Analyse et conception d'algorithmes

INF4705

L4714

Jérémy Cugmas (1814477) & Juliette Tibayrenc (1800292)

3 nov. 2015

Table des matières

0. Introduction

Ce deuxième TP se penche sur le problème de mise en boîte, plus particulièrement dans le cas où toute les boites sont identiques et qu'il faut placer dans chacune d'elle un nombre précis d'items avec des volumes variables sans dépasser leur capacité. Ce problème relève de l'optimisation combinatoire et son objectif principal est d'utiliser de manière optimale le volume des boîtes pour placer les items sans dépasser le volume maximum autorisé. Plus historiquement, ce problème fait partie des 21 problèmes NP-complet, c'est-à-dire qu'il appartient à la classes des problèmes qui peuvent être résolus en temps polynomial sur une machine non déterministe. De plus, ce problème est utilisé dans de nombreuses situations actuelles comme le chargement des bagages dans un avion ou encore dans le découpe des matériaux pour diminuer au maximum les pertes. Afin de le résoudre, nous allons implanter trois algorithmes : vorace, programmation dynamique et recuit simulé afin de comparer leur efficacité et la qualité des solutions que propose chacun de ces algorithmes.

1. Revue de la théorie

Comme énoncé précédemment, ce TP a pour but d'élaborer un algorithme vorace, de programmation dynamique et de recuit simulé. Le premier d'entre eux repose sur une succession de choix qui semblent à chaque fois être le plus optimal mais malheureusement sans la possibilité de revenir en arrière. Les algorithmes de type «vorace » ou « glouton » ont la particularité d'être simple à concevoir, efficace et assez simple à implémenter mais ne représentent souvent pas la méthode la plus optimale. Le plus souvent, l'algorithme vorace dans le cas du problème de mise en boîtes se divise en deux étapes majeures : le tri des items et le placement de ceux-ci dans les boites qui ont respectivement les complexités, pour un nombre n d'items, $\Theta(nlog(n))$ et $\Theta(n)$ donc d'après la règle du maximum la complexité globale est de $\Theta(nlog(n))$.

Ensuite nous avons l'algorithme de programmation dynamique, celui-ci se base sur le principe d'optimalité de Bellman qui affirme qu'une solution optimale s'obtient en résolvant des sous-problèmes de façon optimale. En résumé, cela signifie que la solution optimale d'un problème s'obtient par la combinaison des solutions optimales des sous-problèmes. A contrario de l'algorithme diviser pour régner, la programmation dynamique procède en « bottom-up », c'est-à-dire qu'on débute par la résolution des sous-problèmes les plus petits puis on traite progressivement les plus gros. Avec l'algorithme dynamique, pour faciliter la vision du problème, on définit un tableau contenant la solution de chacun des sous-exemplaires de l'exemplaire originel dans un certain ordre. Dans le cas du problème, la complexité se mesure par le remplissage du tableau avec les n éléments et le rebrousser ensuite pour trouver notre solution. Cela nous donne une complexité globale de ?(n).

Le dernier algorithme est le recuit simulé qui est une méthode type métaheuristique utilisée à la base en thermodynamique. Il repose sur le choix aléatoire d'une solution donnée pour ensuite la modifier afin d'en obtenir une seconde, soit celle-ci améliore le critère d'optimisation soit elle le dégrade. On parle alors de voisinage. Dans le cas de la mise en boîtes, la solution voisine est obtenue en choisissant uniformément au hasard un item non placé et on l'insère dans une boite choisie au hasard. Si cela fait déborder la boite, on choisit un item au hasard dans cette boite pour respecter la capacité de celle-ci. La génération d'un voisinage d'une solution existante sera sans doute, pour n le nombre d'items, en $O(nlog(n))$. Le calcul du volume total d'une solution est au maximum en $O(n * m)$. Le recuit simulé est en $O(k_{max} * P * (n \cdot \log(n) + n * m))$.

2. Protocole expérimental

2.1 Algorithme vorace

Pour l ?algorithme vorace, le travail va se résumer à l ?écriture d ?un programme en langage C++. Afin de nous faciliter la tâche, tout le développement se fait en environnement Linux. Préalablement, le professeur nous a fourni un dossier contenant un panel d ?échantillons avec dans chacun d ?eux : le nombre d ?items, le nombre de boites, la capacité des boites et le volume de chacun des items. Le sujet du TP préconise l ?implémentation de l ?algorithme « Best-Fit-Decreasing » qui consiste à choisir les items par ordre décroissant de volume pour ensuite les placer dans une boite dont la capacité résiduelle est la plus faible après y avoir placé l ?objet. Ainsi notre programme va marcher de la manière suivante :

- On lit le fichier mis en paramètre pour récupérer les valeurs de n,m,c et les valeurs des volumes de items. On stock ces derniers dans des variables.
- On trie le tableau les volumes pour les classer de manière décroissante
- On fait une boucle sur les volumes pour les passer dans une fonction qui va avoir pour but de les placer unitairement dans les boites. Cette fonctionne prend en paramètre la volume à placer, les boites, la capacité des boites et le nombre de boites. On fait une boucle qui aura le même nombre d ?itérations que le nombre de boites. Si le volume à placer plus le volume occupé de la boite est inférieur au minimum actuel qui correspond, au début, à la capacité de la boite alors on met à jour ce minimum à la somme des deux éléments énoncés au début. De plus, on ajoute le volume à placer dans le volume occupé de la boite et on indique via un booléen qu ?on peut terminer la boucle.

Avant-dernière étape, écrire un script bash qui va permettre d'appliquer l'algorithme à tous les exemplaires fournis, et enfin calcul avec matlab des moyennes des temps d'exécution et volume restant pour chaque taille d'exemplaire.

2.2 Algorithme de programmation dynamique

De même que pour l'algorithme vorace, on utilise C++, bash et matlab (ou plus exactement octave). L'algorithme de programmation dynamique est détaillé dans le sujet, on ne le reprendra pas ici. On peut cependant ajouter quelques explications : lors de l'écriture du programme, la difficulté était la manipulation d'un tableau à deux dimensions dont l'une des dimensions était indexée par plusieurs nombres (les μ). De plus, pour prendre en compte l'intégralité des solutions, on est obligé de garder en mémoire l'ensemble des solutions d'une étape. On utilise pour cela des

buffers. Ces buffers sont au nombre de 2, un pour la répartition à l'étape précédente, initialisé à 0, l'autre pour stocker l'étape courante. Ils sont bien entendus swappés à la fin.

Ces buffers ont une taille c parmi m+c correspondant au nombre de combinaisons possibles avec m éléments de différentes capacités inférieures à c.

Par ailleurs, on ne fera pas tourner cet algorithme sur les exemplaires à $n > 32$ ou $m > 2$ (sauf pour le cas m=3 et n=10), par manque de temps.

2.3 Algorithme de recuit simulé

L'algorithme lui-même est fourni dans le sujet. Son implémentation n'a malheureusement pu être terminée (elle a été arrêtée au cours de l'étape de déboggage, tout est donc implémenté mais tout n'est pas correct).

3. Présentation des résultats

3.1 Algorithme vorace

Dans cette partie, nous allons vous présenter les résultats de l ?algorithme vorace, tout d ?abord avec un tableau avec comme valeurs entrantes la taille des exemplaires, le nombre de boites, le temps de calcul moyen de l ?algo et le volume moyen restant.

Pour mieux visualiser les résultats, on va les transférer dans un histogramme empilé. Voici ci-dessous l ?histogramme pour le temps de calcul moyen.

Et enfin un dernier histogramme représentant le volume moyen inutilisé par taille d ?exemplaire et du nombre de boites.

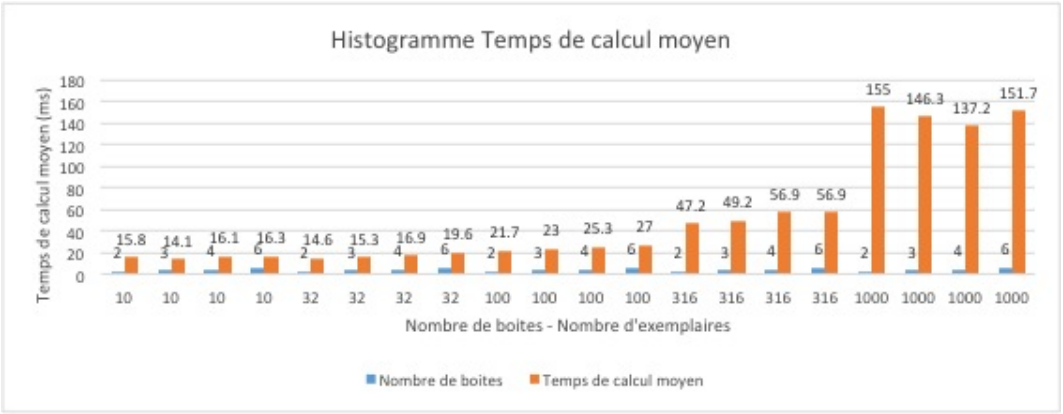
3.2 Algorithme dynamique

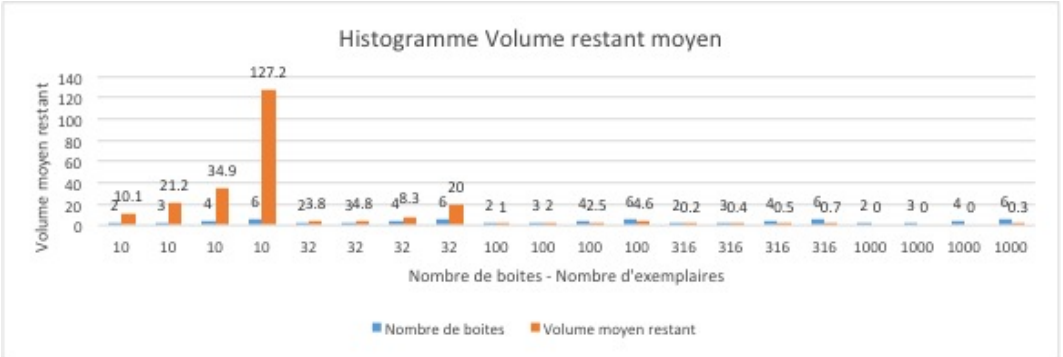
Comme annoncé précédemment, l'algorithme dynamique n'a tourné que sur des exemplaires de taille réduite, sa vitesse étant faible. On a obtenu les résultats suivants (cf .csv attaché) :

- Pour n=10 et m=2, le temps moyen est de l'ordre de 0,6s, le volume restant proche de 0 est
- Pour n=10 et m=3, on passe brusquement à un temps juste sous la minute et le volume restant moyen augmente (environ 15) ;
- Pour n=32 et m=2, on repasse à un temps moyen d'environ 3s et le volume restant moyen est nul.

Tableau 1 : Résultats Algorithme vorace

Taille exemplaire	Nombre de boites	Temps de calcul moyen	Volume moyen restant
10	2	15,8	10,1
10	3	14,1	21,2
10	4	16,1	34,9
10	6	16,3	127,2
32	2	14,6	3,8
32	3	15,3	4,8
32	4	16,9	8,3
32	6	19,6	20
100	2	21,7	1
100	3	23	2
100	4	25,3	2,5
100	6	27	4,6
316	2	47,2	0,2
316	3	49,2	0,4
316	4	56,9	0,5
316	6	56,9	0,7
1000	2	155	0
1000	3	146,3	0
1000	4	137,2	0
1000	6	151,7	0,3





4. Analyse des résultats

4.1 Algorithme vorace

Au vu des histogrammes présentés pour l ?algorithme vorace, on remarque bien que le temps de moyen de calcul augmente avec le nombre d ?exemplaires à traiter mais pour une taille fixe, le temps de calcul ne varie pas beaucoup selon le nombre de boites (quelques millisecondes).

Concernant le temps de calcul, on pouvait s ?y attendre car pour un nombre n d ?items et un nombre m de boites, notre fonction de placement possède une analyse asymptotique de $\Theta(n * m)$. Le nombre de boites de variant pas énormément, c ?est bien le nombre d ?items qui régit le temps de calcul de notre algorithme.

Sur le deuxième histogramme (représentant le volume total inutilisé) , on peut conclure sur le fait que l ?algorithme n ?est pas très optimal pour des petites tailles d ?exemplaires alors qu ?à partir de 100 exemplaires le volume total inutilisé converge fortement vers 0. Mais de manière l ?algorithme vorace est assez efficace, c ?est assez logique car les principaux atouts de ces algorithmes sont qu ?ils ont généralement une complexité de $\Theta(n \log(n))$ donc assez rapide et que leur fonctionnement se base sur la recherche d ?une solution optimale locale.

4.2 Dynamique

Le facteur qui influe le plus sur le temps d'exécution de cet algorithme est le nombre de boîtes. Par manque de temps encore, les coefficients numériques manquent.

Enfin, l'effet de n sur le volume restant est comme précédemment positif : on a une solution optimale !

5. Conclusion

En définitive, des algorithmes examinés, on utiliserait le dynamique seulement dans les cas où le nombre de boîtes est faible et on se contenterait de l'algorithme vorace pour les autres cas, son efficacité suffisant dans la plupart des cas d'utilisation.