AlgoInvest - Analyse algorithmes

Nicolas Habrias (O.C. Parcours D.A. Python)
07/05/2021

Menu

- Objectif
- Hypothèses
- Input
- Traitement des données d’entrée
- Output
- Analyse algorithme de force brute
- Résultats algorithme de force brute
- Pseudocode algorithme optimisé
- Analyse algorithme optimisé
- Résultats algorithme optimisé
- Résultats Sienna vs algo optimisé
- Analyse de performances et d'efficacité
- Conclusion: limites algorithmes choisis

Objectif

- L’objectif est de trouver la liste d’action à acheter, parmi une liste d’action, qui maximise les bénéfices

- Objectifs de bruteforce:
  - Trouver can_buy_stock_list
  - Essayer toutes les combinaisons possible de can_buy_stock_list pour trouver best_stock_list

- Objectifs de optimized:
  - Trouver "le meilleur chemin" qui mène best_stock_list sans essayer toutes les combinaisons possibles
  - Gain de temps: résultat en moins d'une seconde
  - Trouver best_stock_list
  - Objectif secondaire: Maximiser le rendement (profit / prix * 100)

can_buy_stock_list → best_stock_list

Hypothèses

Hypothèses

- Les hypothèses de ce projet ont changé en cours de projet donc voici mes dernières hypothèses:
- Chaque action ne peut être achetée qu'une seule fois.
- On ne peut acheter qu'une action entière.
- Soit MAX_INVEST = 500€ la somme maximale à investir
- Les prix sont en €.

Input

- Soit data_csv, une liste d'actions au format csv avec les champs:
  - name: nom d'une action
  - price: prix d'une action → variable stock_price
  - profit: bénéfice d'une action au bout de 2 ans en % par rapport à price → variable stock_price → variable stock_profit_percent
- dataset_20_actions.csv
- dataset1_Python+P7.csv
- dataset2_Python+P7.csv
- dataset1 et dataset2 ont environ 1000 actions mais certaines données sont manquantes ou incorrectes.

Traitement des données d’entrée

- Soit stock, une action modélisée par un tuple avec les 3 champs (stock_name, stock_price, stock_profit_percent)
- Soit stock_name, le nom d'une action
- Soit stock_price, le prix d'une action
- stock_profit_percent, le profit d'une action en %
- stock_profit égale à stock_profit_percent divisé par 100 fois stock_price
- Soit dispo_stock_list, la liste des actions disponibles
- Soit len_dispo_stock_list, le nombre d'actions dans dispo_stock_list
- La fonction import_data renvoie dispo_stock_list à partir de data_csv

- Soit la fonction filter_and_sort_data qui:
  - Retire les actions au prix négatif de dispo_stock_list
  - Retire les actions au profit négatif de dispo_stock_list
  - Supprime les doublons de dispo_stock_list
  - Tri de dispo_stock_list en fonction de stock_profit_percent par ordre décroissant
  - filter_and_sort_data est utilisée dans optimized.py

Output

- len_dispo_stock_list
- Big0, le nombre d’itérations
- len_can_buy_stock_list (seulement pour force brute)
- best_stock_list, la liste des actions (sous forme de tuple) que l'on devrait acheter pour maximiser le profit d'un client au bout de deux ans
- best_stock_list_price, le coût des actions dans best_stock_list
- best_stock_list_profit, le bénéfice des actions dans best_stock_list
- len_best_stock_list, le nombre d'actions dans best_stock_list
- la durée d'exécution de l'algo en s

Analyse algorithme de force brute

- Algorithme de force brute
- 1e boucle sur len_dispo_stock_list
- Utilisation de la méthode combinations du module itertools pour calculer toutes les combinaisons de liste d'actions dans dispo_stock_list
- combi_stock_list = list(combinations(dispo_stock_list, j))
  - 2e boucle for incluse dans la 1e boucle sur combi_stock_list pour traiter une liste d'actions
    - 3e boucle incluse dans la 2e boucle sur la liste d'actions pour traiter une action
    - Si l'investissement est inférieur à MAX_INVEST, cette liste d'actions est rangée dans can_buy_stock_list

- Tri de can_buy_stock_list en fonction de stock_combi_profit par ordre décroissant

  ➡ Le résultat best_stock_list est le premier élément de can_buy_stock_list

Résultat algorithme de force brute

- Notation Grand O Complexité O(2^n) avec n égale 20: Le nombre d'éléments testés pour toutes les boucles = 2^n
- Pour la liste de 20 actions on obtient:
- BigO: 1 048 576
- len_can_buy_stock_list: 813 347
- best_stock_list: (('Action-4', 70.0, 20.0), ('Action-5', 60.0, 17.0), ('Action-6', 80.0, 25.0), ('Action-8', 26.0, 11.0), ('Action-10', 34.0, 27.0), ('Action-11', 42.0, 17.0), ('Action-13', 38.0, 23.0), ('Action-18', 10.0, 14.0), ('Action-19', 24.0, 21.0), ('Action-20', 114.0, 18.0))
- best_stock_list_price: 498.0
- best_stock_list_profit: 99.08
- len_best_stock_list: 10
- duration in s: 1.9375

Pseudocode algorithme optimisé

- Pas de diagramme mais un pseudo code
- dispo_stock_list obtenue avec la fonction filter_and_sort_data (tri / profit)
- Remarque test boucle wile avec stock_list_price inférieure ou égale à MAX_INVEST et le nombre d'actions sélectionnées dans best_stock_list inférieur à len_dispo_stock_list mais l'algo oubliait certaines actions en fin de liste.
- bigO égale 0 au départ
- Boucle for stock dans dispo_stock_list
  - bigO égale bigO + 1
  - Si stock_list_price + stock_price inférieure ou égale à MAX_INVEST alors:
    - Soit stock_profit, le profit d'une action en €
    - stock_profit égale à stock_profit_percent divisé par 100 fois stock_price
    - best_stock_list_price égale à best_stock_list_price + stock_price
    - best_stock_list_profit égale à best_stock_list_profit plus stock_profit
    - stock est rangé dans best_stock_list
    - recalcule de len_best_stock_list

Résultats algorithme optimisé

- Algorithme glouton dit problème du sac à dos
- Notation Grand O Complexité O(n)
- Pour la liste de 20 actions on obtient:
- len_dispo_stock_list: 20
- bigO: 20
- best_stock_list: [('Action-10', 34.0, 27.0), ('Action-6', 80.0, 25.0), ('Action-13', 38.0, 23.0), ('Action-19', 24.0, 21.0), ('Action-4', 70.0, 20.0), ('Action-20', 114.0, 18.0), ('Action-5', 60.0, 17.0), ('Action-11', 42.0, 17.0), ('Action-18', 10.0, 14.0), ('Action-17', 4.0, 12.0), ('Action-16', 8.0, 8.0), ('Action-14', 14.0, 1.0)]
- best_stock_list_price: 498.0 (vs 498 forcebrute)
- best_stock_list_profit: 97.48 (vs 99.08 forcebrute) delta de 1%
- len_best_stock_list: 12 (vs 10 forcebrute)
- duration in s: 0.0 (vs 1.9 forcebrute)

Résultats Sienna vs algo optimisé

Dataset1 choix parmi (956 actions, après nettoyage des données)		
Algo	Sienna	Optimisé
Choix	1 Action: Share-GRUT	25 actions: Share-XJMO, Share-KMTG, Share-MTLR, Share-GTQK, Share-LRBZ, Share-WPLI, Share-GIAJ, Share-GHIZ, Share-ZSDE, Share-IFCP, Share-FKJW, Share-NHWA, Share-LPDM, Share-QQTU, Share-USSR, Share-EMOV, Share-LGWG, Share-QLMK, Share-UEZB, Share-CBNY, Share-CGJM, Share-EVUW, Share-FHZN, Share-MLGM
Prix	498.76	499.94
Bénéfice	196.61	198.51
Rendement	39.42%	39,71%

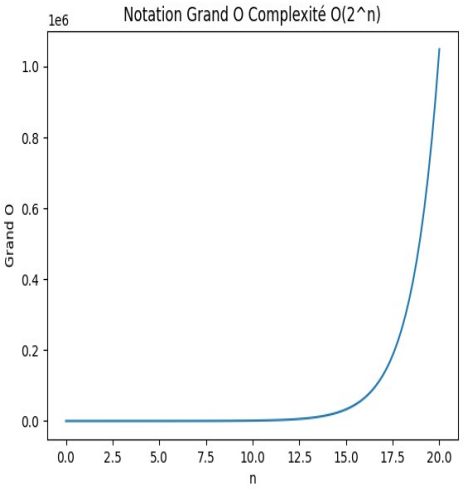
Résultats Sienna vs algo optimisé

Dataset2 (choix parmi 541 actions, après nettoyage des données)		
Algo	Sienna	Optimisé
Choix	18 Action: Share-ECAQ, Share-IXCI, Share-FWBE, Share-ZOFA, Share-PLLK, Share-YFVZ, Share-ANFX, Share-PATS, Share-NDKR, Share-ALIY, Share-JWGF, Share-JGTW, Share-FAPS, Share-VCAX, Share-LFXB, Share-DWSK, Share-XQII, Share-ROOM	22 actions: Share-PATS, Share-ALIY, Share-JWGF, Share-NDKR, Share-PLLK, Share-FWBE, Share-LFXB, Share-ZOFA, Share-ANFX, Share-LXZU, Share-FAPS, Share-XQII, Share-ECAQ, Share-JGTW, Share-IXCI, Share-DWSK, Share-ROOM, Share-VCXT, Share-YFVZ, Share-OCKK, Share-JMLZ, Share-DYVD
Prix	489.24	499.98
Bénéfice	193.78	197.77
Rendement	39.61%	39,56%

Analyse de performances et d'efficacité

- Algorithme de force brute

- Pour la liste de 20 actions on obtient:
- Notation Grand O Complexité O(2^n)
- BigO = nombre d'itérations = 1 048 576 = 2^n avec n = 20
- Durée: 1.9375 s



Fonction exponentielle de base 2

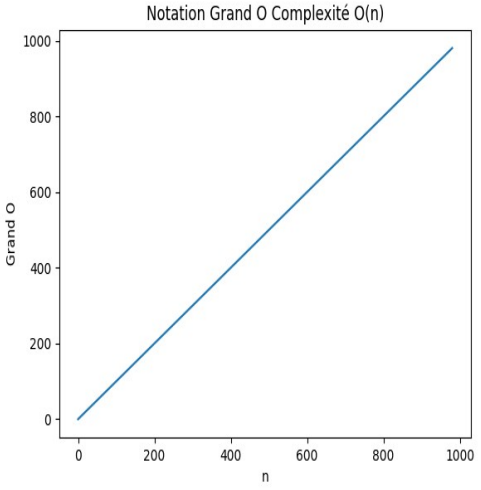
- Pour dataset1
- En inputs 1001 actions, après nettoyage: 956 actions
- nombre d'itérations = 2^956 ≈ 6e+287

- Pour dataset2
- Pour dataset2: En inputs 1000 actions, après nettoyage: 541 actions
- nombre d'itérations = 2^541 ≈ 7e+162

Analyse de performances et d'efficacité

- Algorithme optimisé: algorithme glouton dit problème du sac à dos

- Notation Grand O Complexité O(n)
- Pour dataset1: En inputs 1001 actions, après nettoyage: 956 actions
- Pour dataset2: En inputs 1000 actions, après nettoyage: 541 actions
- Pour liste de 20 actions, dataset1 et dataset 2:
  - BigO = nombre d'itérations = n
  - Durée: 0.0s



Fonction linéaire

Conclusion: limites algorithme choisis

- Solution de force brute
- Recherche toutes les solutions possibles
- Non viable pour un grand nombre d'input en temps
- Non viable pour un grand nombre d'input en mémoire
- Ne retourne pas de solutions pour dataset1 et dataset2 (stoppé après 10')

- Solution algorithme optimisée
- Algorithme glouton dit problème du sac à dos
- Programmation dynamique
- Fonctionne rapidement même pour des inputs importants
- Retourne une solutions pour dataset1 et dataset2
- Ne retourne pas le meilleur bénéfice