TD4

évaluation algorithmique

Mathieu BONNEAU Antoine FEUILLETTE Tom MAFILLE Paul MATHÉ

---

1. Formalisation

1. On définit les variables comme suit :
   - x_n le nombre de camions de capacité C_n choisis, avec $n \in N = \{1, 2, 3\}$.

2. L'objectif est d'optimiser deux critères :
   - **Minimiser le nombre de camions utilisés** : On cherche à minimiser le nombre total de camions utilisés chaque jour, c'est-à-dire $\sum_{n \in N} x_n$.
   - **Minimiser le coût total en carburant** : On cherche également à minimiser le coût en carburant, c'est-à-dire $\sum_{n \in N} P_n \cdot x_n$. Les contraintes sont les suivantes :
   - **Contraintes de capacité des camions** : Le nombre total de palettes à livrer doit être réparti entre les camions, en tenant compte de leurs capacités respectives. Si N_1, N_2, et N_3 sont les nombres de palettes des types T_1, T_2 et T_3, on a :

     $\sum_{n \in N} C_n \cdot x_n \geq \sum_{i=1}^{3} N_i \cdot T_i$
   - **Contraintes sur le nombre de camions** :

     $x_n \geq 0 \quad \forall n \in N$

     Et le nombre total de camions utilisés ne doit pas dépasser le nombre de camions disponibles :

     $\sum_{n \in N} x_n \leq M$
   - **Contraintes sur la capacité d'un camion seul** :

     $\forall i, C_i \geq \sum_{j=1}^{3} t_{i,j} \cdot T_j$

     Avec $t_{i,j}$ le nombre de palettes T_j dans le conteneur C_i.

On cherche donc à minimiser le coût total en carburant et le nombre de camions, soit :

- $\min\left(\sum_{n \in N} P_n \cdot x_n\right)$ **(1)**

et

- $\min\left(\sum_{n \in N} x_n\right)$ **(2)**

La formalisation présente quelques problèmes :

- On cherche à optimiser le nombre de palettes, mais pas le volume occupé. En effet, si on a beaucoup de petites palettes, de telle sorte qu'une palette de taille T_1 équivaut à 1000 palettes de taille T_3, on préferera charger au total 500 palettes T_3 plutôt qu'une palette T_1, ce qui n'est pas l'objectif. Pour résoudre le problème, on prend le parti de considérer pour la suite CAP_T qui représentera la capacité en terme de palettes : par exemple, pour 1 palette T1 qui "vaut" 2 palettes T2 et 1 palette T2, on aura $CAP_T = 1 \cdot 2 + 1 = 3$.

2. Méthode de résolution

1. On identifie un problème du sac à dos pour le remplissage d'un camion d'une capacité fixée : on aurait donc n problèmes du sac à dos différents. C'est ce n qu'on cherche à minimiser. Le problème peut donc être décomposé en sous-problèmes indépendants, qui consiste à déterminer comment répartir les palettes entre les camions de capacité C_1,C_2,C_3. Les solutions optimales des sous-problèmes (répartition pour une capacité donnée) contribuent à la solution optimale globale.

   On utilisera pour répondre au problème une méthode de programmation dynamique.

   Les avantages de cette méthode sont multiples:

   - `Comparé aux algorithmes gloutons, la programmation dynamique garantit une solution optimale.`

   - `Moins complexe à mettre en œuvre qu'un algorithme de branch-and-bound.`

2. Voici un jeu de données arbitraires :

---

1. Nombre de palettes (N) et leur répartition par type :

- T_1 (petites palettes) : $N_1 = 30$
- T_2 (moyennes palettes) : $N_2 = 20$
- T_3 (grandes palettes) : $N_3 = 10$

La **capacité de palettes** à livrer est donné par :

$CAP_T = N_1 \cdot T_1 + N_2 \cdot T_2 + N_3 \cdot T_3$

Si $T_1 = 1$, $T_2 = 2$, et $T_3 = 3$, alors :

$CAP_T = 30 \cdot 1 + 20 \cdot 2 + 10 \cdot 3 = 100$

2. Caractéristiques des camions disponibles (M) : Nous avons trois types de camions :

- C_1 : capacité de 10 palettes, coût $P_1 = 100$€.
- C_2 : capacité de 20 palettes, coût $P_2 = 180$€.
- C_3 : capacité de 30 palettes, coût $P_3 = 250$€.

Nombre total de camions disponibles (M) :

$M = 10$ (répartis selon les types ci-dessous) .

Répartition des camions :

- 4 camions de type C_1,
- 3 camions de type C_2,

- 3 camions de type C_3.

3. Résumé des données :

Type de palettes	**Nombre (N_i)**	**Taille (T_i)**
T_1 (petites)	30 palette	1
T_2 (moyennes)	20 palettes	2
T_3 (grandes)	10 palettes	3

Type de camion	**Capacité (C_i)**	**Coût (P_i)**	**Nombre disponible**
C_1	10 palettes	100 €	4
C_2	20 palettes	180 €	3
C_3	30 palettes	250 €	3

Rappel des contraintes :

- La capacité totale des camions utilisés doit être supérieure ou égale au nombre total de palettes :

  $C_1 \cdot x_1 + C_2 \cdot x_2 + C_3 \cdot x_3 \geq 100$

- Le nombre de camions utilisés ne doit pas dépasser le nombre disponible pour chaque type :

  $x_1 \leq 4,\ x_2 \leq 3,\ x_3 \leq 3$

- Toutes les variables doivent être positives :

  $x_1, x_2, x_3 \geq 0$

---

3. **Résultat final par résolution intuitive** :

   - *Principe du parcours en arrière (Backtracking) utilisé* :

   L'idée est de partir du volume total et de "remonter" les décisions prises lors de la mise à jour du tableau .

   1. Initialisation : On initialise à . On garde un compteur pour chaque type de camion (par exemple, , ,) pour compter le nombre de camions utilisés.
   2. Parcours en arrière : À chaque étape, on vérifie quel camion a permis d'atteindre le coût optimal pour le volume actuel. Si le tableau a été mis à jour par un camion de capacité avec un coût , cela signifie que ce camion a été utilisé pour atteindre ce volume.
   3. Mise à jour des compteurs : On incrémente le compteur correspondant au camion . On réduit de (la capacité du camion utilisé).
   4. Répétition : On répète ce processus jusqu'à ce que (volume nul signifie que tout le volume a été attribué aux camions).

La solution optimale pour transporter $CAP_T = 100$ palettes avec les coûts minimaux est :

- $x_1 = 1$,
- $x_2 = 0$,

- $x_3 = 3$.

D'où le coût total minimal : 850 €.

3. Implémentation :

0. **Explication du code** :

- *Étape 1* : Calcul du volume total à transporter

  Le volume total est donné par :

  $V_{\text{total}} = N_1 \cdot T_1 + N_2 \cdot T_2 + N_3 \cdot T_3$

  En remplaçant les valeurs :

  $V_{\text{total}} = 30 \cdot 1 + 20 \cdot 2 + 10 \cdot 3 = 30 + 40 + 30 = 100$ unités de volume.

- *Étape 2* : Programmation dynamique

  On utilise un tableau où représente le coût minimal pour atteindre un volume .

  Initialisation :

  (aucun coût pour un volume de 0).

  pour (volume non encore atteint).

- *Transition* : Pour chaque camion (capacité , coût), on met à jour pour chaque volume en sens inverse (pour éviter d'utiliser plusieurs fois le même camion) :

  $dp[v] = \min(dp[v], dp[v - C_i] + P_i)$

1. **Pseudo code** :

```
#DONNEES
Camions = {{10, 100},{10, 100}, {10, 100},{10, 100},{20, 180},{20, 180},{20, 180} ,{30, 250},{30, 250}

Fonction Minimiser_Cout_Camions(V_total, Camions, nb_camions)
    Entrée :
        V_total : Volume total à transporter
        Camions : Tableau contenant {capacité, coût} pour chaque camion
        nb_camions : Nombre total de camions disponibles

    Sortie :
        Coût minimal pour transporter V_total, ou -1 si impossible

    // Étape 1 : Initialisation
    Créer un tableau dp de taille V_total + 1
    Pour v allant de 0 à V_total :
        dp[v] ← INF  // INF représente un coût infini
    dp[0] ← 0  // Coût pour un volume nul est 0

    // Étape 2 : Mise à jour avec les camions disponibles
    Pour chaque camion i allant de 0 à nb_camions - 1 :
        capacite ← Camions[i][0]  // Capacité du camion
        cout ← Camions[i][1]      // Coût du camion

        Pour v allant de V_total à capacite (en décrémentant) :
            Si dp[v - capacite] != INF Alors
```

```
                dp[v] ← Min(dp[v], dp[v - capacite] + cout)

    // Étape 3 : Résultat final
    Si dp[V_total] = INF Alors
        Retourner -1  // Impossible de transporter le volume total
    Sinon
        Retourner dp[V_total]  // Coût minimal pour atteindre V_total
```

2. **En langage C** :

```c
#include <stdio.h>
#include <limits.h>

#define MAX_VOLUME 101  // Volume total à transporter + 1 pour les indices
#define INF INT_MAX     // Représentation d'une valeur infinie

// DONNEES
Camions = {{10, 100},{10, 100}, {10, 100},{10, 100},{20, 180},{20, 180},{20, 180} ,{30, 250},{30, 250}

int min(int a, int b) {
    return (a < b) ? a : b;
}

int minimiser_cout_camions(int V_total, int camions[][2], int nb_camions) {
    int dp[MAX_VOLUME];

    // Initialisation : tous les coûts à l'infini sauf dp[0] = 0
    for (int i = 0; i <= V_total; i++) {
        dp[i] = INF;
    }
    dp[0] = 0;

    // Programmation dynamique
    for (int i = 0; i < nb_camions; i++) {
        int capacite = camions[i][0];
        int cout = camions[i][1];

        // Mise à jour du tableau dp pour chaque camion
        for (int v = V_total; v >= capacite; v--) {
            if (dp[v - capacite] != INF) {
                dp[v] = min(dp[v], dp[v - capacite] + cout);
            }
        }
    }

    // Le résultat minimal pour atteindre le volume total
    return dp[V_total] == INF ? -1 : dp[V_total];
}

int main() {
    // Données du problème
    int V_total = 100; // Volume total à transporter

    // Tableau des camions : {capacité, coût}
    int camions[][2] = {
        {10, 100}, // Camion 1 : capacité 10, coût 100
```

```
        {20, 180}, // Camion 2 : capacité 20, coût 180
        {30, 250}  // Camion 3 : capacité 30, coût 250
    };

    int nb_camions = sizeof(camions) / sizeof(camions[0]);

    // Appel de la fonction pour minimiser le coût
    int resultat = minimiser_cout_camions(V_total, camions, nb_camions);

    // Affichage du résultat
    if (resultat != -1) {
        printf("Le coût minimal pour transporter les palettes est : %d\n", resultat);
    } else {
        printf("Impossible de transporter le volume demandé avec les camions disponibles.\n");
    }

    return 0;
}
```