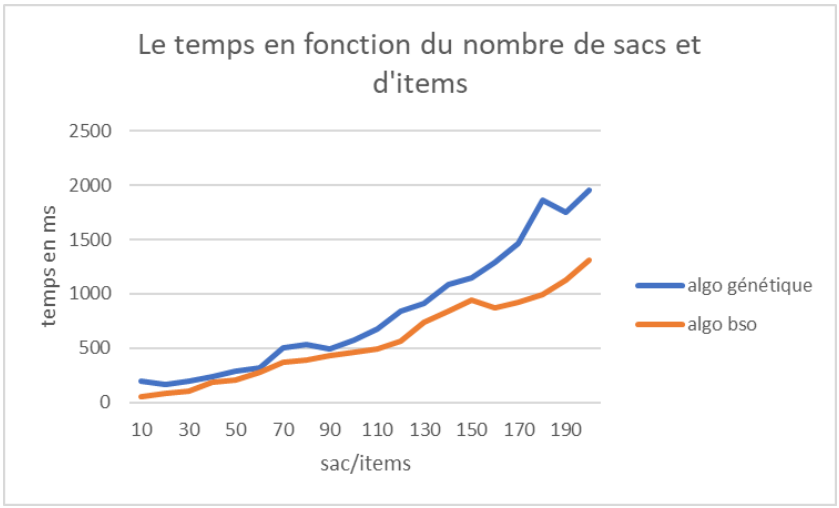

FIGURE 2.12 – Le temps en fonction du nombre de sacs et d'items

Après analyse des deux graphiques, nous avons constaté que les deux algo-

rithmes ont un comportement similaire pour un nombre d'items et de sacs relativement petit. Cependant, en augmentant davantage le nombre d'items et de sacs, on constate que l'algorithme génétique renvoie des solutions de meilleure qualité.

2.4.2 expérimentation numéro 2

dans cette partie du test on fait augmenter le nombre d'articles deux fois plus rapidement que le nombre de sacs.

Après avoir réalisé plusieurs tests sur la taille du problème , nous avons pu dessiner les deux graphiques 2.13 et 2.14.

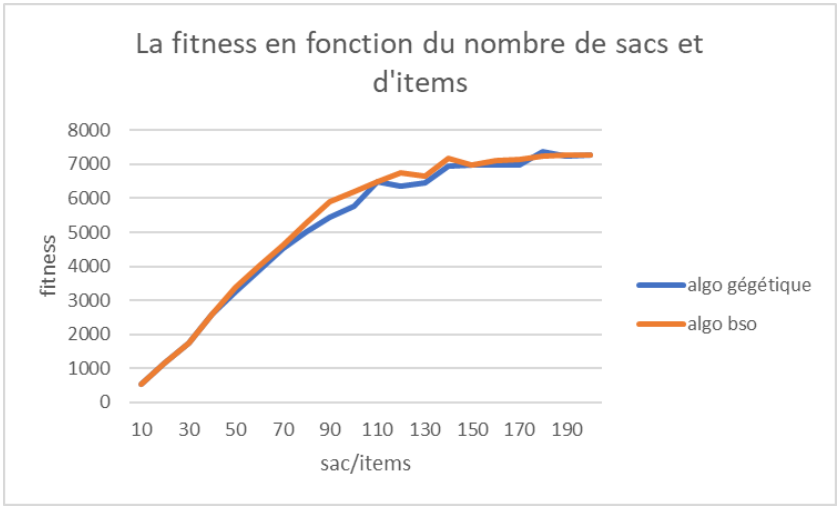

FIGURE 2.13 – La fitness en fonction du nombre de sacs et d'items

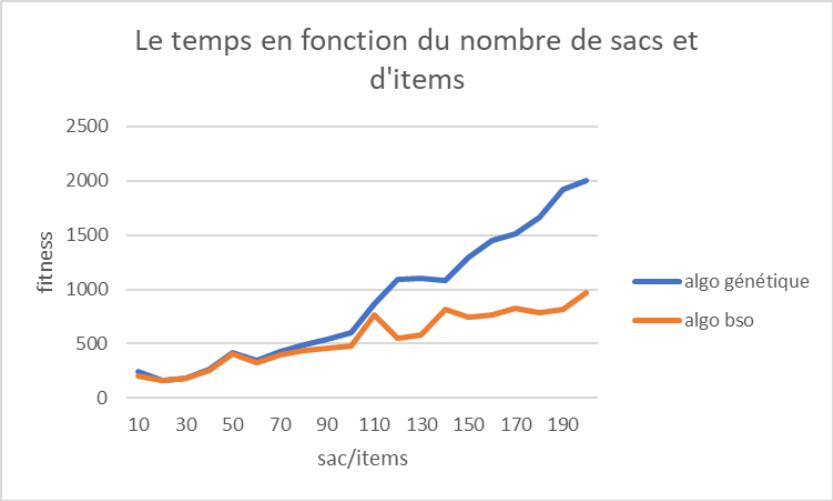

FIGURE 2.14 – Le temps en fonction du nombre de sacs et d'items

Après analyse des deux graphiques, nous avons constaté que les deux courbes de fitness des algorithmes sont presque identiques, ce qui signifie que les deux algorithmes renvoient des solutions de presque même qualité. Cependant, il est important de souligner le fait que l'algorithme **BSO** prend nettement moins de temps pour s'exécuter.

2.4.3 expérimentation numéro 3

dans cette partie du test on fait augmenter le nombre d'articles quatre fois plus rapidement que le nombre de sacs.

Après avoir réalisé plusieurs tests sur la taille du problème , nous avons pu dessiner les deux graphiques 2.15 et 2.16.

Après analyse des deux graphiques, nous avons constaté que les deux algorithmes ont un comportement similaire pour un nombre d'items et de sacs relativement petit. Cependant, en augmentant davantage le nombre d'items et de sacs, on constate que l'algorithme BSO renvoie des solutions de meilleure qualité, ce qui est contraire à l'observation de l'expérimentation 1.

En ce qui concerne le temps d'exécution, l'algorithme BSO consomme nettement moins de temps que l'algorithme génétique.

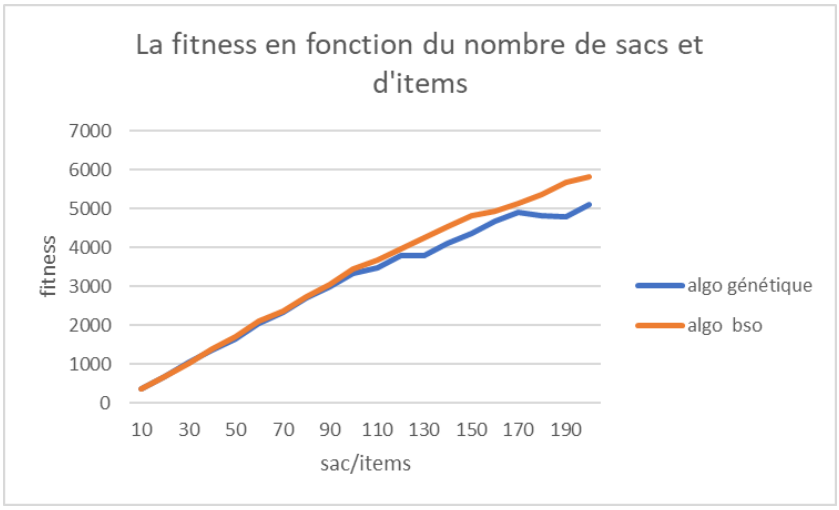

FIGURE 2.15 – La fitness en fonction du nombre de sacs et d'items

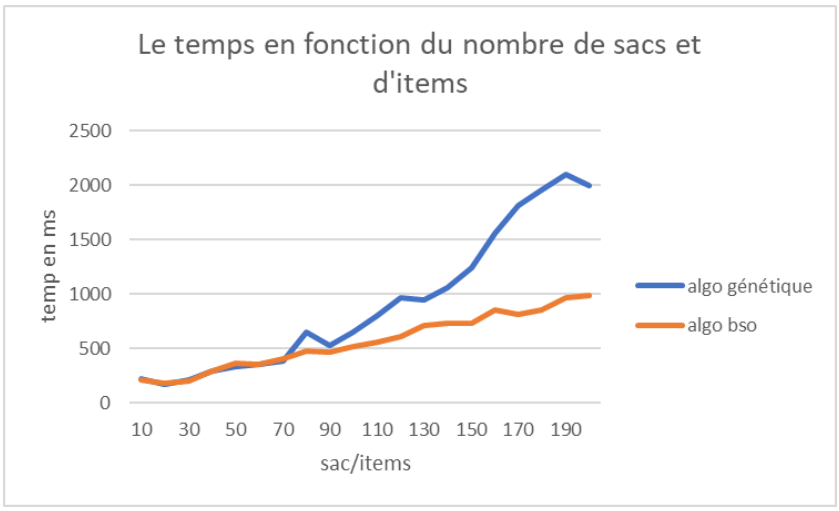

FIGURE 2.16 – Le temps en fonction du nombre de sacs et d'items

2.4.4 Conclusion de l'expérimentation

En conclusion, les expérimentations ont démontré que l'algorithme génétique est efficace pour des ensembles de données avec un nombre plus en moins égal de sacs et d'items, soulignant ainsi sa force dans l'exploration des minima locaux. Cependant, lorsque le nombre d'items augmente significativement par rapport aux sacs, l'algorithme BSO se révèle être plus performant, mettant en évidence sa force dans l'exploration de l'espace de recherche. Cette observation suggère la possibilité de combiner les deux approches, exploitant ainsi les avantages de chacune pour obtenir des solutions optimales dans des scénarios variés.

2.5 Comparaison Partie 1 avec Partie 2

Dans le cadre du problème du sac à dos multiple, la comparaison entre les méta-heuristiques telles que l'algorithme génétique et le BSO avec les méthodes exactes et heuristiques révèle des différences significatives en termes de performance et d'efficacité. Les méthodes exactes offrent une garantie de trouver la solution optimale, mais leur utilisation devient rapidement impraticable lorsque le nombre de sacs et d'items dépasse une certaine taille, généralement au-delà de 10 sacs et 10 items. En revanche, les méta-heuristiques présentent une grande flexibilité et une capacité à explorer efficacement de vastes espaces de recherche. Par exemple, les tests ont montré que des méta-heuristiques peuvent résoudre des instances avec 200 sacs et 200 items sans difficulté notable. En somme, bien que les méthodes exactes puissent garantir une solution optimale pour de petites instances, les méta-heuristiques se révèlent être une solution plus efficace et adaptable pour les problèmes de grande taille, offrant des performances impressionnantes même face à des instances massives.

2.6 Conclusion

La réalisation de ce projet a débuté par une étude approfondie et une mise en œuvre des deux algorithmes, à savoir l'algorithme génétique et le BSO dans le contexte du problème du sac à dos multiple. À travers cette phase initiale, nous avons pu comprendre les principes fondamentaux de ces métaheuristiques et les

adapter à notre problème spécifique. Ensuite, nous avons entrepris une série de tests exhaustifs, générant des données significatives à partir desquelles nous avons élaboré des graphiques pour déterminer les valeurs optimales des paramètres des métaheuristiques. Ce processus de réglage des paramètres a été essentiel pour garantir des performances optimales des algorithmes dans la résolution du problème.

Par la suite, nous avons mené des expérimentations testant nos algorithmes sur des ensembles de données de différentes tailles, en variant le nombre de sacs et d'items. Cette approche nous a permis d'explorer divers scénarios et de comprendre comment les métaheuristiques se comportent face à des instances de plus en plus complexes du problème du sac à dos multiple. Nous avons observé avec fascination comment les algorithmes génétique et BSO ont démontré leur robustesse et leur adaptabilité, fournissant des solutions de qualité même dans des conditions de grande complexité.

Enfin, nous avons comparé les performances des métaheuristiques avec celles des approches exact et heuristiques, mettant en lumière les avantages des méta heuristiques en termes de rapidité, de capacité à trouver des solutions de qualité et d'adaptabilité à des problèmes de grande taille. Cette comparaison a souligné l'importance et la pertinence des métaheuristiques dans la résolution de problèmes complexes tels que le sac à dos multiple. En résumé, ce projet a été une expérience enrichissante qui a permis d'explorer en profondeur le potentiel des métaheuristiques dans la résolution de problèmes réels, ouvrant de nouvelles perspectives pour l'optimisation et l'efficacité des algorithmes dans divers domaines d'application.

Bibliographie

[1] A genetic algorithm for the multiple knapsack problem in dynamic environment. https://www.iaeng.org/publication/WCECS2013/WCECS2013_pp1162-1167.pdf. Accessed : 2024-04-2.

[2] Martín Carpio Marco Sotelo-Figueroa, Rosario Baltazar. Application of the bee swarm optimization bso to the knapsack problem. 2010.