Liste d'exercices

May 20, 2021

Informations préalables

Hormis pour la séance 1, pour toutes les autres questions on considérera pour tester les résultats les instances du projet dont la structure pourra être retrouvée ici `https://www.overleaf.com/read/mpzfrdttbkms`. Les instances à un seul véhicule correspondent au problème du voyageur de commerce. Dans le cas du problème du Sac à Dos multidimensionnel (resp. unidimensionnel), il s'agira de considérer les instances avec plusieurs (resp. un) véhicules en ne tenant compte uniquement que des capacités/utilités/poids sans tenir compte des distances.

1 Séance 1

Modélisation d'un problème d'optimisation

Considérons N tâches indexées par un indice i allant de 1 à N et M machines indexées par j allant de 1 à M. L'objectif est d'affecter ces tâches aux machines, sachant que chaque tâche peut être effectuée par n'importe quelle machine.

- L'affectation de la tâche i à la machine j à un coût de p_{ij}
- Chaque machine j a une capacité maximale c_j
- L'affectation de la tâche i à la machine j consomme b_{ij} de la capacité de la machine.

On suppose dans la première question que le problème est toujours réalisable (p.e. on a une machine de capacité illimitée).

Question 1: Modéliser le problème sous la forme d'un problème d'optimisation

Considérons les variables booléennes X_{ij} indiquant si la tâche i est affectée à la machine j. On cherche à minimiser les coûts d'affectation sachant que les contraintes sont:

- La capacité consommée par l'affectation des tâches sur une machine j ne doit pas dépasser sa capacité

- Chaque tâche doit être affectée à une machine

$$\min \quad \sum_{i=1}^{N}\sum_{j=1}^{M} p_{ij}X_{ij} \tag{1}$$

$$\sum_{i=1}^{N} b_{ij}X_{ij} \leq c_j, \qquad \forall j \in 1,\ldots,M \tag{2}$$

$$\sum_{j=1}^{M} X_{ij} = 1, \qquad \forall i \in 1,\ldots,N \tag{3}$$

$$X_{ij} \in \{0,1\}, \qquad \forall i \in 1,\ldots,N, \quad \forall j \in 1,\ldots,M \tag{4}$$

On suppose maintenant qu'il est possible que toutes les tâches ne puissent être affectées. On définit alors un coût l_i correspondant au fait de ne pas affecter la tâche i.

Question 2: Modéliser ce nouveau problème

On rajoute des variables booléennes Y_i. $Y_i = 1$ si la tâche i n'a pas été affectée.

$$\min \quad \sum_{i=1}^{N} l_i Y_i + \sum_{i=1}^{N}\sum_{j=1}^{M} p_{ij}X_{ij} \tag{5}$$

$$\sum_{i=1}^{N} b_{ij}X_{ij} \leq c_j, \qquad \forall j \in 1,\ldots,M \tag{6}$$

$$Y_i + \sum_{j=1}^{M} X_{ij} = 1, \qquad \forall i \in 1,\ldots,N \tag{7}$$

$$X_{ij} \in \{0,1\}, \qquad \forall i \in 1,\ldots,N, \quad \forall j \in 1,\ldots,M \tag{8}$$

Question 3: Définir une heuristique pour construire une solution réalisable

Heuristique de construction pour le problème du sac à dos

Le problème se définit de la façon suivante:

- On dispose d'un ensemble de N objets
- Chaque objet i dispose d'une utilité u_i et d'un poids p_i.
- On cherche à maximiser l'utilité totale d'un sac de capacité c^{max}.

```python
def testAllSolutions(self):
    listIndexes = [i for i in range(len(self.instance.objects))]
    (value, bestList) = self.testOneSolution(listIndexes, self.instance.bag.capacity)
    print("Total utility:", value)
    print("best list:", bestList)
    return 0

def testOneSolution(self, listIndexes, capacity):
    if capacity < 0 or len(listIndexes) == 0:
        return (0, [])
    totValue = 0
    listIndexesBis = [listIndexes[i] for i in range(1, len(listIndexes))]
    object_ = self.instance.objects[listIndexes[0]]
    #We don't pick the object
    (value1, bestList1) = self.testOneSolution(listIndexesBis, capacity)
    #We pick the object
    (value2, bestList2) = self.testOneSolution(listIndexesBis, capacity-object_.weight)
    if capacity-object_.weight >= 0:
        value2 += object_.utility
        bestList2.append(listIndexes[0])
    totValue = max(value1, value2)
    if value1 > value2:
        return (value1, bestList1)
    else:
        return (value2, bestList2)
```

Figure 1: Exploration exhaustive

Question 1: Implémenter et tester la méthode exacte consistant à essayer toutes les combinaisons possibles et à garder la meilleure

Question 2: Définir une heuristique de construction pour le problème du sac à dos

Question 3: Implémenter cette heuristique

Une solution: On trie chaque objet par valeur $\frac{u_i}{p_i}$ décroissante. Cette valeur correspond à l'utilité par unité de poids et représente l'intérêt marginal de chaque objet. En mettant chaque objet sur la même échelle, l'idée est de les ajouter un par un dans l'ordre défini jusqu'à ce que le sac soit rempli.

```python
#On implemente une affectation des objets au sac a dos
def heuristicConstruction(self):

    #On trie les objets par utilite/poids decroissants
    def utilityWeight(item):
        return item.utility/(0.01 + item.weight)
    listObjects = [item for item in self.instance.objects]
    listObjects.sort(key=utilityWeight, reverse = True)

    i = 0
    bag = self.instance.bag
    item = listObjects[i]
    #On parcourt chacun des objets tries
    #On les ajoute au fur et a mesure tant que l'on a la capacite necessaire
    while i < len(listObjects) and item.weight <= bag.capacityLeft:
        #On ajoute l'objet au sac: l'objet est ajoute et la capacite du sac est mise a jour
        bag.addObject(listObjects[i])
        i += 1
        #On met a jour d'objet a considerer
        if i < len(listObjects):
            item = listObjects[i]
```

Figure 2: Construction d'une solution initiale

2 Séance 2

Heuristique de construction d'un chemin initial pour le problème du voyageur de commerce

On dispose:

- D'un ensemble de N points de livraison définis par leurs coordonnées (x_i, y_i) pour le point i
- D'un véhicule partant d'un dépôt situé en position $(0, 0)$

L'objectif du problème est de déterminer le chemin de longueur minimal:

- Partant du dépôt
- Revenant au dépôt
- Passant par tous les points de livraison

Question 1: Proposer une heuristique de construction d'un chemin initial

Question 2: Implémenter cette heuristique

Utilisation de méthodes de Recherche Locale

Question 1: Proposer puis implémenter une méthode de Recherche Locale pour le problème du voyageur de commerce

Question 2: Proposer puis implémenter une méthode de Recherche Locale pour le problème du Sac à Dos

3 Séance 3

Problème du voyageur de commerce avec capacité limitée

On a le même problème que le problème du voyageur de commerce classique sauf que l'on affecte au véhicule une capacité c^{max} et à chaque point de livraison i une utilité u_i et un poids p_i.

Question 1: Proposer différentes méthodes de construction de solutions de base

Question 2: Proposer et implémenter un algorithme qui effectue de la Recherche Locale sur le problème avec capacité

Modélisation mathématique du problème du sac à dos multidimensionnel

On a:

- M sac à dos de capacité c_j^{max} pour le sac j
- N produits de poids p_i et d'utilité u_i pour l'objet i

On effectuera les tests sur les instances avec plusieurs véhicules. Un véhicule à capacité limitée correspond à un sac à dos, un point de livraison correspond à un produit.

Question 1: Proposer une modélisation mathématique du problème du sac à dos multi-dimensionnel

Question 2: Implémenter et optimiser ce modèle mathématique

Question 3: A partir de la solution partielle, créer une méthode d'affectation des objets au sac

Question 4: Comparer avec une méthode qui consisterait à ne considérer qu'un seul sac à dos (dont la capacité serait la somme de toutes les capacités), et qui affecterait les objets aux différents sacs après optimisation.

4 Séance 4

Utilisation de méthodes de Recherche Locale (2)

Question 1: Modifier l'algorithme $2-opt$ en ajoutant une liste tabou. (voir Section 0.4) On ajoutera à chaque itération dans la liste les deux nouvelles arêtes venant d'être créées. Ces deux nouvelles arêtes ne pourront plus être modifiées pendant 50 itérations.

Question 2: Comparer les résultats avec l'algorithme $2-opt$ classique

Question 3: Considérer d'autres types de liste tabou (*conserver les sommets au lieu des arêtes...*)

Question 4: Implémenter la recherche tabou sur un voisinage qui consisterait à échanger les positions de deux sommets)

5 Séance 5

Génération de bornes inférieures et de bornes supérieures

Question 1: Implémenter l'algorithme de Kruskal pour trouver une borne inférieure de la longueur d'un cycle hamiltonien de longueur minimum

Question 2: Utiliser l'arbre couvrant de poids minimal obtenu pour reconstruire un cycle hamiltonien et ainsi obtenir une borne supérieure