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

Département d'Intelligence Artificielle et Sciences des Données

Partie 2

Les métaheuristiques pour la résolution du problème MKP

MODULE : Métaheuristique

Réalisé par :

TAOURIRT Hamza

SAADA Samir

Année Universitaire : 2023/2024

Table de matières

1. Introduction

Dans cette deuxième partie de notre projet, nous nous concentrons sur l'application de métaheuristiques pour résoudre le Problème des Sacs à Dos Multiples (Multiple Knapsack Problem, MKP). Les métaheuristiques sont des techniques de résolution de problèmes d'optimisation qui offrent des solutions approchées efficaces, souvent nécessaires lorsque les méthodes traditionnelles atteignent leurs limites en raison de la taille ou de la complexité du problème.

Après avoir exploré les approches classiques telles que la recherche exhaustive dans la première partie, nous nous tournons maintenant vers deux métaheuristiques populaires : l'algorithme génétique (GA) et l'optimisation par essaim d'abeille (BSO).

Dans cette partie de notre travail, nous explorons en détail le fonctionnement de chaque métaheuristique, leur inspiration, leur algorithme sous-jacent et les expérimentations menées pour ajuster leurs paramètres afin d'optimiser leurs performances.

Nous évaluons également les performances de ces métaheuristiques en fonction de différentes tailles de problèmes de MKP, afin de comprendre leur évolutivité et leur robustesse face à des instances de problèmes de différentes complexités.

Enfin, nous comparons les résultats obtenus avec ces métaheuristiques avec ceux obtenus dans la première partie de notre projet, où nous avons utilisé des méthodes de recherche exhaustive, à savoir A*, BFS et DFS, pour résoudre le MKP. Cette comparaison nous permettra de mieux comprendre les forces et les limites de chaque approche dans la résolution du MKP.

Dans ce rapport, nous détaillons notre travail, de la conception à la mise en œuvre, en passant par les résultats expérimentaux et leur analyse, pour offrir une perspective complète sur l'efficacité des métaheuristiques dans la résolution du Problème des Sacs à Dos Multiples.

2. Définition et Importance des Métaheuristiques

Les métaheuristiques sont des techniques de résolution de problèmes d'optimisation qui visent à trouver des solutions de qualité dans un temps raisonnable, souvent dans des situations où les méthodes traditionnelles ne peuvent pas être appliquées efficacement en raison de la complexité ou de la taille du problème. Contrairement aux algorithmes déterministes, les métaheuristiques offrent des approches heuristiques flexibles et adaptatives, capables d'explorer efficacement l'espace de recherche pour trouver des solutions prometteuses, sans garantir la meilleure solution absolue.

Les métaheuristiques jouent un rôle crucial dans la résolution de problèmes complexes dans divers domaines tels que l'ingénierie, la logistique, la finance et la science des données. Leur flexibilité et leur capacité à trouver des solutions de qualité dans des délais raisonnables en font des outils précieux pour aborder des problèmes réels où les contraintes de temps et de ressources sont importantes. De plus, les métaheuristiques permettent souvent d'explorer des espaces de recherche de grande dimension de manière efficace, offrant ainsi des solutions pratiques pour des problèmes à grande échelle. En résumé, les métaheuristiques offrent une approche polyvalente et puissante pour résoudre une large gamme de problèmes d'optimisation réels.

3. Algorithmes Génétiques AG

Dans cette section, nous explorons en détail les Algorithmes Génétiques (AG), une métaheuristique largement utilisée pour résoudre divers problèmes d'optimisation, y compris le Problème des Sacs à Dos Multiples (MKP). Nous commençons par définir les principes fondamentaux des Algorithmes Génétiques et discutons de leur inspiration tirée de l'évolution naturelle.

Ensuite, nous examinons en détail les opérateurs génétiques utilisés dans les AG, à savoir la sélection, le croisement et la mutation, en expliquant comment ces opérateurs sont appliqués pour générer de nouvelles solutions et améliorer la qualité des solutions existantes.

Nous discutons également de l'application spécifique des Algorithmes Génétiques au Problème des Sacs à Dos Multiples (MKP), en mettant en évidence les défis et les opportunités dans l'utilisation de cette approche pour résoudre ce problème complexe.

Enfin, nous abordons le sujet du réglage des paramètres dans les Algorithmes Génétiques et présentons les expérimentations menées pour ajuster ces paramètres afin d'optimiser les performances des AG dans la résolution du MKP.

3.1. Définition et Principes des Algorithmes Génétique

Les algorithmes génétiques (AG) sont des techniques d'optimisation inspirées par le processus évolutif naturel de la sélection naturelle et de la génétique. Basés sur des concepts issus de la théorie de l'évolution, les AG simulent le processus de sélection naturelle pour résoudre des problèmes d'optimisation complexes.

Dans un AG, une population initiale de solutions potentielles est générée aléatoirement. Ces solutions, souvent représentées sous forme de chromosomes ou de génotypes, sont ensuite soumises à des opérateurs génétiques tels que la sélection, le croisement et la mutation.

La sélection favorise les solutions les plus performantes, tandis que le croisement et la mutation génèrent de nouvelles solutions en combinant et en modifiant les caractéristiques des solutions existantes.

Au fil des générations, les solutions les plus adaptées survivent et sont transmises à la génération suivante, tandis que les solutions moins performantes sont éliminées. Ce processus itératif se poursuit jusqu'à ce qu'une solution satisfaisante soit trouvée ou que le critère d'arrêt prédéfini soit atteint.

Les AG sont largement utilisés pour résoudre une variété de problèmes d'optimisation dans divers domaines tels que l'ingénierie, la planification, la logistique et l'intelligence artificielle, en raison de leur capacité à trouver des solutions de haute qualité dans des espaces de recherche complexes et de grande dimension.

3.2. Inspiration de l'Évolution Naturelle

Les AG tirent leur inspiration principale du processus évolutif observé dans la nature. En particulier, l'idée de sélection naturelle, où les individus les mieux adaptés à leur environnement ont plus de chances de survivre et de se reproduire, est le fondement des AG.

De même, les opérateurs génétiques tels que le croisement et la mutation imitent les mécanismes de reproduction et de variation génétique observés dans les populations biologiques. En combinant ces opérateurs avec une sélection basée sur la performance, les AG sont capables de générer de nouvelles solutions et d'améliorer progressivement la qualité des solutions au fil des générations.

Cette analogie avec l'évolution naturelle permet aux AG d'explorer efficacement l'espace de recherche et de trouver des solutions de qualité pour une grande variété de problèmes d'optimisation.

3.3. Explication des Opérateurs Génétiques

Les algorithmes génétiques utilisent plusieurs opérateurs génétiques clés pour explorer l'espace de recherche et générer de nouvelles solutions. Parmi ces opérateurs, nous trouvons la sélection, le croisement et la mutation.

3.3.1. Sélection

La sélection est l'opérateur qui détermine quelles solutions individuelles de la population seront conservées pour la génération suivante. Il existe plusieurs méthodes de sélection, chacune avec ses propres avantages et inconvénients.

Sélection basée sur le rang : Dans cette méthode, les individus sont triés en fonction de leur aptitude (fitness), puis chaque individu reçoit un score en fonction de son rang dans la population. Les individus sont ensuite sélectionnés en fonction de leur score de rang, où les individus les mieux classés ont une plus grande probabilité d'être sélectionnés.

Supposons une population de solutions triées par ordre croissant d'aptitude. Chaque individu se voit attribuer un score en fonction de son rang dans la population. Les individus sont ensuite sélectionnés avec une probabilité proportionnelle à leur score de rang.

Sélection par tournoi : Dans cette méthode, des sous-groupes aléatoires d'individus sont sélectionnés (appelés "tournois"), et le meilleur individu de chaque tournoi est sélectionné pour la reproduction. La taille du tournoi peut varier, et les individus peuvent être sélectionnés avec ou sans remplacement.

Exemple : Supposons une population où des tournois de trois individus sont organisés de manière aléatoire. Chaque tournoi est ensuite remporté par l'individu avec la meilleure aptitude parmi les trois.

Ces méthodes de sélection, telles que la sélection basée sur le rang et la sélection par tournoi, permettent aux algorithmes génétiques de maintenir la diversité génétique de la population tout en favorisant les individus les plus performants pour la reproduction.

3.3.2. Croisement

Le croisement (ou crossover) est l'opérateur qui combine les caractéristiques de deux solutions parentes pour créer de nouvelles solutions descendantes. Il existe plusieurs techniques de croisement couramment utilisées dans les algorithmes génétiques.

Croisement en un point : Dans cette technique, un point de croisement est sélectionné au hasard le long des chromosomes parents, et les parties des solutions parents situées après ce point sont échangées pour créer deux nouvelles solutions descendantes.

Exemple : Supposons deux parents représentés sous forme de tableaux binaires où chaque bit représente la présence ou l'absence d'un article dans un sac à dos. Le point de croisement est choisi au hasard, puis les parties après ce point sont échangées entre les deux parents pour créer deux nouvelles solutions descendantes.

Croisement en deux points : Dans cette technique, deux points de croisement sont sélectionnés au hasard le long des chromosomes parents, et les parties des solutions parents situées entre ces points sont échangées pour créer deux nouvelles solutions descendantes.

Exemple : Avec la même représentation binaire que précédemment, deux points de croisement sont choisis au hasard, puis les parties entre ces points sont échangées entre les deux parents pour créer deux nouvelles solutions descendantes.

Croisement uniforme : Dans cette technique, chaque bit des solutions descendantes est sélectionné de manière aléatoire à partir des bits correspondants des solutions parentes. Cela signifie que chaque bit a une probabilité de 50% d'être hérité de l'un ou l'autre parent.

Exemple : Toujours avec la même représentation binaire, chaque bit des solutions descendantes est choisi de manière aléatoire à partir des bits correspondants des solutions parentes pour former deux nouvelles solutions descendantes.

Ces techniques de croisement permettent de créer de nouvelles solutions en combinant les caractéristiques des solutions parentes, ce qui aide à explorer efficacement l'espace de recherche et à générer des solutions de haute qualité pour le Problème des Sacs à Dos Multiples.

3.3.3. Mutation

La mutation est un opérateur qui introduit une petite perturbation aléatoire dans les solutions pour explorer de nouvelles régions de l'espace de recherche. Il est généralement utilisé pour maintenir la diversité génétique de la population et éviter la convergence prématurée vers un optimum local.

Exemple : Dans une solution représentée sous forme binaire, une mutation pourrait changer aléatoirement un bit de la solution, entraînant ainsi une modification mineure de la solution.

Ces opérateurs génétiques sont essentiels pour la capacité des algorithmes génétiques à explorer efficacement l'espace de recherche et à converger vers des solutions de haute qualité.

3.4. Application des algorithmes génétiques au MKP

Les algorithmes génétiques (AG) sont largement utilisés pour résoudre le Problème des Sacs à Dos Multiples (MKP) en raison de leur capacité à trouver des solutions de qualité dans des espaces de recherche complexes. L'application des AG au MKP comprend plusieurs étapes clés :

Génération de la population initiale : Une population initiale de solutions potentielles est générée de manière aléatoire. Chaque solution représente une répartition des articles dans les sacs à dos, et son adéquation est évaluée en fonction de critères tels que la valeur totale des articles et le respect des contraintes de poids des sacs à dos.

Sélection des parents :} Une sélection basée sur le classement est souvent utilisée pour choisir les parents pour la reproduction. Les solutions de la population sont classées en fonction de leur adéquation, et les solutions les mieux adaptées sont plus susceptibles d'être sélectionnées comme parents pour produire des descendants.

Croisement : Les parents sélectionnés sont croisés pour créer de nouvelles solutions descendantes. Différentes techniques de croisement, telles que le croisement en un point, en deux points ou uniforme, sont utilisées pour combiner les caractéristiques des parents et générer une diversité de solutions dans la population descendante.

Mutation : Les solutions descendantes peuvent subir des mutations aléatoires pour introduire de la diversité dans la population. La mutation peut impliquer des changements mineurs dans les solutions, tels que le déplacement d'un article d'un sac à dos à un autre, afin d'explorer de nouvelles régions de l'espace de recherche.

Création de la nouvelle génération : Les parents et les descendants sont combinés pour former la prochaine génération de la population. Les individus les plus adaptés sont sélectionnés pour survivre et être transmis à la génération suivante, tandis que les individus moins adaptés sont éliminés.

Critère d'arrêt : Le processus itératif de sélection, de croisement, de mutation et de création de nouvelles générations se poursuit jusqu'à ce qu'un critère d'arrêt prédéfini soit atteint. Ce critère peut être un nombre maximum de générations, une convergence des solutions vers un optimum, ou d'autres critères spécifiques au problème.

En appliquant ces étapes itératives, les algorithmes génétiques peuvent être utilisés de manière efficace pour résoudre le Problème des Sacs à Dos Multiples en trouvant des répartitions optimales des articles dans les sacs à dos, maximisant ainsi la valeur totale des articles tout en respectant les contraintes de poids des sacs.

Pseudo-code de l'algorithme génétique

Voici le pseudo-code de l'algorithme génétique pour résoudre le Problème des Sacs à Dos Multiples :

```
function activateGeneticAlgorithm(capacities: array of integer; items: array of Item;
    populationSize, maxGenerations: integer; mutationRate: real; )
var

    population: array of State;
    bestState: State;
begin

    population := generatePopulation([], capacities, items, populationSize);// genere
    population initial aleatoitement
    bestState := NULL

    for i := 1 to maxGenerations  do
    begin

        parents := rankBasedSelection(population);// les meilleillers selon la densite de
    valeur sont selectionne pou etre parent

        offspringPop := performUniformCrossover(parents, capacities, items, populationSize);
        // performUniformCrossover utilise un random pur choisir

        mutatePopulation(offspringPop, mutationRate, capacities);
        // la mutation permet d'introduire des objet abscent de la population;

        population := createNextPopulation(parents, offspringPop, populationSize);
        //Les etats avec la plus haute valeur continue vers le nouvelle generation

        bestState := selectBestState(population);
    end;


    Return(bestState);
end;
```

Figure 1: Pseudo-code de l'algorithme principale du GA

Les méthodes principales utilisé dans cet algorithme :

selectParents: Cette méthode sélectionne les parents pour la prochaine génération en fonction de leur aptitude ou de leur valeur. Elle peut utiliser différentes méthodes de sélection, telles que la sélection par rang ou la sélection par tournoi, pour choisir les parents les plus prometteurs.

uniformCrossover: Cette méthode effectue le croisement uniforme entre les parents sélectionnés. Le croisement uniforme consiste à choisir aléatoirement des gènes de chaque parent pour créer les descendants. Cela permet de conserver une diversité génétique tout en combinant les caractéristiques des parents.

mutate: Cette méthode applique une mutation à une partie de la population, ce qui introduit une petite perturbation aléatoire dans les solutions. La mutation peut être réalisée en modifiant certains gènes des individus de manière aléatoire, ce qui permet d'explorer de nouvelles solutions potentielles.

createNextPopulation: Cette méthode crée la prochaine génération en combinant les parents sélectionnés avec les descendants produits par le croisement et la mutation. Elle peut utiliser différentes stratégies pour choisir les individus à inclure dans la nouvelle population, telles que la sélection des meilleurs individus ou une sélection aléatoire.

bestSolution: Cette méthode sélectionne la meilleure solution parmi une population donnée. Elle compare les solutions en fonction de leur aptitude ou de leur valeur et retourne celle qui présente la meilleure performance.

Réglage des paramètres et expérimentation

L'algorithme génétique nécessite souvent un réglage minutieux des paramètres pour obtenir de bons résultats. Voici quelques-uns des paramètres clés et des considérations importantes lors du réglage de l'algorithme :

Taille de la population : La taille de la population détermine le nombre d'individus dans chaque génération. Une plus grande taille de population peut permettre une exploration plus approfondie de l'espace de recherche, mais elle augmente également le coût de calcul. Le choix de la taille de la population dépend souvent de la complexité du problème et des ressources disponibles.

Nombre de générations : Le nombre de générations spécifie le nombre d'itérations ou de générations que l'algorithme va exécuter. Une plus grande valeur peut permettre une exploration plus approfondie de l'espace de recherche, mais elle augmente également le temps d'exécution global. Il est important de trouver un équilibre entre la qualité de la solution et le temps d'exécution.

Taux de mutation: Le taux de mutation contrôle la probabilité qu'une mutation soit appliquée à un individu donné lors de la reproduction. Une valeur plus élevée de ce paramètre peut favoriser l'exploration de nouveaux domaines de recherche, tandis qu'une valeur plus faible peut favoriser l'exploitation des solutions existantes. Le réglage du taux de mutation dépend souvent de la nature du problème et de la diversité souhaitée dans la population.

Exploration vs Exploitation : L'équilibre entre l'exploration et l'exploitation est crucial dans le réglage des paramètres de l'algorithme génétique. L'exploration consiste à rechercher de nouvelles régions de l'espace de recherche, tandis que l'exploitation consiste à exploiter les régions déjà découvertes pour améliorer les solutions. Le choix du réglage des paramètres doit tenir compte de cet équilibre pour obtenir des solutions de haute qualité.

Il est souvent nécessaire de mener des expérimentations et des analyses approfondies pour déterminer les valeurs optimales des paramètres pour un problème donné. Cela peut impliquer des essais empiriques avec différentes combinaisons de paramètres et des analyses des performances de l'algorithme sur des ensembles de données de test.

4. Optimisation par Essaim d'Abeilles (BSO)

L'optimisation par Essaim d'Abeilles (BSO) est une métaheuristique basée sur le comportement de recherche de nourriture des abeilles. Cette méthode s'inspire des interactions et de la communication au sein d'un essaim d'abeilles pour résoudre des problèmes d'optimisation.

4.1. Définition et principes de l'optimisation par Essaim d'Abeilles

L'optimisation par Essaim d'Abeilles repose sur le concept d'essaims d'abeilles, où les abeilles travaillent collectivement pour rechercher de la nourriture et optimiser leur environnement. Ce processus s'appuie sur des principes de coopération, de communication et d'auto-organisation pour trouver des solutions efficaces aux problèmes d'optimisation.

4.2. Inspiration du comportement de recherche de nourriture des abeilles

L'optimisation par Essaim d'Abeilles s'inspire du comportement de recherche de nourriture des abeilles dans la nature. Les abeilles, en quête de nourriture pour leur ruche, utilisent des mécanismes de communication et de coopération pour explorer efficacement leur environnement. Ces interactions entre les abeilles, combinées à leur capacité à s'adapter à des conditions changeantes, ont inspiré le développement de l'algorithme d'optimisation par Essaim d'Abeilles. En imitant les stratégies naturelles des abeilles, cette méthode cherche à trouver des solutions optimales à divers problèmes d'optimisation.

4.3. Application de l'optimisation par Essaim d'Abeilles au Problème des Sacs à Dos Multiples (MKP

L'algorithme d'optimisation par Essaim d'Abeilles (BSO) peut être appliqué au Problème des Sacs à Dos Multiples (MKP) pour trouver des solutions efficaces. Dans cet algorithme, une solution de référence est initialement déterminée à l'aide de l'algorithme Beelnit. Ensuite, tout en respectant

les itérations maximales et tant que la solution optimale n'est pas atteinte, l'algorithme procède comme suit :

- La solution de référence est insérée dans une liste taboue pour éviter de revisiter les mêmes solutions.
- Des points de recherche sont déterminés à partir de la solution de référence.
- Chaque abeille dans l'essaim se voit attribuer une solution à partir des points de recherche.
- Pour chaque abeille, une recherche est effectuée à partir de la solution attribuée, et le résultat est stocké.
- La meilleure solution parmi celles obtenues par les abeilles est assignée comme nouvelle solution de référence.

Ce processus se répète jusqu'à ce que le nombre maximal d'itérations soit atteint ou que la solution optimale soit trouvée. L'optimisation par Essaim d'Abeilles est ainsi utilisée de manière itérative pour explorer l'espace des solutions du Problème des Sacs à Dos Multiples, en s'appuyant sur les interactions et les communications entre les abeilles pour guider la recherche vers des solutions prometteuses.

4.4. Algorithme d'optimisation par Essaim d'Abeilles (BSO)

Dans le processus d'optimisation utilisant l'algorithme d'Optimisation par Essaim d'Abeilles (BSO), divers paramètres jouent des rôles cruciaux dans le façonnage du comportement et des performances de l'algorithme. Parmi ces paramètres figurent le nombre d'abeilles (taille de la population), les itérations et l'équilibre entre l'exploration et l'exploitation.

```
function launchBSO(capacities: array of integer; items: array of Item; MaxIteration, flip,
    bees: integer; );
var

    iteration: integer;
    tabooList: array of Solution;
    sacks: array of Sack;
    sRef: Solution;// solution est une class java qui contient l'etat des sacs
    remainingItems: array of Item;
    searchPoints, newsearchPoints: array of Solution;
begin

    iteration := 0;
    tabooList := [];
    sacks := [];

    for cap in capacities do
        sacks.add(Sack(cap, []));
    end;

    sRef := generateInitialSolution(items, sacks);// on genere une solution aleatoire ou a l
    'aide d'une heuristique

    while (iteration < MaxIteration and noOptimal(Sref))do
    begin
        tabooList.add(sRef);
        remainingItems := items;

        for sack in sRef.Sacks do
            remainingItems.removeAll(sack.Items);

        searchPoints := getSearchPoints(sRef, remainingItems, bees);// de petites
    modification dans la solution initial pour cree les points de recherche

        // Perform local search
        newsearchPoints := localSearch(searchPoints, flip, resultsArea, tabooList);// les
    abeille recherche de nouvelle solutions en cherchant leur envirrants

        // Select the best solution from the search points
        sRef := getBestSolution(newsearchPoints);

        iteration := iteration + 1;
    end;


    return(sRef);
end;

begin
    // Main program logic goes here
end.
```

Figure 2:Pseudo-code de l'algorithme principale du BSO

Nombre d'Abeilles (Taille de la Population) : La taille de la population, représentant le nombre d'abeilles dans l'essaim, influence directement la diversité des solutions explorées lors de l'optimisation. Une plus grande taille de population facilite une exploration plus large de l'espace de recherche, potentiellement conduisant à la découverte de meilleures solutions. Cependant, augmenter la taille de la population augmente également les exigences computationnelles.

L'expérimentation est essentielle pour déterminer une taille de population optimale qui équilibre efficacement l'exploration et l'exploitation tout en tenant compte des contraintes computationnelles.

Nombre d'Itérations : Le nombre d'itérations ou de générations détermine la durée du processus d'optimisation. Chaque itération implique la mise à jour des positions des abeilles et l'évaluation de la qualité des solutions. Augmenter le nombre d'itérations permet une exploration plus approfondie de l'espace de recherche, potentiellement produisant des solutions supérieures. Néanmoins, il existe un compromis entre le temps de calcul et la qualité de la solution. L'expérimentation implique de tester différents nombres d'itérations pour trouver le juste équilibre entre la qualité de la solution et l'efficacité computationnelle.

Exploration vs Exploitation : BSO implique de trouver un équilibre délicat entre l'exploration (diversification) et l'exploitation (intensification) de l'espace de recherche. L'exploration vise à découvrir de nouvelles régions de l'espace de recherche en explorant des solutions diverses, tandis que l'exploitation se concentre sur le raffinement et l'amélioration des solutions dans des régions prometteuses. Le réglage des paramètres implique d'ajuster des paramètres comme le paramètre flip (contrôlant l'étendue de la recherche locale) et la taille du voisinage pour atteindre un équilibre optimal entre l'exploration et l'exploitation. L'expérimentation peut impliquer de tester différentes valeurs de paramètres pour trouver l'équilibre idéal pour le domaine de problème spécifique.

Lors de l'expérimentation, les chercheurs réalisent plusieurs essais avec des configurations de paramètres variées pour évaluer les performances de l'algorithme. Ils analysent des métriques clés telles que le taux de convergence, la qualité de la solution et l'efficacité computationnelle pour évaluer différentes configurations de paramètres. En ajustant et en affinant systématiquement les paramètres en fonction des résultats expérimentaux, les chercheurs peuvent optimiser l'algorithme BSO pour des tâches d'optimisation spécifiques et des domaines de problème.

5. Configuration Expérimentale

5.1. Description des Instances de MKP Utilisées dans les Expériences

Les sacs et les objets : Les instances du Problème du Sac à Dos Multi-dimensionnel (MKP) ont été modélisées à l'aide de classes Java. La classe centrale de cette modélisation est la classe State.

La classe représente un état des sacs contenant les articles. Elle est caractérisée par les attributs suivants :

- **Sacks** : Une liste de listes d'objets Item, représentant les différents sacs et les articles qu'ils contiennent.
- **id** : L'identifiant unique de l'état.
- **totalWeight** : Le poids total de tous les articles dans l'état.
- **itemIndex** : L'indice de l'article en cours de traitement.
- **children**: Une liste contenant les états enfants de l'état actuel.
- **visited**: Un indicateur indiquant si l'état a été visité lors de l'exploration.
- **visitDuration**: La durée de la visite de l'état.

De plus, la classe **State** fournit des méthodes pour calculer le poids total, la valeur totale, ajouter des états enfants, récupérer les enfants, vérifier si un article est contenu dans l'état, ainsi que d'autres méthodes d'accès aux attributs.

La classe **State** est une composante centrale du code, représentant les différents états possibles des sacs avec les articles à l'intérieur dans le cadre de la résolution du Problème du Sac à Dos Multi-dimensionnel.

5.2. Environment de travail

Dans cette section, nous décrivons l'environnement de travail utilisé pour mener nos expériences :

Système d'Exploitation : Windows 10.

Configuration Matérielle : Ordinateur avec 4 Go de RAM et un processeur Intel(R) Atom(TM) x5-Z8350 CPU @ 1.44GHz.

Environnement de Développement : Java 21JE, Eclipse 2023.

Bibliothèques Utilisées :

- **Swing** pour l'interface visuelle.
- **Graphviz** pour la génération d'illustrations.
- **JFreeChart** pour la collecte de données expérimentales.

Interface Utilisateur : Nous avons utilisé une interface spécialement conçue pour l'utilisation des algorithmes métaheuristiques. Cette interface permet de générer des exemples de sacs multiples, de paramétrer l'algorithme de choix, puis d'exécuter l'algorithme. Les données telles que le temps d'exécution, le taux de satisfaction, la moyenne de temps et l'écart type, ainsi que le nombre d'itérations, sont collectées par l'interface utilisateur et présentées dans des graphes et des tableaux. Pour ces tests, nous avons générer un nombre de 10 fichiers contenant les paires poids valeur d'un nombre d'objet allant de 10 jusqu'a 30, ces fichiers serons utiliser pour tous les tests. Par souci de simplicité les sacs qui sont au nombre de trois ont le même poids de 30.

6. Résultats des expérimentations

Dans cette section, nous présentons les résultats expérimentaux de nos algorithmes d'optimisation, à savoir les algorithmes génétiques (GA) et l'optimisation par essaim d'abeilles (BSO), dans le contexte de la résolution du problème du sac à dos multiple (MKP).

Nous nous focalisons sur deux paramètres d'analyse : la durée moyenne d'exécution et le taux de valeur, c'est-à-dire la valeur totale du meilleur résultat obtenu devisé par la valeur totale des données.

6.1. Performances des Algorithmes Génétiques

Nous débutons par une exploration des performances des algorithmes génétiques en variant différents paramètres clés tels que la taille de la population, le taux de mutation, et le nombre de générations. Nous analysons l'impact de ces variations sur la qualité des solutions produites et le temps de convergence.

Nombre d'objet	Temps moyen	Taux de valeur totale
11	7.0308	0.6001
12	5.8221	0.3112
13	6.5572	0.4406
14	6.8888	0.4456
15	6.9187	0.4325
16	6.8123	0.4323
17	6.6990	0.3518
18	7.4584	0.4083
19	6.5641	0.2907
20	8.1926	0.3172

Tableau 1: Durée d'execution et taux de valeur pour l'algorithme génetique GA

Ce tableau présente les résultats moyens obtenus par un algorithme génétique pour résoudre un problème de sac à dos multiple. La première colonne représente le nombre d'objets considérés, tandis que la deuxième colonne indique le temps moyen nécessaire à l'algorithme, exprimé en seconds. La troisième colonne donne le ratio de la valeur atteinte par l'algorithme, c'est-à-dire le rapport entre la valeur totale des objets placés dans les sacs et la valeur totale maximale possible. On constate que le temps moyen augmente généralement avec le nombre d'objets, ce qui est logique étant donné la complexité croissante du problème. Le ratio de valeur atteint reste élevé, se situant autour de 0,3 à 0,45 pour la plupart des tailles de problèmes, ce qui indique que l'algorithme trouve des solutions proches de l'optimal. Ces données permettent d'évaluer les

performances de l'algorithme génétique proposé en termes de temps de calcul et de qualité des solutions trouvées, en fonction de la taille du problème à résoudre.

6.2. Performances de l'Optimisation par Essaim d'Abeilles

Ensuite, nous examinons les résultats obtenus avec l'optimisation par essaim d'abeilles en modifiant ses paramètres, notamment le nombre d'abeilles, la probabilité de recherche locale, et le nombre d'itérations. Nous évaluons ainsi l'efficacité de cette approche dans la résolution du MKP.

Nombre d'objet	Temps moyen	Taux de valeur totale
11	2.8387	0.6001
12	2.9751	0.3512
13	3.0507	0.4406
14	3.1444	0.4456
15	3.4467	0.4825
16	3.6079	0.4323
17	3.6902	0.3718
18	3.9976	0.4083
19	4.0652	0.3207
20	4.1141	0.3372

Tableau 2: Durée d'execution et taux de valeur pour le BSO

Ce tableau présente les résultats moyens obtenus par un algorithme d'essaim de particules (bee swarm algorithm) pour résoudre un problème de sac à dos multiple. La première colonne représente le nombre d'objets considérés, tandis que la deuxième colonne indique le temps moyen nécessaire à l'algorithme, exprimé en secondes. La troisième colonne donne le ratio de la valeur atteinte par l'algorithme, c'est-à-dire le rapport entre la valeur totale des objets placés dans les sacs et la valeur totale maximale possible. On observe que le temps moyen augmente de manière régulière avec le nombre d'objets, conformément à ce qui est attendu étant donné la croissance de la complexité du problème. Le ratio de valeur atteint reste globalement stable, se situant autour de 0,33 à 0,45, ce qui indique que l'algorithme parvient à trouver des solutions de bonne qualité pour différentes tailles de problèmes. Ces données permettent d'évaluer les performances de l'algorithme d'essaim de particules proposé en termes de temps de calcul et de qualité des solutions trouvées, en fonction du nombre d'objets à placer dans les sacs à dos.

6.3. Analyse Comparative

Une fois les performances de chaque algorithme analysé individuellement, nous procédons à une expérience comparative approfondie pour évaluer leur efficacité relative. Nous comparons les solutions obtenues, ainsi que les temps de convergence, afin de déterminer quel algorithme offre les meilleures performances dans le contexte du MKP

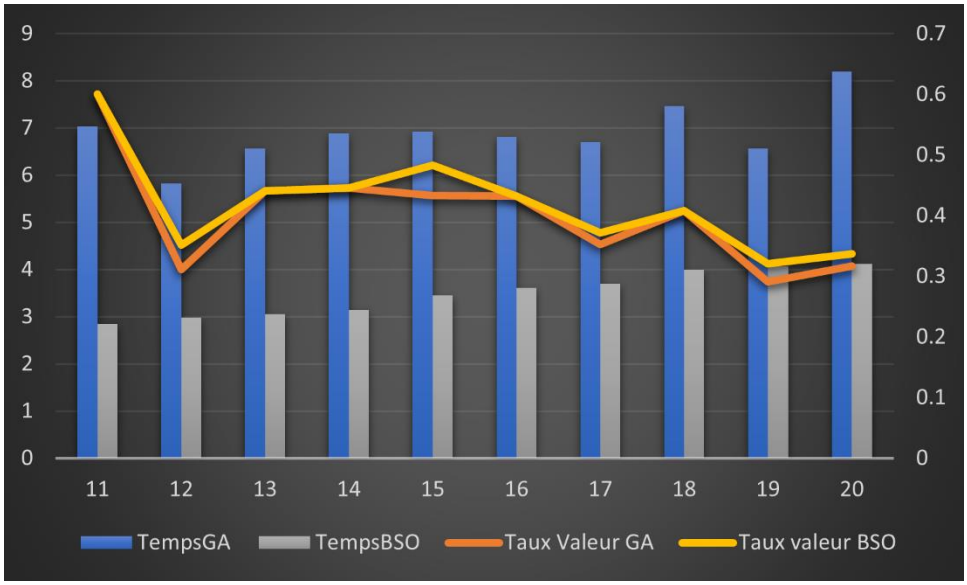

Figure 3: Comparaison des algorithmes BSO et GA pare duréé d'éxecution et taux de valeur atteinte

En comparant les résultats des deux algorithmes résumés dans le graph ci-dessus, on constate que les temps moyens de l'algorithme génétique sont généralement plus élevés que ceux de l'algorithme d'essaim de particules (bee swarm) pour un même nombre d'objets, suggérant ainsi que ce dernier est plus rapide en termes de temps de calcul. Cependant, en ce qui concerne le ratio de valeur atteinte, les performances des deux algorithmes sont similaires, avec des ratios se situant autour de 0,3 à 0,45 pour la plupart des tailles de problèmes, indiquant qu'ils parviennent à trouver des solutions de qualité comparable. Il est intéressant de noter que l'algorithme génétique semble légèrement supérieur en termes de ratio de valeur atteinte pour les petites tailles de problèmes (jusqu'à 13 objets), tandis que l'algorithme d'essaim de particules semble avoir un léger avantage pour les plus grandes tailles de problèmes (à partir de 14 objets). En résumé, si le temps de calcul est un facteur critique, l'algorithme d'essaim de particules semble être un choix plus judicieux, tandis que si la qualité de la solution est la priorité, les deux algorithmes offrent des performances similaires, avec un léger avantage pour l'algorithme génétique sur les petites instances et pour l'algorithme d'essaim de particules sur les plus grandes instances.

6.4. Discussion des Résultats

En analysant les résultats, on constate que le BSO est généralement plus rapide que le GA en termes de temps de calcul moyen. Cette différence peut s'expliquer par la complexité légèrement supérieure du GA dû à une utilisation plus soutenus d'heuristiques, qui implique davantage de calculs pour simuler le comportement de la sélection naturel. Cependant, en ce qui concerne la qualité des solutions (ratio de valeur atteinte), les deux algorithmes ont des performances très similaires, oscillant autour de 0,3 à 0,45 pour la plupart des tailles de problèmes. Cela suggère qu'ils sont tous deux capables de trouver des solutions de bonne qualité pour ce problème d'optimisation complexe. Un point intéressant à noter est que le BSO et le GA semble avoir des performances similaires pour les petites instances (jusqu'à 13 objets), tandis que le BSO prend légèrement l'avantage pour les plus grandes instances (à partir de 14 objets). Cette différence subtile pourrait être liée à la manière dont chaque algorithme explore l'espace de recherche et converge vers les régions prometteuses. Bien que le BSO soit plus rapide, les deux algorithmes offrent des performances comparables en termes de qualité de solution pour ce problème de sac à dos multiple. Le choix entre les deux pourrait dépendre des contraintes spécifiques du problème, comme le temps de calcul disponible ou la taille de l'instance à résoudre, mais aussi des paramétrages. En effet, dans notre expérimentation, nous avons exécuté les algorithmes sur les mêmes données plusieurs fois en variant leurs hyperparamètres, tels que le nombre de générations, la taille de la population et le taux de mutation pour l'algorithme génétique, ainsi que le nombre d'abeilles, le "flip", et le nombre d'itérations pour le BSO. Bien que dans notre cas nous avons trouvé que le BSO est plus rapide que le GA pour atteindre des résultats similaires, cela ne peut être généralisé à ces deux algorithmes, mais est plutôt le résultat de notre implémentation particulière de ces deux algorithmes, qui contiennent des heuristiques différentes, ce qui vraisemblablement joue dans leur performance.

7. Analyse des performances des métaheuristiques avec les algorithmes de recherche de graph DFS

Ici nous comparons les résultats des algorithmes métaheuristiques avec un des algorithmes a base de recherche graphique. Nous choisissant l'algorithme DFS pour bien démontre le contraste qui existe entre ses deux catégories d'algorithmes.

	Durée Moyenne (seconds)			Taux de valeur totale		
Nombre d'objet	**DFS**	**BSO**	**GA**	**DFS**	**BSO**	**GA**
11	27.20	2.84	7.03	61%	60.01%	60.01%
12	34.42	2.98	5.82	37.24%	35.12%	31.12%
13	40.23	3.05	6.56	44.33%	44.06%	44.06%
14	47.23	3.14	6.89	44.17%	44.56%	44.56%
15	86.12	3.45	6.92	49.79%	48.25%	43.25%
16	115.63	3.61	6.81	53.42%	43.23%	43.23%
17	174.75	3.69	6.70	39.56%	37.18%	35.18%
18	233.61	4.00	7.46	45.66%	40.83%	40.83%
19	274.33	4.07	6.56	33.72%	32.07%	29.07%
20	307.22	4.11	8.19	33.48	33.72%	31.72%

Tableau 3 : Comparaison des temps et taux de valeurs entre les algorithmes de recherche par espace d'état et les métaheuristiques

D'après les résultats présentés dans ce tableau, nous pouvons analyser et comparer les performances des trois algorithmes (DFS, BSO et GA) pour résoudre le problème de sac à dos multiple, en termes de temps d'exécution et de ratio de valeur atteinte.

Tout d'abord, comparons les temps d'exécution moyens :

L'algorithme BSO (Bees Swarm Optimization) est généralement le plus rapide pour la plupart des tailles de problèmes.

L'algorithme génétique (GA) est plus lent que BSO, mais plus rapide que DFS (Depth-First Search) pour les grandes instances.

L'algorithme DFS est le plus lent des trois pour la majorité des tailles de problèmes.

Ensuite, analysons le ratio de valeur atteinte (taux valeur totale) :

- L'algorithme DFS étant un algorithme de recherche par espace des états, trouvera la solution optimale si les recoures sont suffisantes
- Les trois algorithmes obtiennent des ratios de valeur similaires, oscillant autour de 60% pour les petites instances (11 objets) et diminuant progressivement jusqu'à environ 32-35% pour les plus grandes instances (20 objets). Les algorithmes GA et BSO trouve des solutions légèrement inferieur ou bien aussi bonne que celle du DFS

Quelques observations supplémentaires :

- Pour les petites instances (jusqu'à 13 objets), DFS et BSO semblent légèrement plus performants que GA en termes de ratio de valeur atteinte.
- Pour les plus grandes instances (à partir de 14 objets), DFS et BSO dépassent légèrement GA en termes de ratio de valeur atteinte.
- La diminution du ratio de valeur atteinte avec l'augmentation de la taille du problème est probablement due à la complexité croissante de l'espace de recherche, rendant plus difficile la découverte des solutions optimales.

En résumé, si le temps d'exécution n'est pas la priorité et les données sont de faible taille, l'algorithme DFS semble être le choix le plus judicieux, en particulier pour les petites et moyennes instances. Cependant, pour les grandes instances, l'algorithme BSO pourrait être une alternative intéressante, offrant un bon compromis entre temps d'exécution et qualité de solution. L'algorithme GA, bien que plus lent, peut être envisagé s'il n'y a pas de contrainte de temps critique et que la qualité de la solution est primordiale.

8. Conclusion

Notre étude a examiné divers algorithmes pour résoudre le problème du sac à dos multiple. Nous avons d'abord défini et modélisé le problème, puis nous avons introduit les algorithmes exhaustifs de type recherche graphique, en commençant par la recherche en profondeur, puis la recherche en largeur, et enfin l'algorithme A-star. Nous avons constaté que l'utilisation d'heuristiques permet de réduire significativement le nombre de nœuds dans l'espace de recherche de notre algorithme. Ensuite, nous avons discuté des algorithmes méta-heuristiques, notamment l'algorithme génétique et l'optimisation BSO. Notre étude a montré que notre implémentation de l'optimisation BSO était plus performante que l'algorithme génétique, probablement en raison de l'implémentation particulière de ces deux algorithmes dans notre code ; en particulier, l'impact des heuristiques utilisé à dans différentes étapes de l'exécution des deux algorithmes. Nous avons également comparé les méthodes exhaustives aux méthodes méta-heuristiques et avons constaté qu'il existe un compromis entre la rapidité des méthodes méta-heuristiques et la recherche de la solution optimale des méthodes exhaustives. En fin de compte, choisir le bon algorithme dépend de trouver un équilibre entre exploration et exploitation.