Planification d'activités avec un algorithme glouton

Première NSI

qkzk & chatGPT

2023-02-23

Voici quelques exemples et une résolution (rectifiée...) proposés par ChatGPT pour illustrer un exemple classique : la planification des tâches avec un algorithme glouton ainsi qu'une recherche exhaustive.

pdf - Script accompagnant ces exemples

Un exemple classique de choix d'activité à planifier qui peut être résolu par un algorithme glouton est le problème de sélection d'activités.

Supposons qu'il y ait un ensemble d'activités à planifier, chacune ayant une heure de début et une heure de fin.

L'objectif est de sélectionner les activités ayant une durée totale maximale sans qu'elles ne se chevauchent.

Exemple introductif et résolution gloutonne

Voici un exemple de jeu de données d'activités :

Activité	Heure de début	Heure de fin
A	9h00	10h00
B	9h30	11h30
C	10h00	11h00
D	11h00	12h00

Pour résoudre ce problème par un algorithme glouton, nous pouvons trier les activités par heure de fin croissante. Ensuite, nous sélectionnons la première activité de la liste, qui a l'heure de fin la plus basse. Nous ajoutons ensuite toutes les activités suivantes qui ont une heure de début supérieure à l'heure de fin de l'activité sélectionnée précédemment. Nous répétons ce processus jusqu'à ce qu'il n'y ait plus d'activités restantes.

Voici les étapes de l'algorithme glouton :

Trier les activités par heure de fin croissante.

Activité	Heure de début	Heure de fin
A	9h00	10h00
C	10h00	11h00
B	9h30	11h30
D	11h00	12h00

Sélectionner la première activité, A.

Activité sélectionnée	Heure de début	Heure de fin
A	9h00	10h00

Ajouter toutes les activités suivantes qui ont une heure de début supérieure à l'heure de fin de A. Dans ce cas, la prochaine activité qui respecte cette condition est C.

Activités sélectionnées	Heure de début	Heure de fin
A	9h00	10h00
C	10h00	11h00

Continuer à ajouter des activités jusqu'à ce qu'il n'y en ait plus de compatibles. Dans ce cas, l'activité suivante compatible est D.

Activités sélectionnées	Heure de début	Heure de fin
A	9h00	10h00
C	10h00	11h00
D	11h00	12h00

Ainsi, l'algorithme glouton a sélectionné trois activités : A, C et D, qui ne se chevauchent pas. L'activité B a été écartée car elle se chevauche avec C.

Autre exemple avec 10 activités

Voici un exemple de jeu de données d'activités plus grand, avec 10 activités :

Activité	Heure de début	Heure de fin
A	9h00	10h30
B	9h00	10h00
C	10h00	11h00
D	10h00	12h00
E	11h00	12h30
F	12h30	13h00
G	12h00	14h00
H	13h00	14h00
I	13h30	15h00
J	14h00	15h00

En triant les activités par heure de fin croissante, nous obtenons :

Activité	Heure de début	Heure de fin
B	9h00	10h00
A	9h00	10h30
C	10h00	11h00
D	10h00	12h00
E	11h00	12h30
F	12h30	13h00
H	13h00	14h00
G	12h00	14h00
I	13h30	15h00
J	14h00	15h00

En appliquant l'algorithme glouton, nous obtenons la solution suivante :

Activités	sélectionnées Heure de début	Heure de fin
B	9h00	10h00
C	10h00	11h00
E	11h00	12h30
F	12h30	13h00

Activités	sélectionnées Heure de début	Heure de fin
H	13h00	14h00
J	14h00	15h00

L'algorithme glouton ne fournit pas toujours la solution optimale, mais il permet souvent d'obtenir une solution acceptable en un temps raisonnable. Cependant, si l'objectif est d'obtenir la solution optimale, il est nécessaire de recourir à un algorithme de recherche exhaustive qui va énumérer toutes les combinaisons possibles d'activités et sélectionner celle qui est compatible et qui a la durée maximale. Cela peut être très coûteux en termes de temps de calcul si le nombre d'activités est important.

Exemple où l'algorithme glouton ne fournit pas une solution optimale

Ici il est notable de remarquer que ChatGPT n'a jamais été en mesure de fournir un contre exemple valide. J'ai dû lui proposer cet exemple simpliste après une dizaine de rectifications infructueuses. Il s'entétait à proposer des tâches qui se chevauchent. Je l'ai questionné sans réponse sur la source de ces erreurs.

Voici un petit exemple où l'algorithme glouton ne fournit pas la solution optimale :

Activité	Heure de début	Heure de fin
A	9h00	10h30
B	9h30	14h00
C	10h30	13h00

En triant les activités par heure de fin croissante, nous obtenons :

Activité	Heure de début	Heure de fin
A	9h00	10h30
C	10h30	13h00
B	9h30	14h00

En appliquant l'algorithme glouton, nous obtenons la solution suivante :

Activités sélectionnées	Heure de début	Heure de fin
A	9h00	10h30
C	10h30	13h00

La durée totale des activités sélectionnées est de 3 heures.

Cependant, la solution optimale consiste à sélectionner l'activité B dont la durée est de 4h30...

Résolution python

Voici une proposition d'implémentation d'une solution gloutonne en Python pour la planification de tâches, en utilisant des tuples de la forme ("A", 9.5, 10.5) ou ("B", 10.5, 11.5) pour représenter les tâches :

```
def planification_taches(taches):
    # Tri des tâches par ordre croissant de fin de la tâche
    taches_triees = sorted(taches, key=lambda x: x[2])

    # Création de la liste de tâches planifiées
    taches_planifiees = []

    # Définition de la fin de la dernière tâche planifiée
    fin_derniere_tache = 0
```

```
    # Parcours des tâches triées par ordre croissant de fin de la tâche
    for tache in taches_triees:
        # Si la tâche peut être planifiée avant la fin de la dernière tâche planifiée
        if tache[1] >= fin_derniere_tache:
            taches_planifiees.append(tache)
            fin_derniere_tache = tache[2]

    return taches_planifiees
```

Explication du code :

La fonction `planification_taches` prend en entrée une liste de tâches représentées par des tuples de la forme `("A", 9.5, 10.5)` ou `("B", 10.5, 11.5)`. La première étape consiste à trier les tâches par ordre croissant de fin de la tâche, en utilisant la fonction sorted avec une fonction lambda qui extrait le troisième élément de chaque tuple (c'est-à-dire la fin de la tâche). Ensuite, la liste de tâches planifiées est initialisée avec la première tâche triée. La variable `fin_derniere_tache` est initialisée avec la fin de la première tâche planifiée. Ensuite, chaque tâche triée est parcourue, et si elle peut être planifiée avant la fin de la dernière tâche planifiée, elle est ajoutée à la liste de tâches planifiées et la variable `fin_derniere_tache` est mise à jour avec la fin de cette tâche.

Finalement, la liste de tâches planifiées est renvoyée. Cette solution gloutonne consiste à toujours choisir la tâche qui se termine le plus tôt parmi celles qui peuvent être planifiées. Cela ne garantit pas une planification optimale, mais peut fournir une solution raisonnablement efficace pour des instances de petite à moyenne taille.

Recherche exhaustive d'une solution optimale

Solution proposée par ChatGPT et légèrement rectifiée.

```
def planification_taches_exhaustif(taches: list[tuple]):
    """
    Recherche exhaustive d'une solution optimale
    """
    # Générer toutes les permutations possibles de la liste de tâches
    permutations = itertools.permutations(taches)

    # Initialiser les variables de la meilleure solution trouvée
    meilleur_planification = None
    meilleur_duree_totale = -float("inf")

    # Parcourir chaque permutation possible
    for permutation in permutations:
        # Vérifier si la permutation est valide (pas de chevauchement de tâches)
        duree_totale = 0
        fin_derniere_tache = 0
        valide = True
        for tache in permutation:
            if tache[1] < fin_derniere_tache:
                valide = False
                break
            duree_totale += tache[2] - tache[1]
            fin_derniere_tache = tache[1]

        # Si la permutation est valide et donne une durée totale plus courte
        # que la meilleure solution actuelle
        if valide and duree_totale > meilleur_duree_totale:
            meilleur_planification = permutation
            meilleur_duree_totale = duree_totale

    return meilleur_planification
```