AlgoInvest&Trade

Sommaire

- Avantages/inconvénients
- Explication du code
- Comparaison de l'algorithme brute force et optimisé

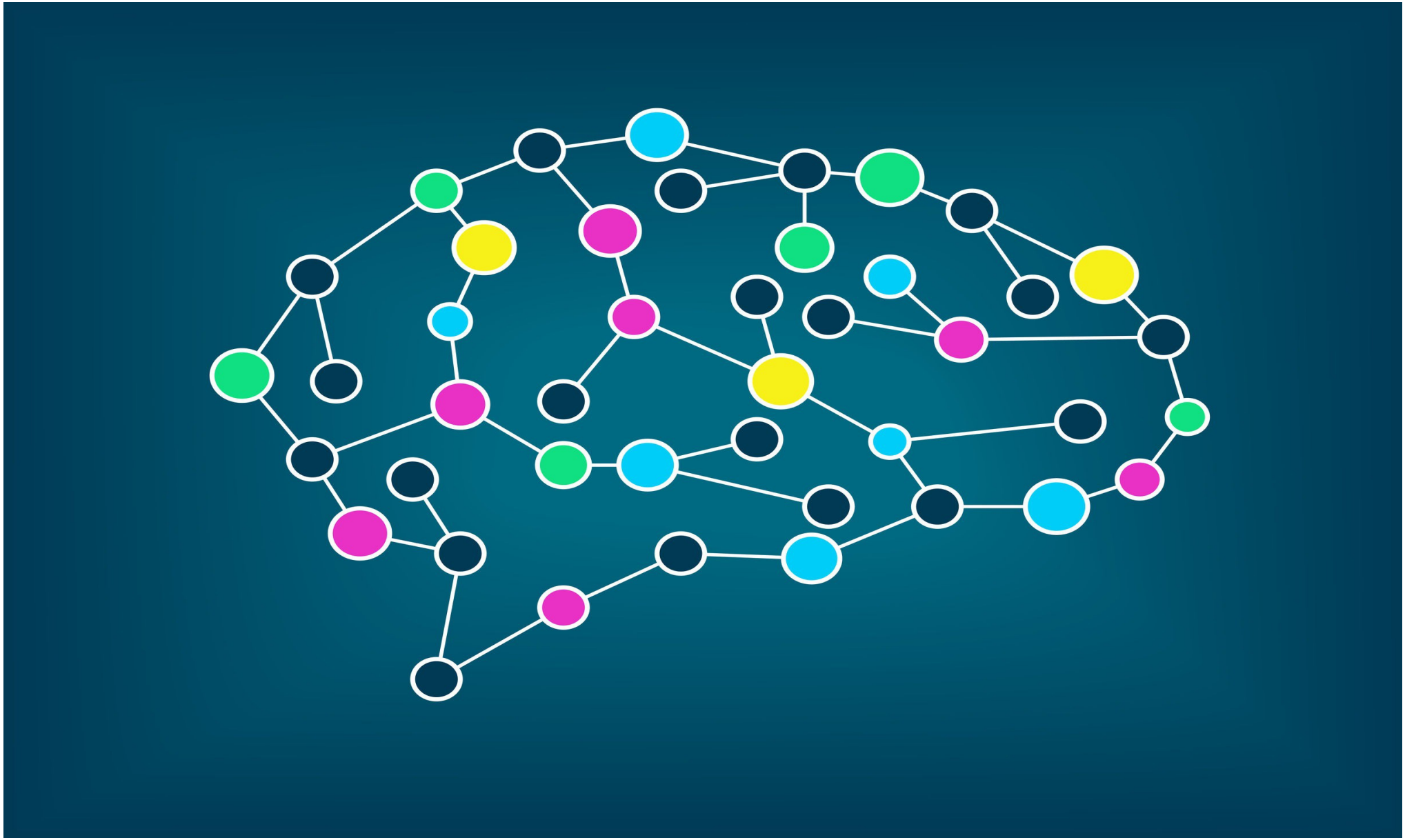
  > Big 0

  > Limites de l’algorithme

  > Performances, efficacité

- Comparaison avec l'algorithme de Sienna
- Conclusion

Algorithme Brute Force



Avantages

- Algorithme de force brute, essayant toutes les combinaisons possibles

- Très bon algorithme si peu de données

Inconvénients

- N'est pas utilisable pour un grand nombre d'actions (complexité : O(2^n))

Test de toutes les combinaisons possibles

Combinaisons avec 3 actions

action1	action2	action3
oui	non	oui
non	non	oui
oui	oui	oui
non	oui	oui
oui	non	non
non	non	non
oui	oui	non
non	oui	non

Toutes les combinaisons testées avec 20 actions

Nombre d'opérations a tester	
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768
16	65536
17	131072
18	262144
19	524288
20	1048576

Dépense maximum qui est définie à 500,
les actions sont représentées par les actions
et la liste des actions sélectionnées
représente les actions gardées par l'algorithme

```python
def bruteforce(depense_max, donnees, lst_actions_selectionees=[]):
    if donnees:
        val1, lst_val1 = bruteforce(depense_max, donnees[1:], lst_actions_selectionees)
        action_selection = donnees[0]
        if action_selection[1] <= depense_max:
            val2, lst_val2 = bruteforce(depense_max - action_selection[1], donnees[1:],
                                        lst_actions_selectionees + [action_selection])
            if val1 < val2:
                return val2, lst_val2
        return val1, lst_val1
```

On rappelle
récursivement la fonction
Brute force,

On enlève le montant
de l'action en cours
de la dépense max
et on ajoute cette action
dans la liste des actions

Résultat de la
meilleure rentabilité

```python
else:
    return f"la rentabilité maximum obtenue est : \
        {round(sum([i[1] * i[2] for i in lst_actions_selectionees]), 2)}", \
        f"La depense maximum est : {sum([i[1] for i in lst_actions_selectionees])} euros, " \
        f"avec ces actions: {[i[0] for i in lst_actions_selectionees]}"
```

Si toutes les actions ont été traitées,
alors on renvoie la dépense max
ainsi que la liste des actions sélectionnées

Algorithme optimisé

Avantages

- Efficacité
- Rapidité
- Permet de trouver une solution rapidement avec un grand nombre de données

Inconvénients

- La solution trouvée n'est pas la meilleure

Valeur de la dépense max

On reprend la liste des actions
ainsi que le budget max

La variable matrix représente une matrice
où le budget max est représenté par les colonnes
et les actions par les lignes

```python
MAX_VALUE_INVEST = 500


def dynamic_search(list_actions, max_value=MAX_VALUE_INVEST):
    matrix = [[0 for x in range(max_value + 1)] for x in range(len(list_actions) + 1)]
    for i in range(1, len(list_actions) + 1):
        for wallet in range(1, max_value + 1):
            if list_actions[i - 1][1] <= wallet:
                matrix[i][wallet] = max(list_actions[i - 1][2] + matrix[i - 1][int(wallet - list_actions[i - 1][1])],
                                        matrix[i - 1][wallet])
            else:
                matrix[i][wallet] = matrix[i - 1][wallet]
```

On créait deux boucles,
une sur la liste d'actions
et l'autre sur le budget max

Si la dépense est supérieure
à la dépense max définie
Alors on récupère la solution
de la ligne précédente

On choisit la meilleure valeur

Rechercher les actions
en fonction du résultats

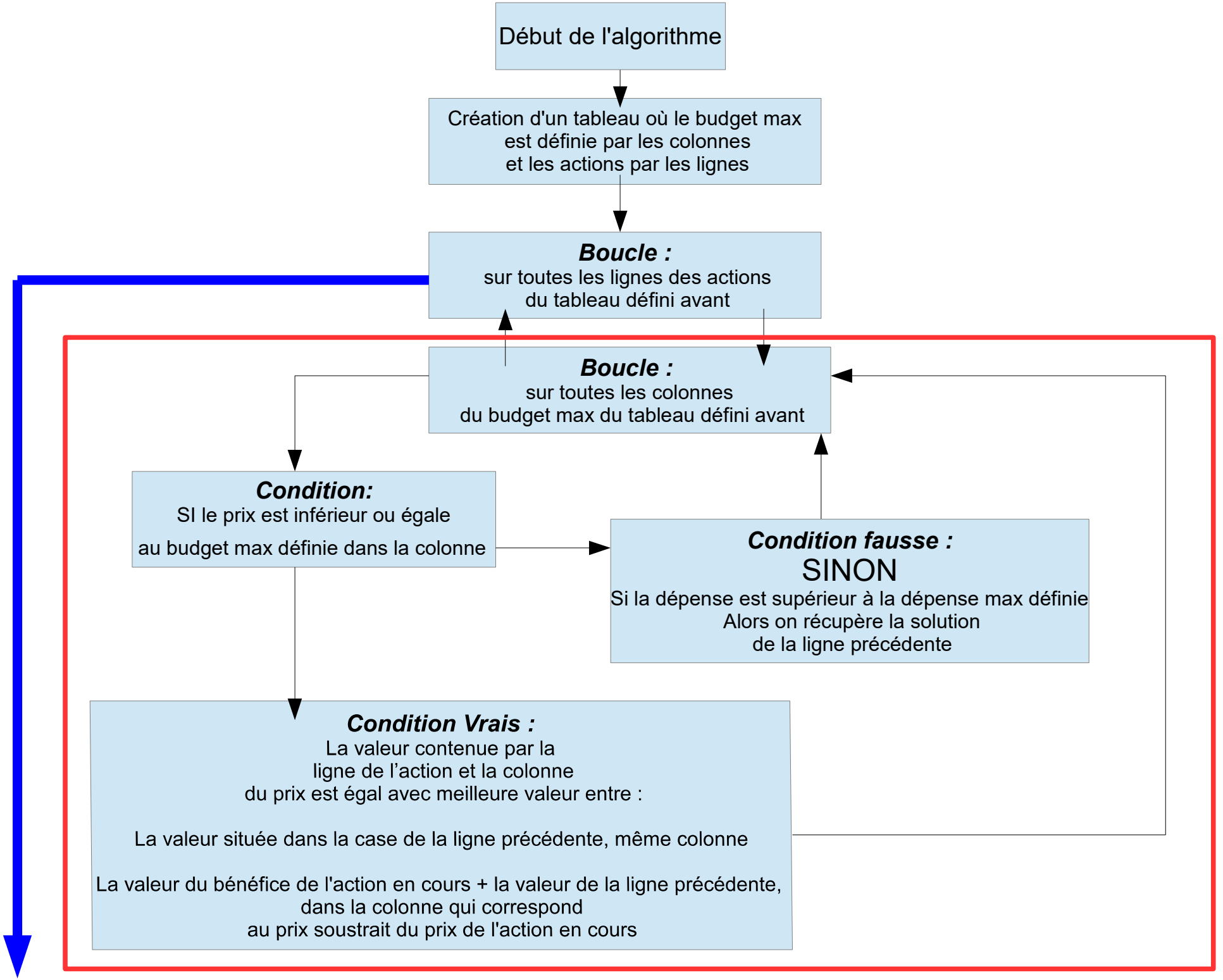
Tant que la dépense est supérieure à 0
et qu'il reste encore des actions

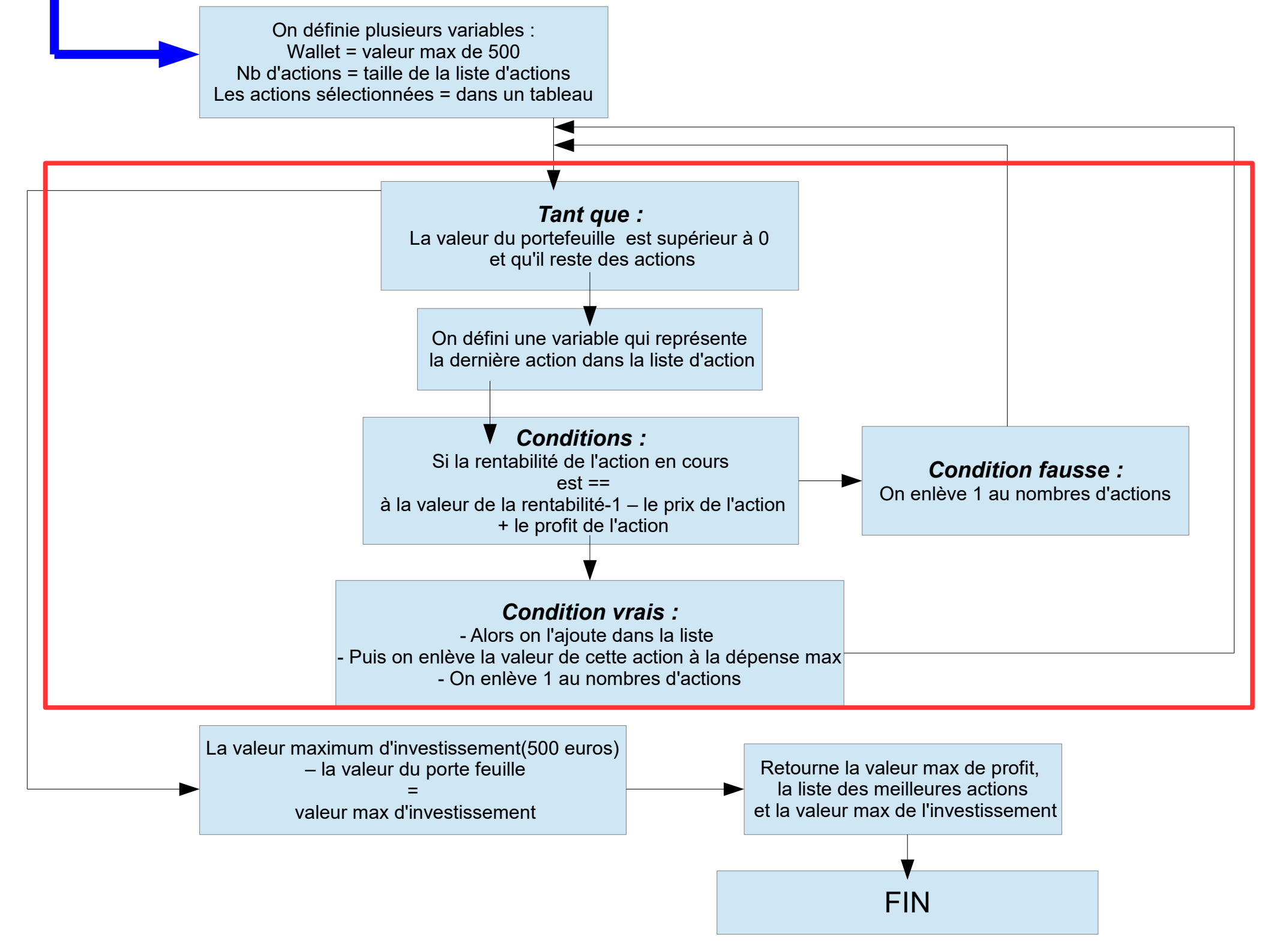
```
wallet = max_value
nb_actions = len(list_actions)
actions_selection = []
while wallet >= 0 and nb_actions >= 0:
    action = list_actions[nb_actions - 1]
    if matrix[nb_actions][int(wallet)] == matrix[nb_actions - 1][int(wallet - action[1])] + action[2]:
        actions_selection.append(action)
        wallet -= action[1]
    nb_actions -= 1
max_invest = MAX_VALUE_INVEST - wallet
return matrix[-1][-1], actions_selection, max_invest
```

Retourne la valeur max de profit,
la liste des meilleures actions
et la valeur max de l'investissement

Si la rentabilité de l'action en cours
est ==
à la valeur de la rentabilité-1 – le prix de l'action
+ le profit de l'action

- Alors on l'ajoute dans la liste
- Puis on enlève la valeur de cette action à la dépense max
- On enlève 1 au nombres d'actions

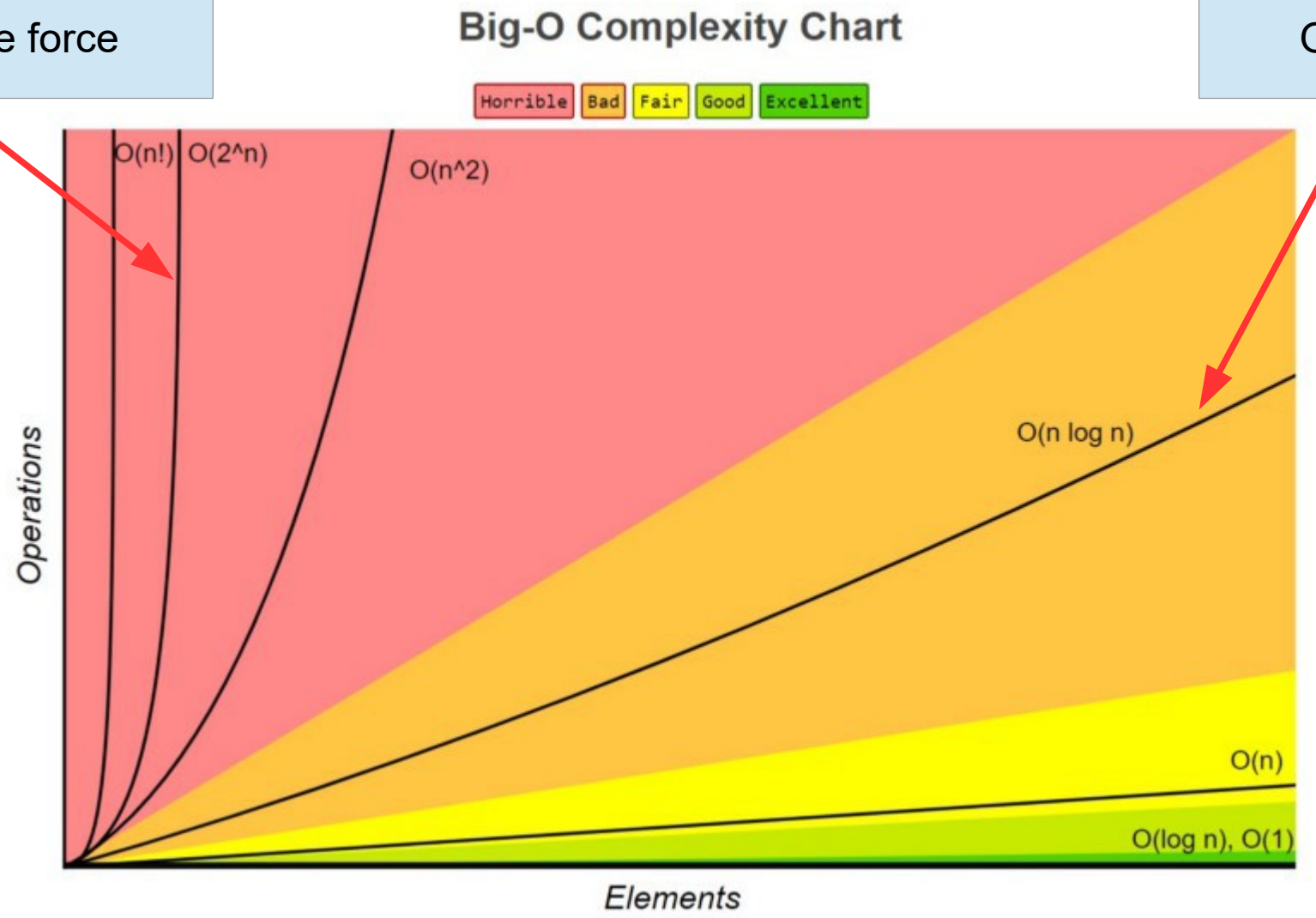

Début de l'algorithme
Création d'un tableau où le budget max
est définie par les colonnes
et les actions par les lignes
Boucle :
sur toutes les lignes des actions
du tableau défini avant
Boucle :
sur toutes les colonnes
du budget max du tableau défini avant
Condition:
SI le prix est inférieur ou égale
au budget max définie dans la colonne
Condition fausse :
SINON
Si la dépense est supérieur à la dépense max définie
Alors on récupère la solution
de la ligne précédente
Condition Vrais :
La valeur contenue par la
ligne de l'action et la colonne
du prix est égal avec meilleure valeur entre :
La valeur située dans la case de la ligne précédente, même colonne
La valeur du bénéfice de l'action en cours + la valeur de la ligne précédente,
dans la colonne qui correspond
au prix soustrait du prix de l'action en cours


On définie plusieurs variables :
Wallet = valeur max de 500
Nb d'actions = taille de la liste d'actions
Les actions sélectionnées = dans un tableau
Tant que :
La valeur du portefeuille est supérieur à 0
et qu'il reste des actions
On défini une variable qui représente
la dernière action dans la liste d'action
Conditions :
Si la rentabilité de l'action en cours
est ==
à la valeur de la rentabilité-1 – le prix de l'action
+ le profit de l'action
Condition fausse :
On enlève 1 au nombres d'actions
Condition vrais :
- Alors on l'ajoute dans la liste
- Puis on enlève la valeur de cette action à la dépense max
- On enlève 1 au nombres d'actions
La valeur maximum d'investissement(500 euros)
– la valeur du porte feuille
=
valeur max d'investissement
Retourne la valeur max de profit,
la liste des meilleures actions
et la valeur max de l'investissement
FIN

Big 0 :

Brute force

Optimisé



Les limites de l'algorithme optimisé

Comparaison :

- **<u>20 actions :</u>**

Brute force : Optimisé :

- Durée :
- Nb d'actions :
- Bénéfices :
- Dépenses max :

Comparaison :

- **<u>1000 actions(dataset1 de Sienna) :</u>**

Brute force : Optimisé : Sienna :

Comparaison :

- **<u>1000 actions(dataset2 de Sienna) :</u>**

Brute force : Optimisé : Sienna :

Conclusion

- Brute force :

  A utiliser si peu de données

- Optimisé :