Chapitre I : Initiation à l'Optimisation

Appelée aussi optimisation discrète, est une branche de l'optimisation en mathématiques appliquées et en informatique, également liée à l'algorithmique et la théorie de la complexité.

Problème d'optimisation combinatoire

Un problème d'optimisation combinatoire est défini à partir d'un ensemble fini S et d'une application $f : S \to \mathbb{R}$. Il s'agit de déterminer $\hat{s} \in S$ tel que :

$$\boxed{(P) \equiv \ f(\hat{s}) = \min\{f(s) : s \in S\}}$$

Formulation 1

Étant donnés : un ensemble fini $E = \{e_1, e_2, \ldots, e_n\}$ et une fonction coût $c : E \to \mathbb{R}$

$$\left.\begin{array}{l} S \subset \mathscr{P}(E) \\ f(s) = \sum_{e \in s} c(e) \end{array}\right| \to f(\hat{s}) = \min\{f(s) : s \in S\}$$

Formulation 2

Étant donnés : un vecteur $s = (s_1, s_2, \ldots, s_n)$ tel que $s_i \in D_i, (i = 1, \ldots, n)$ et une fonction objective $f : D_1 \times D_2 \times \cdots \times D_n \to \mathbb{R}$

$$S \subset D_1 \times D_2 \times \cdots \times D_n \Big| \to f(\hat{s}) = \min\{f(s) : s \in S\}$$

Exemple(TD): Soit le programme d'optimisation suivant :

$$\boxed{\begin{array}{ll} \min z = & 4x_1 + 5x_2 + 6x_3 \\ (1) & 3x_1 + 2x_2 + 5x_3 \geq 6 \\ (2) & x_i \in \{0;1\}, \ (i = 1,2,3) \end{array}}$$

Formulation et interprétation 1

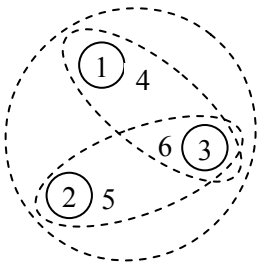
$E = \{1,2,3\}$

$\mathscr{P}(E) = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

$S = \{\{1,3\}, \{2,3\}, \{1,2,3\}\}$

$\hat{s} = \{1,3\} \to f(\hat{s}) = 10$

$$c : \{1,2,3\} \to \mathbb{R}, \quad e \mapsto c(e)$$



Formulation et interprétation 2

$s = (s_1, s_2, s_3), \quad D_i \equiv \{0;1\}, \ i \in \{1,2,3\}$

$D_1 \times D_2 \times D_3 \equiv \{0;1\}^3 = \{(0,0,0), (1,0,0), (0,1,0), \ldots, (1,1,1)\}$

$S = \left\{s \in \{0;1\}^3 : 3s_1 + 2s_2 + 5s_3 \geq 6\right\} = \{(1,0,1), (0,1,1), (1,1,1)\}$

$$f : \{0;1\}^3 \to \mathbb{R}, \quad s \mapsto 4s_1 + 5s_2 + 6s_3$$

$\hat{s} = (1,0,1) \to f(\hat{s}) = 10$

Modélisation de quelques problèmes de l'optimisation combinatoire

Couplage généralisé, **Partitionnement et Recouvrement**

Étant donné une matrice de taille $m \times n$ dont les éléments valent 0 ou 1, le m-vecteur e dont toutes les composantes sont égales à 1 et un n-vecteur de ligne c, on considère les trois programmes linaires en variables bivalentes :

Couplage généralisé	Partitionnement	Recouvrement
$(PC)\begin{cases} cx = z(\max) \\ Ax \leq \text{e} \\ x \in \{0;1\}^n \end{cases}$	$(PP)\begin{cases} cx = z(\min), \text{ou } (\max) \\ Ax = \text{e} \\ x \in \{0;1\}^n \end{cases}$	$(PR)\begin{cases} cx = z(\min) \\ Ax \geq \text{e} \\ x \in \{0;1\}^n \end{cases}$

Interprétation :

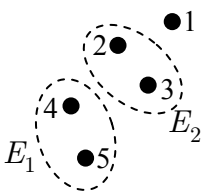
Soit $E = \{e_1, e_2, \ldots, e_m\}$, et n sous-ensembles $E_1, E_2, \ldots, E_n$ de E caractérisé par : $\boxed{e_i \in E_j \Leftrightarrow A_{ij} = 1}$

Il s'agit de trouver une famille $F \subseteq \{E_1, E_2, \ldots, E_n\}$ telle que :

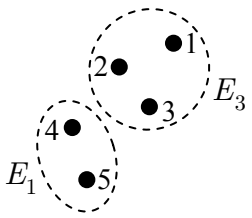
- couplage : les sous-ensembles de la famille aient deux à deux une intersection vide.
- recouvrement : tout élément de E soit contenu dans au moins un sous-ensemble de la famille.
- partitionnement : on ait à la fois un couplage et un recouvrement.

Exemple

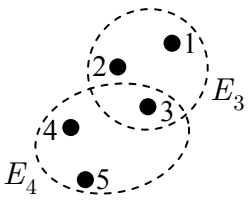
Par exemple $E = \{1,2,3,4,5\}$ et $E_1 = \{4,5\}, E_2 = \{2,3\}, E_3 = \{1,2,3\}, E_4 = \{3,4,5\}$



$F = \{E_1, E_2\}$: couplage



$F = \{E_1, E_3\}$: partition



$F = \{E_3, E_4\}$: recouvrement

Problème du sac à dos

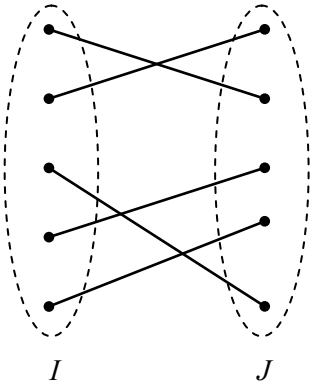
Considérons n objets possédant chacun un bénéfice et un poids $(c_i, a_i), i \in \{1,\ldots,n\}$. On cherche à ranger ces objets dans un sac-à-dos de capacité b. Les objets mis dans le sac doivent maximiser la valeur totale du bénéfice sans dépasser la capacité du sac. On pose : $x_i = 1$, si l'objet $i \in$ sac ; sinon 0.

$$PSD \equiv \begin{cases} z = \displaystyle\sum_{i=1}^{n} c_i x_i & (\max) \\ (1) \quad \displaystyle\sum_{i=1}^{n} a_i x_i \leq b & \\ (2) \quad x_i \in \{0,1\} & i \in \{1,\ldots,n\} \end{cases}$$

Problème de l'affectation classique

Soit un graphe biparti $G = (I \cup J, E)$ tel que $\left|I\right| = \left|J\right|$, muni d'une application $c : E \to \mathbb{R}$. Le problème est de trouver un couplage parfait de coût minimum dans G.

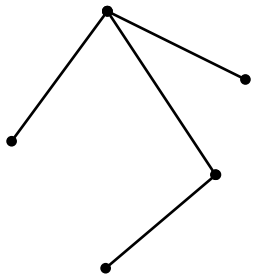
$$PAC \equiv \begin{cases} z = & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & (\min) \\ (1) & \sum_{j=1}^{n} x_{ij} = 1 & i \in \{1,\ldots,n\} \\ (2) & \sum_{i=1}^{n} x_{ij} = 1 & j \in \{1,\ldots,n\} \\ (3) & x_{ij} \in \{0,1\} & (i,j) \in \{1,\ldots,n\}^2 \end{cases}$$



Problème d'arbre couvrant de poids minimum

Soit un graphe $G = (V, E)$ connexe, muni d'une application $c : E \to \mathbb{R}$. Le problème est de trouver dans G , un arbre <u>maximal</u> $A = (V, E')$, $E' \subset E$ de poids total minimum.

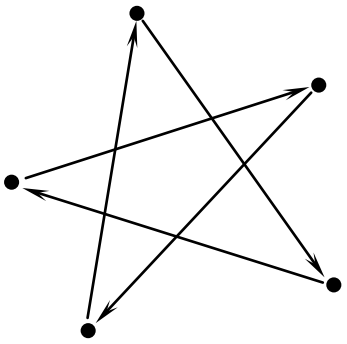
$$PACP \equiv \begin{cases} z = & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & (\min) \\ (1) & \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} = n-1 & \\ (2) & \sum_{(i,j)\in S^2} x_{ij} \le \left|S\right| - 1 & S \subset \{1,\ldots,n\} : 2 \le \left|S\right| \le n-1 \\ (3) & x_{ij} \in \{0,1\} & (i,j) \in \{1,\ldots,n\}^2 \end{cases}$$



Problème du voyageur de commerce

Un représentant doit visiter n villes une et une seule fois, et revenir à sa ville de départ, en minimisant le coût total du trajet. Soit un graphe complet $G = (V, \vec{E})$ muni d'une application $c : \vec{E} \to \mathbb{R}$. Le problème est de trouver le plus court <u>circuit Hamiltonien</u> dans le graphe G.

$$PVC \equiv \begin{cases} z = & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & (\min) \\ (1) & \sum_{j=1}^{n} x_{ij} = 1 & i \in \{1,\ldots,n\} \\ (2) & \sum_{i=1}^{n} x_{ij} = 1 & j \in \{1,\ldots,n\} \\ (3) & \sum_{(i,j)\in S^2} x_{ij} \le \left|S\right| - 1 & S \subset \{1,\ldots,n\} : 2 \le \left|S\right| \le n-1 \\ (4) & x_{ij} \in \{0,1\} & (i,j) \in \{1,\ldots,n\}^2 \end{cases}$$



Problème de localisation de dépôts

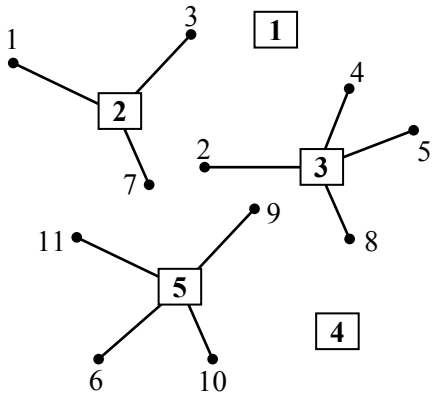
Le problème est spécifié par :

n	Le nombre de sites (dépôts) envisageables.
m	Le nombre de clients, chaque client doit être affecté à un dépôt.
f_i	Le coût fixe pour construire (ouvrir) le dépôt i.
c_{ij}	Le coût d'affectation du client j au dépôt i.

Les variables de décision :

$y_i = 1$, si le dépot i est ouvert ; sinon 0
$x_{ij} = 1$, si le client j est affecté au dépot i ; sinon 0

$$PSLD \equiv \begin{cases} z = \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{i=1}^{n} f_i y_i & (\min) \\ (1) \quad \displaystyle\sum_{i=1}^{n} x_{ij} = 1 & j \in \{1,\ldots,m\} \\ (2) \quad \displaystyle\sum_{j=1}^{m} x_{ij} \le m y_i & i \in \{1,\ldots,n\} \\ (3) \quad x_{ij} \in \{0,1\}, y_j \in \{0,1\} & i \in \{1,\ldots,n\}, j \in \{1,\ldots,m\} \end{cases}$$



Problème de reconnaissance (ou de décision) associé au problème d'optimisation

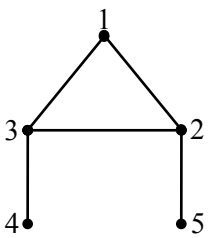
Étant donné un problème d'optimisation combinatoire (P) et un nombre a. Le problème de reconnaissance (R_a) associé à (P) est défini :

(P)	(R_a)
$\min\{f(s) : s \in S\}$	$(\exists \overline{s} \in S : f(\overline{s}) \le a)$?
$\max\{f(s) : s \in S\}$	$(\exists \overline{s} \in S : f(\overline{s}) \ge a)$?

Exemple(TD) : Le problème de reconnaissance associé au problème du **stable de taille maximum**.

(R_k): Existe-t-il un stable de taille supérieure ou égale à k ?

La réponse peut prendre 2 valeurs seulement: (oui =1)/(non = 0).

k	$(\exists \overline{s} \in S : \text{Taille}(\overline{s}) \ge k)$	$\overline{s}$		$\begin{cases} \sum_{i \in V} x_i = z(\max) \\ x_i + x_j \le 1, \quad (i,j) \in E \\ x_i \in \{0;1\}, \quad i \in V \end{cases}$
1	oui	$\{2\}$		
2	oui	$\{3,5\}$		
3	oui	$\{1,4,5\}$		
4	non	/	$G = (V,E)$	Le programme mathématique

Remarque : Pour $k = 3$, on peut facilement vérifier que $\hat{s} = \{1,4,5\}$ est une solution (Réponse positive).

Complexité algorithmique

La complexité algorithmique s'attache à comparer les algorithmes selon leur temps d'exécution.

Définition : On appelle complexité d'un algorithme le nombre asymptotique d'opérations élémentaires (+, -, ×, /, :=, test) qu'il doit effectuer en fonction de la taille de l'entrée qu'il a à traiter.

Notations n : taille des données; et $T(n)$: nombre d'opérations élémentaires.

Comportement asymptotique $(n \to \infty)$ **de** $T(n)$ (nous adoptons le critère « au pire des cas ».)

Définition : $f(n)$ est une borne supérieure asymptotique pour $T(n)$, si est seulement si :

$$\boxed{(\exists C > 0),(\exists n_0 \geq 0) : \forall n \geq n_0, T(n) \leq C \times f(n)}$$

et on note : $T(n) = \mathcal{O}(f(n))$ pour dire que $T \in \mathcal{O}(f)$.

Algorithmes efficaces : Un algorithme est dit efficace si le nombre d'opérations élémentaires est polynomial (i.e. borné supérieurement par un polynôme) en la taille de problème.

Exemple : $T(n) = 5n^3 + 2n^2 - n + 2 = \mathcal{O}(n^3)$ car si : $n \geq 1 \Rightarrow T(n) \leq 8n^3$, $(n_0 = 1, \text{et } c = 8)$.

Remarques

- Un algorithme est autant plus efficace que sa complexité est minimale.
- La complexité algorithmique est indépendante de l'environnement, du compilateur et du programmeur.

Complexité des problèmes et efficacité des algorithmes (Notions élémentaires)

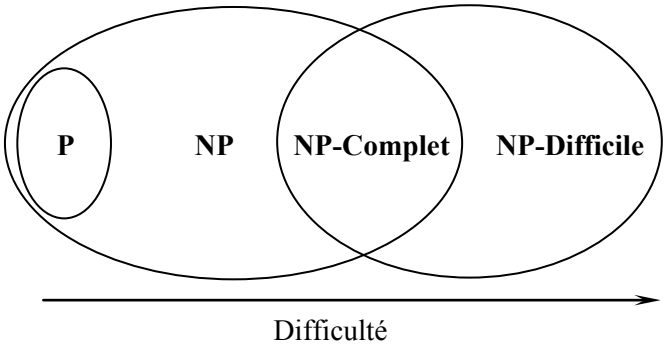
La théorie de la complexité s'attache à classifier les problèmes selon leur difficulté.

- **P** : la classe des problèmes qui peuvent être résolus en temps polynomial sur une machine déterministe. Ces problèmes sont souvent considérés comme ceux pour lesquels il existe un algorithme efficace.
- **NP** : la classe des problèmes qui peuvent être résolus en temps polynomial sur une machine non déterministe.

Problèmes faciles et difficiles

- Un problème est **Facile** s'il appartient à la classe des problèmes **P** pour lesquels il existe un algorithme efficace pour les résoudre.
- Un problème de décision est **Difficile** s'il appartient à la classe **NP-Complet**, pour lesquels il n'existe pas à l'heure actuelle un algorithme efficace pour les résoudre, cependant, il existe un moyen pour vérifier, en temps polynomial, la véracité d'une solution donnée.
- Un problème d'optimisation appartient à la classe **NP-Difficile** si son problème décisionnel appartient à la classe **NP-Complet**.

Dans ce cours, nous adoptons le schéma suivant :



Désignation des complexités algorithmique courantes

Notation	Désignation	Notation	Désignation
$\mathcal{O}(1)$	Constante	$\mathcal{O}(n^2)$	Quadratique
$\mathcal{O}(\log(n))$	Logarithmique	$\mathcal{O}(n^3)$	Cubique
$\mathcal{O}(n)$	Linéaire	$\mathcal{O}(a^n),\ (a>1)$	Exponentielle
$\mathcal{O}(n\log(n))$	Quasi-linéaire	$\mathcal{O}(n!)$	Factorielle

Exercice (TD)

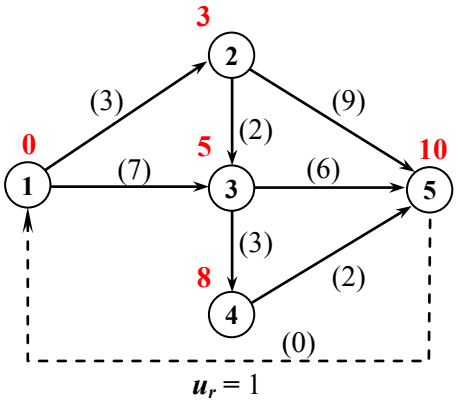
Supposons que trois ordinateurs M_1, M_2 et M_3 effectuent, respectivement, 10.000, 20.000 et 40.000 opérations par seconde. Quelle taille maximum de problème n peut-on résoudre en une minute par des algorithmes effectuant respectivement $n, n^2, n^3,$ et 2^n opérations ?

Ordinateur	n	n^2	n^3	2^n
M_1	600.000	775	84	19
M_2	1.200.000	1.095	106	20
M_3	2.400.000	1.549	134	21

Exercices sur la modélisation mathématique (TD)

1) Problème du plus court chemin

$$(P_1)\begin{cases} z = \min \sum_{e\in E} c_e x_e \\ \sum_{e\in\delta^+(i)} x_e - \sum_{e\in\delta^-(i)} x_e = \begin{cases} +1 & i = s \\ 0 & i \neq s,p \\ -1 & i = p \end{cases} \qquad i \in V \\ x_e \geq 0, \qquad e \in E \end{cases}$$



$$(P_2)\begin{cases} \sum_{u\in E} t(u) = z(\max) \\ t(u) \leq d(u) \end{cases} \rightarrow \begin{cases} \pi(s) - \pi(p) = z(\max) \\ \pi(T(u)) - \pi(I(u)) \leq d(u), \quad u \in E \\ \pi(s) = 0 \end{cases}$$

2) Problème de placement de boîtes (Remplissage)

Problème : Placer un ensemble de boîtes de longueurs différentes dans le moins de cartons possibles de longueur fixée.

Données : n boîtes de longueurs $w_i, (i = 1,\ldots,n)$ et W est la longueur de chaque carton.

Objectif : minimiser le nombre de cartons utilisés.

Contraintes : ne pas dépasser la capacité d'un carton.

Exemple :($n = 5,\ W = 6$)

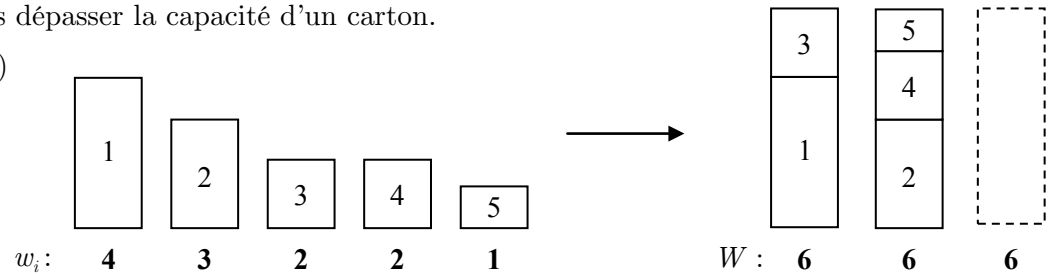
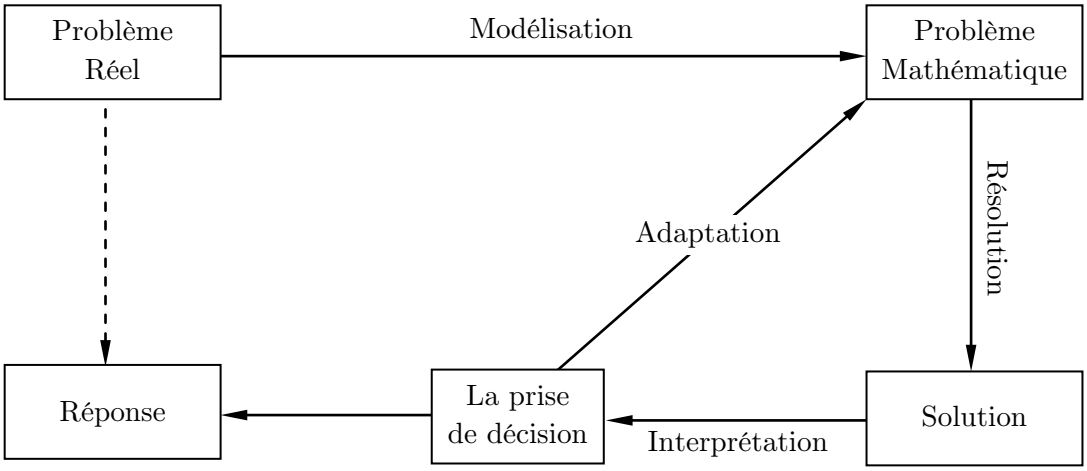


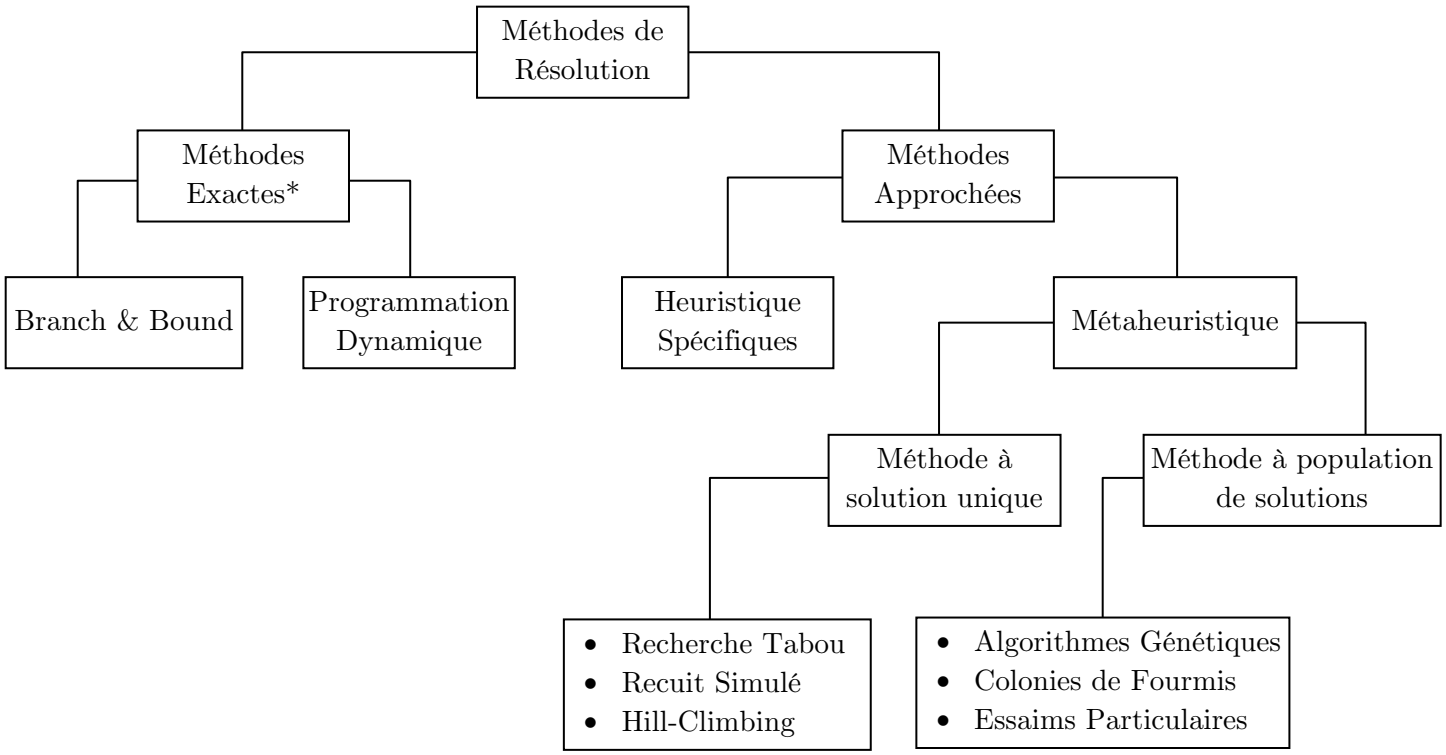
Schéma conceptuel de la RO pour la résolution d'un problème d'optimisation

En général, la résolution d'un problème d'optimisation passe par plusieurs étapes :



Optimisation = Modélisation + Résolution

Schéma général des méthodes de résolution



*Les méthodes exactes ne sont efficaces que pour les instances de problèmes de petite taille.