

**INF4705**

**Analyse et conception d’algorithmes**

**Travail pratique 2 : algorithmes vorace, dynamique et local**

|  |  |
| --- | --- |
| **Étudiants** |  |
| **Clément Gamache** | **1642792** |
| **Louise Piotte** | **1683988** |
|  |  |

**Présenté à M. Samuel Gagnon**

**Le 18 mars 2016**

# Description du sujet et objectifs du laboratoire

Lors de cet exercice, nous devons implémenter un algorithme pour planifier l’installation de succursales de restauration rapide pour maximiser les profils dans chaque ville au W&A (une chaine de restauration fictive) sans dépasser la capacité des distributeurs de poulet locaux.

Au cours de conception d’algorithmes INF4705, nous avons vu plusieurs façons d’approcher la création d’algorithmes. Dans ce travail pratique, nous nous attaquons aux algorithmes voraces probabilistes, aux algorithmes de programmation dynamique et aux heuristiques d’amélioration locale. Le but de l’exercice est de pratiquer la conception de chaque type d’algorithme et de les comparer entre eux pour mieux comprendre les forces et les faiblesses de ces algorithmes sur l’exactitude de la solution, le temps de calcul et l’utilisation des ressources. L’exercice va aussi nous permettre de poursuivre les objectifs du premier travail pratique en nous faisant adapter et appliquer les notions théoriques sur un problème réel simplifié.

# Description du jeu de données

Les données utilisées par l’algorithme sont organisées de façon à ce que chaque fichier représente une ville. Dans chaque fichier se trouve le nombre d’emplacements de succursale possible. Pour chaque emplacement, il y a un identifiant unique, les revenus produit par la succursale et la quantité de poulet utilisé. Finalement, la dernière ligne de chaque fichier est la capacité maximale de génération de poulet par le fournisseur local :

<nombre d’emplacement>\n

<id> <revenue> <poulet>\n (fois le <nombre de ville>)

<capacité du fournisseur>

# Résultats expérimentaux

Voici la moyenne des résultats que nous avons obtenus pour chaque série de fichiers de jeux de données :

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| algorithme | fournisseur | Nombre d’emplacements possibles | | | | | | |
| 10 | 20 | 50 | 100 | 200 | 500 | 1000 |
| vorace | 10 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.001 | 0.005 |
| 100 | 0.000 | 0.000 | 0.000 | 0.001 | 0.001 | 0.002 | 0.005 |
| 1000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.005 | 0.020 |
| Tous | 0.000 | 0.000 | 0.000 | 0.001 | 0.001 | 0.003 | 0.010 |
| dynamique | 10 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 |
| 100 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.006 |
| 1000 | 0.001 | 0.000 | 0.000 | 0.001 | 0.002 | 0.015 | 0.085 |
| Tous | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.005 | 0.031 |
| local | 10 | 0.000 | 0.000 | 0.001 | 0.023 | 0.231 | 22.136 | 618.758 |
| 100 | 0.000 | 0.000 | 0.001 | 0.027 | 0.325 | 33.737 | 938.015 |
| 1000 | 0.000 | 0.000 | 0.001 | 0.015 | 0.404 | 39.310 | 1027.140 |
| Tous | 0.000 | 0.000 | 0.001 | 0.022 | 0.320 | 31.727 | 861.304 |

Pour mieux visualiser nos données, nous avons construit un graphique à échelle logarithmique sur les deux axes présentant les données du tableau précédent.

Le tableau ci-haut a une forte chance de suivre une complexité de où k serait une constante positive. Cela supporte notre idée initiale.

Le tableau ci-haut montre également que l’algorithme suive une complexité parce que est constant. Selon notre analyse théorique, si n’avait pas été constant, nous n’aurions pas obtenu ce résultat.

Le graphique ci-haut montre également la tendance à avoir une complexité de . Par contre, ne semble pas avoir d’incidence sur le résultat obtenu.

Pour l’exactitude des données, le tableau suivant montre les résultats que nous avons obtenus. Le pourcentage d’erreur a été calculé en fonction de la solution trouvée avec l’algorithme de programmation dynamique.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithme | fournisseur | Nombre d’emplacements possibles | | | | | | |
| 10 | 20 | 50 | 100 | 200 | 500 | 1000 |
| Vorace | 10 | 0.8547 | 5.0147 | 14.1134 | 17.0498 | 22.5180 | 24.3529 | 26.1834 |
| 100 | 1.1000 | 3.0225 | 10.7117 | 16.4893 | 17.8575 | 21.9673 | 23.5073 |
| 1000 | 0.3456 | 3.2042 | 11.2730 | 14.5308 | 19.4471 | 21.3358 | 23.8353 |
| Tous | 0.7668 | 3.7471 | 12.0327 | 16.0233 | 19.9409 | 22.5520 | 24.5087 |
| Dynamique | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Tous | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Local | 10 | 0.0000 | 0.6667 | 1.6848 | 0.3980 | 0.4561 | 0.8688 | 0.4861 |
| 100 | 0.0000 | 0.0793 | 0.9606 | 0.1776 | 0.1574 | 0.1321 | 0.0423 |
| 1000 | 0.0000 | 0.3221 | 0.6525 | 0.1082 | 0.0985 | 0.0574 | 0.0378 |
| Tous | 0.0000 | 0.3560 | 1.0993 | 0.2279 | 0.2374 | 0.3528 | 0.1887 |

Comme pour les temps d’exécution, il est pertinent de construire des graphiques illustrant l’exactitude en fonction de la taille des villes analysées :

Le graphique ci-haut montre que plus la taille de l’exemplaire est élevée, plus l’erreur de l’algorithme vorace sera grande. L’erreur maximale semble tendre vers une valeur qui nous est inconnue.

Nous n’avons pas de graphique pour l’algorithme de programmation dynamique parce que ce dernier trouve toujours la meilleure solution.

Ce graphique montre que, contrairement à l’algorithme vorace, le pourcentage d’erreur de l’algorithme d’heuristique à améliorations locales diminue avec la taille de l’échantillon. Nous voyons ainsi que l’avantage de cet algorithme se trouve dans l’exactitude de ses résultats.

# Analyse

Analyse asymptotique théorique

= nombre d’emplacement

= capacité du fournisseur

### Algorithme vorace probabiliste

Au pire cas (tous les emplacements dépassent la capacité du fournisseur pas eux-mêmes et ceux-ci sont choisi en ordre inverse), l’algorithme a une complexité , car il y a deux boucle imbriquées qui au pire cas passe tous les deux à travers toutes les données. Le meilleur cas serait , si un emplacement remplis la capacité du fournisseur tout seul et est choisi dès le premier cout. Par contre, le pire cas (tout comme le meilleur cas) arrivera rarement.

### Dynamique

L’algorithme possède une complexité dans tous les cas. Il y a deux boucles imbriquées, une sur et une sur .

### Amélioration local

L’algorithme possède une complexité de par itération d’amélioration. Chaque itération a quatre boucles impliquées sur dans le pire cas. Par contre la complexité de l’algorithme en général n’est pas évaluable, car le nombre d’itération d’amélioration n’est pas prévisible parce qu’il dépend des optimums locaux et de la précision des solutions vorace. Nous ne pouvons donc que conclure qu’en meilleur cas, l’algorithme aurait une complexité parce qu’il n’y aurait qu’une itération d’amélioration. En pire cas, il est difficile pour nous de définir le pire scénario possible.

## Analyse empirique avec les données :

### Algorithme vorace

Prouver que l’algorithme est . Diviser les données par . Nos temps pour l’algorithme sont trop petits pour être significatif. Les différences de timing sur coup d’horloge (~0.0016 sec) rendent les valeurs de moins que deux coups d’horloge invalide. La valeur de 0.001 sec veut seulement dire que l’on a détecté un coup d’horloge, donc que le temps est entre 0.000 et 0.002. comme notre plus grande valeur est 0.005 pour et 0.020 pour , nous n’avons pas assez de valeur pour faire une bonne analyse.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 10 | 20 | 50 | 100 | 200 | 500 | 1000 |
| Données(x) | 10 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.001 | 0.005 |
| 100 | 0.000 | 0.000 | 0.000 | 0.001 | 0.001 | 0.002 | 0.005 |
| 1000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.005 | 0.020 |
|  | 10 | 3.33E-06 | 0 | 4E-08 | 1.44E-07 | 5E-09 | 3.2E-09 | 5.4E-09 |
| 100 | 1.00E-06 | 0 | 0 | 1.22E-07 | 1.5E-08 | 6.8E-09 | 4.9E-09 |
| 1000 | 2.50E-06 | 0 | 1.2E-07 | 2E-08 | 2.78E-08 | 2.04E-08 | 1.96E-08 |

Conclusion : Selon les chiffres, l’algorithme est bel et bien car les résultat de sont tous très proche de 0.

### Algorithme dynamique

Prouver que l’algorithme est . Diviser les données par . Encore une fois, les données sont trop petites pour faire une bonne analyse empirique.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 10 | 20 | 50 | 100 | 200 | 500 | 1000 |
| Données(x) | 10 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 |
| 100 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.006 |
| 1000 | 0.001 | 0.000 | 0.000 | 0.001 | 0.002 | 0.015 | 0.085 |
|  | 10 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 2.22E-07 | 0.00E+00 | 8.00E-08 | 2.00E-07 |
| 100 | 2.00E-07 | 0.00E+00 | 0.00E+00 | 3.33E-08 | 1.00E-08 | 2.60E-08 | 5.70E-08 |
| 1000 | 1.13E-07 | 0.00E+00 | 4.00E-09 | 8.00E-09 | 7.78E-09 | 2.90E-08 | 8.48E-08 |

Conclusion : Selon les chiffres, l’algorithme est bel et bien , car les résultats de sont tous très proche de 0.

### Heuristique d’amélioration locale

Voici la preuve que l’algorithme est . Nous avons divisé les données par . Par contre, cette approche empirique n’est peut-être pas significative car le nombre d’itération d’amélioration n’est pas pris en comptes.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 10 | 20 | 50 | 100 | 200 | 500 | 1000 |
| Données(x) | 10 | 0.000 | 0.000 | 0.001 | 0.023 | 0.231 | 22.136 | 618.758 |
| 100 | 0.000 | 0.000 | 0.001 | 0.027 | 0.325 | 33.737 | 938.015 |
| 1000 | 0.000 | 0.000 | 0.001 | 0.015 | 0.404 | 39.310 | 1027.140 |
|  | 10 | 3.33E-08 | 0.00E+00 | 9.60E-11 | 2.30E-10 | 1.45E-10 | 3.54E-10 | 6.19E-10 |
| 100 | 2.00E-08 | 1.25E-09 | 1.12E-10 | 2.73E-10 | 2.03E-10 | 5.40E-10 | 9.38E-10 |
| 1000 | 2.50E-08 | 0.00E+00 | 9.60E-11 | 1.49E-10 | 2.53E-10 | 6.29E-10 | 1.03E-09 |

Conclusion : Selon les chiffres, l’algorithme est bel et bien , car les résultats de sont tous très proche de 0 dans le même ordre de grandeur. Ce résultat vient probablement du fait que l’algorithme fasse plusieurs itérations pour chaque version. Son effet est alors mitigé.

# Discussion des algorithmes

## Précision

En ce qui concerne la précision de la réponse, le meilleur algorithme est sans aucun doute l’algorithme dynamique, car il donne une solution optimale et les deux autres donnent une réponse approximative. L’algorithme d’amélioration locale donne une meilleure approximation de la solution optimale que vorace, car il prend la réponse de vorace et tente de faire des améliorations. Par exemple, pour le fichier WC-1000-1000-10.txt, l’algorithme dynamique donne la solution 61155, alors que vorace donne 51453 et les améliorations locales trouvent 61142.

## Temps d’exécution

Pour le temps d’exécution, vorace avec , est comparable au temps d’exécution de l’algorithme dynamique qui a un temps. Par contre, l’algorithme d’amélioration locale, avec une complexité de , a un temps d’exécution beaucoup plus grand pour tous les tailles d’échantillon (voir les tableaux contenant les moyennes de temps pour tous les algorithme donnés précédemment). En effet, en moyenne pour les échantillons de 1000 emplacement avec une capacité de fournisseur de 1000 est 0.020 secondes pour vorace, 0.085 secondes pour dynamique et 1027.140 secondes pour locale. Nous n’avons pas traité d’exemples possédant des valeurs de très élevées par rapport à . Si nous l’avions fait, nous aurions obtenu des temps d’exécution de l’algorithme dynamique plus élevés que ceux de l’algorithme d’amélioration locale.

## Consommation de mémoire

Enfin, pour la consommation de mémoire, l’algorithme vorace et celui d’amélioration locale ont une consommation de mémoire de l’ordre de . L’algorithme dynamique à une consommation de .

## Choix de l’algorithme

Enfin, le seul désavantage de l’algorithme dynamique est qu’il utilise mémoire. Par contre, avec un plus grand que , cet algorithme risque d’être plus coûteux en temps. Dans les manipulations que nous avons effectuées, ces désavantages ne se sont pas manifestés et ainsi leurs effets n’ont pas été significatifs par rapport au temps d’exécution et de la précision de la réponse. L’algorithme vorace n’est pas assez précis et les améliorations locales prennent beaucoup trop de temps.

## Améliorations possibles

L’algorithme vorace, étant beaucoup plus rapide que les deux autres, aurait bien pu avoir été implémenté avec plus de dix répétitions. Cela pourrait avoir un impact bénéfique, mais non mesurable sur l’exactitude des résultats.

L’algorithme de programmation dynamique fonctionne parfaitement dans notre cas. Par contre, avec un très grand par rapport à , l’algorithme devient beaucoup moins rapide que l’algorithme local. Pour résoudre ce problème, on pourrait faire en sorte que la grandeur des bonds soit plus grande que 1. De cette façon, on n’obtiendra pas de résultat exact, mais cela rendrait l’algorithme compétitif à l’algorithme local. De plus, cet algorithme n’aurait pas fonctionné si les valeurs entre 0 et n’avaient pas été énumérables de façon finie. Il faudrait alors utiliser une méthode semblable à celle que nous venons de décrire.