École Polytechnique de Montréal

Département de génie informatique et génie logiciel

INF4705

Analyse et conception d’algorithmes

Travail pratique 2

Soumis par

Francis de Lasalle, #1706231

Zihui Zhong, #1687994

18 mars 2016

Table des matières

[Introduction 2](#_Toc446042585)

[Description du jeu de données 3](#_Toc446042586)

[Résultats expérimentaux 3](#_Toc446042587)

[Analyse et discussion 7](#_Toc446042588)

[Analyse asymptotique des algorithmes 7](#_Toc446042589)

[Algorithme vorace probabiliste 7](#_Toc446042590)

[Algorithme de programmation dynamique 7](#_Toc446042591)

[Heuristique d’amélioration locale 7](#_Toc446042592)

[Analyse empirique des algorithmes 7](#_Toc446042593)

[Algorithme vorace probabiliste 7](#_Toc446042594)

[Algorithme de programmation dynamique 7](#_Toc446042595)

[Heuristique d’amélioration locale 8](#_Toc446042596)

[Qualité des algorithmes 8](#_Toc446042597)

[Conclusion 9](#_Toc446042598)

[Références 9](#_Toc446042599)

# Introduction

Ce second travail pratique concerne l’implémentation et l’analyse de 3 types d’algorithmes permettant de déterminer les locations (restaurants) à installer pour maximiser les revenus d’une chaîne de restauration, sous la contrainte d’une certaine capacité maximale de distribution de la matière première des restaurants (du poulet). Plus particulièrement, cette chaîne de restauration compte ouvrir un nombre indéterminé de restaurants parmi une liste de sites potentiels, pour plusieurs villes données (il y a donc plusieurs sites par ville et plusieurs villes). Chacun de ces sites est caractérisé par les revenus qu’il générera et la quantité de poulet nécessaire pour son fonctionnement. Chaque ville est liée à un seul fournisseur de poulet qui dispose d’une capacité maximale de poulet qu’il peut fournir. Les 3 différents algorithmes à implémenter devront donc permettre de déterminer la sélection de sites optimisant les revenus générés par la ville, tout en respectant la capacité maximale du fournisseur pour la ville donnée.

Ce problème peut être entièrement généralisé au problème du sac à dos, un problème NP-complet, n’ayant donc aucun algorithme connu pouvant le résoudre à la fois rapidement (en temps polynomial) et de façon exacte, pour tout exemplaire de données. Quelques algorithmes pour solutionner ce problème existent, notamment un algorithme de programmation dynamique[[1]](#footnote-1) permettant de résoudre le problème (de trouver l’optimum) en temps pseudo-polynomial , avec le nombre d’objets dans l’exemplaire et la capacité du sac à dos, exprimé de façon proportionnelle à la taille de , en bit, soit . Afin de résoudre le problème de désignation des sites (d’optimisation du revenu des villes), cet algorithme de programmation dynamique devra être implémenté. Cet algorithme sera également comparé à un algorithme de type vorace ainsi qu’à l’heuristique d’optimisation locale. Ces deux autres algorithmes seront également implémentés et une composante aléatoire sera aussi ajoutée à l’algorithme vorace. En effet, l’implémentation classique d’un algorithme vorace est de faire des choix « gloutons », c’est-à-dire de construire une solution en prenant d’abord les « meilleurs » choix (ceux ayant la meilleure santé (*fitness*), par exemple). L’algorithme vorace probabiliste qui sera implémenté affecte cependant une certaine probabilité à chaque objet de l’exemplaire, selon son revenu, donc plus un site est rentable, plus il est probable qu’il soit sélectionné.

Enfin, une fois implémentés, l’analyse asymptotique du temps de calcul de chacun de ces algorithmes sera faite. À l’aide de cette analyse et des résultats empiriques, il sera ensuite possible de faire une analyse hybride du problème, afin de préciser la constante multiplicative et le coût fixe de ces algorithmes et de confirmer la forme de la fonction de consommation de ressources. Une brève discussion quant à la qualité des solutions obtenues (en termes de temps de calcul, de consommation de mémoire, d’exactitude des résultats et de la difficulté d’implémentation) sera faite. Il sera enfin possible de déterminer sous quelles conditions il est préférable d’utiliser lequel de ces algorithmes. Le rapport se conclura par un bref rappel des résultats.

# Description du jeu de données

Le jeu de données se présente sous la forme de fichiers texte dont la première ligne indique le nombre de sites (restaurants) contenus dans le fichier et la dernière ligne indique la capacité du fournisseur, en nombre de poulets. Les lignes intermédiaires correspondent chacune à une location potentielle. Le premier nombre est l’identifiant du restaurant, le second nombre sa rentabilité et le troisième sa demande de poulet.

Les ensembles de sites sont de tailles 10, 20, 50, 100, 200, 500 et 1000 (donc, autrement dit, les villes (chaque fichier) peuvent contenir de 10 à 1000 sites potentiels). Pour chaque fichier, il y a une version dont la capacité totale est de 10, 100 ou 1000. Pour chaque combinaison de taille d’exemplaire et de capacité se trouvent 10 ensembles de données différents, afin de pouvoir réaliser une moyenne sur les 10 ensembles. Toutefois, la capacité totale varie, tel que mentionné par le chargé de laboratoire, dans son courriel. Cette variation s’accentue d’ailleurs plus fortement pour de plus gros exemplaires. Par exemple, le fichier contenant 500 sites et une capacité de 100 a en réalité une capacité de 248 pour le premier fichier des 10 ensembles de données et une capacité de 2538 pour le dernier fichier. En contraste, les 10 ensembles de fichiers pour un échantillon de taille 50 avec une capacité de 100 vont de 101 pour le premier fichier à 248 pour le dernier. L’écart est donc moins prononcé.

# Résultats expérimentaux

Tel que mentionné dans l’énoncé du travail, chacun des 3 algorithmes a été exécuté de façon à retourner le temps d’exécution et le revenu total de l’exemplaire en question. Comme demandé, la moyenne du temps d’exécution de chaque ensemble de 10 exemplaires et la moyenne du revenu total sont rapportées dans le tableau suivant. Toutefois, comme mentionnée dans la section *Description du jeu de données*, la capacité totale de chaque exemplaire varie généralement au sein d’un ensemble de 10 exemplaires supposés identiques. Cette moyenne a donc plus ou moins d’intérêt. Le premier tableau illustre les résultats pour l’algorithme de programmation dynamique, le second tableau pour l’algorithme vorace et le troisième pour l’algorithme d’amélioration locale.

Tableau 1. Moyenne sur 10 exemplaires des résultats expérimentaux pour l’algorithme de programmation dynamique.

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre de sites | Capacité (non réelle) | Temps moyen de calcul (ns) | Revenu moyen calculé |
| 10 | 10 | 67269.3 | 13.1 |
| 10 | 100 | 70000.1 | 108.2 |
| 10 | 1000 | 34808 | 1086.6 |
| 20 | 10 | 187228.4 | 13.9 |
| 20 | 100 | 181799.7 | 103 |
| 20 | 1000 | 319094.8 | 1243.1 |
| 50 | 10 | 308284.9 | 22.6 |
| 50 | 100 | 770435.5 | 203.6 |
| 50 | 1000 | 2091364.9 | 1967.6 |
| 100 | 10 | 572103.9 | 43.6 |
| 100 | 100 | 1502141.9 | 371.5 |
| 100 | 1000 | 9179893.8 | 3675.7 |
| 200 | 10 | 1072040.9 | 83.8 |
| 200 | 100 | 4838382 | 741 |
| 200 | 1000 | 28724735.6 | 7452.6 |
| 500 | 10 | 3612200.9 | 209.5 |
| 500 | 100 | 20631949.4 | 1858.8 |
| 500 | 1000 | 122482807.5 | 17672.9 |
| 1000 | 10 | 11213106 | 421 |
| 1000 | 100 | 54973262.2 | 3724.7 |
| 1000 | 1000 | 444331411.5 | 36469.1 |

Tableau 2. Moyenne sur 10 exemplaires des résultats expérimentaux pour l’algorithme vorace.

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre de sites | Capacité (non réelle) | Temps moyen de calcul (ns) | Revenu moyen calculé |
| 10 | 10 | 1299130.3 | 12.6 |
| 10 | 100 | 1134920.4 | 106.8 |
| 10 | 1000 | 1194826.3 | 1061.7 |
| 20 | 10 | 1430580.9 | 13.1 |
| 20 | 100 | 1523540.8 | 116 |
| 20 | 1000 | 1454952.2 | 1165.8 |
| 50 | 10 | 2012025.1 | 20.2 |
| 50 | 100 | 2207972.9 | 181.3 |
| 50 | 1000 | 2370106.6 | 17567 |
| 100 | 10 | 2204462.2 | 36.2 |
| 100 | 100 | 2594572.4 | 314.4 |
| 100 | 1000 | 3229490.9 | 3110.7 |
| 200 | 10 | 3958168.2 | 67.1 |
| 200 | 100 | 4808211.2 | 624.7 |
| 200 | 1000 | 7308563.7 | 6190.9 |
| 500 | 10 | 5253009.1 | 162.6 |
| 500 | 100 | 9515351 | 1494.4 |
| 500 | 1000 | 22811863.2 | 14110.6 |
| 1000 | 10 | 8465526.7 | 320.2 |
| 1000 | 100 | 16969843.8 | 2954.3 |
| 1000 | 1000 | 42960279.6 | 28710.8 |

Tableau 3. Moyenne sur 10 exemplaires des résultats expérimentaux pour l’algorithme d’amélioration locale.

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre de sites | Capacité (non réelle) | Temps moyen de calcul (ns) | Revenu moyen calculé |
| 10 | 10 | 1604915.5 | 13.1 |
| 10 | 100 | 1523586 | 106.4 |
| 10 | 1000 | 1454199.6 | 1080.8 |
| 20 | 10 | 2611062.5 | 13.5 |
| 20 | 100 | 2724831.9 | 119.1 |
| 20 | 1000 | 2746561.8 | 1196.7 |
| 50 | 10 | 9911250.6 | 21.5 |
| 50 | 100 | 19198287.1 | 210.7 |
| 50 | 1000 | 17510107.7 | 2119.8 |
| 100 | 10 | 33778922.2 | 39.4 |
| 100 | 100 | 110626284.4 | 375.6 |
| 100 | 1000 | 143151809.3 | 3588.2 |
| 200 | 10 | 367022157.8 | 74 |
| 200 | 100 | 1355339448 | 688.6 |
| 200 | 1000 | 1342525669 | 6759.1 |
| 500 | 10 | 5063961047 | 181.1 |
| 500 | 100 | 31619978889 | 1812.6 |
| 500 | 1000 | 48591852087 | 16461.8 |
| 1000 | 10 | 78442092161 | 360.6 |
| 1000 | 100 | n/a | n/a |
| 1000 | 1000 | n/a | n/a |

Il est à noter l’absence de 2 résultats pour l’heuristique d’amélioration locale, justifiée par le fait que ces calculs auraient exigé beaucoup trop de temps (quelques dizaines de minutes).

Comme l’analyse des résultats empiriques doit se faire sur la base de 2 paramètres, soit la taille de l’exemplaire et la capacité du fournisseur, les valeurs ci-dessus sont difficilement exploitables. Étant donné que l’écart entre la capacité du premier exemplaire d’un ensemble de 10 exemplaires et le dixième augmente avec la taille des échantillons (le nombre de sites), il a été choisi de limiter l’analyse empirique aux ensembles WC-10-10, WC-10-100, WC-10-1000, WC-20-10, WC-20-100, WC-20-1000, WC-50-10, WC-50-100 et WC-100-10. Ces ensembles présentent peu ou pas d’écart entre le premier exemplaire (eg. WC-10-10-01) et le dernier (eg. WC-10-10-10). De plus, pour tenter de compenser les différences de capacité entre les fichiers d’un même ensemble de 10 exemplaires, une moyenne pondérée selon la capacité a été faite pour ajuster le temps de calcul obtenu :

De plus, chaque calcul a été fait 2 fois et la moyenne des deux résultats a été calculée afin d’obtenir un temps de calcul plus juste (indépendant d’événements fortuits).

Les temps de calcul pondérés sont donnés dans le tableau suivant, pour les 3 différents algorithmes.

Tableau 4. Temps de calcul pondéré pour l’algorithme de programmation dynamique.

|  |  |  |
| --- | --- | --- |
| Nombre de sites | Capacité (non réelle) | Temps moyen de calcul pondéré selon la capacité réelle (ns) |
| 10 | 10 | 74898 |
| 10 | 100 | 62371.1 |
| 10 | 1000 | 34128.85 |
| 20 | 10 | 174948.4 |
| 20 | 100 | 177699.2 |
| 20 | 1000 | 326657.2 |
| 50 | 10 | 308535.8 |
| 50 | 100 | 816956.6 |
| 100 | 10 | 756120.8 |

Tableau 5. Temps de calcul pondéré pour l’algorithme vorace.

|  |  |  |
| --- | --- | --- |
| Nombre de sites | Capacité (non réelle) | Temps moyen de calcul pondéré selon la capacité réelle (ns) |
| 10 | 10 | 1285620 |
| 10 | 100 | 1121222 |
| 10 | 1000 | 1186614 |
| 20 | 10 | 1450564 |
| 20 | 100 | 1530349 |
| 20 | 1000 | 1460664 |
| 50 | 10 | 1975228 |
| 50 | 100 | 2225889 |
| 100 | 10 | 2160061 |

Tableau 6. Temps de calcul pondéré pour l’algorithme d’amélioration locale.

|  |  |  |
| --- | --- | --- |
| Nombre de sites | Capacité (non réelle) | Temps moyen de calcul pondéré selon la capacité réelle (ns) |
| 10 | 10 | 1632269 |
| 10 | 100 | 1458825 |
| 10 | 1000 | 1456257 |
| 20 | 10 | 2586250 |
| 20 | 100 | 2693864 |
| 20 | 1000 | 2778966 |
| 50 | 10 | 10310371 |
| 50 | 100 | 1.85E+08 |
| 100 | 10 | 4.62E+08 |

Notons que les temps obtenus par la moyenne pondérée demeurent très semblables à ceux obtenus par la moyenne « normale ».

# Analyse et discussion

## Analyse asymptotique des algorithmes

### Algorithme vorace probabiliste

blah

### Algorithme de programmation dynamique

blah

### Heuristique d’amélioration locale

blah

## Analyse empirique des algorithmes

Les méthodes d’analyse empiriques possibles sont le test de puissance, le test du rapport et le test des constantes. Le test de puissance est utile lorsqu’il n’y a aucune idée claire de la fonction de consommation de l’algorithme; le test de puissance permet de déterminer cette fonction, si elle croît de façon polynomiale. Le test du rapport est utile pour vérifier l’hypothèse selon laquelle la consommation en ressource est représentée par une certaine fonction . Enfin, le test des constantes permet de préciser l’hypothèse selon laquelle la consommation en ressources est de la forme de . Ce dernier test est d’ailleurs le seul applicable à des fonctions à plusieurs variables.

### Algorithme vorace probabiliste

blah

### Algorithme de programmation dynamique

En observant les résultats empiriques, il est clair que l’algorithme de programmation dynamique croît non seulement en fonction de la taille de l’exemplaire, mais également en fonction de la capacité du fournisseur. En effet, il est facile d’observer que, pour une même taille d’exemplaire, le temps de calcul augmentait de façon très considérable en augmentant la capacité. L’analyse asymptotique révèle d’ailleurs que la fonction de consommation est dans l’ordre exact de ; il s’agit d’ailleurs de ce que la littérature suggère1. Comme la consommation croît selon 2 paramètres (la taille de l’exemplaire et la capacité), seul le test des constantes est applicable.

Figure X. Test des constantes de l’algorithme de programmation dynamique

Le test des constantes confirme en effet l’hypothèse selon laquelle l’algorithme de programmation dynamique est dans l’ordre exact de . En effet, une droite peut-être tracée, joignant les points avec un facteur de corrélation fort acceptable de 0.8942. Un tel facteur de corrélation permet d’affirmer avec bonne certitude les conclussions que ce test permet de tirées sont valides. La droite est d’équation :

suggérant donc une constante multiplicative de 1784.

### Heuristique d’amélioration locale

blah

## Qualité des algorithmes

Tout d’abord, un aperçu rapide des résultats empiriques obtenus permet d’observer que l’algorithme de programmation dynamique est celui duquel est obtenue systématiquement la solution la plus optimale, ce qui était certes attendu. À cet effet, il s’agit, de ces 3 algorithmes, de l’algorithme de choix si la solution la plus optimale possible est nécessaire. L’heuristique d’amélioration locale offre des solutions quand même assez proches de celles de l’algorithme de programmation dynamique et il est évident que cette heuristique améliore considérablement les solutions obtenues par l’algorithme vorace, dont les solutions sont assez loin derrière, en termes d’optimalité.

En termes de temps de calcul, empiriquement, il est observé que l’heuristique d’amélioration locale est de loin la plus coûteuse en temps de calcul, ce qui est certes attendu, considérant sa plus grande complexité. Il devient vite difficilement utilisable pour des exemplaires de grande taille. En effet, pour les exemplaires de taille 1000 avec capacité de 10, cet algorithme a nécessité en moyenne 78442092161 ns, soit environ 1 minute 30. En contraste, l’algorithme vorace nécessitait 8.46 millisecondes et l’algorithme de programmation dynamique 11.2 millisecondes, pour le même exemplaire. Ces deux derniers algorithmes doivent donc nécessairement être privilégiés pour de très grands exemplaires.

En termes de consommation de mémoire, l’algorithme de type vorace et l’heuristique d’amélioration locale consomment tous deux une quantité relativement négligeable de mémoire, linéaire selon la taille de l’exemplaire. Même avec les plus gros exemplaires, il a été noté une consommation de mémoire nettement inférieure à 1% de 16GB.

BLAH

L’algorithme le plus simple à implémenter fut sans doute celui de type vorace probabiliste, suivit de l’algorithme de programmation dynamique et enfin, de l’heuristique d’amélioration locale. En somme, tous les algorithmes furent implémentés sans trop de difficulté. Les conseils, par exemple sur le choix d’algorithme de sélection au hasard2, ont certes grandement aidé, toutefois.

# Conclusion

Afin de pouvoir déterminer les sites (restaurants) permettant d’optimiser les revenus de différentes villes, sous la contrainte de la capacité de production en matière première (poulet) d’un fournisseur, une concrétisation du problème du sac à dos, 3 algorithmes différents ont été développés : un algorithme de type vorace probabiliste, un algorithme de programmation dynamique et l’heuristique d’amélioration locale.

Une analyse asymptotique de ces algorithmes a été faite, permettant de conclure que la complexité de l’algorithme de type vorace est dans l’ordre exact de , que la complexité de l’algorithme de programmation dynamique est dans l’ordre exact de et que la complexité de l’heuristique d’amélioration locale est dans l’ordre exact de .

Ensuite, une analyse empirique a été faite pour chacun des 3 algorithmes, afin de confirmer les hypothèses tirées de l’analyse asymptotique.

BLAH

# Références

Wikipedia, Knapsack problem :

https://en.wikipedia.org/wiki/Knapsack\_problem#Solving

Wikipedia, Fitness proportionate selection :

https://en.wikipedia.org/wiki/Fitness\_proportionate\_selection

1. https://en.wikipedia.org/wiki/Knapsack\_problem#Solving [↑](#footnote-ref-1)