Résoudre des problématiques à l'aide d'algorithmes - python

Soutenance du 24/06/2022

Léo Garrigoux

Addi Ait-Mlouk

Analyse de problématique

Dataset de type Action, Coût, Profit

- Import des données
- Formatage des données
- Traitement des données
- Analyse algorithmique
- Résultat

Analyse de performance

- Comparaison avec investissement passés
- Temps d'exécution
- Rentabilité du résultat
- Exploitation du budget
- Densité du code

Première approche : Brute force

- Cherche à évaluer toutes les possibilités (exhaustif)
- Pas de recherche d’efficience (rapport temps/résultat)
- Approche directe et brutale
- Rapide à mettre en place

Brute Force : Pseudo code

Import des données

Extraction des données

Définition d’une action (nom, prix, revenu, gain)

Construction d’une liste d’actions

Définition de toutes les combinaisons possibles d’actions

Détermination de la rentabilité de chaque combinaison

Choisis et renvoie la meilleure combinaison avec : Coût, Gain, Temps d'exécution

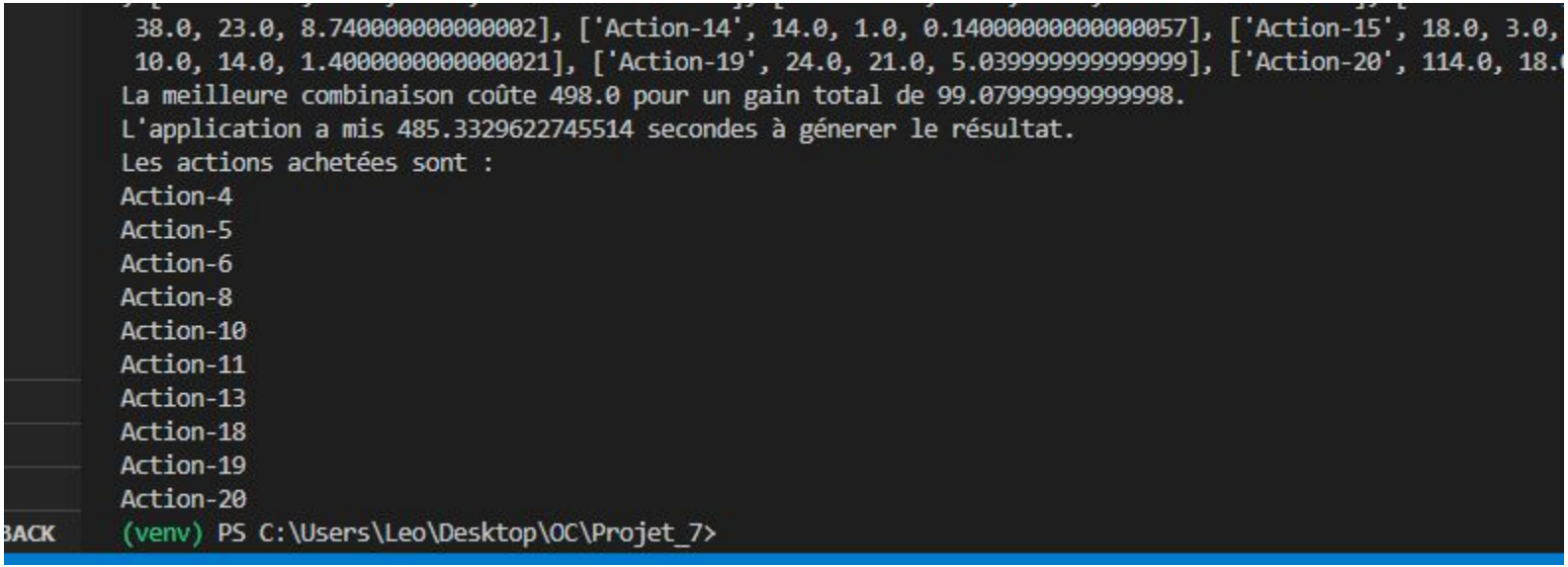
Brute Force

```
7   # Importing and formating CSV datas to a list
8   data = pd.read_csv("Actions.csv")
9   actions = data.name
10  initial_price = data.price
11  sell_price = data.profit
12
13  # Analyzing the actions to set a list of best return on investment
14  actions_revenus = []
15  for i in range(len(actions)):
16      action = actions[i]
17      price = float(initial_price[i])
18      revenu = float(sell_price[i])
19      final_price = price * (1 + (revenu / 100))
20      gain = final_price - price
21      if price > 0:
22          action_summary = [action, price, revenu, gain]
23          actions_revenus.append(action_summary)
```

Brute Force

```
24
25  # Brute Force Algorithm
26  buyable_combination = []
27  gain = 0
28  cost = 0
29  for i in range(len(actions_revenus)):
30      combinations = list(itertools.combinations(actions_revenus, i))
31      for j in range(len(combinations)):
32          combination_test = combinations[j]
33          actions_prices = [action[1] for action in combination_test]
34          if sum(actions_prices) < 500:
35              combination_gain = sum([action[3] for action in combination_test])
36              if combination_gain > gain:
37                  cost = sum(actions_prices)
38                  gain = combination_gain
39                  best_combination = combination_test
40
```

Brute Force


38.0, 23.0, 8.740000000000002], ['Action-14', 14.0, 1.0, 0.14000000000000057], ['Action-15', 18.0, 3.0,
10.0, 14.0, 1.4000000000000021], ['Action-19', 24.0, 21.0, 5.039999999999999], ['Action-20', 114.0, 18.
La meilleure combinaison coûte 498.0 pour un gain total de 99.07999999999998.
L'application a mis 485.3329622745514 secondes à génerer le résultat.
Les actions achetées sont :
Action-4
Action-5
Action-6
Action-8
Action-10
Action-11
Action-13
Action-18
Action-19
Action-20
(venv) PS C:\Users\Leo\Desktop\OC\Projet_7>

Seconde approche : Optimisée

- Demande d’avantage d’anticipation : Analyse du problème plus exhaustive
- Doit restreindre le champs d’analyse nécessaire au programme
- Réduction du nombre d’opération (notion big O : avoir O(n) avec plus petit n possible
- Doit être efficiente
- Doit simplifier le travail de l’ordinateur

Optimisation : Pseudo code

Import des données

Extraction des données

Définition d'une action (nom, prix, revenu, gain)

Construction d'une liste d'actions **triée sur le revenu de chaque action**

Achat d'action **dans l'ordre de la liste (donc de rentabilité)**, ce dans la limite du budget

Stockage des actions dans une liste

Renvoie la liste d'actions achetées, le coût total, le bénéfice total et le temps d'execution

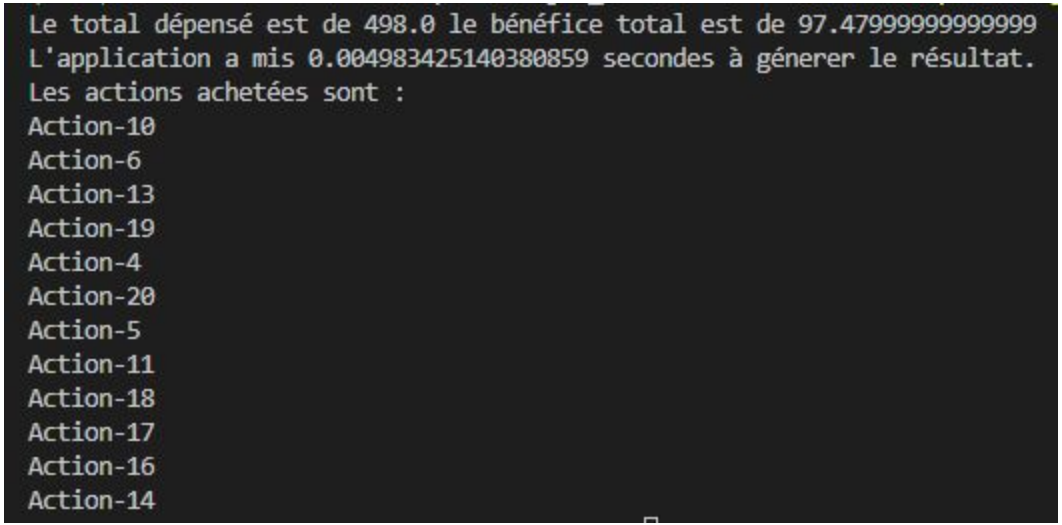
Approche Optimisée

```python

# Importing and formating CSV datas to a list
data = pd.read_csv("Actions.csv")
actions = data.name
initial_price = data.price
sell_price = data.profit


# Data preprocessing prior to analysis.
actions_revenus = []
for i in range(len(actions)):
    action = actions[i]
    price = float(initial_price[i])
    revenu = float(sell_price[i])
    final_price = price * (1 + (revenu / 100))
    gain = final_price - price
    if price > 0:
        action_dictionary = {
            "action": action,
            "price": price,
            "revenu": revenu,
            "gain": gain,
        }
        actions_revenus.append(action_dictionary)
sorted_actions = sorted(actions_revenus, key=itemgetter("revenu"), reverse=True)

```

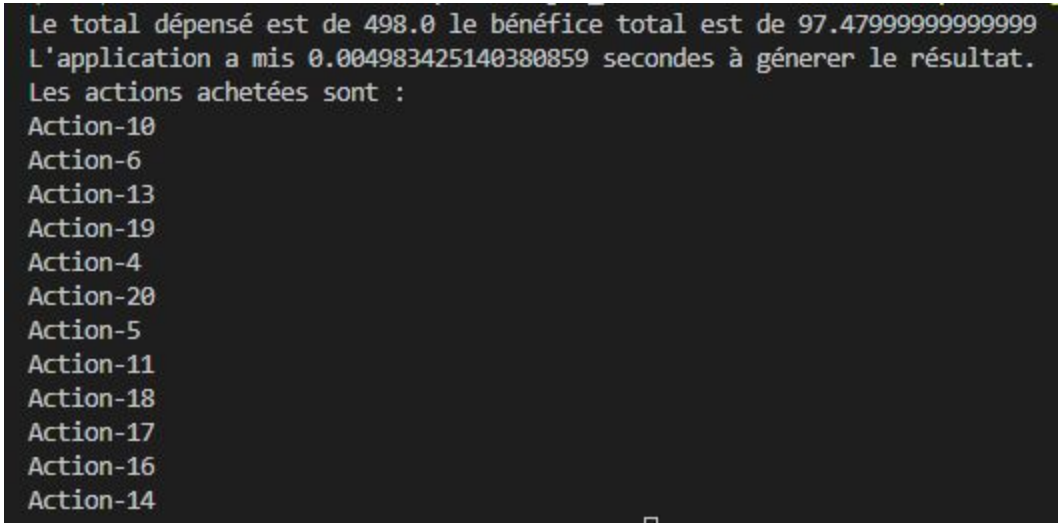
Approche Optimisée

```python

# Work on the sorted list in order to deduce the best actions.
def getActions(List, budget):
    total_spent = 0
    action_to_buy = []
    total_gain = 0
    for i in range(len(List)):
        action = List[i]
        price = action["price"]
        new_total = total_spent + price
        if new_total < budget:
            gain = action["gain"]
            total_gain += gain
            action_to_buy.append(action)
            total_spent += price
    print(
        f"Le total dépensé est de {total_spent} le bénéfice total est de {total_gain}"
    )
    return action_to_buy


```

Approche Optimisée



Approche Optimisée



Comparaison

Brute Force

- Brute plus complète car approche global : Toutes possibilités
- rapport Temps/efficience mauvais : O(n) avec n élevé

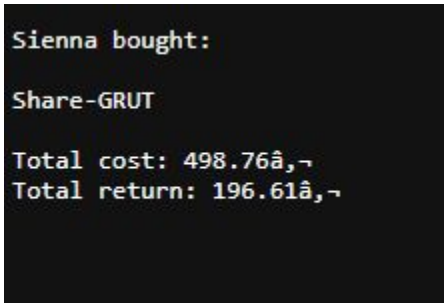
Optimisée

- Extrêmement proche : statistiquement non significatif
- Rapport temps/efficience excellent : O(n) faible
- Permet l’analyse de gros sets de données : Échelle de temps viable

Approche Optimisée: Dataset 1

```
Le total dépensé est de 499.94000000000005 le bénéfice total est de 198.507805
L'application a mis 0.02156829833984375 secondes à génerer le résultat.
Les actions achetées sont :
Share-XJMO
Share-MTLR
Share-KMTG
Share-LRBZ
Share-GTQK
Share-WPLI
Share-GIAJ
Share-GHIZ
Share-IFCP
Share-ZSDE
Share-FKJW
Share-NHWA
Share-LPDM
Share-QQTU
Share-USSR
Share-EMOV
Share-LGWG
Share-SKKC
Share-QLMK
Share-UEZB
Share-CBNY
Share-CGJM
Share-EVUW
Share-FHZN
Share-MLGM
```

Approche Optimisée: Dataset 1



Résultats meilleurs avec l’algorithme.

- Analyse plus profonde des données
- Meilleure exploitation du budget
- Diversification de l’investissement
- Sienna a acheté l’action avec le plus gros bénéfice et non les actions les plus rentables

Approche Optimisée: Dataset 2

```
Le total dépensé est de 499.9800000000001 le bénéfice total est de 197.76834499999998
L'application a mis 0.01595783233642578 secondes à génerer le résultat.
Les actions achetées sont :
Share-PATS
Share-JWGF
Share-ALIY
Share-NDKR
Share-PLLK
Share-FWBE
Share-LFXB
Share-ZOFA
Share-ANFX
Share-FAPS
Share-LXZU
Share-XQII
Share-ECAQ
Share-JGTW
Share-IXCI
Share-DWSK
Share-ROOM
Share-VCXT
Share-YFVZ
Share-OCKK
Share-JMLZ
Share-DYVD
```

Approche Optimisée: Dataset 2

```
Sienna bought:
Share-ECAQ 3166
Share-IXCI 2632
Share-FWBE 1830
Share-ZOFA 2532
Share-PLLK 1994
Share-YFVZ 2255
Share-ANFX 3854
Share-PATS 2770
Share-NDKR 3306
Share-ALIY 2908
Share-JWGF 4869
Share-JGTW 3529
Share-FAPS 3257
Share-VCAX 2742
Share-LFXB 1483
Share-DWSK 2949
Share-XQII 1342
Share-ROOM 1506

Total cost: 489.24â,¬
Profit: 193.78â,¬
```

Résultats meilleurs avec l'algorithme.

- Analyse plus profonde des données
- Meilleure exploitation du budget
- Sienna a acheté les actions avec les plus gros bénéfices et non les actions les plus rentables

Conclusion

- Algorithme optimisé à utiliser : O(n) excellent donc utilisable sur gros dataset
- Résultats meilleurs car analyse plus profonde, plus exhaustive
- Tri sur le retour sur investissement pour gagner de l’efficience
- Brute force exhaustive mais pas réaliste pour gros datasets