# Description du sujet et objectifs du laboratoire

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

Durant le cours de conception d’algorithmes INF4705, nous avons vu plusieurs façons d’approcher la création d’algorithmes. Dans de travail pratique, nous nous attaquons aux algorithmes voraces probabilistes, aux algorithmes de programmation dynamique et aux heuristiques d’amélioration locales. Le but de l’exercice est de pratiquer la conception de chaque type d’algorithme et de les comparés ensemble pour mieux comprendre les force et les faiblesses de chacun sur l’exactitude de la solution, le temps de calcul et l’utilisation des ressources. L’exercice va aussi nous permettre de continuer 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 organisé de façon à ce que chaque fichier représente une ville. Dans ce fichier, nous avons le nombre d’emplacement de succursales possible et pour chaque emplacement, il y a un identifiant unique, les revenus produit par la succursale et le poulet utilisé. Finalement, la dernière ligne de chaque fichier est la capacité maximale de poulet du fournisseur local.

<nombre d’emplacement>\n

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

<capacité du fournisseur>

# Résultats expérimentaux

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| algorithme | fournisseur | 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 |
| Tout | 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 |
| Tout | 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 |
| Tout | 0.000 | 0.000 | 0.001 | 0.022 | 0.320 | 31.727 | 861.304 |

# Analyse et discussion

Analyse asymptotique théorique

n = nombre d’emplacement

m = 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 à une complexité O(n2), car il y a deux boucle imbriquées qui au pire cas passe tous les deux à travers toutes les données. Le meilleur cas serait O(1), 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 : O(n\*m) dans tous les cas. Il y a deux boucles imbriquées, une sur n et une sur m.

Amélioration local : O(n4) au pire cas pour chaque itérations d’améliorations. Chaque itération à quatre boucles impliquées sur n 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. (Il dépend des optimums locaux et de la précision des solutions vorace.

## Analyse empirique avec les données :

Vorace : Prouver que l’algorithme est O(n2). Diviser les données par n2. 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 m = 10 et 0.020 pour m=1000, 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 |
| x/ n2 | 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 O(n2), car les résultat de x/n2 sont tous très proche de 0.

Dynamique : Prouver que l’algorithme est O(n\*m). Diviser les données par n\*m. Encore une fois, les données sont trop petite 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 |
| x/(n\*m) | 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 O(n\*m), car les résultat de x/(n\*m) sont tous très proche de 0.

Local : Prouver que l’algorithme est O(n4). Diviser les données par n4. 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 |
| x/(n4) | 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 O(n4), car les résultat de x/n4 sont tous très proche de 0. Ce résultat viens probablement du fait que l’algorithme fait plusieurs itération pour chaque versions alors sont effet est mitigé.

Discussion des algorithmes :

En ce qui concerne la précision de la réponse, l’algorithme choisit est le dynamique car il donne une solution optimale et les deux autre donne 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. Pour le fichier WC-1000-1000-10.txt, l’algorithme dynamique donne la solution 61155, alors que vorace donne 51453 et les améliorations locale trouve 61142.

Pour le temps d’exécution, vorace avec O(n2), est comparable au temps d’exécution de l’algorithme dynamique qui a un temps O(n\*m). Par contre, l’algorithme d’amélioration locale, avec une complexité de O(n4), a un temps d’exécution beaucoup plus grand pour tous les tailles d’échantillon (voir les tableaux contenant les moyenne de temps pour tous les algorithme donné 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.

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

Choix de l’algorithme

Enfin, le seul désavantage de l’algorithme dynamique est qu’il utilise n\*m mémoire, mais c’est utilisation reste raisonnable et elle n’est pas significatif 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 trop longtemps.