CHAPITRE 2

EXEMPLE DE PROBLÈMES D'OPTIMISATION COMBINATOIRE

Sommaire

Introduction

La *Combinatoire* en mathématique est une branche dont le but est de dénombrer les dispositions que l'on peut former à l'aide des éléments d'un ensemble fini .

La définition de l'optimisation combinatoire est liée directement par l'espace de recherche du problème . Le but de la résolution d'un tel problème est de chercher l'élément optimal à partir d'un ensemble fini dénombrable. Cet objet peut être un *nombre entier*, un *sous-ensemble*, ou une *structure de graphe* .

Ce chapitre présente d'abord la démarche de l'optimisation combinatoire (Sivazlian, 2009). Ensuite, quelques exemples classiques de l'optimisation combinatoire seront expliqués(Belhoul, 2014), (Ouaarab, 2015), (Lobjois, 1999), (Korte *et al.*, 2010), (Laribi, 2018), (Verfaillie, 2008).

1 Démarche en Optimisation Combinatoire

1.1 Exemple

C'est la fin de l'année ! Les étudiants doivent se préparer pour passer leurs examens. Omar passera 3 modules et ne possède que 100 heures pour la révision. Il révise pendant t_i heures le module i $(i = 1, 2, 3)$. On suppose que la note n_i du module i dépend uniquement du temps t_i passé à réviser. Supposons que :

$$n_1 = 2.t_1, \qquad n_2 = 2.t_2 - 4, \qquad n_3 = 3/2.t_3$$

Omar veut optimiser sa tâche et avoir la moyenne maximale en respectant la contrainte de temps. Les variables de décisions x_i pour ce problème représentent les temps de révision (t_i), estimés en minutes, attribués à chaque module . Le problème revient à trouver $\overline{x} = (\overline{x}_1, \overline{x}_2, \overline{x}_3)$ qui donne le maximum de moyenne $f(x) = (2.x_1 + 2.x_2 - 4 + 3/2.x_3)/3$ où $x = (x_1, x_2, x_3) \in X = \{x_1, x_2, x_3 \in \mathbb{N}, : x_1 + x_2 + x_3 \leqslant 100 * 60; \}$.

Le problème est alors noté comme suit : $\max_{x \in X} f(x)$.

1.2 Démarches

L'optimisation suit les mêmes démarches de travail de la Recherche Opérationnelle . La figure 2.1 résume les différentes étapes pour résoudre un problème d'optimisation combinatoire.

FIGURE 2.1 – Démarche de résolution d'un problème d'optimisation

1.2.1 Identification

Cette phase est cruciale et très importante au bon fonctionnement des étapes suivantes. A ce stade on tire les informations nécessaires à la modélisation du problème :

1. **Variables de décision :** Identification des variables du problème d'optimisation. Les variables de décision dans l'exemple 1.1 sont présents dans le vecteur du temps $x = (x_1, x_2, x_3)$.
2. **Critère :** Définition d'une fonction objectif permettant d'évaluer l'état du système ainsi que le besoin de maximisation ou de minimisation (ex : rendement, performance, coût...). La fonction de la moyenne $f(x) = (2.x_1 + 2.x_2 - 4 + 3/2.x_3)/3$ est la fonction objectif de l'exemple 1.1.
3. **Contraintes :** Description des contraintes imposées aux variables de décision. Les contraintes tirées de l'exemple 1.1 sont :

$$x_1, x_2, x_3 \in \mathbb{N}$$
$$x_1 + x_2 + x_3 \leqslant 100 * 60$$

1.2.2 Modélisation

Dans cette étape, on définit les équations nécessaires pour donner une description mathématique au problème. La modélisation peut être mathématique (linéaire et non linéaire), graphique (sous formes de graphes, arbres, graphes bipartis etc.), etc. Dans ce cours on s'intéresse seulement à la modélisation mathématique.

On modélise un problème d'optimisation combinatoire en Programme Mathématique qui est définit comme suit :

$$\begin{array}{ll} \text{Maximiser} & f(x) \\ s.c. & \text{(sous les contraintes)} \\ & g_i(x) \leq 0 \qquad i = 1, .., m \\ & x \in X \end{array} \tag{2.1}$$

Où

— $X \subset \mathbb{R}^n$ est l'espace de recherche et n est la dimension du problème.
— $x \in X$ est le vecteur de variables de décision.
— $f : X \to \mathbb{R}$ est la fonction objectif.
— $g_i : X \to \mathbb{R}, i = 1, .., m$ sont les m égalités et inégalités qui représentent les contraintes du problème.

Selon la nature de l'ensemble X, on définit les programmes mathématiques **continus** et **discrets**. Dans le cas discret on parle de :

1. **Programme mathématique entier**, si l'espace de recherche X ne contient que des nombres entiers ($X \subset \mathbb{Z}^n$).

$$\begin{aligned} &\text{Maximiser} f(x) \\ &s.c. \\ &\quad g_i(x) \leq 0 \qquad i = 1, .., m \\ &\quad x \in \mathbb{Z}^n \end{aligned}$$

2. **Programme mathématique binaire**, si l'espace de recherche X ne contient que des nombres binaires ($X \subset \{0,1\}^n$).

$$\begin{aligned} &\text{Maximiser} f(x) \\ &s.c. \\ &\quad g_i(x) \leq 0 \qquad i = 1, .., m \\ &\quad x \in \{0,1\}^n \end{aligned}$$

3. **Programme mathématique mixte**, si l'espace de recherche X est formé de plusieurs sous-ensembles hétérogènes : des sous-ensembles avec des variables qui prennent des valeurs entières et d'autres qui prennent de valeurs continues.

$$\begin{aligned} &\text{Maximiser} f(x, y) \\ &s.c. \\ &\quad g_i(x, y) \leq 0 \qquad i = 1, .., m \\ &\quad x \in \mathbb{Z}^n \\ &\quad y \in \mathbb{R}^n \end{aligned}$$

Si la fonction objectif f et les fonctions contraintes g sont toutes linéaires (les coefficients de variables sont constants et il n'a pas de produit de variables) : on parle de **Programme linéaire**. On s'intéresse dans ce cours à la programmation linéaire discrète (entière ou mixte).

1.2.3 Résolution

Il existent plusieurs approches de résolution de problèmes d'optimisation selon l'optimalité de la solution globale. Les deux classes essentielles sont les méthodes exactes et les méthodes approchées. Cette étape sera expliquée dans les chapitres suivants.

1.2.4 Implémentation

Une fois la solution établie (via une méthode de résolution) est validée, l'implémentation du système peut être concrétisée.

2 Exemples classiques de problèmes combinatoires

On qualifie un problème de *combinatoires*, un problème dont la résolution confronte un nombre de combinaisons énorme à explorer . Ces dernières ne sont pas toutes acceptées sauf celles répondant aux exigences du problème lui même.

2.1 Le problème du sac-à-dos (knapsack)

2.1.1 Introduction

Supposant que nous prévoyons une journée à la plage. Nous devons alors remplir nos sacs à dos avec des éléments nécessaires pour le voyage. Vu que la capacité du sac à dos est limitée (un poids ou un volume maximum), on doit faire une décision pour pouvoir le remplir.

FIGURE 2.2 – Problème sac-à-dos

Chaque type d'élément est caractérisé par un poids (ou un volume) et une valeur (niveau d'importance).

2.1.2 Formalisme

On dispose d'un sac-à-dos dont le contenu ne peut pas excéder une capacité c. Considérons un ensemble de n objets, numérotés par l'indice i de 1 à n, possédant chacun un poids w_i (weight en anglais) et une valeur p_i (profit en anglais).

Le problème du sac-à-dos consiste à trouver le sous-ensemble d'objets à charger dans le sac de capacité c afin de **maximiser** la somme des profits.

2.1.3 Instances

Il existe plusieurs instances du problème sac à dos : problème de sac à dos à variables binaires, problème de sac à dos à variables entières, problème de sac à dos multidimensionnel, problème de sac à dos multi-objectif, etc.

Le problème de sac à dos permet de modéliser de très nombreux problèmes comme : problèmes de gestion de ressource, de chargement de cargaison, etc.

2.1.4 Instance problème de sac à dos à variables binaires

Cette instance appelée aussi sac à dos fractionnaire, utilise un programme linéaire binaire pour modéliser le problème sac-à-dos. Cette modélisation se présente comme suit :

1. **Variables de décision :** Les variables sont définies sur l'ensemble $0, 1$: La variable x_i du i-ème élément, $x_i = 1$ si l'élément i est mis dans le sac, $x_i = 0$ s'il est laissé de côté.

2. **Contraintes :** Le problème sac-à-dos est limité par deux contraintes. En plus de la contrainte d'intégrité $x_i \in 0, 1$, le sac-à-dos ne doit pas dépasser sa capacité c (volume ou poids) : $\sum_{i=1}^{n} w_i.x_i \leq c$.

3. **Critère :** La fonction objectif du problème sac-à-dos se définit comme la maximisation des profits des objets mis dans le sac :

$$f(x) = \sum_{i=1}^{n} p_i.x_i \tag{2.2}$$

   où p_i est le profit de l'objet i.

Résoudre le problème du sac à dos revient à trouver le vecteur solution $\overline{x} = (\overline{x}_1, .., \overline{x}_n)$ qui **maximise** la fonction objectif définie par l'équation (2.2).

Mathématiquement, un problème de sac à dos est représenté comme suit :

Soient n nombre d'objets,p_i profit de l'objet i, $w_i \in \mathbb{N}*$ poids de l'objet i et $x = (x_1, .., x_n)$ vecteur de décision.

$$\text{Maximiser} \qquad f(x) = \sum_{i=1}^{n} p_i.x_i$$

$$s.c.$$

$$x_i \in \{0, 1\}, \qquad i \in \{1, .., n\}$$

$$\sum_{i=1}^{n} w_i.x_i \leq c$$

2.2 Problème de voyageur de commerce (TSP : Travelling Salesman Problem)

2.2.1 Introduction

Un agent commercial doit visiter une liste de villes et retourner à sa ville de départ. Cette tournée entre toutes les villes doit prendre le chemin le plus cours. Le plus court chemin peut se traduire par la distance, le temps ou même le carburant.

FIGURE 2.3 – Exemple d'itinéraire d'un voyageur de commerce entre les villes : Annaba - Constantine - Souk Ahras et El Kala

Quelques conditions limitent le voyage de l'agent. Chaque ville doit être visitée une est une seule fois. Pour chaque paire de ville, on connaît la distance entre les deux villes.

La situation présentée ci-dessus définit le Problème de Voyageur de Commerce.

2.2.2 Formalisme

Considérons n villes représentées par un graphe complet [1] de n sommets, valué par les distances entre ces villes. On note ce graphe $G = (V, E, c)$ où $V = \{1, .., n\}$ est l'ensemble de sommets (de villes), E est l'ensemble d'arêtes et c est une fonction de coût qui représente la distance entre les 2 villes (c_{ij} est la distance entre les villes i et j).

Le problème du voyageur de commerce (TSP, Travelling Salesman Problem) consiste à trouver le cycle de longueur minimale [2] passant par chaque ville une et une seule fois. Un tel cycle est dit hamiltonien (appelé aussi tour ou tournée).

Pour un nombre de n villes, il existe $n!$ chemin possible. La difficulté de ce problème vient de l'explosion combinatoire du nombre de chemins possibles lorsqu'on augmente le nombre de villes à visiter.

2.2.3 Instances

Le graphe représentant le problème de voyageur de commerce peut être orienté ou non-orienté. Il est orienté si la distance entre les deux villes n'est pas la même pour l'aller et le retour ($c_{ij} \neq c_{ji}$). Dans ce cas, on parle de problème de voyageur de commerce asymétrique.

Un problème de voyageur de commerce symétrique est définit par un graphe non-orienté (c'est-à-dire $c_{ij} = c_{ji}$). La figure 2.4 représente le graphe complet non-orienté du problème de voyageur de commerce symétrique de l'exemple de la figure 2.3.

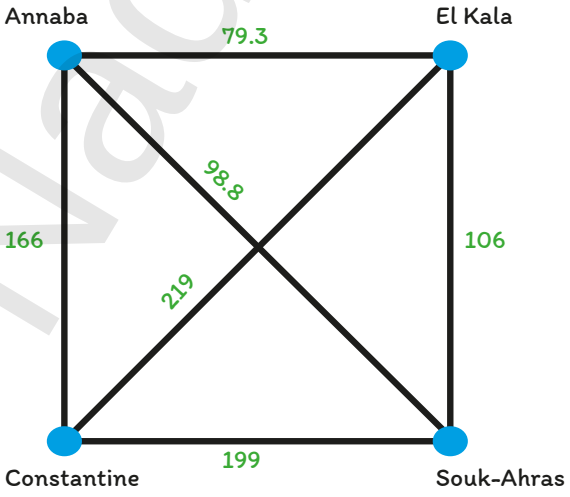


FIGURE 2.4 – Graphe complet non-orienté valué

1. Un graphe est complet si chaque sommet du graphe est relié directement à tous les autres sommets.
2. Un cycle (circuit) dans un graphe est un chemin fermé.

2.2.4 Instance Problème de voyage de commerce symétrique

On peut utiliser un programme linéaire binaire ou entier pour modéliser le problème de voyageur de commerce symétrique. La modélisation binaire se présente comme suit :

1. **Variables de décision :** Les variables sont définies sur l'ensemble $\{0, 1\}$: La variable x_{ij} correspond à la présence de l'arc ij dans le cycle hamiltonien du voyageur. On note $x_{ij} = 1$ si le voyageur prend le chemin existant entre les villes i et j, sinon $x_{ij} = 0$.

2. **Contraintes :** Le problème de voyageur de commerce est limité par plusieurs contraintes :

   (a) Contrainte d'intégrité : $x_{ij} \in \{0, 1\}$.

   (b) Le voyageur doit entrer une seule fois dans une ville : $\sum_{i=1}^{n} x_{ij} = 1$.

   (c) Le voyageur doit sortir une seule fois d'une ville : $\sum_{j=1}^{n} x_{ij} = 1$.

3. **Critère :** La fonction objectif du problème de voyageur de commerce se définit comme la minimisation de la longueur du chemin :

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}.x_{ij} \tag{2.3}$$

   où c_{ij} est la distance entre les villes i et j.

Résoudre le problème de voyageur de commerce revient à trouver la matrice solution $\overline{x} = \begin{pmatrix} \overline{x}_{11} & \dots & \overline{x}_{1n} \\ \vdots & \ddots & \vdots \\ \overline{x}_{n1} & \dots & \overline{x}_{nn} \end{pmatrix}$ qui **minimise** la fonction objectif définie par l'équation 2.3.

Mathématiquement, un problème de sac à dos est représenté comme suit :

Soient n nombre de villes, $c_{ij} \in \mathbb{N}*$ distance entre les villes i et j,

$$x = \begin{pmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nn} \end{pmatrix}$$

$$\text{Minimiser} \qquad f(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}.x_{ij}$$

$s.c.$

$$x_i \in \{0,1\}, \qquad i \in \{1,..,n\}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,..,n$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,..,n$$

2.3 Problème d'ordonnancement de tâches

La production dans un atelier est la création d'un produit à travers l'utilisation et la transformation des ressources. Pour le bon fonctionnement de ce système, certains nombres de processus sont implémentés prenant en compte le temps et le coût de production.

En général, le problème d'ordonnancement est relatif aux points suivants :

— Un ensemble de tâches ou jobs à effectuer.

— Un ensemble de ressources ou machines à utiliser par ces jobs.

— Un programme à identifier, pour allouer les ressources aux taches.

Les problèmes d'ordonnancement sont des problèmes d'optimisation définis en terme de : tâches et ressources.

2.3.1 Formalisme

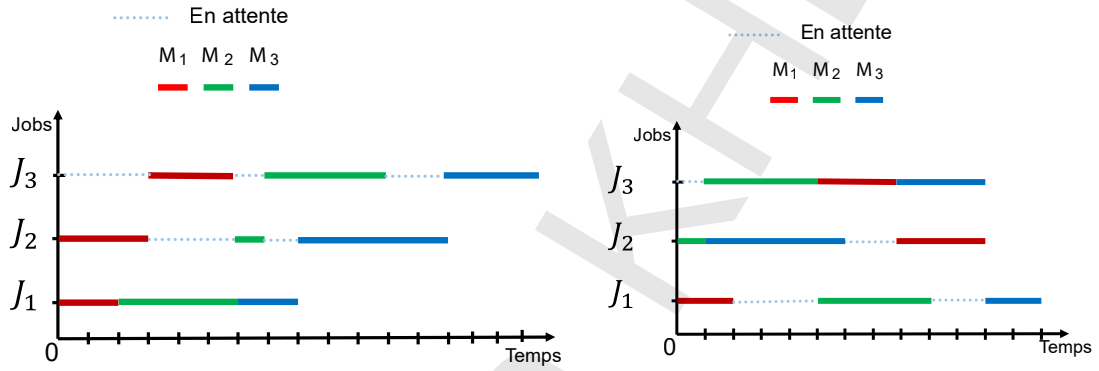
Une tâche (un job) i est une unité élémentaire du travail à effectuer qui commence dans une date t_i (start time) et se termine dans une date c_i (completion time). Elle utilise les ressources pendant un temps $p_i = c_i - t_i$ (processing time). Un job peut être interrompu, c'est-à-dire il est composé de plusieurs opérations. Dans le cas où le job ne peut pas être interrompu, il est vu comme une seule tâche à effectuer.

Une **ressource** est tout ce qu'on utilise pour effectuer un job. Les ressources sont limitées en temps et en quantités. Elles peuvent être des machines, ouvriers, équipements, etc.

Soit $J = \{1, .., n\}$ un ensemble de n jobs et $M = \{1, .., m\}$ un ensemble de machines. Chaque job $j \in J$ est composé au plus de m opérations ordonnées $O_{j1}, .., O_{jm}$. Chaque opération doit être effectuée sur une machine $k \in M$ spécifique durant un temps spécifique. Chaque job doit passer par chaque machine une et une seule fois, et chaque machine peut exécuter seulement un job (ou une opération) à la fois sans interruption.

Un **ordre** est une allocation des opérations aux intervalles de temps sur toutes les machines. Un problème d'ordonnancement de tâches consiste à trouver un ordre réalisable qui minimise C_{max} le temps nécessaire pour compléter tous les jobs.

Une représentation possible du problème d'ordonnancement de tâches est le diagramme de Gantt. On associe à chaque opération une barre horizontale avec une longueur proportionnelle au temps de l'opération (Figure 2.5).



(a) Flow Shop

(b) Job Shop

FIGURE 2.5 – Représentation de Gantt d'un ordonnancement de 3 tâches sur 3 machines, cas du Flow shop et Job shop.

2.3.2 Instances

Il existe plusieurs instances pour le problème d'ordonnancement selon le nombres de machines et la nature des jobs. On trouve les ordonnancements sur une machine unique ou sur des machines parallèles. Il existent aussi les ordonnancements linéaire ou multiples.

- **Machine Unique :** L'ensemble des tâches à réaliser est fait par une seule machine. On cherche l'ordre des tâches sur cette machine.
- **Machines parallèles :** L'ensemble des tâches à réaliser est fait par des machines identiques. Les tâches ne sont pas toutes exécutées sur les mêmes machines.
- **Ateliers à cheminement unique (Flow Shop) :** C'est-à-dire, le processus de fabrication est linéaire (Figure 2.5a). Les jobs sont découpés en tâches qui doivent s'exécuter sur toutes les machines.

— **Ateliers à cheminements multiples (Job Shop) :** Le processus de fabrication n'est pas linéaire (Figure 2.5b). Les jobs sont découpés en tâches qui doivent s'exécuter sur toutes les machines.

2.4 Problème du plus court chemin

Le problème du plus court chemin entre deux sommets dans un graphe orienté et valué par des coûts positifs est un problème de théorie de graphe. Il est résout efficacement par l'algorithme de Dijkstra (1959) (Figure 2.6).

(a) Graphe orienté et valué

(b) Plus court chemin

FIGURE 2.6 – Le plus court chemin du graphe orienté et valué de la figure (A) coloré en bleu dans la figure (B)

2.5 Problème de l'Arbre couvrant de poids minimum

Un arbre est un graphe connexe sans cycles. La recherche d'un arbre de poids minimum qui couvre tous les sommets d'un graphe G = (V,E) est un problème polynomial (Figure 2.7).

(a) Graphe orienté et valué

(b) Arbre couvrant de poids minimal

FIGURE 2.7 – Représentation d'un Arbre couvrant dans un Graphe orienté et valué