Cours 5': Algorithmes gloutons

Olivier Bournez

bournez@lix.polytechnique.fr
LIX, Ecole Polytechnique

2011-12

Algorithmique

Principe général

- Pour un problème d'optimisation, on construit la solution de façon séquentielle, en faisant à chaque étape le meilleur choix local.
- Pas de retour en arrière: on va directement vers une solution.
- Progression descendante = choix puis résolution d'un problème plus petit.

Location d'un camion

- On veut offrir un unique véhicule à la location.
- On veut maximiser le nombre de clients satisfaits.
- Soit E l'ensemble des demandes. Pour $e \in E$, on note $d(e)$ et $f(e)$ pour la date de début et fin de la demande.
- Les demandes e_1 et e_2 sont compatibles si

$$]d(e_1), f(e_1)[\cap]d(e_2), f(e_2)[= \emptyset.$$

- On cherche un sous ensemble de E de demandes deux-à-deux compatibles de cardinal maximal.

Algorithme

$LOCATION - CAMION(E)$

- Trier les éléments de E par date de fin croissante $f(e_1) \leq f(e_2) \leq \cdots \leq f(e_n)$.
- $s_1 := e_1$
- $k := 1$
- Pour $i := 2$ à n
  - si $d(e_i) \geq f(s_k)$ alors
    - $s_{k+1} := e_i$
    - $k := k + 1$
- Retourner $s_1, \ldots, s_k$.

- Théorème: l'algorithme donne bien un résultat optimal.
- Complexité: $O(n \log n)$.
- Remarques:
  1. ne marche pas pour maximiser le temps local de location;
  2. ne marche pas si on trie par durées décroissantes.

Exemple

	e_1	e_2	e_3	e_4
d	0	0	2	3
f	1	3	6	6

Résultat: $\{e_1, e_3\}$.

Preuve du théorème

- Des demandes $x_1, \ldots, x_r$ ordonnées par date de fin croissantes sont compatible ssi

$$d(x_1) < f(x_1) \leq d(x_2) < f(x_2) \cdots d(x_r) < f(x_r).$$

- On montre par récurrence sur $1 \leq j \leq k$ que (HR_j) "il existe une solution optimale $t_1, \ldots, t_l$ avec $t_1 = s_1, \ldots, t_j = s_j$.
- On montrera ensuite qu'une solution optimale $t_1, \ldots, t_l$ avec $t_1 = s_1, \ldots, t_k = s_k$ vérifie $k = l$.
-

HR_1

- Pour (HR_1): si $t_1, \ldots, t_l$ est une solution optimale ordonnée par date de fin croissantes, on a

$$d(t_1) < f(t_1) \leq d(t_2) < f(t_2) \cdots d(t_r) < f(t_r).$$

- s_1 est la demande qui termine le plut tôt, donc $f(s_1) \leq f(t_1)$ et donc

$$d(s_1) < f(s_1) \leq d(t_2) < f(t_2) \cdots d(t_r) < f(t_r).$$

- $\{s_1, t_2, \ldots, t_l\}$ est une solution optimale qui prouve (HR_1).

$HR_j \rightarrow HR_{j+1}$

- Supposons (HR_j). Soit $t_1, \ldots, t_l$ une solution optimale ordonnée par date de fin croissantes avec $t_1 = s_1, \ldots, t_j = s_j$.
- Par construction s_{j+1} est une demande de $E - \{s_1, \ldots, s_j\}$ compatible avec $s_1, \ldots, s_j$ de date de fin minimale.
- t_{j+1} est une demande de $E - \{s_1, \ldots, s_j\}$ compatible avec $s_1, \ldots, s_j$: donc $f(s_{j+1}) \leq f(t_{j+1})$.
- On en déduit que s_{j+1} est compatible avec $t_{j+2}, \cdots, t_l$, et donc $\{s_1, \ldots, s_j, s_{j+1}, t_{j+2}, \cdots, t_l\}$ est une solution optimale qui prouve (HR_{j+1}).

Nécessairement $k = l$

Si par l'absurde $k < l$, alors

- t_{k+1} est une demande compatible avec les demandes précédentes,
- Absurde, car l'algorithme aurait pris t_{k+1}, considéré après s_k, si c'était le cas.

Rappels: graphes

- Graphe $G = (S, A)$: S ensemble de sommets, E ensemble d'arêtes. Une arête est une paire de sommets.
- Chemin: suite d'arêtes

$$e_1 = \{x_1, x_2\}, e_2 = \{x_2, x_3\}, \cdots, e_p = \{x_p, x_{p+1}\}$$

- Cycle: chemin tel que $x_{p+1} = x_1$.
- Chemin élémentaire: qui ne passe pas deux fois par le même sommet.
- Graphe connexe: toute paire de sommets est reliée par un chemin.

Arbres, Forêts

- Arbre: graphe connexe sans cycle.
- Forêt: graphe sans cycle.
- Théorème: si $G = (S, A)$ connexe, alors G est un arbre ssi $a = s - 1$.
- Arbre couvrant de G = sous-ensemble $A' \subset A$ tel que (S, A') soit un arbre (connexe sans cycle).

Arbres couv. de pds maximum

- Pondération $w : A \to R^{\geq 0}$.
- Pour $A' \subset A$, le poids de A' est

$$f(A') = \sum_{x \in A'} w(x).$$

- But: trouver un arbre couvrant de poids maximum.

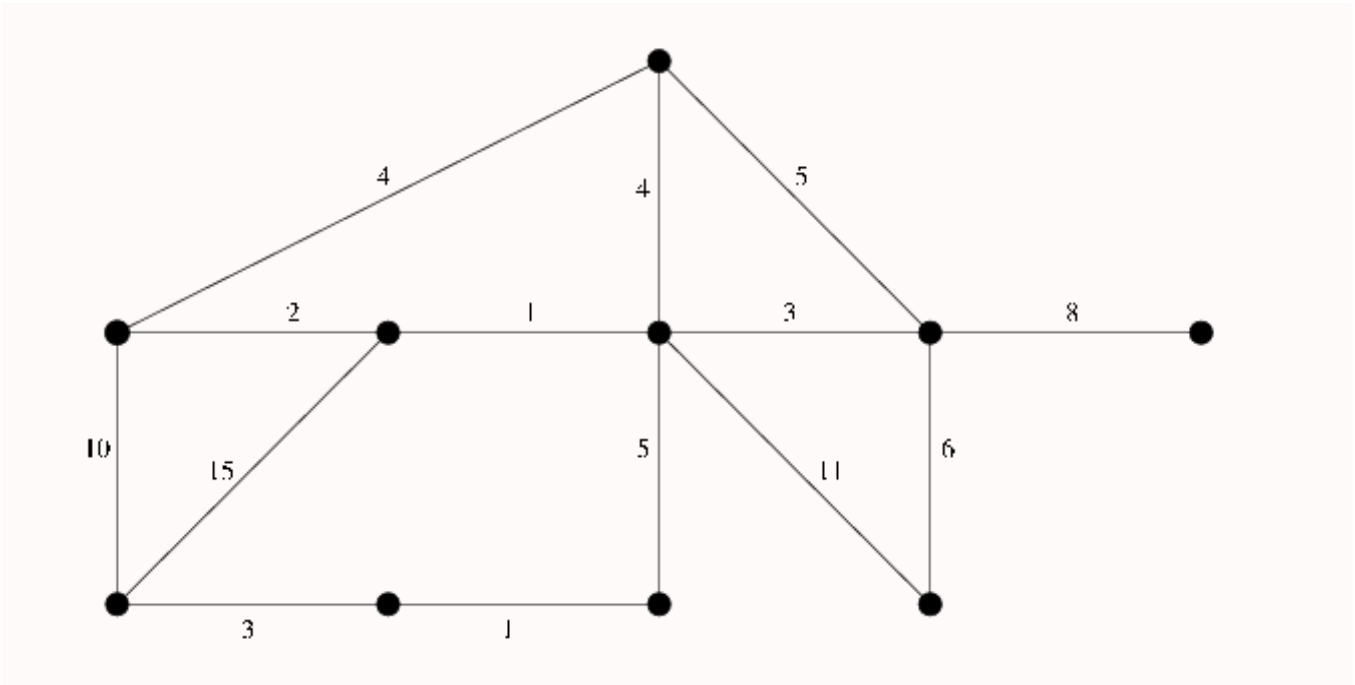
Algorithme de Kruskal

Algorithme *Kruskal*(G, w)

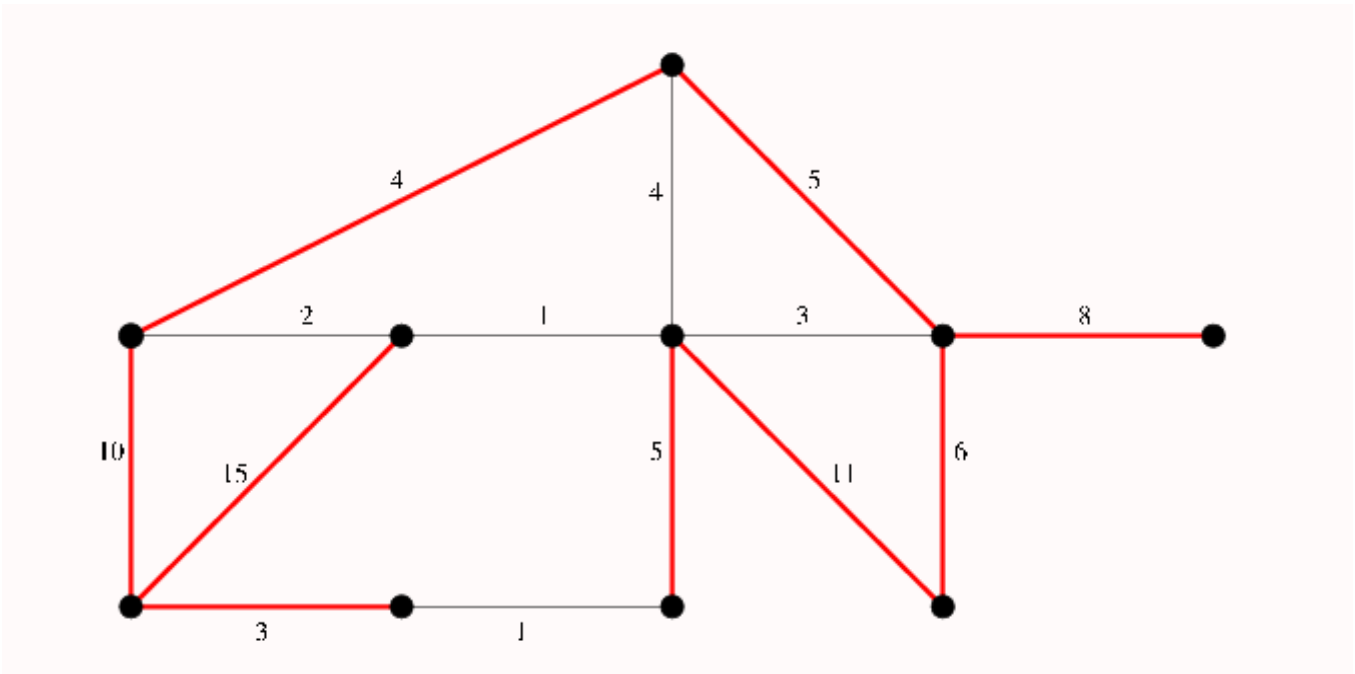
- Trier les éléments de A par ordre de poids décroissants $w(e_1) \geq w(e_2) \geq \ldots \geq w(e_a)$
- $T := \emptyset$
- Pour $i := 1$ à a
  - si $T \cup \{e_i\}$ est acyclique alors
    - $T := T \cup \{e_i\}$
- Retourner T.

Remarque: à chaque étape T est une forêt.

Exemple



Résultat



Optimalité

- Théorème: l'algorithme retourne bien une solution optimale.
- Conséquence du lemme suivant:
- Lemme: Soit T et U deux arbres couvrants de G. Soit $a \in U$, $a \notin T$, alors il existe $b \in T$ tel que $U - a + b$ soit un arbre couvrant. De plus, b peut être choisi dans le cycle formé par des arêtes de T et de a.

Preuve du Lemme

Preuve:

- la suppression de a dans U divise l'arbre en 2 composantes connexes U_1 et U_2.
- Le cycle γ crée par a dans T contient nécessairement un nombre pair d'arêtes reliant ces deux composantes.
- Comme a est une telle arête, il en existe au moins une autre b dans T.
- On vérifie qu'une telle arête b satisfait bien aux conditions du lemme.

Preuve du théorème

Preuve:

- Soit T l'arbre donné par l'algorithme glouton.
- Si l'algorithme ne donne pas l'optimal c'est qu'il existe des arbres couvrants de poids supérieur.
- Notons U celui parmi ceux-ci dont le nombre d'arêtes communes avec T est maximal.
- Soit a l'arête de U de plus grand poids qui n'est pas dans T, et b l'arête donnée par le lemme pour ce cas.
- Comme a n'a pas été choisie par l'algorithme glouton, c'est qu'elle crée un cycle avec les éléments de T, et ce cycle est formé d'arètes toutes de poids supérieur ou égal à $w(a)$.

- Puisque b est dans ce cycle, on a $w(b) \geq w(a)$.
- Ainsi $f(U - a + b) \geq f(U) > f(T)$, mais $U - a + b$ est un arbre qui a une arête commune avec T de plus que U.
- Ceci contredit le choix de U.

Remarques

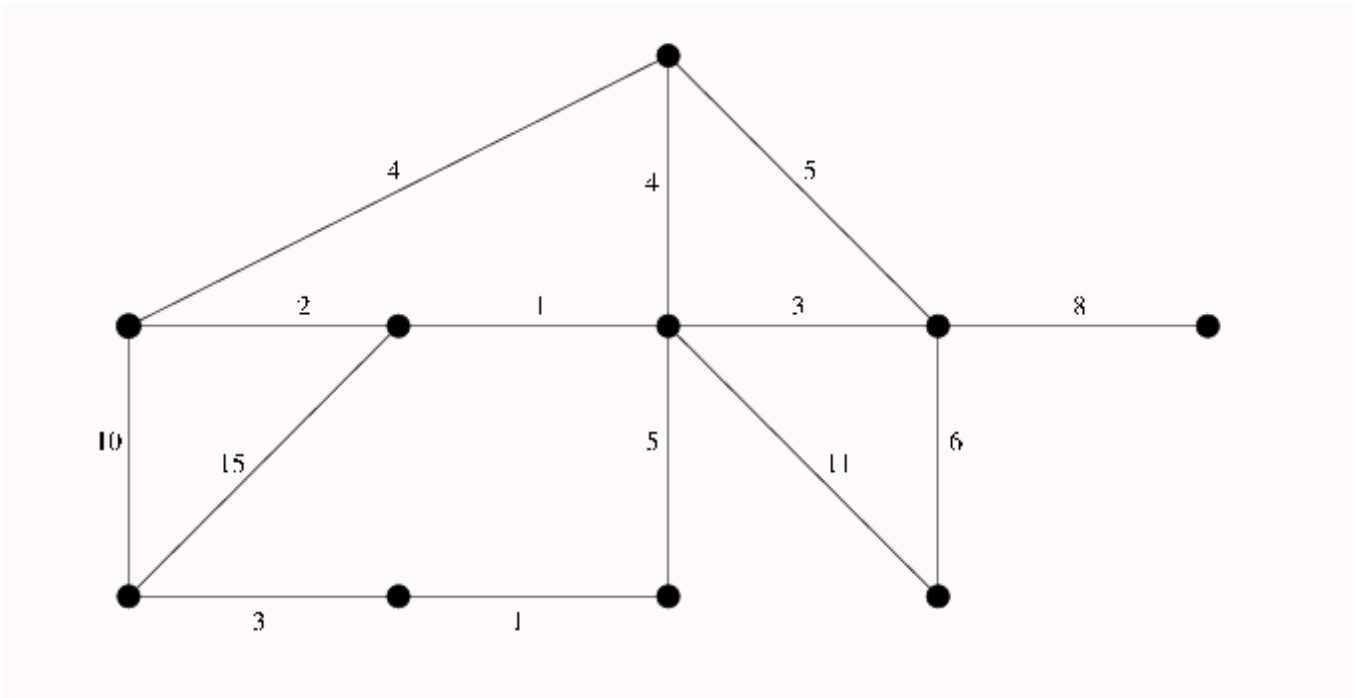
- L'algorithme fonctionne si G n'est pas connexe.
- On peut aussi obtenir l'arbre de poids MINIMUM: considérer $w'(e) = M - w(e)$ pour $M = \max\{w(e)|e \in A\}$.

Complexité: $O(a \log s)$.

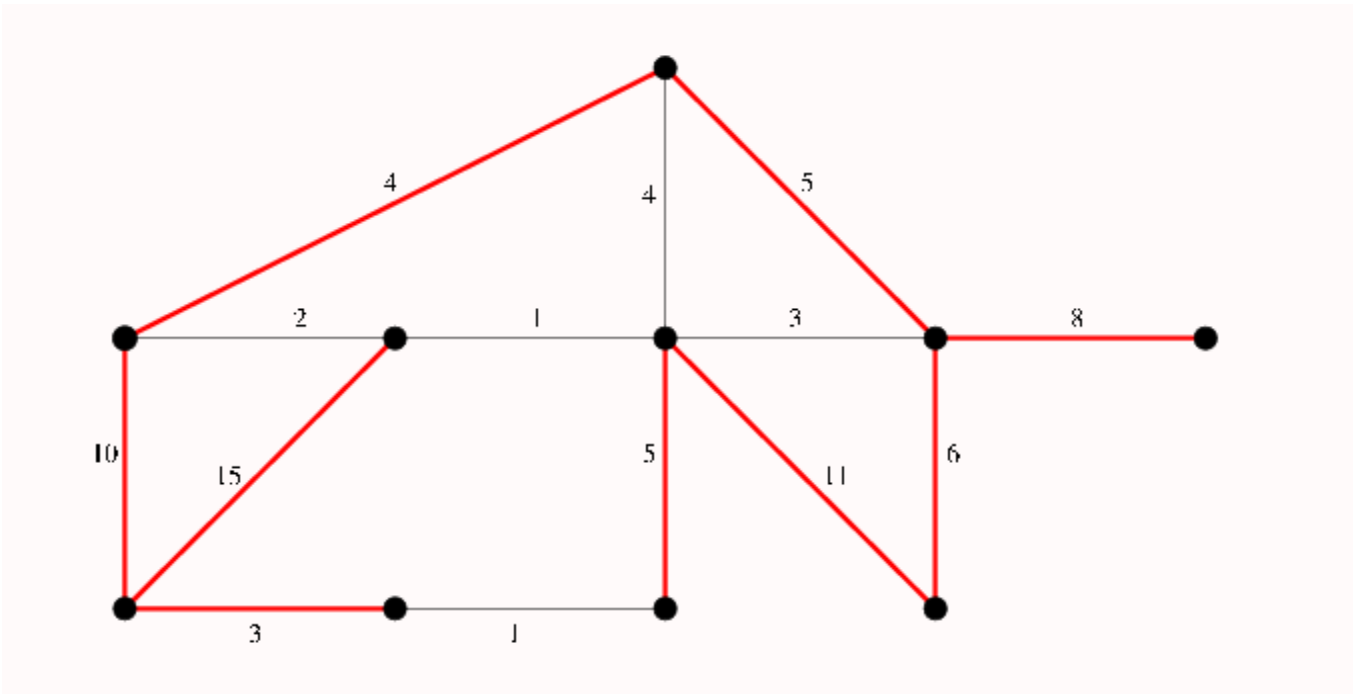
Algorithme de Prim

- Alternative: G supposé connexe.
- Algorithme $PRIM(G, w, v_0)$ ou v_0 est un sommet.
  - $T := \emptyset$
  - Tant que T n'est pas couvrant faire
    - choisir une arête de poids maximum tel que $T \cup \{e\}$ soit un arbre contenant v_0.
    - $T := T \cup \{e\}$
- Remarque: à chaque étape T est une forêt contenant v_0.
- Complexité: $O(a \log s)$, comme Kruskal.

Exemple



Résultat



Exercice: pièces de monnaies

On considère le problème où l'on doit rendre la monnaie pour x euros avec le moins possible de pièces de monnaies.

- On a des pièces de 1, 2, 5, 10. Écrire l'algorithme glouton qui donne une solution optimale.
- On dispose d'un ensemble de pièces $c^0, c^1, c^2, \ldots, c^k$ pour $c > 1$, et $k \geq 1$. Écrire l'algorithme glouton qui donne la solution optimale.
- Donner un ensemble de pièces tel que l'algorithme glouton ne retourne pas une solution optimale.
- Donne un algorithme qui retourne une solution optimale pour n'importe quel ensemble de pièces.

Théorie des matroïdes

- (S, I) est un matroïde si S est un ensemble de n éléments, I est une famille de parties de S ($I \subset \mathcal{P}(S)$) vérifiant
  1. "Hérédité": $X \in I$ implique $\forall Y \subset X,\ Y \subset I$
  2. "Échange": $(A, B \in I, |A| < |B|)$ implique $\exists x \in B - A$ tel que $A \cup \{x\} \in I$.
- Les éléments de I sont appelés "indépendants".
- Exemples
  1. Les familles libres d'un espace vectoriel.
  2. Les forêts d'un graphe (hérédité évidente, propriété d'échange = lemme précédent).

Propriétés

- Un indépendant est dit “maximal” s’il est maximal au sens de l’inclusion.
- Prop: tous les indépendants maximaux ont le même cardinal.
- Preuve: si ce n’était pas le cas, on pourrait les étendre par la propriété d’échange.

Matroïde pondéré

- On ajoute une fonction de poids $w : S \to \mathbb{N}$,
- On pose

$$w(X) = \sum_{x \in X} w(x).$$

- Question: étant donné un matroïde pondéré, trouver un indépendant de poids maximal.

Algorithme glouton

- Trier les éléments de S par poids décroissants

$$w(s_1) \geq w(s_2) \geq \cdots \geq w(s_n).$$

- $A := \emptyset$.
- Pour $i := 1$ à $n = |S|$
  - Si $A \cup \{s_i\} \in I$ alors
    - $A := A \cup \{s_i\}$
- Retourner A.

Théorème: Cet algorithme retourne une solution optimale.

Preuve du théorème

- Soit s_k le premier élément indépendant de S (= le premier indice i de l'algorithme précédent tel que $\{s_i\} \in I$.
- Lemme: Il existe une solution optimale qui contient s_k.

- Soit B un indépendant de poids maximal, et A la solution retournée par l'algorithme. On a $s_k \in A$.
- Supposons $s_k \notin B$ (sinon, rien à prouver): on applique $|B| - 1$ fois la propriété d'échange pour compléter $\{s_k\}$ par des éléments de B: on obtient un indépendant A' qui contient tous les éléments de B sauf un seul que l'on peut appeler b_i.
- On a $w(A') = w(B) - w(b_i) + w(s_k)$, or $w(s_k) \geq w(b_i)$, car $\{b_i\} \in I$ (hérédité) et b_i a été considéré après s_k par ordre décroissant.
- Donc $w(A') \geq w(B)$, donc $w(A') = w(B)$, et A' est une solution optimale qui contient s_k.

- Puis par récurrence, on obtient que glouton donne la solution optimale: on se restreint à une solution contenant $\{s_k\}$, et on recommence avec

$$S' = S - \{s_k\}$$

et

$$I' = \{X \subset S' | X \cup \{s_k\} \in I\}.$$

Application: ordonnancement

Le problème de l'ordonnancement de tâches sur un seul processeur se compose

- d'un ensemble de taches $S = \{1..n\}$ de durée 1,
- avec des dates limites $d_1, \ldots, d_n$: la tâche i doit finir avant la date d_i, sinon on doit payer la pénalité w_i.

Problème: comment ordonnancer pour minimiser la somme des pénalités.

- Définition: Un ensemble A de tâches est dit "indépendant" si il est possible de les ordonnancer tel que personne ne soit en retard.
- $N_t(A)$: nombre de tâches de l'ensemble A dont la date limite est $\leq t$.
- Proposition: Les 3 propriétés suivantes sont équivalentes
  - A est indépendant
  - Pour $t = 1..n$, on a $N_t(A) \leq t$.
  - Si les tâches de A sont ordonnancées dans l'ordre croissant de leur dates limites, alors aucune tache n'est en retard.

Preuve

- $1 \rightarrow 2$: supposons $N_t(A) > t$, alors il y a au moins une tâche qui se déroulera après le temps t, et donc A ne peut pas être indépendant.
- $2 \rightarrow 3$: trivial.
- $3 \rightarrow 1$: par définition.

Exercice: en déduire un algorithme pour tester l'indépendance d'un ensemble A.

Remarque

- Minimiser les pénalités des tâches en retard = Maximaliser les pénalités des tâches qui ne sont pas en retard.
- Théorème: (S, I), où S est l'ensemble des tâches, I la famille des sous-ensembles de tâches indépendantes est un matroïde.

Preuve

- Hérédité: triviale.
- Échange: Soit A, B indépendants avec $|A| < |B|$. Il faut montrer qu'il existe $x \in B$ tel que $A \cup \{x\}$ soit indépendant.
  Idée: comparer $N_t(A)$ et $N_t(B)$ pour $t = 1, 2, \ldots, n$.
  On cherche le plus grand $t \leq n$ tel que $N_t(A) \geq N_t(B)$. On sait que $t < n$ donc $N_t(A) \geq N_t(B)$, $N_{t+1}(A) < N_{t+1}(B)$, $\ldots$, $N_n(A) < N_n(B)$.
  Dans B il y a plus de tâches de date limite $t+1$ que dans A: on en prend une x qui n'est pas déjà dans A: $A \cup \{x\}$ est indépendant.

Algorithme glouton

- Trier les éléments de S par date limites décroissantes

$$w(s_1) \geq w(s_2) \geq \cdots \geq w(s_n).$$

- $A := \emptyset$.
- Pour $i := 1$ à $n = |S|$
  - Si $A \cup \{s_i\}$ est indépendant alors
    - $A := A \cup \{s_i\}$
- Retourner A.

Exemple

- Entrée: 7 tâches

i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	7	6	5	4	3	2	1

- Itérations:
  - $A = \{1\}$;
  - $A = \{2, 1\}$;
  - $A = \{2, 1, 3\}$
  - $A = \{2, 4, 1, 3\}$;
  - $A = \{2, 4, 1, 3, 7\}$.
  - Résultat: $\{2, 4, 1, 3, 7\}$.