TP01 : Le problème du sac à dos et son application à un exemple concret en 3D.

Binôme :

- Mokrane Kaouthar
- Khettache Nour El Houda

Année : 2018/2019

1. Introduction

De nombreux problèmes, de la vie de tous les jours, peuvent être modélisés sous la forme de problème d'optimisation combinatoire de manière que cette modélisation soit la plus proche possible d'un modèle théorique de la littérature pour lequel différentes approches de résolution ont été proposée.

Le problème du sac à dos étant un problème classique parmi les différents modèles théoriques qu'on peut les citer représente une difficulté à résoudre d'une manière optimale en un temps raisonnable, le but de ce TP est de proposer des solutions informatiques optimale à ce type de problème (maximisation), en comparant les différents temps d'exécution des différentes approches.

2. L'énoncé du TP :

Le problème du sac à dos est un problème d'optimisation combinatoire. Le but étant de trouver un sous ensemble d'objets de valeur totale maximale tout en restant soumis à certaines contraintes. Dans le cadre de l'étude du chargement d'un camion de livraison, le problème est défini de la sorte :

• Le camion possède une capacité maximale 520 kg et un volume maximal 3m3 ;

• Les objets sont des tuples (« nom », poids, valeur, volume) présentés sous forme de liste ;

• On dispose d'une liste de 20 objets de masse totale 693 kg et 3,47m3.

(Le tableau des objets se trouve dans l'énoncé du TP donné par l'enseignant).

3. L'implémentation de la solution :

a. Le problème du sac à dos simple

Afin de résoudre ce problème d'optimalité on revient vers le problème du sac à dos simple (à une seule contrainte), on peut formuler ce problème comme suit :

« Etant donné plusieurs objets possédants chacun un poids et une valeur et étant donné un poids maximum pour le sac, quels objets faut-il mettre dans le sac de manière à maximiser la valeur totale sans dépasser le poids maximal autorisé pour le sac ? »[1]

Pour la formulation mathématique on a :

$$\text{Maximiser } Z(x) = \sum_{j=1}^{n} v_j x_j$$

Sous la condition $\sum_{j=1}^{n} p_{ij} x_j \leq C_i \quad i = 1,2 \ldots., m$

$x_j \in \{0,1\} \quad j = 1,2 \ldots \ldots \ldots, n$

Tel que :

- Les $\boldsymbol{x_j}$ Sont des variables de décision quand $\boldsymbol{x_j} = \mathbf{0}$: on ne prend pas l'objet j et quand $\boldsymbol{x_j} = \mathbf{1}$: on prend l'objet j
- $\boldsymbol{Z(x)}$ est la fonction à maximiser
- $\boldsymbol{v_j}$ Représente la valeur de l'objet j
- $\boldsymbol{p_j}$ Représente le poids de l'objet j
- $\boldsymbol{Pmax}$ est le poids maximal du sac à dos
- Et $\sum_{j=1}^{n} \boldsymbol{p_j x_j} \leq \boldsymbol{Pmax}$ est la condition à respecter

[1] **https://interstices.info/le-probleme-du-sac-a-dos/**

b. Le problème du sac à dos multidimensionnel :

Ce type de problème du sac à dos est l'objet de ce TP, à partir de son nom on peut facilement déduire qu'il s'agit un problème multidimensionnel, c'est à dire plus d'une contrainte, c'est une généralisation de problème de sac à dos simple.

La formulation mathématique générale de ce problème est la suivante :

$$\text{Maximiser } Z(x) = \sum_{j=1}^{n} v_j x_j$$

Sous la condition $\sum_{j=1}^{n} p_j x_j \leq Pmax$

$x_j \in \{0,1\} \quad j = 1,2 \dots \dots \dots, n$

Le **m** représente le nombre de contraintes et **i** l'indice de cette contrainte.

Dans notre cas on a deux contraintes : une sur le poids et l'autre sure le volume donc la formule devient :

$$\text{Maximiser } Z(x) = \sum_{j=1}^{n} v_j x_j$$

Les conditions

$\sum_{j=1}^{n} p_j x_j \leq Pmax$ Et $\sum_{j=1}^{n} vol_j x_j \leq Vmax$

$x_j \in \{0,1\} \quad j = 1,2 \dots \dots \dots, n$

Pour la résolution du problème, on adopte les deux approches suivantes :

Par récurrence

C'est la solution la plus intuitive pour résoudre ce problème. L'algorithme nous assure que le gain retourné est le plus optimale.

Le principe de la résolution du problème à n éléments revient à résoudre le problème à n-1 éléments et ainsi de suite, jusqu'à ce qu'on arrive au cas trivial.

La formulation du la solution est bien la suivante :

$$F\,(n,P,V)\begin{cases} F(n-1,P,V) & Si\ l'objet\ "n"\ n'est\ pas\ pris \\ F(n,P-p_n,V-vol_n)+\ v_n & sinon \end{cases}$$

Le programme en C :

```c
/****Solution réccursive****/
int sacADosReccursive(int nbElem,int tabVal[],int tabPd[],int pdMax,int tabVl[],int vlMax){
  int k,v,z;
   if (nbElem == 0 || pdMax == 0||vlMax==0)
        {return 0;}
   else
   {
       if (tabPd[nbElem-1] > pdMax||tabVl[nbElem-1] >vlMax)
       {

         return sacADosReccursive(nbElem-1, tabVal,tabPd,pdMax,tabVl,vlMax);
       }
       else  {
         v= tabVal[nbElem-1] + sacADosReccursive(nbElem-1, tabVal,tabPd,pdMax-tabPd[nbElem-1],tabVl,vlMax-tabVl[nbElem-1]);
         z=sacADosReccursive(nbElem-1, tabVal, tabPd,pdMax,tabVl,vlMax);
         k=max(v,z);
         return k;
        }
}
}
```

Le résultat après l'exécution de l'exemple du TP :

```
Execution reccurcive:
 Gain max = 183
 Temps Execution:15 ms
```

Par programmation dynamique :

Pour l'approche récursive on trouve qu'on résout le même problème plusieurs fois c'est à dire on calcule la même valeur de la fonction plusieurs fois. Ce qui serait intéressant est de mémoriser les valeurs calculées dans une structure afin d'éviter de les recalculer à chaque fois c'est l'approche de programmation dynamique.

Le programme en C :

```
int sacADosProgDynamique(int nbElem,int tabVal[],int tabPd[],int pdMax,int tabVl[],int vlMax){
int i,p,v,x,y,z;

int (*tab)[pdMax+1][vlMax+1];

tab = malloc((nbElem+1) * sizeof *tab);

for (i = 0; i <= nbElem; ++i)
{
  for (p = 0; p <= pdMax; ++p){

    for(v=0;v<=vlMax;++v) {

    if(i==0||p==0||v==0) {tab[i][p][v]=0;}
    else {
                y=tab[i-1][p][v];
               if(tabPd[i-1]>p||tabVl[i-1]>v)   {tab[i][p][v]=y;}
               else{
                       x=tab[i-1][p-tabPd[i-1]][v-tabVl[i-1]]+tabVal[i-1];
                       tab[i][p][v]=max(x,y);
                    }
          }

          }
}
} x=tab[nbElem][pdMax][vlMax];
free(tab);
return x;
}
```

Le résultat après l'exécution de l'exemple du TP :

```
Execution programmation dynamique:
 Gain max = 183
 Temps Execution:31 ms
```

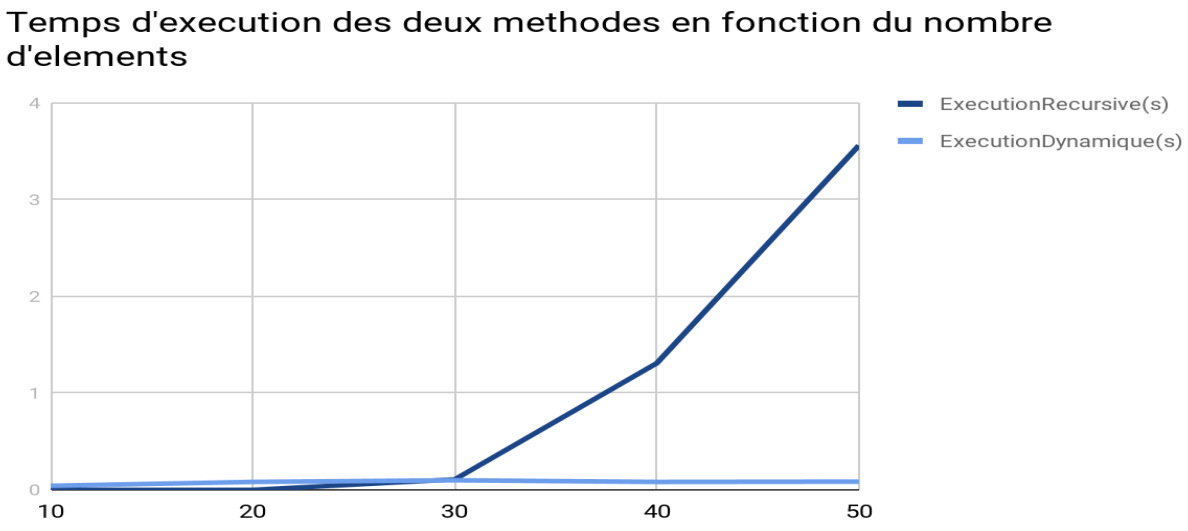
4.La comparaison entre les deux approches

Pour la comparaison on exécute le programme plusieurs fois avec un nombre d'objet différent à chaque fois et on récupère le temps d'exécution :

Tableau récapua

NombreD'Elements	ExecutionRecursive(s)	ExecutionDynamique(s)
10	0	0.042
20	0	0.082
30	0.108	0.099
40	1.31	0.082
50	3.561	0.085

Le graphe associé :



Interprétation

On constate que les deux approches sont des méthodes exactes parce qu'ils arrivent à une solution optimale, dans notre cas, un gain de 183.

De plus, d'après le graphe ci-dessous, et par rapport au temps d'exécution, on remarque que la deuxième méthode (programmation dynamique) est bien plus efficace que la première méthode (récursive), quand le nombre des objets est grand ce qui est justifié, car la première recalcule à chaque fois toutes les valeurs mêmes si elles étayent déjà calculées (Le temps d'exécution augmente dans ce cas exponentiellement.)

Or que l'approche par programmation dynamique les enregistres dans une structure afin de les utiliser pour le calcul des nouvelles valeurs. (Le temps d'exécution augmente dans ce cas linéairement. Il dépend du poids et volumes maximaux.)

La complexité des deux approches est par conséquent comme suit :

Première méthode (Programme récurrent) = $o(2^n)$

Deuxième méthode (Programme dynamique) = $o(nbElem * pdMax * vlMax)$