Projet 7 OpenClassrooms

Résolvez des problèmes en utilisant des algorithmes en Python

Fabien ROYER

Table des matières

Présentation du problème

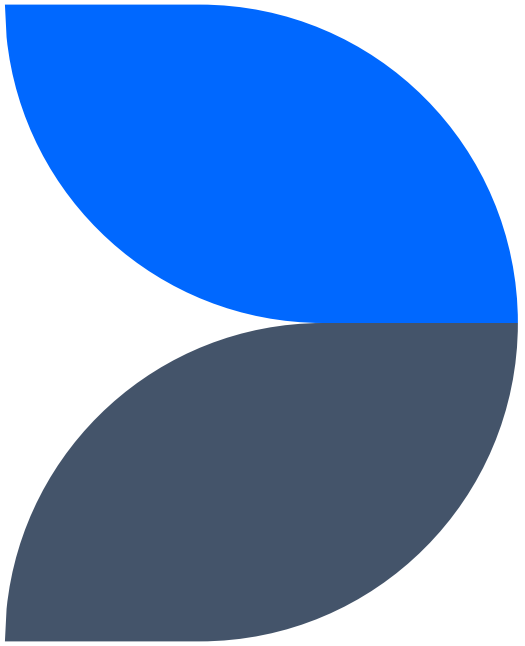
À partir d'une liste de N actions, le but est de trouver une combinaison d'actions dont le rapport entre le prix (W) et le profit (V) constituera la combinaison optimale permettant le profit maximum.

Les contraintes sont les suivantes :

- L'investissement total ne peut dépasser 500 euros
- Chaque action ne peut être achetée qu'une seule fois.
- Il n'est pas possible d'acheter une fraction d'action.

Algorithme de force brute

Avec une première liste réduite de 20 actions, il est possible de tester l'ensemble des combinaisons possibles et de ne conserver que celle qui permettra le profit le plus élevé tout en remplissant les conditions imposées.



Analyse de l’algorithme de force brute

1. Récupération des données à analyser dans un fichier csv.

```
csv_data = "actions.csv"
actions = []

with open(csv_data, mode='r') as file:
    reader = csv.DictReader(file)
    for row in reader:
        price = float(row["price"])
        if price > 0:
            actions.append({
                "name": row["name"],
                "price": float(row["price"]),
                "profit": float(row["profit"]),
            })
```

2. Pour chaque combinaison, on calculera avec cette fonction le coût total et le profit total de la combinaison.

```
def calculate_profit(combination):
    total_cost = sum(action["price"] for action in combination)
    total_profit = sum(action["price"] * action["profit"] / 100 for action in combination)
    return total_cost, total_profit
```

3. En utilisant le module itertools, on répertorie l'ensemble des combinaisons possibles. Pour chaque combinaison, la fonction calculate_profit permet d'en déterminer le coût et le profit.

Si une combinaison ne dépasse pas le budget de 500 euros et a un profit supérieur à toutes les autres alors elle est retenue comme étant la meilleure combinaison.

```
def algo_bruteforce(actions, budget):
    best_profit = 0
    for r in range(1, len(actions) + 1):
        for combination in itertools.combinations(actions, r):
            actual_cost, actual_profit = calculate_profit(combination)

            if actual_cost ⩽ budget and actual_profit > best_profit:
                best_combination = combination
                best_profit = actual_profit
```

Pseudocode de l’algorithme optimisé

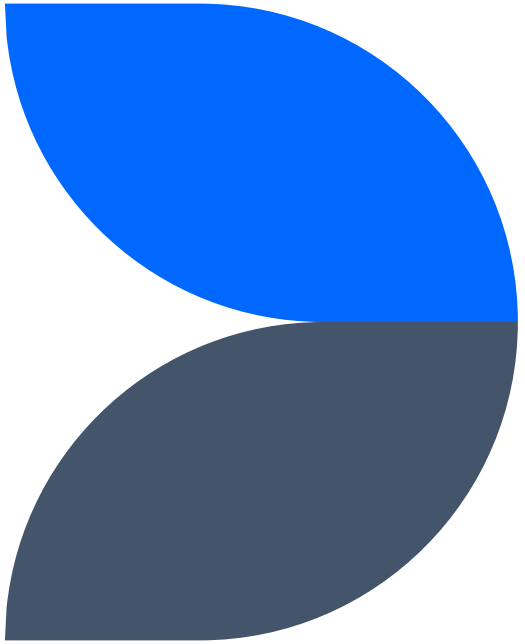
```
Boucle pour chaque action :
    Valeur d’une action = profit/prix
Trier les actions par ordre de Valeur

Fonction algo_optimized:
    Budget = 500
    Budget restant = 500
    Budget dépensé= 500-budget restant
    Profit = 0
    Initialisation liste meilleure combinaison d’actions
    Boucle pour chaque action par ordre de valeur :
        Si prix + budget dépensé <= budget:
            Ajouter le prix à budget dépensé
            Ajouter le profit à la variable profit
            Ajouter le nom à la liste
```

Présentation de l’algorithme glouton

L’algorithme de force brute a pour principale faiblesse un temps de calcul élevé, chaque combinaison possible devant être testée. Pour de grandes listes de combinaisons d’autres algorithmes existent afin de réduire considérablement le temps de calcul, c’est le cas de l’algorithme glouton.

Ce dernier consiste à trier les éléments selon le meilleur rapport profit/prix, puis à ajouter chaque élément à la liste des actions de la meilleur combinaison possible jusqu’à ce que la limite des 500 euros soit atteinte.

Inconvénient : la solution procurée par un algorithme glouton n’est pas systématiquement la meilleure. En effet, les éléments étant sélectionnés dans l’ordre, toutes les combinaisons ne sont donc pas testées. La solution retenue demeure néanmoins proche du meilleur résultat possible.



Comparaison de l'efficacité et des performances de l'algorithme de force brute par rapport à l'algorithme optimisé

Algorithme de force brute

Bénéfice obtenu : 227 euros
Temps de calcul : 1,424 seconde
Complexité temporelle : O(2^n)
Analyse de la mémoire : O(2^n)

Algorithme glouton

Bénéfice obtenu : 227 euros
Temps de calcul : 0,001 seconde
Complexité temporelle : O(n log n)
Analyse de la mémoire : O(n)

Comparaison des résultats

Dataset 1	Sienna	Algorithme glouton
Coût total	498,76	499,94
Profit total	196,61	2387,63

Dataset 2	Sienna	Algorithme glouton
Coût total	489,24	499,88
Profit total	193,78	1609,68

Conclusion

- L’algorithme glouton donne de meilleurs résultats que ceux de Sienna.

- L’algorithme de force brute est plus précis que l’algorithme glouton mais inutilisable pour de grandes bases de données. L’algorithme glouton est donc meilleur dans cette situation.

Merci

Fabien Royer