Optimisation Discrète

Méthode exacte

Yannick Kergosien
Polytech Tours
LIFAT EA 6300– Equipe ROOT

Une méthode exacte c’est un algorithme

A. **qui trouve toujours une solution en un temps précis connu à l’avance.**
B. **qui trouve une solution la plus optimale possible.**
C. **qui trouve la solution optimale.**
D. **Oula ça commence déjà…**

Plan

- Utilisation de solveur
- Méthode arborescente
- Programmation Dynamique

Utilisation de solveur

- **Basé sur un « format » (modélisation mathématique)**
  - Programmation linéaire simple
  - Programmation linéaire mixte
  - Programmation quadratique
  - Programmation par contrainte
  - …
- **De nombreux solveurs spécifiques ou génériques**

Utilisation de solveur

- **Méthode de résolution d'un solveur :**
  - Basée sur des méthodes arborescentes
  - Utilise de nombreuses techniques d'optimisation/déduction/coupes/…
  - Critère de choix :
    - Type de problème à résoudre (feasabilité vs optimilatié)
    - Performances (temps de résolution, GAP, etc.)
    - Niveau de configuration/paramétrisation
    - Prix
    - Langage de programmation

Utilisation de solveur

- **« Aider » votre solveur :**
  - Réfléchir à une modélisation pertinente
    - nombre et type de variables
    - nombre et formulation des contraintes
    - Eviter l'utilisation de grande valeur (ex : le fameux « big M »)
  - Effectuer du préprocessing
  - Générer des coupes
  - Donner une borne inférieure, une borne supérieure ou même une solution
  - Paramétrer la recherche (stratégie de branchement ou parcours de l'arbre, intensifier ou non les techniques d'amélioration des solveurs, etc.).

Utilisation de solveur

- **Illustration sur le $1|r_i,d_i|L_{max}$**
  - Données :
    - Une machine sur laquelle doit s'exécuter n tâches non préemptables
    - Chaque tâche i a une durée (p_i), une date au plus tôt d'exécution (r_i) et une date due (d_i)
  - Objectif : ordonnancer les tâches en minimisant le plus grand retard.
  - Exemple d'instance au tableau
- **Modélisation mathématique (PLNE/PLM)**
  - Choix des variables
    - t_i : date de début d'exécution de la tâche i
    - x_{ij} ou x_{ip} ou x_{it} ?

Selon vous, quel type de variable binaire est le plus pertinent …

A. **X_{ij} : 1 si la tâche i précède la tâche j, 0 sinon.**
B. **X_{ip} : 1 si la tâche i est en position p sur la machine, 0 sinon.**
C. **X_{it} : 1 si la tâche i commence son exécution à date t sur la machine, 0 sinon.**
D. **Aucune des trois, « it's a trap ! ! »**
E. **Peu importe**

Utilisation de solveur

- **Modèle :**
  - Variables :
    - t_p : date de début de la tâche en position p
    - x_{ip} : 1 si la tâche i est en position p, 0 sinon
    - L_{max} : le plus grand retard
  - Contraintes :
    - $\sum_{i=1}^{i=n} x_{ip} = 1 \qquad \forall p \in \{1..n\}$
    - $\sum_{p=1}^{p=n} x_{ip} = 1 \qquad \forall i \in \{1..n\}$
    - $t_p + \sum_{i=1}^{i=n} p_i x_{ip} \leq t_{p+1} \qquad \forall p \in \{1..n-1\}$
    - $\sum_{i=1}^{i=n} r_i x_{ip} \leq t_p \qquad \forall p \in \{1..n\}$
    - $t_p - \sum_{i=1}^{i=n} d_i x_{ip} \leq L_{max} \qquad \forall p \in \{1..n\}$
  - Fonction Objectif : min L_{max}

Utilisation de solveur

- **Une idée de préprocessing :**
  - Calculer une borne supérieure UB
  - Déterminer pour chaque tâche i le nombre de tâche qui doit forcément s'exécuter avant r_i+p_i pour respecter $L_{max} \leq UB$, noter $nbAv_i$
  - Imposer : $x_{i1} = x_{i2} = x_{i3} = \ldots = x_{i\ nbAvi} = 0$
  - Exemple au tableau
- **Une idée de coupe :**
  - Déterminer la séquence σ des tâches par ordre r_i croissant puis ajouter : $r_{\sigma(p)} \leq t_p \quad \forall p \in \{1..n\}$
  - Grâce à la UB, déduire des précédences entre tâches i et j, puis ajouter des contraintes du type : $\sum_{p=1}^{p=n} px_{ip} \leq \sum_{p=1}^{p=n} px_{jp}\ \forall (i,j)$ tq i avant j *(à tester)*
  - Exemple au tableau

Utilisation de solveur

- **Des bornes :**
  - Supérieure : EDD croissant puis recherche locale ?
  - Inférieure : résoudre le problème polynomial : $1|r_i,d_i,prmt|L_{max}$
- **Stratégie de branchement :**
  - …
- **…**

Stratégie de branchement : il est plus intéressant de brancher d'abord …

A. **Sur les variables x_{ip} puis sur les variables t_p**
B. **Sur les variables t_p puis sur les variables x_{ip}**
C. **Les variables x_{ip} suffisent**
D. **De quoi ?**

Stratégie de branchement : il est plus intéressant de brancher d'abord …

A. **Sur la valeur 0**
B. **Sur la valeur 1**
C. **Sur le valeur 220…**

Utilisation de solveur

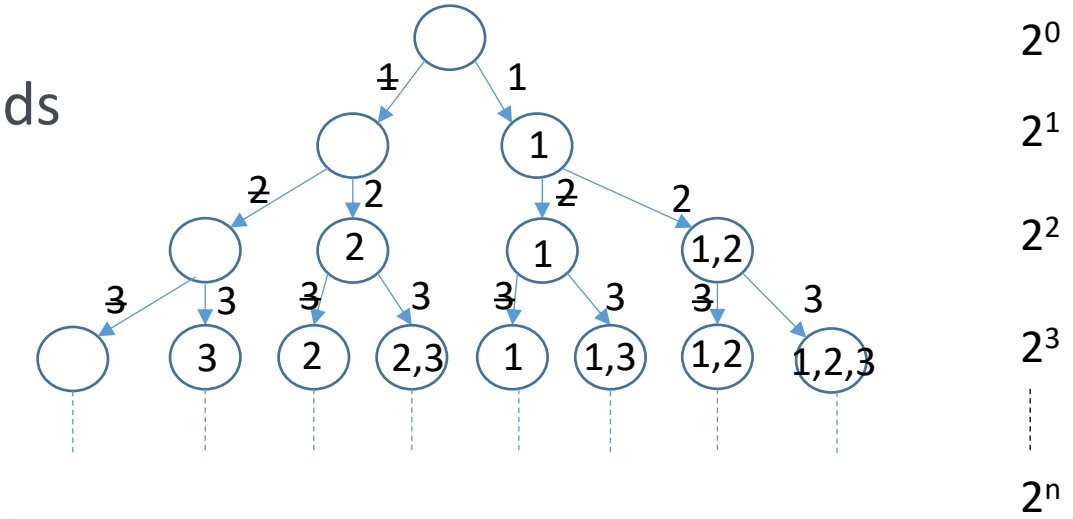
- **Il existe des méthodes de décomposition pour résoudre efficacement des PL/PLM, par exemple :**
  - L'algorithme de décomposition de Dantzig et Wolf (et génération de colonne)
  - La méthode de décomposition de Benders
  - Et d'autres…

Méthode arborescente

- **Principe :**
  - Ces méthodes sont basées sur une énumération "intelligente" des solutions admissibles d'un problème d'optimisation combinatoire sous forme d'arbre
  - Prouver l'optimalité d'une solution en partitionnant l'espace des solutions (« Diviser pour régner »)

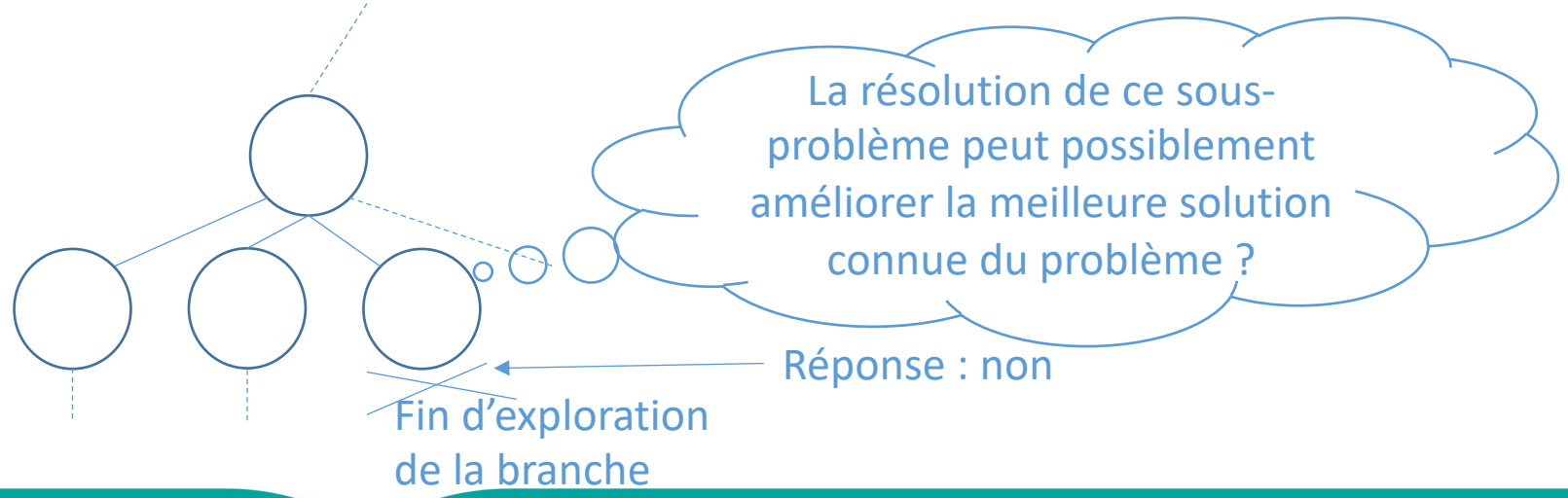
    => **Branch (séparation)**

- Soit le problème du sac à dos (n objets de poids w_i et de valeurs a_i et un sac de capacité W).
  - Branchement : objet i dans le sac ou non
  - Inutile d'énumérer les solutions non admissibles



Méthode arborescente

- **Branch and bound :**
  - Chaque nœud de l'arbre représente un sous-problème qui peut être analysé afin d'obtenir une évaluation (**bound**) de la solution optimale de ce sous-problème
  - En fonction de cette évaluation et de la meilleure solution connue du problème, des coupes de branches de l'arbre pourront être possibles.

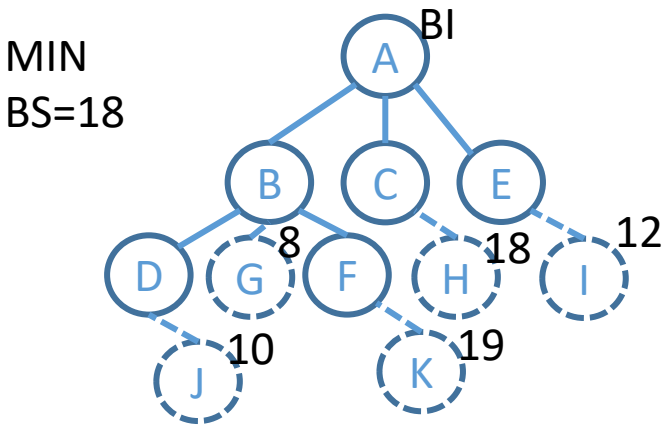


Méthode arborescente

- **Branch and bound :**
  - L'évaluation d'un nœud consiste très souvent à trouver une borne inférieure ou supérieure selon si le problème est à minimiser ou maximiser
  - Cas d'un problème à minimiser :
    - Evaluation de la meilleure solution connue : BS (borne supérieure)
    - Evaluation d'un nœud (sous-problème) : BI_i (borne inférieure nœud i)
    - La branche du nœud i est non explorée si $BI_i \geq BS$
  - La meilleure solution connue :
    - Doit être mis à jour à chaque nœud feuille si une meilleure solution est connue
    - L'initialisation d'une première solution par une heuristique permet souvent d'élaguer plus vite
  - L'évaluation d'un nœud :
    - Doit être rapide
    - Plusieurs méthodes : relaxation du PL (+simplex), algorithme se basant sur des propriété du problème, etc.

Méthode arborescente

- **Branch and bound :**
  - Parcours/exploration de l'arbre :
    - L'ordre d'exploration des nœuds est très important pour un meilleur « élaguage »
    - Les plus connus : profondeur d'abord, largeur d'abord, meilleure évaluation du sous problème, etc.
  - Afin d'être considéré comme méthode exacte, l'algorithme doit satisfaire 3 règles :
    - En appliquant systématiquement le principe de séparation, le nombre total de nœuds engendrés doit être fini
    - Aucune solution ne peut être éliminée par le principe de séparation uniquement
    - Un nœud feuille est défini comme un nœud qu'il n'est plus possible de séparer



Avec la stratégie meilleure évaluation d'abord, dans quel ordre vont être explorés les nœuds ?

A. **K H**

B. **K H J I G**

C. **G J I**

D. **G J I H**

E. **G J I H K**

Méthode arborescente

- **Branch and bound :**
  - Exemple : PSE de Little pour le PVC
    - Séparation : ajout ou retrait d'un arc dans la solution (arbre binaire)
    - Evaluation : Par réduction de la matrice de distance (Méthode hongroise)
    - Parcours : Profondeur d'abord en prenant l'arc ayant 1) une case nulle dans le tableau de matrice et 2) le plus grand surcoût minimal si on ne le prend pas
    - A chaque création de nœud :
      - Si l'arc (i,j) sélectionné : enlever la ligne i et la colonne j dans la matrice de distance et mettre la case [j,i] à $+\infty$
      - Si l'arc (i,j) n'est pas sélectionné : et mettre la case [i,j] à $+\infty$ dans matrice de distance
  - Exemple au tableau

Méthode arborescente

- **Branch and bound :**
  - Amélioration possible :
    - Inclure des règles de dominance entre nœuds à priori ou posteriori (mémorisation)
    - Eliminer la symétrie
    - Trouver des solutions réalisables aux sous-problèmes (certains nœuds) à l'aide d'heuristique (sondage)
    - …

Méthode arborescente

- **Branch and bound : Algorithme pour un problème à minimiser**
  - Initialiser une BS
  - LN <- {Racine} //liste des nœuds
  - Tant que LN!={} faire
    - Nœud N <- extraire le prochain nœud à explorer de LN (et le retirer de LN)
    - Pour chaque nœud fils nf de N
      - Si nf n'est pas un nœud feuille
        - Evaluer sa BI(nf)
        - Si BI(nf) < BS et que les règles de dominances sont respectées alors
          - Ajouter nf à LN
          - Sonder le nf pour trouver une BS(nf) (si BS actualisée : retirer les nœuds de LN tel que leur BI>=BS)
      - Sinon mettre à jour BS (si BS actualisée : retirer les nœuds de LN tel que leur BI>=BS)

Méthode arborescente

- **D'autres méthodes arborescentes performantes :**
  - Branch and cut
    - Améliorer l'évaluation de chaque nœud en ajoutant des inégalités valides
    - Algorithme de plans coupants (cf. cours PL)
    - Permet parfois d'obtenir directement la solution optimale du sous-problème du nœud
  - Branch and price
    - = Branch and bound + « Génération de colonne » à chaque nœud
    - Génération de colonne : un problème maître restreint + un problème esclave (pricing) qui ajoute des « colonnes » au problème maître
  - Branch and check
    - Décomposition du problème en deux sous problèmes, P1 (optimisation) et P2 (faisabilité)
    - P1 est résolu par un branch and bound et à chaque solution ou solution partielle de P1, on vérifie si P2 est faisable, sinon on coupe le ou les nœuds correspondants.
    - A chaque cas d'infaisabilité pour P2, des coupes/contraintes peuvent être ajoutées dans P1
    - Similaire à une décomposition de Benders.

Peut-on transférer une des ces méthodes exactes en heuristique ?

A. **Non**
B. **Ca dépend des méthodes**
C. **Ca dépend des problèmes**
D. **Oui dans tous les cas**
E. **Vous pouvez répéter la question ?**

Programmation Dynamique

- **Commençons par un exemple**
  - Soit le problème du sac à dos (n objets de poids w_i et de valeurs a_i et un sac de capacité W).
  - Soit la fonction $F_i(w)$ la valeur de la fonction objectif qui considère les i premiers objets seulement et un poids maximal w (<W).
  - L'objectif est de trouver $F_n(W)$
  - Le programme dynamique pour résoudre le problème du sac à dos est le suivant :
    - $F_i(w)=\max(F_{i-1}(w) ; a_i + F_{i-1}(w-w_i))$ avec $F_0(w) = 0$
    - Exemple au tableau

Complexité de l'algorithme ?

A. **Polynomial en O(n)**
B. **Polynomial en $O(n^2)$**
C. **Pseudo-polynomial en O(nW)**
D. **Exponentiel en $O(2^n)$**
E. **Zbradaraldjan**

Programmation Dynamique

- **Un autre exemple**
  - Soit le problème du plus court chemin hamiltonien (n villes à visiter en minimisant la distance totale parcourue).
  - Soit, i le nombre de villes visitées, e_i un ensemble de i villes, j la dernière ville visitée de e_i, la fonction $F_i(e_i,j)$ retournant le plus court chemin passant par les villes de e_i et finissant par j.
  - Le programme dynamique pour résoudre le problème du plus court chemin hamiltonien est le suivant :
    - $F_i(e_i,j) = \min_{k \in e_i \setminus j} \{F_{i-1}(e_i \setminus j,k) + d_{kj}\}$
    - Exemple au tableau

Programmation Dynamique

- **Idée générale :** consiste à décomposer un problème en une suite de sous-problèmes de même nature et de relier les solutions optimales de ces sous-problèmes par des relations de récurrence
- Un algorithme de programmation dynamique n'est pas toujours possible
- **Un problème doit posséder une sous structure optimale :** toute solution optimale contient la solution optimale aux sous-problèmes
- Pour obtenir une solution efficace, l'espace des sous-structures d'un problème ne doit pas être trop grand.

Lequel de ces problèmes n’est pas vraiment adapté à la programmation dynamique ?

A. **Le plus court chemin**
B. **2-Partition**
C. **Le plus long chemin**
D. **Rendu de monnaie**
E. **Aucune idée**