OPTIMISATION EN NOMBRES ENTIERS :

BRANCH & BOUND

Hacène Ouzia

MAIN (5 ème année)
Université Pierre et Marie Curie

2020

Table des matières

AGENDA

Généralités

■ **MODÈLE** Nous considérerons le modèle suivant :

$$\begin{array}{ll} \min & c^T x + f^T y \\ s.c. & \\ & Ax + By \leq b, \\ & x \in \mathbb{Z}^n, y \in \mathbb{R}^m. \end{array} \tag{1}$$

■ **DÉFINITION** BRANCH AND BOUND

La méthode par *séparation et évaluation* (Branch and Bound) pour résoudre le problème (1) est une méthode d'énumération exhaustive et effective du domaine combinatoire du problème (1).

Généralités

■ MODÈLE Nous considérerons le modèle suivant :

$$\begin{array}{ll} \min & c^T x + f^T y \\ s.c. & \\ & Ax + By \leq b, \\ & x \in \mathbb{Z}^n, y \in \mathbb{R}^m. \end{array} \tag{1}$$

■ DÉFINITION BRANCH AND BOUND

La méthode par *séparation et évaluation* (Branch and Bound) pour résoudre le problème (1) est une méthode d'énumération exhaustive et effective du domaine combinatoire du problème (1).

Exemple

Soit à résoudre, par séparation et évaluation, le problème suivant :

$$\begin{array}{ll} \max & 3x + 2y + 7z \\ s.c. & \\ & 3x + 4y + 6z \leq 11, \\ & 2x + 5y + 4z \leq 7, \\ & x, y, z \in \{0, 1\}. \end{array} \tag{2}$$

Exemple : solution I

Noeud-0 A la racine il faut résoudre la relaxation continue suivante :

$$\begin{array}{ll} \max & 3x + 2y + 7z \\ s.c. & \\ & 3x + 4y + 6z \leq 11, \\ & 2x + 5y + 4z \leq 7, \\ & x, y, z \in [0, 1]. \end{array}$$

- La solution optimale (après 4 itérations) est la suivante :

$$\hat{x} = 1; \hat{y} = \frac{1}{5}; \hat{z} = 1;$$

  et sa valeur est $10 + \frac{2}{5}$.
- Séparer sur la variable y. Donc, il faudra considérer pour le **branchement** les deux contraintes $y \leq 0$ et $y \geq 1$.
- La solution n'est pas entière alors $LB = +\infty$.

Noeud-1 Dans ce nœud nous fixons la variable de la variable y à 0. Cela donnera la relaxation continue suivante :

$$\begin{array}{ll} \max & 3x + 7z \\ s.c. & \\ & 3x + 6z \leq 11, \\ & 2x + 4z \leq 6, \\ & x, z \in [0, 1]. \end{array}$$

Exemple : solution II

- La solution optimale est :

$$\hat{x} = 1; \hat{y} = 0; \hat{z} = 1;$$

  et sa valeur est 10.
- Inutile d'explorer plus cette branche de l'arbre
- La solution est entière alors $LB = 10$.

Noeud-2 Dans ce noeud nous fixons la variable y à 1. cela donnera la relaxation continue suivante :

$$2 + \begin{array}{ll} \max & 3x + 7z \\ s.c. & \\ & 3x + 6z \leq 7, \\ & 2x + 4z \leq 6, \\ & x, z \in [0, 1]. \end{array}$$

- La solution optimale est :

$$\hat{x} = 0; \hat{y} = 1; \hat{z} = \frac{1}{2};$$

  et sa valeur est $\frac{11}{2}$.
- Inutile d'explorer plus cette branche car la borne supérieure est inférieure à la valeur de la borne inférieure.

- Donc, la solution optimale est :

$$\hat{x} = 1; \hat{y} = 0; \hat{z} = 1;$$

et sa valeur est 10

Algorithme

Fonction BAB(Prob)

Entrée : `Prob` :: Problème ! *Implémentation du problème à résoudre.*

Sortie : `Xopt` :: Solution optimale ! *Implémentation d'une solution.*

```
Début
    Xopt.init()  ! Intialisation de la solution.
    Pool.init()  ! Initialisation de la pool de problèmes.
    Pool.add( Prob.relax() )  ! Problème racine : relaxation
      continue du problème.
    Tant que Pool.active() faire
        Q ← Pool.first()  ! Extraction du problème à résoudre
        Sol ← Bound(Q)  ! Sol est la solution du problème Q
        Si Sol.feasible() alors
            Si Sol.value() ≥ Xopt.value() alors
                Pool.fathome(Q) ! Élagage par valeur.
            sinon
                Si Sol.integer() alors
                    Xopt.update(Sol)
                    Pool.fathome(Q)  ! Élagage car solution entière
                sinon
                    B ← Branch(Q)  ! B est une structure contenant les
                      détails du branchement.
                    Pool.remove(Q)
                    Pool.add(leftBranch(Q,B))
                    Pool.add(rightBranch(Q,B))
                Fin si
            Fin si
        sinon
            Pool.fathome(Q)  ! Élagage car problème non réalisable.
        Fin si
    Fin tant que
    return Xopt
Fin
```

Principales fonctions

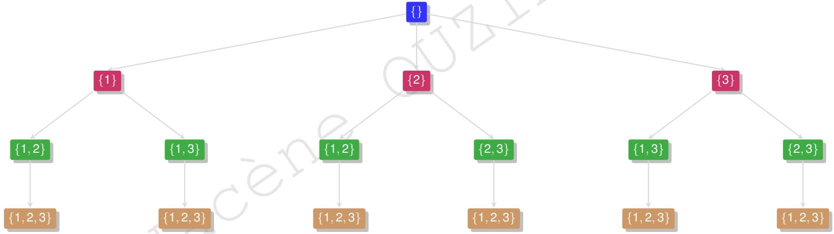
- `Prob.relax()` : Méthode retournant la relaxation continue du problème.
- `Sol.init()` : Méthode pour initialiser la solution entière du problème. Cette initialisation concerne, entre autre, la valeur de cette même solution. Cette valeur constitue la borne supérieure du problème (*minimisation !*)
- `Bound(Problem)` : Procédure pour résoudre la relaxation continue du problème en argument. La valeur de cette relaxation est une borne inférieure de la valeur optimale.
- `Pool.fathome(Problem)` : Supprime de la pool de problème le problème en argument.
- `Branch(Problem)` : Détermine la variable sur laquelle se fera la séparation du domaine. Cette séparation à, en générale, la forme d'inégalités du type $x_k \leq 0$ et $x_k \geq 1$.

POLYTECH SORBONNE

AGENDA

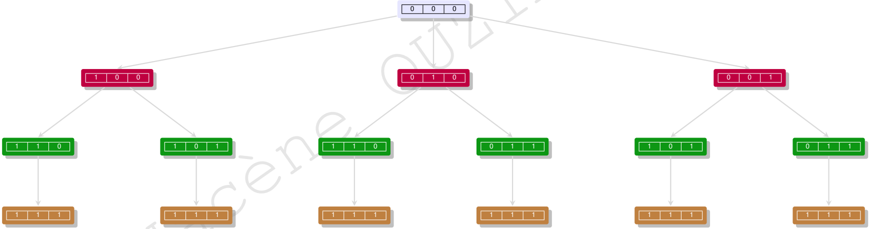
Génération des sous-ensembles

Exemple



Génération des sous-ensembles

Exemple



Génération des sous-ensembles

Algorithme 2: Enumération naïve des sous-ensembles de *Ens*

Entrées : *N* :: Entier non nul ! *Taille de l'ensemble*
Ens :: Tableau d'entiers ! *Ensemble à énumérer*
Visites :: Tableau de booléens ! *Ensemble courant*

Sortie : Liste des sous-ensembles de *Ens*

```
Début
    Pour k = 1 à N faire
        Visites[k] ← non
    Fin pour
    Enumset (1)
Fin
```

Génération des sous-ensembles

Fonction Enumset(j)

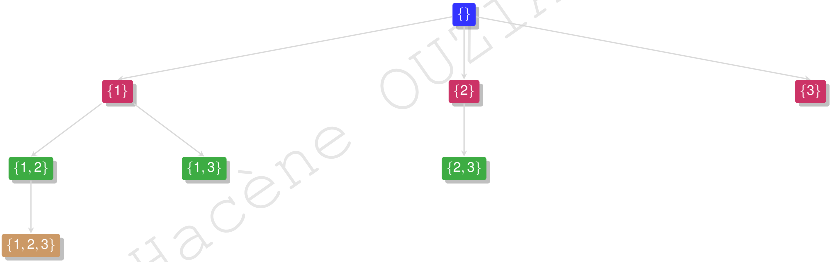
Entrée : j :: Entier *! Indice indiquant la position de début de l'exploration*

Sortie : Les sous-ensembles de $\{Ens\,(j)\,, \ldots, Ens\,(N)\}$

```
Début
    Pour k = 1 à N faire
        Si non Visites[k] alors
            Visites[k] ← oui
            afficher() ! Utilisera les tableaux Ens et Visites
            Enumset (j + 1)
            Visites[k] ← non
        Fin si
    Fin pour
Fin
```

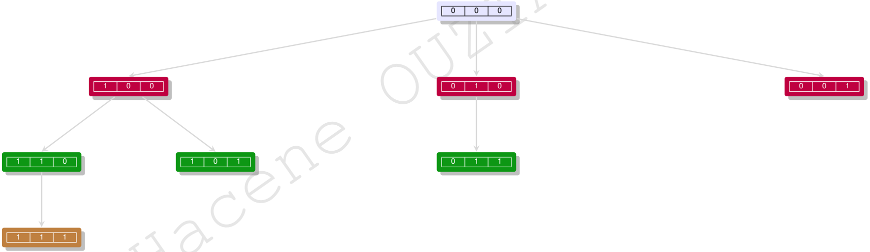
Génération des sous-ensembles

Exemple



Génération des sous-ensembles

Exemple



Génération des sous-ensembles

Fonction Enumset(j)

Entrée : j :: Entier *! Indice indiquant la position de début de l'exploration*

Sortie : Les sous-ensembles de $\{Ens(j), \ldots, Ens(N)\}$

```
Début
    Pour k = j à N faire
        Si non Visites[k] alors
            Visites[k] ← oui
            afficher() ! Utilisera les tableaux Ens et Visites
            Enumset (j + 1)
            Visites[k] ← non
        Fin si
    Fin pour
Fin
```

Problème du Sac-à-dos

Données

- Une étudiante
- Un sac-à-dos de capacité γ
- n livres $(l_1, \ldots, l_n)$
- $(u_1, \ldots, u_n)$ vecteur des utilités
- $(p_1, \ldots, p_n)$ vecteur des poids

Question

- Déterminer une composition du sac-à-dos telle que son utilité soit maximale et sa capacité ne soit pas dépassée.

Problème du Sac-à-dos

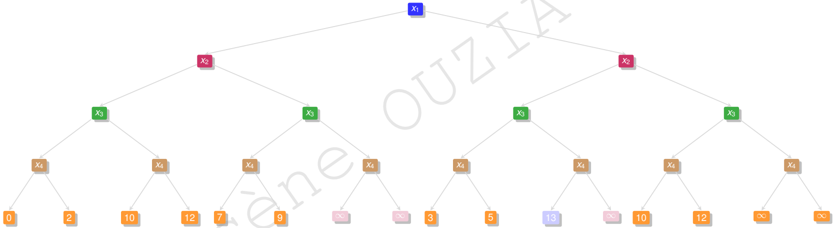
Modèle mathématique Problème du sac-à-dos

$$\begin{array}{ll} \text{Max} & \sum_{j=1}^{n} u_j x_j \\ \text{s.c.} & \\ & \sum_{j=1}^{n} p_j x_j \leq \gamma \\ & x_j \in \{0,1\} \qquad j \in \{1,\dots,n\} \end{array}$$

$$x_j = \begin{cases} 1 & \textit{si le livre } j \textit{ est pris} \\ 0 & \textit{sinon} \end{cases}$$

Problème du Sac-à-dos

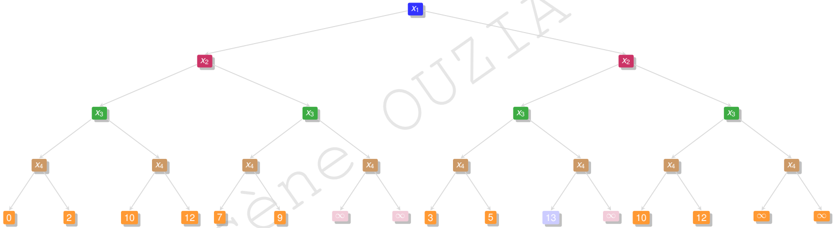
■ EXEMPLE $u = \langle 3, 7, 10, 2 \rangle$, $p = \langle 4, 6, 7, 2 \rangle$ et $\gamma = 12$



☞ Pour tout $1 \leq k \leq n$ nous avons $\mathcal{C}_k = \{0, 1\}$.

Problème du Sac-à-dos

■ EXEMPLE $u = \langle 3, 7, 10, 2\rangle, p = \langle 4, 6, 7, 2\rangle$ et $\gamma = 12$



☞ Pour tout $1 \leq k \leq n$ nous avons $\mathcal{C}_k = \{0, 1\}$.

Problème du Sac-à-dos

Algorithme 5: Enumération explicite pour le problème du sac-à-dos

Entrées : N :: Entier non nul *! Nombre d'objets*
u :: Tableau de réels *! Vecteurs des utilités*
p :: Tableau de réels *! vecteurs des poids*

Sorties : *xopt* :: Tableau de booléens *! Solution optimale*
UtiliteOpt :: Réel *! Utilité de la solution optimale*

```
Début
    UtiliteOpt ← −∞
    sacados (1)
Fin
```

Problème du Sac-à-dos

Fonction Sacados(k)

Entrée : k :: Entier ! *Explorer les valeurs de la variable x_k*

```
Début
    Si k = n + 1 alors
        Si ⟨p, x⟩ ≤ γ alors
            Utilite ← ⟨u, x⟩
            Si Utilite > UtiliteOpt alors
                UtiliteOpt ← Utilite
                xopt ← x
            Fin si
        Fin si
    sinon
        x_k ← 0
        sacados(k + 1)
        x_k ← 1
        sacados(k + 1)
    Fin si
Fin
```

Problème du sac-à-dos

Données

- Une solution partielle $(x_1, \ldots, x_k)$
- Charge actuelle : $\pi = \sum_{j=1}^{k} p_j x_j$
- Poids p_{k+1} de l'objet $k+1$.

Nous avons :

$$\mathcal{C}_{k+1} = \begin{cases} \{0\} & \text{si } \pi + p_{k+1} > \gamma \\ \{0, 1\} & \text{sinon} \end{cases}$$

Problème du Sac-à-dos

Fonction Sacados(k,w)

Entrées : k :: Entier ! *Explorer les valeurs de la variable x_k*
w :: Réel ! *Charge actuelle du sac-à-dos*

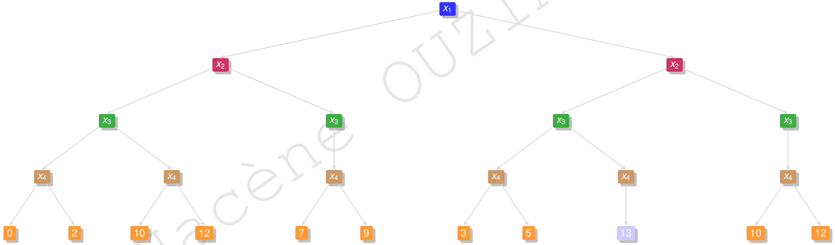
Début
- **Si** $k = n+1$ **alors**
  - **Si** $\langle p, x\rangle \leq \gamma$ **alors**
    - $Utilite \leftarrow \langle u, x\rangle$
    - **Si** $Utilite > UtiliteOpt$ **alors**
      - $UtiliteOpt \leftarrow Utilite$
      - $xopt \leftarrow x$
    - **Fin si**
  - **Fin si**
- **sinon**
  - $x_k \leftarrow 0$
  - **sacados**$(k+1, w)$
  - **Si** $w + p_k \leq \gamma$ **alors**
    - $x_k \leftarrow 1$
    - **sacados**$(k+1, w+p_k)$
  - **Fin si**
- **Fin si**

Fin

POLYTECH SORBONNE

Problème du Sac-à-dos

■ EXEMPLE $u = \langle 3, 7, 10, 2 \rangle, p = \langle 4, 6, 7, 2 \rangle$ et $\gamma = 12$



Problème du sac-à-dos général

Modèle mathématique Problème du sac-à-dos général

$$
\begin{array}{rll}
\text{Max} & \displaystyle\sum_{j=1}^{n} u_j x_j & \\
\text{s.c.} & & \\
& \displaystyle\sum_{j=1}^{n} p_j x_j \leq \gamma & \\
& 0 \leq x_j \leq b_j & j \in \{1, \ldots, n\} \\
& x_j \in \mathbb{Z} & j \in \{1, \ldots, n\}
\end{array}
$$

Problème du Sac-à-dos général

Algorithme 8: Enumération explicite pour le problème du sac-à-dos

Entrées : N :: Entier non nul *! Nombre d'objets*
u :: Tableau de réels *! Vecteurs des utilités*
p :: Tableau de réels *! vecteurs des poids*
b :: Tableau de réels *! vecteurs des bornes supérieures*

Sorties : *xopt* :: Tableau de booléens *! Solution optimale*
UtiliteOpt :: Réel *! Utilité de la solution optimale*

```
Début
    UtiliteOpt ← −∞
    sacados (1)
Fin
```

Problème du Sac-à-dos général

Fonction Sacados(k,w)

Entrées : k :: Entier *! Explorer les valeurs de la variable x_k*
w :: Réel *! Charge actuelle du sac-à-dos*

```
Début
    Si k = n + 1 alors
        Si ⟨p, x⟩ ≤ γ alors
            Utilite ← ⟨u, x⟩
            Si Utilite > UtiliteOpt alors
                UtiliteOpt ← Utilite
                xopt ← x
            Fin si
        Fin si
    sinon
        Limite ← calculerLimite(k, w); y ← 0
        Tant que y ≤ Limite faire
            x_k ← y
            sacados(k + 1, w + y p_k)
            y ← y + 1
        Fin tant que
    Fin si
Fin
```

Clique maximale dans un graphe

Données

- $G = \langle V, E \rangle$ un graphe non orienté d'ordre n
- $V = \{1, \ldots, n\}$

Question

- Quelle est la taille de la *clique maximale* de G, c.-à-d., le nombre $w(G)$.

Définition Clique dans un graphe

Un sous-ensemble $\mathcal{C}$ de sommets du graphe G est une clique si :

$$\forall u, v \in \mathcal{C} : (u, v) \in E.$$

Une clique est maximale si elle n'est pas incluse dans une autre clique.

Clique maximale dans un graphe

Données

- $G = \langle V, E\rangle$ un graphe non orienté d'ordre n
- $V = \{1, \ldots, n\}$

Question

- Quelle est la taille de la *clique maximale* de G, c.-à-d., le nombre $w(G)$.

Définition Clique dans un graphe

Un sous-ensemble $\mathcal{C}$ de sommets du graphe G est une clique si :

$$\forall u, v \in \mathcal{C} : (u, v) \in E.$$

Une clique est maximale si elle n'est pas incluse dans une autre clique.

Clique maximale

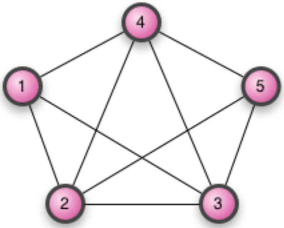


FIGURE – Exemple

Clique maximale

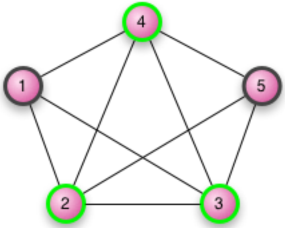


FIGURE – Exemple : $\mathcal{C} = \{2, 3, 4\}$ est une clique

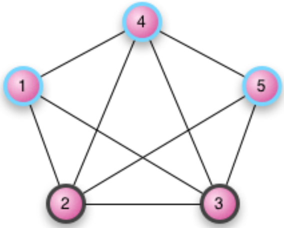


FIGURE – Exemple : : $\mathcal{C} = \{1, 4, 5\}$ n'est pas une clique

Clique maximale

Algorithme 10: Énumération naïve des cliques d'un graphe G

Entrées : $G = \langle V, E \rangle$ :: Graphe non orienté *! Liste des successeurs*
Visites :: Tableau de booléens *! Sommets traités, c.-à-d., $\mathcal{C}_k$*

Sortie : $w(G)$ :: Entier *! Taille de la clique maximale*

```
Début
    Pour k = 1 à n faire
        Visites[k] ← non
    Fin pour
    Cliquemax(0)
Fin
```

Clique maximale

Fonction Cliquemax(k)

Entrées : k :: Entier *! Taille de la clique courante*
Candidats :: Tableau de Booléens *! De taille n*

Sortie : Les cliques de taille k

```
Début
    Si k > 0 alors
        ζ ← Max {y : y ∈ C_{k−1}}
        Candidats ← {x ∈ V \ {1, . . . , ζ} : (x, v) ∈ E, ∀v ∈ C_{k−1}}
    sinon
        Candidats ← V
    Fin si
    Pour j = 1 à N faire
        Si Candidat[j] alors
            Visites[j] ← oui
            Cliquemax (k + 1)
            Visites[j] ← non
        Fin si
    Fin pour
Fin
```

Cliques maximales

```
Liste des cliques :

  1
  1   2
  1   2   3
  1   2   3   4  clique maximale
  1   2   4
  1   3
  1   3   4
  1   4
  2
  2   3
  2   3   4
  2   3   4   5  clique maximale
  2   3   5
  2   4
  2   4   5
  2   5
  3
  3   4
  3   4   5
  3   5
  4
  4   5
  5

  Nombre de cliques             :  23
  Nombre de cliques maximales   :   2

STOP Fin cliques : generation explicite ...
lama-2:cliques ouzia$
```

Clique maximale

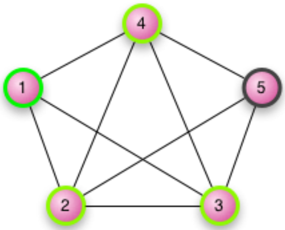


FIGURE – $\mathcal{C} = \{1, 2, 3, 4\}$ clique maximale

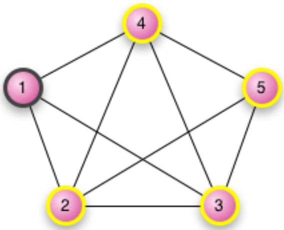


FIGURE – $\mathcal{C} = \{2, 3, 4, 5\}$ clique maximale

Bibliographie

- D. G. Luenberger and Yinyu Ye (2008),
  Linear and Nonlinear Programming,
  Springer
- M.S. Bazaraa, J.J. Jarvis and H.D. Sherali (2006),
  Linear Programming and Network Flows
- G.B. Dantzig and N.T. Mukund (1997),
  Linear Programming,
  Springer
- R. J Venderbei (2008),
  Linear programming, Fondations and extensions,
  Springer