Algorithme d’optimisation des bénéfices sur achat d’actions

Objectif de l’algorithme

- ➔ Sélectionner la combinaison d’actions la plus rentable après 2 ans.
- ➔ Contraintes :
  - ◆ Budget maximal de 500€.
  - ◆ Une seule unité de chaque action.
  - ◆ Pas d’achat fractionné.
- ➔ Approche brute-force : tester toutes les combinaisons possibles pour trouver la meilleure.

Fonctionnement de l’algorithme

- ➔ Lire toutes les actions du fichier CSV.
- ➔ Générer toutes les combinaisons possibles (avec itertools.combinations).
  - ◆ Il y a $2^n - 1$ combinaisons pour n actions.
- ➔ Pour chaque combinaison :
  - ◆ Calculer son coût total.
  - ◆ Si le coût est ≤ 500€, calculer son profit total.
- ➔ Sélectionner la meilleure combinaison valide en maximisant le profit.

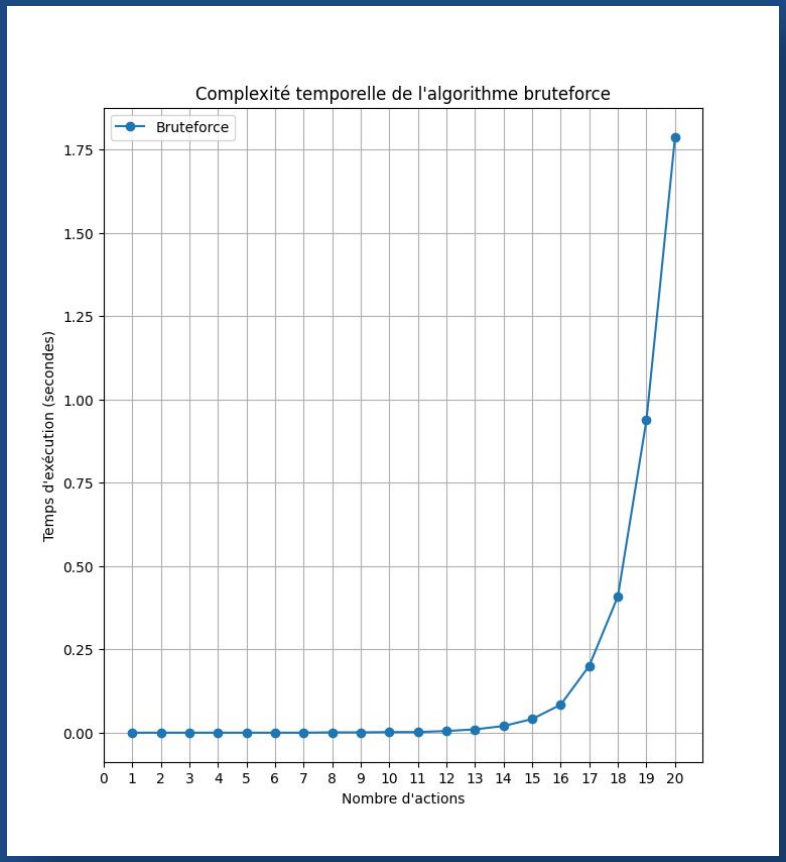
Complexités de l'algorithme bruteforce

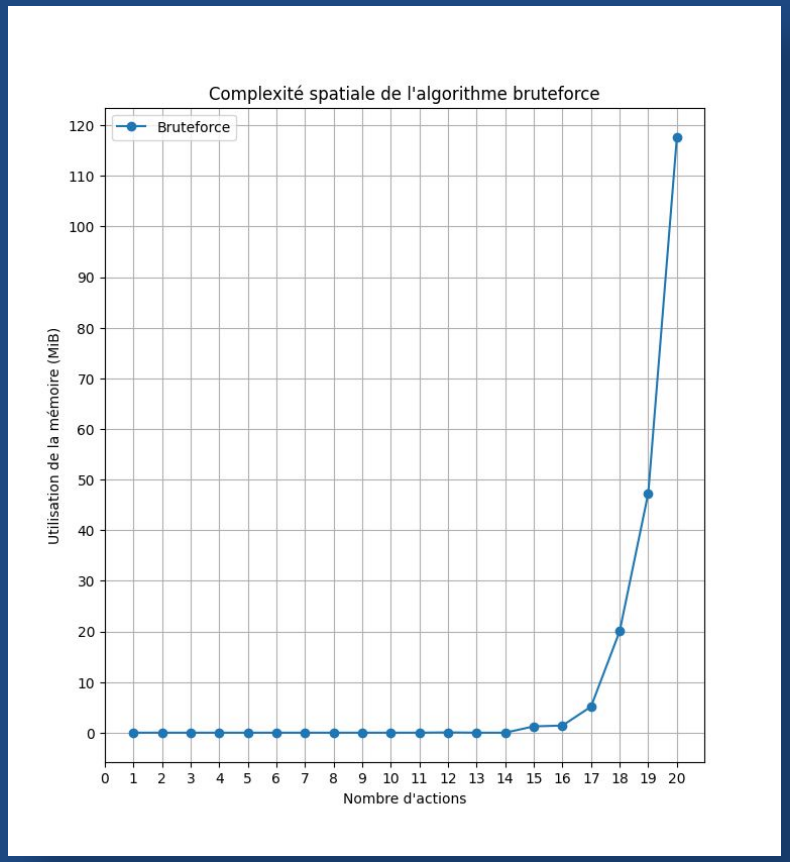
- ➔ Complexité temporelle :
  - ◆ Génération de toutes les combinaisons : $O(2^n)$
  - ◆ Calcul du coût et profit pour chaque combinaison : $O(n)$

    ⇒ Complexité globale : $O(n \cdot 2^n)$

- ➔ Complexité spatiale :
  - ◆ Stockage de toutes les combinaisons : $O(2^n)$
  - ◆ Peut entraîner une consommation mémoire élevée dès que $n > 20$.

Complexités de l’algorithme bruteforce





Limites

- ➔ Dès que n > 20, le nombre de combinaisons devient énorme
  - ◆ 2^20 ≈1million
  - ◆ 2^30 ≈1milliard
- ➔ Inefficace pour de très gros ensembles de données.
- ➔ Pas de sélection intelligente : toutes les combinaisons sont évaluées, même celles très improbables.

Objectif de l’optimisation

- ➔ Réduire le temps :
  - ◆ Ne pas calculer toutes les combinaisons
- ➔ Réduire la mémoire :
  - ◆ Ne pas stocker toutes les combinaisons
- ➔ Garder un résultat optimal :
  - ◆ Ne pas sacrifier la qualité des résultats

Algorithme choisi
Programmation dynamique (sac à dos 0/1)

- ➔ Construction d'une table de sous-problèmes, où chaque case représente le meilleur profit possible avec un certain budget et un sous-ensemble d'actions.
- ➔ On évite de recalculer les mêmes situations → gains en temps et en mémoire.

- ➔ Réduire le temps :
  - ◆ On évite les doublons de calculs
- ➔ Réduire la mémoire :
  - ◆ Pas de stockage de toutes les combinaisons
- ➔ Garder un résultat optimal :
  - ◆ Trouve toujours la solution optimale

Pseudocode

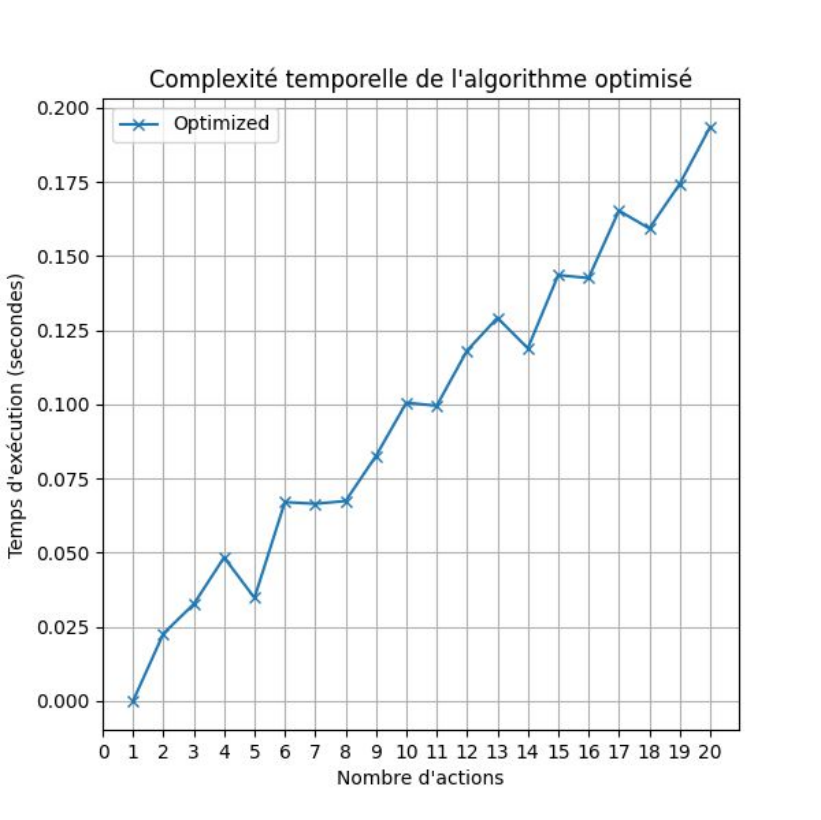
```
fonction knapsack(actions, max_cost):
    filtrer les actions avec coût > 0 et ≤ max_cost
    convertir tous les coûts en entiers (ex: *100)
    initialiser table[n+1][max_cost+1] avec des 0

    pour i de 1 à n:
        pour w de 1 à max_cost:
            si action_i peut être ajoutée au sac (coût_i ≤ w):
                table[i][w] = max(
                    table[i-1][w],                    // on ne prend pas l'action
                    profit_i + table[i-1][w - coût_i]    // on la prend
                )
            sinon:
                table[i][w] = table[i-1][w]

    reconstruire la solution (traceback) à partir de table[n][max_cost]
    retourner la combinaison optimale
```

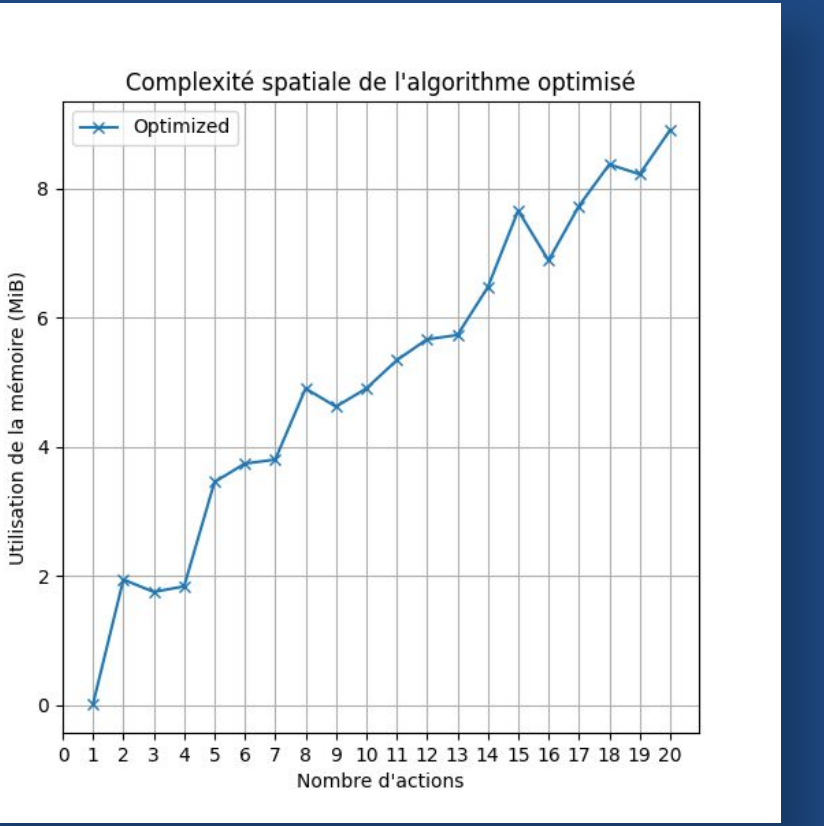
Complexités de l'algorithme optimisé

- Complexité temporelle :
  - Remplissage de la table : O(n × C)
    - (n = nombre d'actions, C = capacité max en centimes)
  - Traceback pour reconstituer la solution : O(n)
  - Complexité globale : O(n × C)

- Complexité spatiale :
  - Taille de la table : O(n × C)
  - La mémoire augmente proportionnellement au budget et à la précision choisie
  - Peut devenir lourde pour de très grands budgets.

Complexités de l’algorithme optimisé

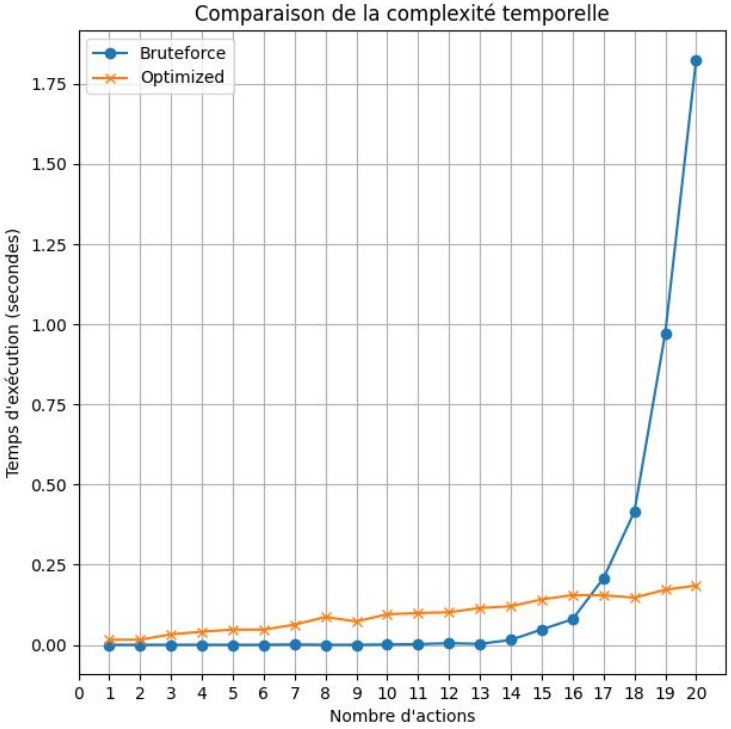


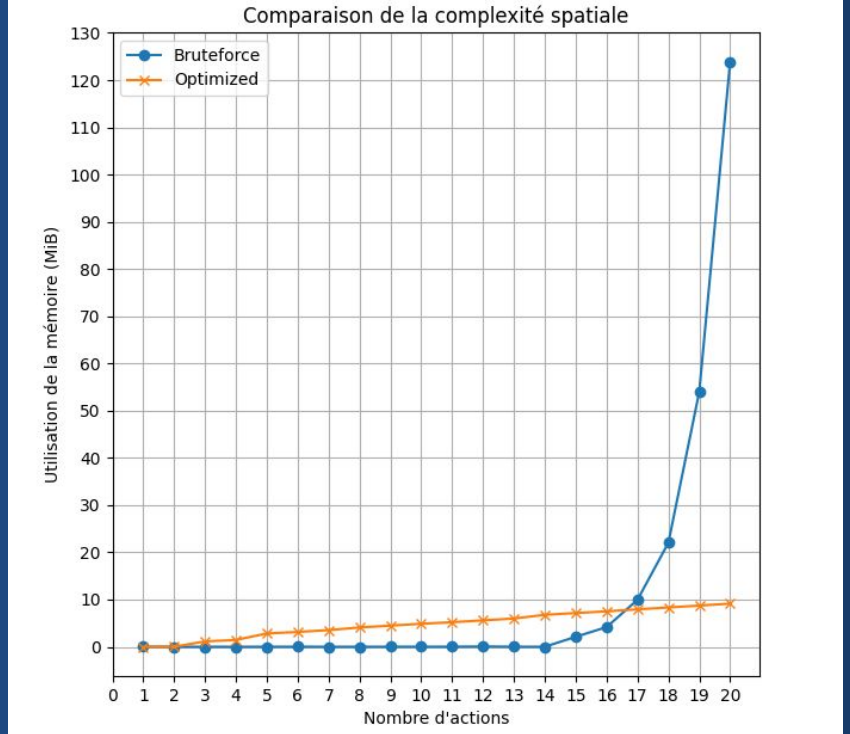


Limites

- ➔ La taille de la table dépend à la fois :
  - ◆ Du nombre d'actions (n)
  - ◆ De la capacité budgétaire exprimée en entiers ©
    - Ex. : 500€ en centimes → 50 000 colonnes
- ➔ La mémoire peut devenir problématique si :
  - ◆ Le budget est très élevé
  - ◆ La précision devient trop fine (ex. x10 000)
- ➔ Moins efficace si la majorité des actions ont des ratios coût/profit très similaires
  - ◆ La table sera remplie sans que beaucoup de cas soient élagués

Comparaison des complexités





Backtesting

Dataset_1

Critère	Résultats Sienna	Résultats optimized
Coût (€)	498.76	499.94
Profit (€)	196.61	198.54
Profit (%)	39.42%	39,71%
Nb d’actions	1	21

Backtesting

Dataset_2

Critère	Résultats Sienna	Résultats optimized
Coût (€)	489.24	499.90
Profit (€)	193.78	197.96
Profit (%)	39.61%	39,60%
Nb d’actions	18	20

Rapports d’exploration de l’ensemble des données

```
Rapport de l'exploration du dataset dataset_1.csv:
Actions: 1001
Actions avec un coût nul: 43
Actions avec un coût négatif: 1
Actions avec un profit nul: 1
Actions avec un profit négatif: 0
--------------------------------
Actions inutilisables: 45 (4.50%)
```

```
Rapport de l'exploration du dataset dataset_2.csv:
Actions: 1000
Actions avec un coût nul: 219
Actions avec un coût négatif: 240
Actions avec un profit nul: 0
Actions avec un profit négatif: 0
--------------------------------
Actions inutilisables: 459 (45.90%)
```

Merci