Spécialiste de l'investissement

Algo Invest & Trade

Société financière spécialisée dans l'investissement

Accueil

Par Laurent Jouron

Spécialiste de l'investissement

Lecture fichier .CSV

Le format .csv (comma separated values) est un fichier dont les éléments sont séparés par des virgules.
On doit, dans un premier temps, transformer ces données en une liste.

Calcul de la performance

J'ajoute le calcul de la performance de chaque élément. Ce sont ces performances qui indiquent la meilleure rentabilité des actions.

Le fichier indique le coût de chaque action et son pourcentage de profit, il suffit de faire le calcul :pourcentage / 100 * coûts.

Lecture

Tri sur les performances

Pour avoir les actions les plus rentables en premier, il faut trier les performances de façon décroissante.

Spécialiste de l'investissement

Résultat

```
Action,Cost,Profit
Action-1,20.0,5.0
Action-2,30.0,10.0
Action-3,50.0,15.0
Action-4,70.0,20.0
Action-5,60.0,17.0
Action-6,80.0,25.0
Action-7,22.0,7.0
Action-8,26.0,11.0
Action-9,48.0,13.0
Action-10,34.0,27.0
Action-11,42.0,17.0
Action-12,110.0,9.0
Action-13,38.0,23.0
Action-14,14.0,1.0
Action-15,18.0,3.0
Action-16,8.0,8.0
Action-17,4.0,12.0
Action-18,10.0,14.0
Action-19,24.0,21.0
Action-20,114.0,18.0
```

Action	Coût	%
Action-1	20	5
Action-2	30	10
Action-3	50	15
Action-4	70	20
Action-5	60	17
Action-6	80	25
Action-7	22	7
Action-8	26	11
Action-9	48	13
Action-10	34	27
Action-11	42	17
Action-12	110	9
Action-13	38	23
Action-14	14	1
Action-15	18	3
Action-16	8	8
Action-17	4	12
Action-18	10	14
Action-19	24	21
Action-20	114	18

Action	Coût	%	Profit
Action-20	114	18	20,52
Action-6	80	25	20
Action-4	70	20	14
Action-5	60	17	10,2
Action-12	110	9	9,9
Action-10	34	27	9,18
Action-13	38	23	8,74
Action-3	50	15	7,5
Action-11	42	17	7,14
Action-9	48	13	6,24
Action-19	24	21	5,04
Action-2	30	10	3
Action-8	26	11	2,86
Action-7	22	7	1,54
Action-18	10	14	1,4
Action-1	20	5	1
Action-16	8	8	0,64
Action-15	18	3	0,54
Action-17	4	12	0,48
Action-14	14	1	0,14

Par Laurent Jouron

Algorithme Brute force

Spécialiste de l'investissement

C’est un algorithme qui parcourt l’ensemble d’une liste d’actions et qui teste toutes les possibilités afin de définir la valeur de rendement maximal d’une liste finale.

- Chaque action a la possibilité de figurer dans la liste, cependant elle ne doit l’intégrer qu’une seule fois.

- Le coût de chaque liste ne peut pas dépasser la somme de contrainte (500€ pour l’exemple).

- Pour chaque liste d’action, on compare le rendement de toutes les actions.

- Le résultat de la comparaison permet de déterminer si chaque action doit figurer, ou non, dans la liste optimale.

Complexité

Cette solution parcourt plusieurs fois certaines combinaisons, même si elles ne sont pas optimales.

Par Laurent Jouron

Code Brute force

Spécialiste de l'investissement

Code

```
def brute_force(cost_invest, actions, final_list=None):
        """
            Create a lists that are maximum investment sum.
            :param:
                int: cost_invest (maximum cost the client wants to invest)
                list: action_list (name, cost, profit, performance)
                list: final_list free
            :return:
                list:  best actions lists
        """
    if final_list is None:
        final_list = []
    if not actions:
        return sum(i[1] for i in final_list), final_list
    cost_invest1, cost_list1 = brute_force(cost_invest, actions[1:], final_list)
    cost = actions[0]
    if cost[1] <= cost_invest:
        cost_invest2, cost_list2 = brute_force(cost_invest - cost[1],actions[1:], final_list + [cost])
        if cost_invest1 < cost_invest2:
            return cost_invest2, cost_list2
    return cost_invest1, cost_list1
```

Par Laurent Jouron

Spécialiste de l'investissement

Complexité temporelle

Il suffit d'ajouter une seule action pour que le nombre de calcul double systématiquement.

Cet algorithme de complexité temporelle ***O*(2ⁿ)**, est une démonstration mathématique par recurrence.

Action	Action-1	Action-2	Action-3	Action-4
Boucle-0	0	0	0	0
Boucle-1	1	0	0	0
Boucle-2	0	1	0	0
Boucle-3	1	1	0	0
Boucle-4	0	1	1	0
Boucle-5	0	0	1	0
Boucle-6	1	0	1	0
Boucle-7	1	1	1	0
Boucle-8	0	0	0	1
Boucle-9	1	0	0	1
Boucle-10	0	1	0	1
Boucle-11	1	1	0	1
Boucle-12	0	1	1	1
Boucle-13	0	0	1	1
Boucle-14	1	0	1	1
Boucle-15	1	1	1	1

Complexité

Par Laurent Jouron

Spécialiste de l'investissement

Fichier action.csv

Avec cet algorithme, sur le fichier de test, j'obtiens le résultat suivant:

Total cost: 500€
Total return: 64,32€

Action	Coût	%	Profit
Action-5	60	17	10,2
Action-12	110	9	9,9
Action-13	38	23	8,74
Action-3	50	15	7,5
Action-11	42	17	7,14
Action-9	48	13	6,24
Action-2	30	10	3,0
Action-8	26	11	2,86
Action-7	22	7	1,54
Action-18	10	14	1,4
Action-1	20	5	1,0
Action-16	8	8	0,64
Action-15	18	3	0,54
Action-17	4	12	0,48
Action-14	14	1	0,14

Résultats

Résultats algorithme Brute force

Spécialiste de l'investissement

Avec l'algorithme brute force, je n'obtiens pas de résultat sur les fichiers suivants:

- dataset1_Python+P7.csv
- dataset2_Python+P7.csv

Le côté exponentiel fait que je n'ai pas de résultat même au-delà d'une minute d'attente.

L'affichage des résultats devent être inférieur à 1 seconde, peu importe le nombre d'actions, cette solution ne peut être considérée comme valable.

Cependant, elle resterait sûre et rapide avec un nombre d'actions très restreint, mais pas avec 1000 actions comme sur les fichiers proposés.

Résultats

Spécialiste de l'investissement

Code naif

Le vrai Brute force ne solutionnant pas le besoin du client, je vous propose le code suivant. Il répond à l'attente mais de façon moins précise, mais de complexité moins gourmande ***O*(n).**

Code

```
def naif(actions, final_list=None, i: int = 0):
        """
         Create a lists that are optimized on the maximum investment sum.
           :param:
              list: action_list (name, cost, profit, performance)
              list: final_list free
              int: 0 at initialization and +1 at each recursion
           :return:
              list:  best actions lists
        """
    if final_list is None:
        final_list = []
    temporary_list: list = actions[i:]
       while DataList.get_cost_invest(temporary_list) > MAX_EXPENDITURE:
        temporary_list.pop()
    final_list.append(temporary_list)
    i += 1
    if i < len(temporary_list):
        brute_force(actions, final_list, i)
    return final_list
```

Conclusion algorithme Brute force

Spécialiste de l'investissement

Il faut une version plus optimale pour répondre aux attentes des clients.

L'autre solution serait de faire varier cette contrainte pour diviser le problème initial en sous-problèmes de même nature, mais de taille moindre.

La limite de cet algorithme est de raisonner selon sa contrainte principale
-> **un budget précis**

La complexité spatiale de brute force est $\boldsymbol{O(2^{10n})}$

Si l'on multiplie le nombre d'actions par 10, l'algorithme force brute nécessitera $\mathbf{2^{10n}/2^{n} = 2^{9n}}$ fois plus de temps et d'espace pour s'exécuter.

Conclusion

Analyse optimisée

Spécialiste de l'investissement

Si je peux remplir une liste d'actions de capacité n, je peux essayer de remplir cette liste de capacités n+1.

Seule la limite locale pour une liste d'actions de taille n est comparée avec les autres options pour une liste de taille n+1.

Les options non-optimales pour la liste de taille n sont écartées des comparaisons dans la liste de taille n+1.

Le processus est itéré jusqu'au coût maximal de la liste d'actions:

-> **Le budget précis**

Optimisée

Spécialiste de l'investissement

Code Optimisé

Code

```
def optimized(actions, final_list=None, i: int = 0)
        """
            returns an action list with the best investment and profitability.
            :param:
                list: action_list (name, cost, profit, performance)
                list: final_list Empty list at initialization
                int: an integer set to 0
            :return:
                list: action list with best on first
        """
j: int = i
    if final_list is None:
        final_list = []
    temporary_list: list = []
    cost_invest: float = 0.0
    while cost_invest < MAX_EXPENDITURE and j < len(actions):
        cost_invest += actions[j][1]
        if cost_invest <= MAX_EXPENDITURE and actions[j][3] > 0.0:
            temporary_list.append(actions[j])
        else:
            cost_invest -= actions[j][1]
        j += 1
    final_list.append(temporary_list)
    i += 1
    if i < len(actions):
        optimized(actions, final_list, i)
    return final_list
```

Spécialiste de l'investissement

Demo

Action	Bin.	Coût	%	Profit	Bin. * Coût	Evol. Coût	Evol. contrainte
Action-20	**0**	114	18	20,52	**0**	**0**	**500**
Action-6	**1**	80	25	20	**80**	**80**	**420**
Action-4	**1**	70	20	14	**70**	**150**	**350**
Action-5	**1**	60	17	10,2	**60**	**210**	**290**
Action-12	**1**	110	9	9,9	**110**	**320**	**180**
Action-10	**1**	34	27	9,18	**34**	**354**	**146**
Action-13	**1**	38	23	8,74	**38**	**392**	**108**
Action-3	**1**	50	15	7,5	**50**	**442**	**58**
Action-11	**1**	42	17	7,14	**42**	**484**	**16**
Action-9	**0**	48	13	6,24	**0**	**484**	**16**
Action-19	**0**	24	21	5,04	**0**	**484**	**16**
Action-2	**0**	30	10	3	**0**	**484**	**16**
Action-8	**0**	26	11	2,86	**0**	**484**	**16**
Action-7	**0**	22	7	1,54	**0**	**484**	**16**
Action-18	**1**	10	14	1,4	**10**	**494**	**6**
Action-1	**0**	20	5	1	**0**	**494**	**6**
Action-16	**0**	8	8	0,64	**0**	**494**	**6**
Action-15	**0**	18	3	0,54	**0**	**494**	**6**
Action-17	**1**	4	12	0,48	**4**	**498**	**2**
Action-14	**0**	14	1	0,14	**0**	**498**	**2**

Contrainte: **500€**
TEST
VALIDE
INVALIDE

Stockage de la liste dans une liste temporaire.

Par Laurent Jouron

Spécialiste de l'investissement

Tri

Action	Coût	Profit
Liste 1	500	89,48
Liste 2	498	88,54
Liste 3	498	79,48
Liste 4	498	73,22
Liste 5	498	65,34
Liste 6	388	55,44
Liste 7	354	46,26
Liste 8	316	37,52
Liste 9	266	30,02
Liste 10	224	22,88
Liste 11	176	16,64
Liste 12	152	11,60
Liste 13	122	8,60
Liste 14	96	5,74
Liste 15	74	4,2
Liste 16	64	2,8
Liste 17	44	1,8
Liste 18	36	1,16
Liste 19	18	0,62
Liste 20	14	0,14

Tri des résultats

Il en ressort 20 listes (autant que d'actions) pour le fichier en exemple.

- Pour un investissement maximum de 500€:
- 20 listes
- 500 €, pour l'investissement le plus élevé.
- Un profit de 89,48€ sur les 2 dernières années

Les 5 premières listes sont toutes très valables.

Cependant, il faut les comparer les unes aux autres pour identifier **la** plus rentable.
Nous gardons celle qui a le meilleur résultat, coût / rentabilité.

Spécialiste de l'investissement

Fichier dataset1_Python+P7.csv

Action	Coût	Profit
Share-GRUT	198,76	196,61

Pour ce fichier, Sienna obtient le résultat suivant:
Total cost: 498,76€
Total return: 196,61€

Ce résultat n'est pas totalement faux, mais il peut être plus optimal.

Comparaison

Action	Coût	Profit
Share-GRUT	198,76	196,61
Share-CBNY	1,22	0,48

Pour ce même fichier, j'obtiens le résultat suivant:

Total cost: 499,98€
Total return: 197,09€

Spécialiste de l'investissement

Comparaison

Action	Coût
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	1406

Pour le fichier « dataset2_Python+P7.csv » testé, Sienna obtient le résultat suivant:
Total cost: 489,24€
Total return: 193,78€

Mis à part 2 actions (Share-FWBE et Share-ANFX) , les résultats sont tous différents.

Pour ce même fichier, j'obtiens le résultat suivant:
Total cost: 499,98€
Total return: 185,25€

Action	Coût	Profit
Share-IJFT	40,91	15,91
Share-ANFX	38,55	15,31
Share-MALJ	46,37	15,25
Share-OPBR	39,0	15,19
Share-FWMV	41,68	14,92
Share-HATC	43,45	14,83
Share-XGNC	41,86	14,71
Share-VWZM	40,41	14,63
Share-QLWT	37,63	14,4
Share-ENZZ	37,24	14,22
Share-SFQC	36,53	14,22
Share-VUMM	36,71	14,06
Share-FWBE	18,31	7,29
Share-JMLZ	1,27	0,31

Résultat algorithme Optimisé

Spécialiste de l'investissement

Il faut un tri préalable des actions les plus lucratives pour que cette solution soit viable.

Cette nouvelle version aborde un type d'algorithmes nommé "glouton".

Il choisit les actions en fonction de la rentabilité coût/performance. Une action est rentable si elle profite plus qu'elle ne coûte.

Les actions sont ordonnées en diminuant les coûts/performances (retour maximum pour la première action).

L'algorithme parcourt la liste des actions : tant que le budget le permet, l'action est ajoutée à la liste finale.

Résultats

Conclusion algorithme Optimisé

Spécialiste de l'investissement

Cet algorithme est de complexité temporelle ***O*(n*log(n))**.

Bien que “glouton”, l’algorithme est moins gourmand en mémoire que brute force.

Une fois les données triées, il n'y a aucun besoin de comparer chaque action avec les actions précédentes.

La liste définitive des actions comprend toutes les actions pour lesquelles le coût/performance est le plus élevé et dont le coût total est le plus proche du budget.

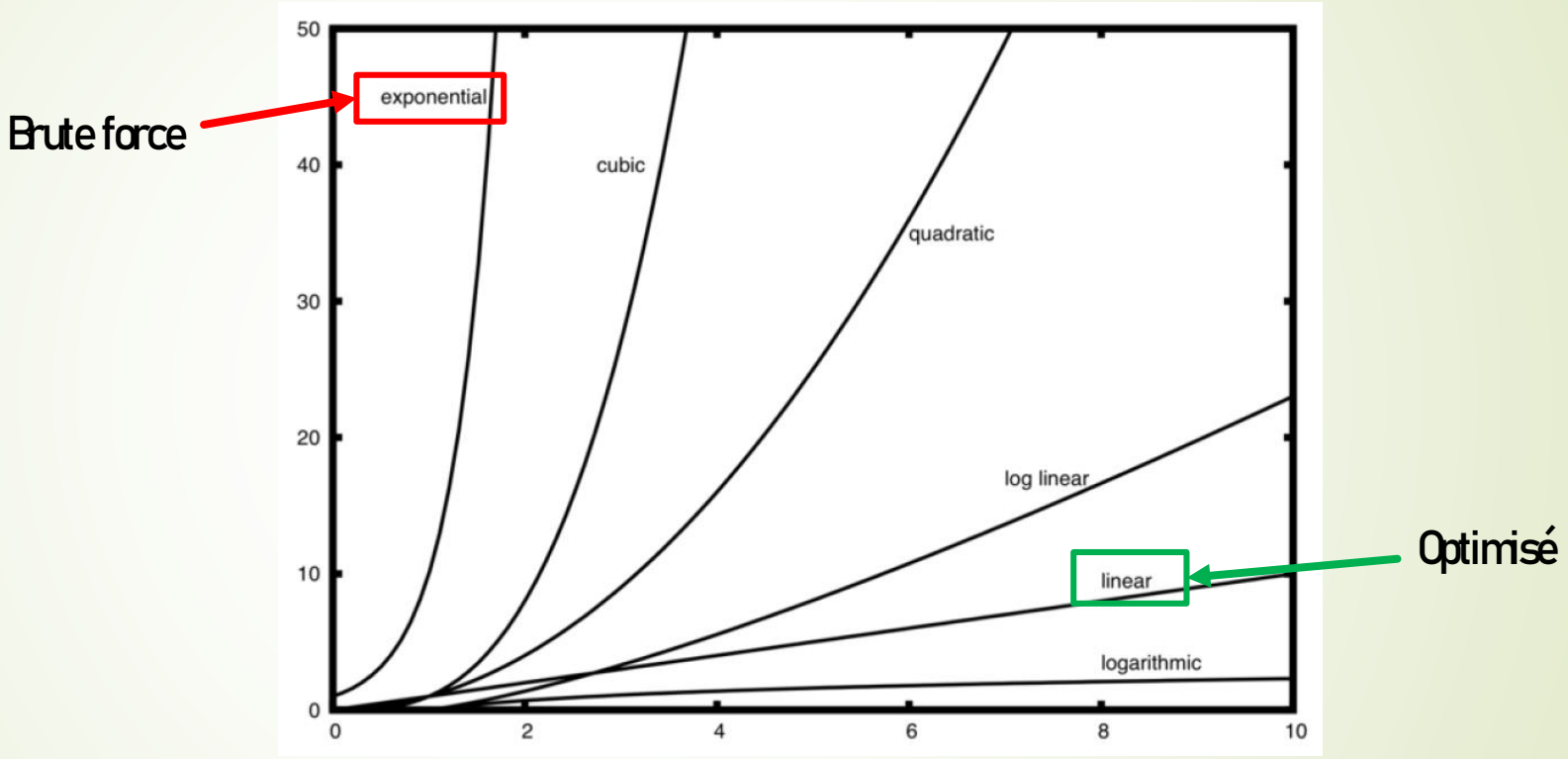
La compléxité spatiale est ***O*(W10n).**

Conclusion

Graphique des complexités

Spécialiste de l'investissement

Graphique



L'algorithme optimisé, qui utilise le tri, a peu de complexité ***O*(n*log(n))**

Conclusions

Spécialiste de l'investissement

Conclusion

Pour les fichiers dataset fournis par Trade & Invest, l'algorithme glouton se révèle être le plus performant, comparé à Brute force, quant à la maximisation des performances des actions.

L'algorithme glouton semble être une solution viable, tant en termes de complexité

- Temporelle ***O*(n*log(n))**
- Spatiale ***O*(W10n)**