Recherche Opérationnelles : TP n°2
Algorithme Branch and Bound

1 Objectifs

Ce TP a pour objectifs de :
— Comprendre le principe de l'algorithme de Branch and Bound.
— Appliquer la méthode à un problème concret d'optimisation.
— Analyser l'intérêt pratique de cet algorithme.

2 L'algorithme de Branch and Bound

2.1 Principe

L'algorithme de Branch and Bound est une méthode générale pour résoudre des problèmes d'optimisation combinatoire. Il explore l'espace des solutions de manière implicite en construisant un arbre de recherche. Le principe repose sur deux idées clés :
— **Branch (Branchement)** : Diviser le problème en sous-problèmes plus petits.
— **Bound (Borne)** : Calculer des bornes inférieures (pour un problème de minimisation) ou supérieures (pour un problème de maximisation) pour la solution optimale de chaque sous-problème. Si la borne d'un sous-problème est moins bonne que la meilleure solution trouvée jusqu'à présent, ce sous-problème peut être ignoré (élagage).

2.2 Intérêt pratique

L'algorithme de Branch and Bound permet de réduire l'espace de recherche en éliminant des branches entières de l'arbre de recherche. Cela le rend plus efficace que l'exploration exhaustive pour les problèmes de grande taille.

2.3 Applications

L'algorithme de Branch and Bound est utilisé dans de nombreux domaines, notamment :
— Recherche opérationnelle (Problème du voyageur de commerce, problème du sac à dos)
— Intelligence artificielle (Jeux, planification)
— Optimisation en général

3 Exercice : Le problème du sac à dos

On considère le problème du sac à dos (Knapsack Problem) : étant donné un ensemble d'objets, chacun ayant un poids et une valeur, et un sac à dos avec une capacité maximale de poids, trouver le sous-ensemble d'objets qui maximise la valeur totale sans dépasser la capacité du sac à dos.

3.1 Implémentation

La fonction `branch_and_bound` appelle `calculer_borne` pour évaluer si une branche doit être explorée ou non. Si la borne calculée pour une branche est inférieure à la meilleure valeur déjà trouvée, la branche est abandonnée.

3.1.1 Fonction `calculer_borne`

Cette fonction estime une borne supérieure pour la valeur maximale que l'on peut obtenir dans un sous-problème du sac à dos.

```python
def calculer_borne(objets, capacite_restante):
    objets_tries = sorted(objets, key=lambda x: x[1]/x[0], reverse=True)
    valeur_totale = 0
    poids_total = 0

    for poids, valeur in objets_tries:
        if poids_total + poids <= capacite_restante:
            valeur_totale += valeur
            poids_total += poids
        else:
            fraction = (capacite_restante - poids_total) / poids
            valeur_totale += fraction * valeur
            break  # La capacité est atteinte

    return valeur_totale
```

Listing 1 – Fonction de calcul de la borne

3.1.2 Fonction `branch_and_bound`

Cette fonction implémente l'algorithme de Branch and Bound. Elle explore l'arbre de recherche en créant des nœuds pour chaque décision (inclure ou non un objet dans le sac à dos).

```python
def branch_and_bound(objets, capacite):
    meilleure_solution = []
    meilleure_valeur = 0
    noeuds = [([], 0, 0, objets)] # solution courante, valeur courante, poids courant, objets restants


    while noeuds:
        solution_courante, valeur_courante, poids_courant, objets_restants = noeuds.pop()

        if not objets_restants:
            if valeur_courante > meilleure_valeur:
                meilleure_valeur = valeur_courante
                meilleure_solution = solution_courante
            continue

        poids, valeur = objets_restants[0]

        # Branche "inclure l'objet"
        if poids_courant + poids <= capacite:
              noeuds.append((solution_courante + [objets_restants[0]], valeur_courante + valeur, poids_courant + poids, objets_restants[1:]))

        # Branche "exclure l'objet"
        borne = calculer_borne(objets_restants[1:], capacite - poids_courant) + valeur_courante
        if borne > meilleure_valeur :
              noeuds.append((solution_courante, valeur_courante, poids_courant, objets_restants[1:]))


    return meilleure_solution, meilleure_valeur
```

Listing 2 – Fonction de branch and bound

3.2 Application

Appliquer l'algorithme implémenté à la même instance du problème du sac à dos du TP précédent : Capacité maximale de 10 kg. 6 objets dont les poids et les valeurs respectives sont données comme suit :

$$Poids[] = \{2, 3, 4, 3, 2, 7\}$$

$$Valeur[] = \{4, 5, 2, 1, 4, 14\}$$

le code de lancement sera alors

```
1
2  # Données du problème
3  poids = [2, 3, 4, 3, 2, 7]
4  valeur = [4, 5, 2, 1, 4, 14]
5  objets = list(zip(poids, valeur))
6  capacite = 10
7
8  # Résolution du problème
9  meilleure_solution, meilleure_valeur = branch_and_bound(objets, capacite)
10
11
12 print("Meilleure solution:", meilleure_solution)
13 print("Valeur maximale:", meilleure_valeur)
```

Listing 3 – Programme principal

4 Travail à faire

— Donner une explication approfondie du fonctionnement de la solution proposée.
— Refaire une implémentation en C/C++ de la solution.
— Analyser les résultats et la performance de l'algorithme.