INF4705 — Analyse et conception d'algorithmes

TP 2

Ce travail pratique se répartit sur deux séances de laboratoire et porte sur un problème de mise en boîte :

> Étant donné n items, chacun d'un certain volume (un entier positif) v_i $(1 \leq i \leq n)$, et m boîtes identiques de capacité c, trouvez un sous-ensemble des items de volume total maximum qu'on puisse mettre dans l'ensemble des boîtes sans dépasser leur capacité.
>
> Évidemment, on ne peut pas fragmenter les items (ce serait trop facile). On suppose également que $v_i \leq c$ $(1 \leq i \leq n)$ et que $n \geq m$.

Voici un exemple d'application réelle de ce problème. Chaque semaine, un envoi de m barres de marbre est reçu par une compagnie qui les découpe puis les transforme en divers produits. Ces barres sont de taille uniforme et beaucoup plus longues que larges. Chaque produit i requiert d'abord une section de la barre d'une certaine longueur v_i, qui est ensuite transformée. Selon son inventaire courant, la compagnie dresse une liste des n produits qu'elle souhaiterait fabriquer. De cette liste, on doit choisir l'ensemble des produits à fabriquer qui minimisera les pertes de marbre pour les m barres.

1 Implantation

Trois algorithmes seront implantés, mettant en pratique les patrons de conception : vorace, programmation dynamique et recuit simulé.

1.1 Algorithme vorace

Vous avez vu en classe ce qu'est un algorithme vorace : celui-ci fait un choix à chaque étape, possiblement basé sur les choix précédents mais pas sur les choix à venir, et ne revient jamais sur ce choix.

Vous devez d'abord implanter l'algorithme vorace "Best-Fit-Decreasing" pour résoudre ce problème de mise en boîte. Il procède ainsi :

> On choisit les items en ordre décroissant de volume, plaçant chacun dans la boîte dont la capacité résiduelle est la plus petite après l'y avoir mis (ou laissant l'objet de côté s'il n'entre dans aucune boîte).

Contrairement au problème d'arbre sous-tendant minimum, il ne faut pas s'attendre ici à ce que votre algorithme trouve une solution optimale à tout coup.

1.2 Algorithme de programmation dynamique

Vous devez également implanter un algorithme de programmation dynamique. Définissons $V[i][(\mu_0, \mu_1, \mu_2, \ldots, \mu_{c-1}, \mu_c)]$ comme le volume total maximum qu'on puisse mettre dans μ_0 boîtes de capacité 0, μ_1 boîtes de capacité

1, μ_2 boîtes de capacité 2, ..., μ_{c-1} boîtes de capacité $c-1$ et μ_c boîtes de capacité c, en utilisant des items parmi les i premiers. La solution recherchée correspond alors à $V[n][(0,0,\ldots,0,m)]$. La récurrence servant à remplir notre tableau à deux dimensions sera la suivante :

$$V[i][(\mu_0,\ldots,\mu_c)] = \max\{V[i-1][(\mu_0,\ldots,\mu_c)], \max_{v_i \leq k \leq c}\{v_i + V[i-1][(\mu'_0,\ldots,\mu'_c)]\}\}$$

où

$$\begin{aligned} \mu'_j &= \mu_j - 1 && \text{si } j = k \\ \mu'_j &= \mu_j + 1 && \text{si } j = k - v_i \\ \mu'_j &= \mu_j && \text{sinon} \end{aligned}$$

La première alternative de la récurrence correspond à la décision de ne pas utiliser l'item i alors que chacune des alternatives suivantes correspond à une décision de placer i dans une boîte de capacité v_i, $v_i + 1$, ..., $c-1$ ou c, respectivement. Considérons $V[4][(2,1,2,0)]$: il y a deux boîtes de capacité 0, une boîte de capacité 1 et deux boîtes de capacité 2. Supposons que $v_4 = 1$ et que nous examinions l'alternative correspondant à placer l'item 4 dans une boîte de capacité $k = 2$. Nous devrons consulter la case $V[3][(2,2,1,0)]$ puisque le sous-problème possède encore deux boîtes de capacité 0 mais maintenant une boîte supplémentaire de capacité 1 et une boîte de moins de capacité 2.

Le tableau ainsi défini contiendra un grand nombre de colonnes. À première vue, il semble y en avoir $(m+1)^{c+1}$ mais nous pouvons considérablement réduire ce nombre en réalisant que $\Sigma_{j=0}^{c}\mu_j = m$ (il y a m boîtes au total). On peut ne générer que les colonnes "légales" : il y en aura $\binom{m+c}{c} = \frac{(m+c)!}{c!m!}$ (voir les "weak compositions into a fixed number of parts" à `https ://en.wikipedia.org/wiki/Composition_(combinatorics)`).
Par exemple, avec $m = 5$ boîtes de capacité $c = 3$, nous aurons 56 colonnes légales dont $(5,0,0,0)$, $(4,1,0,0)$, $(3,2,0,0)$, $(2,3,0,0)$, $(1,4,0,0)$, $(0,5,0,0)$, $(4,0,1,0)$, $(3,1,1,0)$, $(2,2,1,0)$, ... plutôt que 1296.

1.3 Recuit simulé

Enfin vous devez implanter une méta-heuristique basée sur le recuit simulé (Algorithm 1). La fonction *voisin* renvoie une solution voisine de la solution passée en argument. Une solution voisine est obtenue en choisissant uniformément au hasard un item parmi ceux qui ne sont pas encore placés et en l'insérant dans une des m boîtes, choisie uniformément au hasard. Il est possible que cela fasse déborder cette boîte : on retirera alors un item choisi uniformément au hasard dans celle-ci tant que sa capacité c n'est pas respectée. Notez qu'il est possible de calculer la variation du volume total Δ de manière incrémentielle.

Algorithm 1: Recuit Simulé

input: I /* Ensemble des items */,m /* Nombre de boîtes */, c /* Capacité des boîtes */, k_{max} /* Nombre de pas maximum */, T /* Temperature initiale */, P /* Palier de refroidissement */, α /* Coefficient de refroidissement */

output: $\langle I_1, I_2, \ldots, I_m \rangle$, un m-tuple de sous-ensembles disjoints de I décrivant le contenu des boîtes

```
1  S ← ⟨∅, ∅, ..., ∅⟩;
2  S_meilleur ← S;
3  θ_1 ← T;
4  for k = 1 ... k_max do
5  |  for j = 1 ... P do
6  |  |  Considérer un voisin au hasard : S' ← voisin(S);
7  |  |  Calculer Δ ← volume(S') − volume(S);
8  |  |  if CritereMetropolis(Δ, θ_k) then
9  |  |  |  S ← S';
10 |  |  |  if volume(S) > volume(S_meilleur) then
11 |  |  |  |  S_meilleur ← S;
12 |  Mettre à jour la température : θ_{k+1} ← θ_k × α;
13 retourner S_meilleur;
```

Algorithm 2: CritereMetropolis

input: Δ, θ

output: Vrai, Faux

```
1  if Δ ≥ 0 then
2  |  retourner Vrai;
3  if exp(Δ/θ) ≥ Unif([0,1]) then
4  |  retourner Vrai;
5  retourner Faux;
```

2 Jeu de données

Vous trouverez sur le site Moodle du cours des exemplaires à traiter. Les fichiers portent le nom "ex_nN_mM.X" où N = 10, 32, 100, 316 ou 1000 (ce qui représente le nombre d'items) ; M = 2, 6, 20, 63, 200 ou 632 (ce qui représente le nombre de boîtes) ; et X = 0,..,9 (le numéro d'exemplaire). Nous aurons 10 exemplaires pour chacune des 20 classes (N, M) possibles : (10,2), (10,6), (32,2), (32,6), (32,20), (100,2), (100,6), (100,20), (100,63), (316,2), (316,6), (316,20), (316,63), (316,200), (1000,2), (1000,6), (1000,20), (1000,63), (1000,200) et (1000,632).

Chaque fichier débute avec la valeur des paramètres n, m et c sur la première ligne. Les lignes suivantes énumèrent le volume occupé par chacun des items. Les valeurs sont séparées par des espaces. En voici un exemple :

10 2 20
11 8 4 6 19 12 16 2 5 14

Vous ne pourrez pas résoudre les plus gros exemplaires avec l'approche par programmation dynamique. Prenez soin d'évaluer l'espace mémoire qui vous sera nécessaire avant de lancer une exécution !

3 Résultats

Pour chacune des approches, mesurez le temps d'exécution pour chaque exemplaire et rapportez dans un tableau le temps moyen par taille d'exemplaire. Vous devrez borner à 5 minutes le temps maximal alloué pour la résolution d'un exemplaire de problème. Rapportez également le volume total inutilisé moyen par taille d'exemplaire. Vous pourrez ainsi comparer la qualité des solutions trouvées par les trois algorithmes.

4 Analyse et discussion

1. Tentez une analyse asymptotique du temps de calcul pour chaque algorithme.
2. Servez-vous de vos temps d'exécution pour confirmer et/ou préciser l'analyse asymptotique théorique de vos algorithmes avec la méthode hybride de votre choix (cette méthode peut varier d'un algorithme à l'autre). Justifiez ces choix.
3. Discutez des trois algorithmes en fonction de la qualité respective des solutions obtenues, de la consommation de ressources (temps de calcul, espace mémoire) et de la difficulté d'implantation.
4. Indiquez sous quelles conditions vous utiliseriez l'un de ces algorithmes plutôt que les deux autres.

5 Remise

Avant votre cinquième séance de laboratoire, vous devez faire une remise électronique en suivant les instructions sur le site du cours. Votre répertoire devra comprendre les éléments suivants :

1. un rapport comprenant :
   - une brève description du sujet et des objectifs de ce travail (svp pas de redite de l'énoncé),
   - la description des jeux de données,
   - les résultats expérimentaux,
   - l'analyse et discussion
2. les trois exécutables nommés : vorace, dynamique et recuitSimule.
3. un fichier *ReadMe.txt* indiquant les arguments à entrer en ligne de commande lors de l'exécution de vos programmes.

Les exécutables doivent respecter le strandard suivant :

1. Chaque exécutable doit avoir un des noms mentionné plus haut.
2. Chaque exécutable doit prendre les paramètres suivant : -f chemin vers l'exemplaire et -p pour imprimer le résultat.
3. Chaque exécutable, lorsqu'il est exécuté sans le paramètre -p doit afficher le temps de calcul et le volume total. Si l'option -p est ajoutée, le programme doit afficher en plus le contenu de chaque boîte sur une ligne distincte.

Si le standard n'est pas respecté, des pénalités s'ensuivront. Ce format est cependant pratique, car il vous permettra de standardiser facilement vos tests, avec un script par exemple. Par ailleurs, ne remettez pas les fichiers de données (exemplaires).

6 Barème de correction

1 pts : exposé du travail pratique

2 pts : présentation des résultats

5 pts : analyse et discussion

3 pts : les programmes (corrects,structurés,commentés,...)

2 pts : présentation générale et qualité du français