OPTIMISATION EN NOMBRES ENTIERS :

ALGORITHMES APPROCHÉS

Hacène Ouzia

MAIN (5 ème année)
Sorbonne Université

2020

AGENDA

Méthodes exactes vs méthodes approchées

DONNÉE Soit le problème d'optimisation suivant :

$$\begin{array}{ll} \min & f(x) \\ s.c. & \\ & x \in \Omega. \end{array} \tag{1}$$

QUESTION Résoudre le problème (1) ?

① Méthode exacte : sol. réalisable + certificat d'optimalité.

② Heuristique : sol. réalisable.
- pas de certificat d'optimalité.
- éventuellement, une garantie de performance.
- dépendante du problème.

③ Méta-heuristique : sol. réalisable.
- pas de certificat d'optimalité.
- pas de garantie de performance.
- indépendante du problème.

Méthodes exactes vs méthodes approchées

■ DONNÉE Soit le problème d'optimisation suivant :

$$\begin{array}{ll} \min & f(x) \\ s.c. & \\ & x \in \Omega. \end{array} \qquad (1)$$

■ QUESTION Résoudre le problème (1) ?

① Méthode exacte : sol. réalisable + certificat d'optimalité.

② Heuristique : sol. réalisable.
- ☞ pas de certificat d'optimalité.
- ☞ éventuellement, une garantie de performance.
- ☞ dépendante du problème.

③ Méta-heuristique : sol. réalisable.
- ☞ pas de certificat d'optimalité.
- ☞ pas de garantie de performance.
- ☞ indépendante du problème.

Algorithmes approchés

Définition Problème d'optimisation combinatoire

Un problème d'optimisation combinatoire P est la donnée de :

- $\texttt{Inst}(\mathbf{P})$: classe d'instances du problème P,
- $\texttt{Sol}(\mathbf{I})$: ensemble des solutions réalisables de l'instance I de P,
- $f : \texttt{Sol}(\mathbf{I}) \to \mathbb{R}$: une fonction d'évaluation, à chaque solution réalisable x elle associe sa valeur $f(x)$.

Problème de coloriage

Exemple Problème de coloriage

- $\texttt{Inst}(\textbf{Coloring})$: classe de graphes $G = \langle V, E \rangle$,
- $\texttt{Sol}(\mathbf{I})$: partitions de l'ensemble des sommets V. Les sommets de chaque partition ont la même couleur,
- $f : \texttt{Sol}(\mathbf{I}) \to \mathbb{R}$: retournant le nombre de classes.

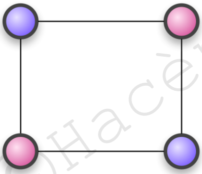

Figure – Graphe 2-coloriable.

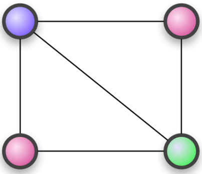

Figure – Graphe 3-coloriable.

Mesure de performance absolue

Définition Mesure absolue de performance

Soit P un problème d'optimisation combinatoire. Soit ρ un entier donné. Une heuristique A (pour le problème P) présente une garantie absolue de performance ρ si :

$$|A(I) - \texttt{Opt}(I)| \leq \rho, \ \forall I \in \texttt{Inst}(\mathbf{P}). \tag{2}$$

☞ On dira que A est un algorithme ρ-approché du problème P.

Mesure de performance absolue

■ **DÉFINITION** MESURE ABSOLUE DE PERFORMANCE

Soit P un problème d'optimisation combinatoire. Soit ρ un entier donné. Une heuristique A (pour le problème P) présente une garantie absolue de performance ρ si :

$$|A(I) - \mathtt{Opt}(I)| \leq \rho,\ \forall I \in \mathtt{Inst}(\mathbf{P}). \tag{2}$$

☞ On dira que A est un algorithme ρ-approché du problème P.

Mesure de performance absolue

Algorithme 1-approché Problème Plan-Coloring

Algorithme 1: Algorithme 1-approché pour Coloring

Entrée : $G = \langle V, E \rangle$ :: Graphe planaire

Sortie : Nombre de chromatique de G

```
Début
    χ ← 2
    Si non est2coloriable(G) alors χ ← 4
    return χ
Fin
```

- Tout graphe planaire est 4-coloriable. Mais savoir s'il est 3-coloriable est plus compliqué !!
- $|A(I) - \mathtt{Opt}(I)| \leq 1, \forall I \in \mathtt{Inst}(\textbf{Plan-Coloring})$.
- L'existence d'un algorithme approché avec garantie de performance absolue n'est pas toujours garantie (voir TD).

Mesure de performance absolue

ALGORITHME 1-APPROCHÉ PROBLÈME PLAN-COLORING

Algorithme 2: Algorithme 1-approché pour COLORING

Entrée : $G = \langle V, E \rangle$ **::** Graphe planaire

Sortie : Nombre de chromatique de G

```
Début
    χ ← 2
    Si non est2coloriable(G) alors χ ← 4
    return χ
Fin
```

- Tout graphe planaire est 4-coloriable. Mais savoir s'il est 3-coloriable est plus compliqué !!
- $|A(I) - \mathtt{Opt}(I)| \leq 1, \; \forall I \in \mathtt{Inst}$(**PLAN-COLORING**).
- L'existence d'un algorithme approché avec garantie de performance absolue n'est pas toujours garantie (voir TD).

Mesure de performance relative

Définition Mesure relative de performance

Soit P un problème d'optimisation combinatoire. Une heuristique A (pour le problème P) présente une garantie relative de performance ρ_A si :

$$\rho_A = \sup \left\{ \frac{A(I)}{\mathtt{Opt}(I)} : I \in \mathtt{Inst}(\mathbf{P}) \right\}. \tag{3}$$

- On dira que A est un algorithme ρ_A-approché du problème P.
- Pour un problème en maximisation on utilisera :

$$\rho_A = \sup \left\{ \frac{\mathtt{Opt}(I)}{(I)} : I \in \mathtt{Inst}(\mathbf{P}) \right\}.$$

- En général, on calcule un majorant de ρ_A.

Mesure de performance relative

Définition Mesure relative de performance

Soit P un problème d'optimisation combinatoire. Une heuristique A (pour le problème P) présente une garantie relative de performance ρ_A si :

$$\rho_A = \sup \left\{ \frac{A(I)}{\texttt{Opt}(I)} : I \in \texttt{Inst}(\mathbf{P}) \right\}. \tag{3}$$

- On dira que A est un algorithme ρ_A-approché du problème P.
- Pour un problème en maximisation on utilisera :

$$\rho_A = \sup \left\{ \frac{\texttt{Opt}(I)}{(I)} : I \in \texttt{Inst}(\mathbf{P}) \right\}.$$

- En général, on calcule un majorant de ρ_A.

AGENDA

Arbre couvrant de poids minimum

Énoncé Min-Span-Tree

- $\texttt{Inst}$(**Min-Span-Tree**) : classe de graphes $G = \langle V, E, \gamma \rangle$ non orientés et valués,
- $\texttt{Sol}(\mathbf{I})$: arbres couvrants de I,
- $f : \texttt{Sol}(\mathbf{I}) \rightarrow \mathbb{R}$:

$$f(I) = \sum_{e \in E[I]} \gamma(e) .$$

Exemple : arbre couvrant de poids minimum

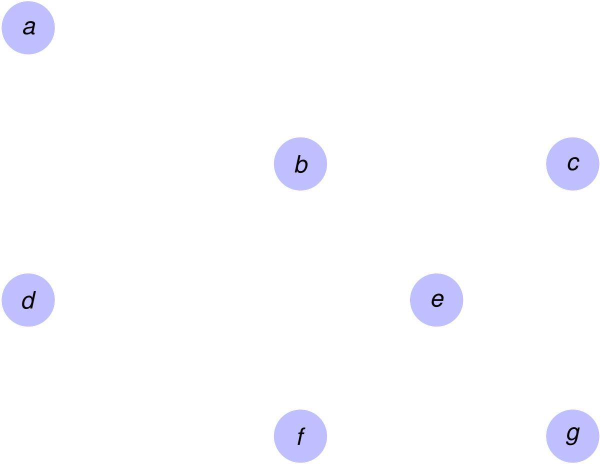

FIGURE – Instance du MIN-SPAN-TREE.

Exemple : arbre couvrant de poids minimum

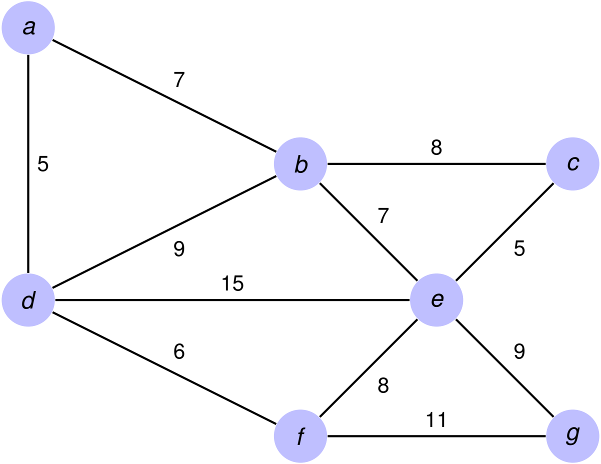

FIGURE – Instance du MIN-SPAN-TREE.

Algorithme de Kruskal

Algorithme Problème Min-Span-Tree

Algorithme 3: Algorithme de Kruskal pour le Min-Span-Tree

Entrée : $G = \langle V, E, \gamma \rangle$ :: Graphe non orienté et valué

Sortie : Arbre couvrant de poids minimum T

```
Début
    T ← ∅
    σ ← triCroissant (E, γ)
    Pour k = 1 à |E| faire
        Si acyclique(T ∪ {e_σ(k)}) alors T ← T ∪ {e_σ(k)}
    Fin pour
    return T
Fin
```

- `triCroissant` (E, γ) : tri les arêtes de E suivant l'ordre croissant. Le résultat est une permutation.
- `acyclique` $(T \cup \{e_{\sigma(k)}\})$: retourne vrai ou faux suivant que le graphe en argument est acyclique ou pas.

Algorithme de Kruskal

Algorithme Problème Min-Span-Tree

Algorithme 4: Algorithme de Kruskal pour le MIN-SPAN-TREE

Entrée : $G = \langle V, E, \gamma \rangle$ :: Graphe non orienté et valué

Sortie : Arbre couvrant de poids minimum T

```
Début
    T ← ∅
    σ ← triCroissant (E, γ)
    Pour k = 1 à |E| faire
        Si acyclique(T ∪ {e_σ(k)}) alors T ← T ∪ {e_σ(k)}
    Fin pour
    return T
Fin
```

- `triCroissant` (E, γ) : tri les arêtes de E suivant l'ordre croissant. Le résultat est une permutation.
- `acyclique` $(T \cup \{e_{\sigma(k)}\})$: retourne vrai ou faux suivant que le graphe en argument est acyclique ou pas.

Algorithme de Prim

Algorithme de Prim — Problème Min-Span-Tree

Algorithme 5: Algorithme de Prim pour le MIN-SPAN-TREE

Entrée : $G = \langle V, E, \gamma \rangle$ :: Graphe non orienté et valué

Sortie : Arbre couvrant de poids minimum T

```
Début
    T ← ∅
    X ← choisirSommet (V)
    Tant que X ≠ V faire
        e = (uv) ← argmin {γ(e) : e ∈ E, e = (xy), x ∈ X, y ∉ X}
        X ← X ∪ {v}
        T ← T ∪ {e}
    Fin tant que
    return T
Fin
```

- choisirSommet (V) : retourne un sommet quelconque de V,
- Dans le calcule de l'argmin, il s'agit de déterminer l'arête de plus petit poids appartenant à la coupe définie par l'ensemble des sommets déjà choisis X.

Algorithme de Prim

Algorithme de Prim Problème Min-Span-Tree

Algorithme 6: Algorithme de Prim pour le Min-Span-Tree

Entrée : $G = \langle V, E, \gamma \rangle$ :: Graphe non orienté et valué

Sortie : Arbre couvrant de poids minimum T

Début
  $T \leftarrow \emptyset$
  $X \leftarrow$ `choisirSommet` (V)
  Tant que $X \neq V$ **faire**
    $e = (uv) \leftarrow \texttt{argmin}\,\{\gamma(e) : e \in E, e = (xy), x \in X, y \notin X\}$
    $X \leftarrow X \cup \{v\}$
    $T \leftarrow T \cup \{e\}$
  Fin tant que
  return T
Fin

- `choisirSommet` (V) : retourne un sommet quelconque de V,
- Dans le calcule de l'`argmin`, il s'agit de déterminer l'arête de plus petit poids appartenant à la coupe définie par l'ensemble des sommets déjà choisis X.

Algorithme de Prim : exemple

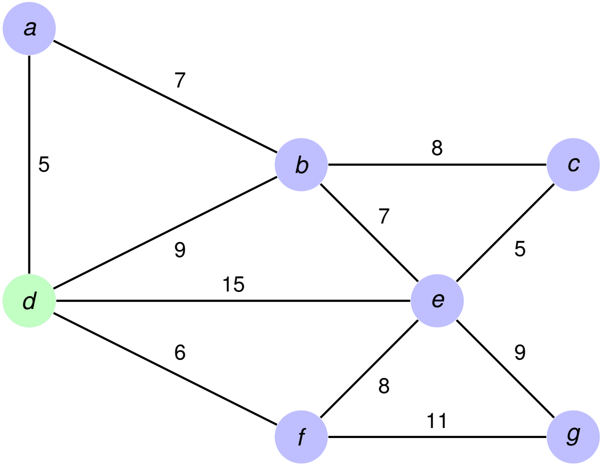

FIGURE – Exécution de l'algorithme de Prim.

Algorithme de Prim : exemple

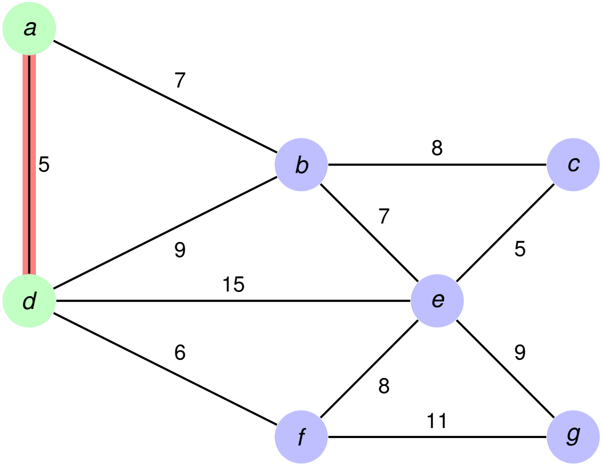

FIGURE – Exécution de l'algorithme de Prim.

Algorithme de Prim : exemple

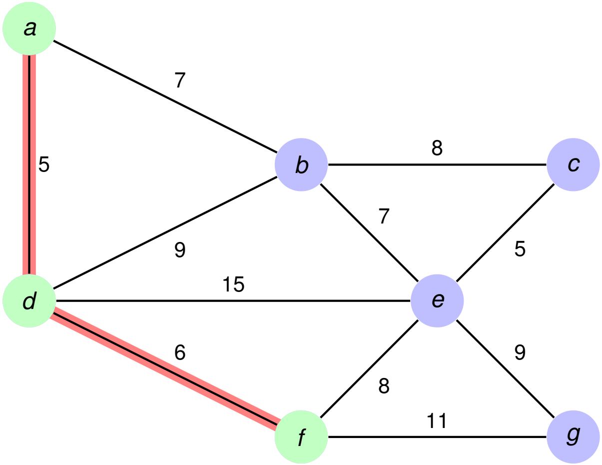

FIGURE – Exécution de l'algorithme de Prim.

Algorithme de Prim : exemple

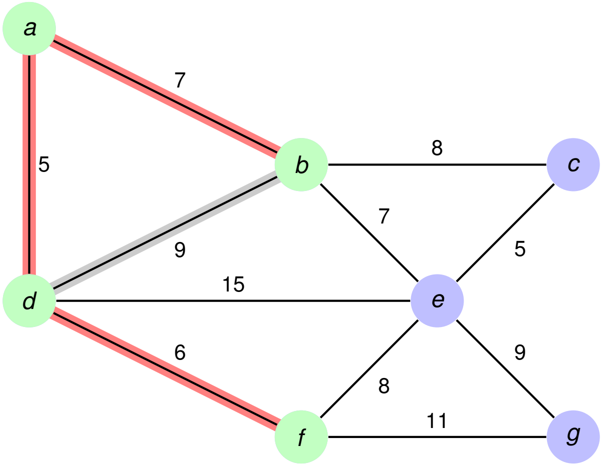

FIGURE – Exécution de l'algorithme de Prim.

Algorithme de Prim : exemple

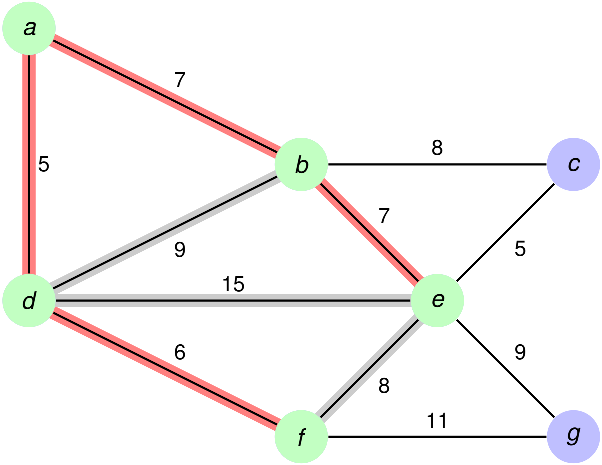

FIGURE – Exécution de l'algorithme de Prim.

Algorithme de Prim : exemple

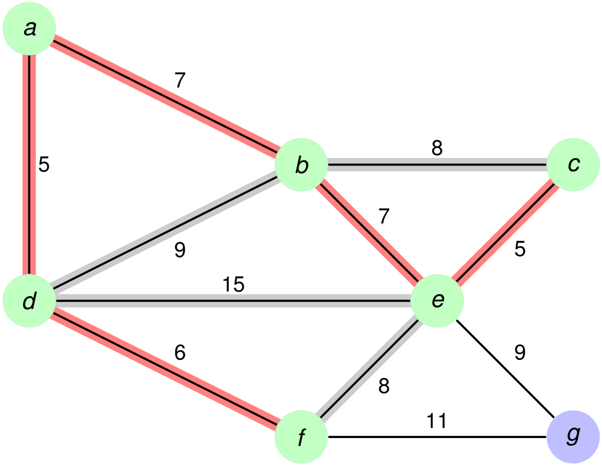

FIGURE – Exécution de l'algorithme de Prim.

Algorithme de Prim : exemple

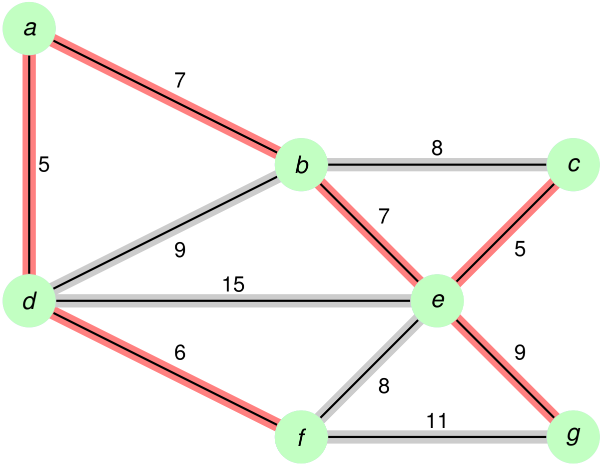

FIGURE – Exécution de l'algorithme de Prim.

Analyse des heuristiques

THÉORÈME **OPTIMALITÉ DE PRIM ET KRUSKAL**

Les deux heuristiques gloutonnes *Kuskal* et *Prim* pour le problème de l'arbre couvrant de poids minimum dans un graphe non orienté et valué sont optimales.

✍ Les preuves découlent du lemme de l'arête minimale suivant.

Analyse des heuristiques

THÉORÈME OPTIMALITÉ DE PRIM ET KRUSKAL

Les deux heuristiques gloutonnes *Kuskal* et *Prim* pour le problème de l'arbre couvrant de poids minimum dans un graphe non orienté et valué sont optimales.

✍ Les preuves découlent du lemme de l'arête minimale suivant.

Analyse des heuristiques

Lemme Arête minimale

Soit S un sous-ensemble propre de V. Considérons la coupe $\mathcal{C} = \left(S, \overline{S}\right)$. Notons X le multi-ensemble :

$$X = \texttt{argmin}\,\{\gamma\,(e) : e \in \mathcal{C}\}\,.$$

Toute solution optimale du problème Min-Span-Tree contient au moins une arête de $\mathcal{C}$.

- Supposons que $|X|$ vaut $\{e\}$;
- Si e n'appartient pas à l'arbre optimal (T) alors, de part la connexité de cet arbre, il existe un cycle dans G contenant e ; Soit e' l'arête de ce cycle appartenant à T ;
- Ainsi, l'arbre $T - e' + e$ est de poids strictement inférieur à celui de T ; Ce qui est une contradiction.

Analyse des heuristiques

LEMME ARÊTE MINIMALE

Soit S un sous-ensemble propre de V. Considérons la coupe $\mathcal{C} = \left(S, \overline{S}\right)$. Notons X le multi-ensemble :

$$X = \texttt{argmin}\left\{\gamma\left(e\right) : e \in \mathcal{C}\right\}.$$

Toute solution optimale du problème MIN-SPAN-TREE contient au moins une arête de $\mathcal{C}$.

- ✍ Supposons que $|X|$ vaut $\{e\}$;
- ✍ Si e n'appartient pas à l'arbre optimal (T) alors, de part la connexité de cet arbre, il existe un cycle dans G contenant e ; Soit e' l'arête de ce cycle appartenant à T ;
- ✍ Ainsi, l'arbre $T - e' + e$ est de poids strictement inférieur à celui de T ; Ce qui est une contradiction.

Analyse de l'heuristique de Prim

■ PREUVE VALIDITÉ DE PRIM POUR LE MIN-SPAN-TREE

- Le sous-graphe retourné par l'algorithme de Prim est bien connexe et contient $|V| - 1$ arêtes. Donc, c'est bien un arbre !
- L'arbre retourné est bien minimal. Sinon, le lemme de *l'arête minimale* se serait pas valide. Contradiction !

AGENDA

Problème du sac-à-dos fractionnaire

$$
\begin{aligned}
\textit{Max} \quad & \sum_{j=1}^{n} p_j x_j \\
\textit{s.c.} \quad & \\
& \sum_{j=1}^{n} w_j x_j \leq B \\
& x_j \in [0, 1], \qquad j = 1, \ldots, n
\end{aligned}
$$

où

- n nombre d'objets,
- p_j coût unitaire associé au j-ème objet,
- w_j poids du j-ème objet,
- B la capacité.

Heuristique

$$
\begin{aligned}
Max \quad & \sum_{j=1}^{n} p_j x_j \\
s.c. \quad & \\
& \sum_{j=1}^{n} w_j x_j \leq B \\
& x_j \in [0, 1], \qquad j = 1, \ldots, n
\end{aligned}
$$

Définition — Efficacité d'un élément

Pour un indice $j \in \{1, \ldots, n\}$ donné, *l'efficacité* du j-ème objet est la valeur suivante :

$$e_j = \frac{p_j}{w_j}$$

Heuristique

Algorithme 7: Algorithme glouton

Entrées :
n :: Entier naturel *! Nombre d'objets*
B :: Entier naturel *! Capacité du sac-à-dos*
σ :: Permutation *! Suivant l'efficacité décroissante*
$[p_{\sigma(1)}, \ldots, p_{\sigma(n)}]$ :: Tableau de réels *! coût des objets*
$[w_{\sigma(1)}, \ldots, w_{\sigma(n)}]$ :: Tableau de réels *! Poids des objets*

Sortie : $\hat{x}$ *! Solution optimale* ; z^{LP} *! Utilité optimale du sac-à-dos*

Début
- **Pour** $j = 0$ **à** n **faire** $\hat{x} \leftarrow 0$
- $j \leftarrow 1$; $z^{LP} \leftarrow 0$
- $\kappa \leftarrow w_{\sigma(j)}$ *! Capacité courante*
- **Tant que** $\kappa \leq B$ **et** $j \leq n$ **faire**
  - $x_{\sigma(j)} \leftarrow 1$; $z^{LP} \leftarrow z^{LP} + p_{\sigma(j)}$
  - $j \leftarrow j + 1$
  - $\kappa \leftarrow \kappa + w_{\sigma(j)}$
- **Fin tant que**
- **Si** $j \leq n$ **alors** $\hat{x}_{\sigma(j)} \leftarrow \dfrac{1}{w_{\sigma(j)}}(B - \kappa)$; $z^{LP} \leftarrow z^{LP} + \hat{x}_{\sigma(j)} p_{\sigma(j)}$
- **return** $\hat{x}, z^{LP}$

Fin

✍ L'heuristique 7 est optimale pour le problème du sac-à-dos fractionnaire.

Heuristique

$$\begin{aligned} Max \quad & \sum_{j=1}^{n} p_j x_j \\ s.c. \quad & \\ & \sum_{j=1}^{n} w_j x_j \leq B \\ & x_j \in \{0, 1\}, \qquad j = 1, \dots, n \end{aligned}$$

où

- n nombre d'objets,
- p_j coût unitaire associé au j-ème objet,
- w_j poids du j-ème objet,
- B la capacité (ou budget, ...).

Heuristique

Algorithme 8: Algorithme glouton

Entrées : n :: Entier naturel *! Nombre d'objets*
B :: Entier naturel *! Capacité du sac-à-dos*
σ :: Permutation *! Suivant l'efficacité décroissante*
$[p_{\sigma(1)}, \dots, p_{\sigma(n)}]$ :: Tableau de réels *! coût des objets*
$[w_{\sigma(1)}, \dots, w_{\sigma(n)}]$ :: Tableau de réels *! Poids des objets*

Sortie : $\hat{x}$ *! Solution optimale*; z^{LP} *! Utilité optimale du sac-à-dos*

Début
  Pour $j = 0$ **à** n **faire** $\hat{x} \leftarrow 0$
  $j \leftarrow 1; z^{LP} \leftarrow 0$
  $\kappa \leftarrow w_{\sigma(j)}$ *! Capacité courante*
  Tant que $\kappa \leq B$ **et** $j \leq n$ **faire**
    $x_{\sigma(j)} \leftarrow 1$; $z^{LP} \leftarrow z^{LP} + p_{\sigma(j)}$
    $j \leftarrow j + 1$
    $\kappa \leftarrow \kappa + w_{\sigma(j)}$
  Fin tant que
  Si $j < n$ **et** $Z^{LP} < p_{\sigma(j)}$ **alors**
   $Z^{LP} \leftarrow p_{\sigma(j)}; \hat{x} \leftarrow (0, \dots, 0, p_{\sigma(j)}, 0, \dots, 0)$
  return $\hat{x}, z^{LP}$
Fin

Analyse

Théorème Qualité de la solution gloutonne

L'heuristique 8 est un algorithme 2-approché pour le problème du sac-à-dos entier.

AGENDA

Problème de la couverture

Énoncé Weighted-Vertex-Cover

- Inst(**W-Vertex-Cover**) : classe de graphes $G = \langle V, E, \gamma \rangle$ non orientés avec une valuation γ sur les sommets,
- Sol(**I**) : Couvertures de I,
- f : Sol(**I**) $\to \mathbb{R}$:

$$f(I) = \sum_{v \in I} \gamma(v).$$

Identifier la couverture de poids minimum !

Problème de la couverture

Énoncé WEIGHTED-VERTEX-COVER

- `Inst`(**W-VERTEX-COVER**) : classe de graphes $G = \langle V, E, \gamma \rangle$ non orientés avec une valuation γ sur les sommets,
- `Sol`(**I**) : Couvertures de I,
- f : `Sol`(**I**) $\to \mathbb{R}$:

$$f(I) = \sum_{v \in I} \gamma(v).$$

Identifier la couverture de poids minimum !

Problème de la couverture

MODÉLISATION PROBLÈME W-VERTEX-COVER

$$
\begin{aligned}
\min \quad & \sum_{u \in V} \gamma(u)\, x_u \\
s.t. \quad & \\
& x_u + x_v \geq 1, \forall\, uv \in E, \\
& x_u \in \{0, 1\}, \forall\, u \in V.
\end{aligned}
$$

Heuristique

Problème W-Vertex-Cover Relaxation linéaire

$$
\begin{aligned}
\min \quad & \sum_{u \in V} \gamma(u)\, x_u \\
s.t. \quad & \\
& x_u + x_v \geq 1, \forall\, uv \in E, \\
& x_u \in [0, 1], \forall\, u \in V.
\end{aligned}
\tag{4}
$$

Heuristique

Heuristique Problème W-Vertex-Cover

Algorithme 9: Heuristique pour le W-Vertex-Cover

Entrée : $G = \langle V, E, \gamma \rangle$ :: Graphe non orienté sommets-valués.

Sortie : η :: Taille d'une couverture de G

```
Début
    x̂ ← resoudreRelaxationWVCover(G)
    Pour u ∈ V faire
        ŷ_u ← 0
        Si x̂_u ≥ 1/2 alors ŷ_u ← 1
    Fin pour
    η ← 0
    Pour u ∈ V faire
        η ← η + γ(u) y_u
    Fin pour
    return η
Fin
```

☞ `resoudreRelaxationWVCover`(G) : retourne la solution optimale du problème d'optimisation linéaire (4).

Analyse

■ THÉORÈME

L'heuristique 9 est un algorithme 2-approché.

✍ Nous avons :

$$
\begin{aligned}
H(I) &= \sum_{u \in V} \gamma(u)\, \hat{y}_u \\
&\leq \sum_{u \in V} 2\gamma(u)\, \hat{x}_u \\
&\leq 2\mathtt{Opt}(I).
\end{aligned}
$$

Analyse

■ THÉORÈME

L'heuristique 9 est un algorithme 2-approché.

✍ Nous avons :

$$\begin{aligned} H(I) &= \sum_{u \in V} \gamma(u)\, \hat{y}_u \\ &\leq \sum_{u \in V} 2\gamma(u)\, \hat{x}_u \\ &\leq 2\mathtt{Opt}(I). \end{aligned}$$

AGENDA

Le problème MAX-3SAT

■ ÉNONCÉ MAX-3SAT

- $\mathtt{Inst}$(**MAX-3SAT**) : un ensemble fini de variables propositionnelles $\mathbb{V}$; Une formule propositionnelle conjonctive normale ϕ.
- $\mathtt{Sol}(\mathbf{I})$: l'ensemble $\{0, 1\}^n$.
- $f : \mathtt{Sol}(\mathbf{I}) \to \mathbb{R}$:

$$f(I) = \sharp\text{clauses satisfaites.}$$

Heuristique

Algorithme Problème Max-3SAT

Algorithme 10: Heuristique pour le Max-3SAT

Entrée : $\phi = \bigwedge_{k=1}^{m} C_k$ :: Formule CNF sur $\{x_1, \ldots, x_n\}$

Sortie : η :: Nombre de clauses satisfaite dans ϕ

```
Début
    Pour k = 1 à n faire
        x_k ← 0
        ρ ← alea(0, 1)
        Si ρ ≥ 1/2 alors x_k ← 1
    Fin pour
    η ← 0
    Pour k = 1 à m faire
        Si C_k alors η ← η + 1
    Fin pour
    return η
Fin
```

☞ `alea` $(0, 1)$: retourne un réel appartenant à l'intervalle $[0, 1]$.

Analyse de l'heuristique

Définition Mesure absolue de performance (cas randomisé)

Soit P un problème d'optimisation combinatoire. Soit ρ un entier donné. Une heuristique A (pour le problème P) présente une garantie absolue de performance ρ si :

$$\mathbb{E}\left(A\left(I\right)\right) \leq \rho \mathtt{Opt}(I), \; \forall I \in \mathtt{Inst}(\mathbf{P}). \tag{5}$$

☞ La valeur $\mathbb{E}\left(A\left(I\right)\right)$ représente le coût espéré.

Analyse de l'heuristique

DÉFINITION MESURE ABSOLUE DE PERFORMANCE (CAS RANDOMISÉ)

Soit P un problème d'optimisation combinatoire. Soit ρ un entier donné. Une heuristique A (pour le problème P) présente une garantie absolue de performance ρ si :

$$\mathbb{E}\left(A(I)\right) \leq \rho \mathtt{Opt}(I),\ \forall I \in \mathtt{Inst}(\mathbf{P}). \tag{5}$$

☞ La valeur $\mathbb{E}\left(A(I)\right)$ représente le coût espéré.

Analyse de l'heuristique

THÉORÈME

L'heuristique 10 est un algorithme $\frac{8}{7}$-approché.

- Considérer $\{X_1, \ldots, X_n\}$ des variables aléatoires uniformes sur l'ensemble $\{0, 1\}$;
- Le nombre de clauses satisfaites vaut :

$$Y = \sum_{j=1}^{m} Y_j,$$

où : Y_j vaut $\max\{X_{j_1}, X_{j_2}, X_{j_3}\}$ avec $\{j_1, j_2, j_3\}$ les indices des littéraux formant la clause C_j.
- Le coût moyen vaut

$$\mathbb{E}(Y) = \frac{7}{8} m.$$

Analyse de l'heuristique

THÉORÈME

L'heuristique 10 est un algorithme $\frac{8}{7}$-approché.

- Considérer $\{X_1, \ldots, X_n\}$ des variables aléatoires uniformes sur l'ensemble $\{0, 1\}$;
- Le nombre de clauses satisfaites vaut :

$$Y = \sum_{j=1}^{m} Y_j,$$

où : Y_j vaut $\max\left\{X_{j_1}, X_{j_2}, X_{j_3}\right\}$ avec $\{j_1, j_2, j_3\}$ les indices des littéraux formant la clause C_j.
- Le coût moyen vaut

$$\mathbb{E}\left(Y\right) = \frac{7}{8} m.$$

Bibliographie

- M. R. Garey and D. S. Johnson (1979),
  Computers and Intractability,
  Freeman
- I. Wegener (2005),
  Complexity Theory (1997),
  Springer
- Ding-Zhu Du, Ker-I Ko and Xiaodong Hu (2012),
  Design and Analysis of Approximation Algorithms,
  Springer
- J. Kleinberg and E. Tardos (2006),
  Algorithm Design,
  Pearson International Edition