Algorithmes Gloutons

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 $f(e_1) \leq f(e_2) \leq \ldots \leq f(e_n)$.
- $s_1 := e_1$
- $k := 1$
- Pour $i := 2$ à n
  - si $d(e_i) \geq f(s_k)$ alors
    - $s_{k+1} := c_i$
    - $k := k + 1$
- 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 $t_1 = s_1$,...,$t_j = s_j$.
- 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 $G = (V, E)$: V 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 $G = (V, E)$ connexe, alors G est un arbre ssi $v = a - 1$.
- Arbre couvrant de G = sous-ensemble $E' \subset E$ tel que (V, E') soit un arbre (connexe sans cycle).

Arbres couv. de pds maximum

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

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

- But: trouver un arbre couvrant de poids maximum.

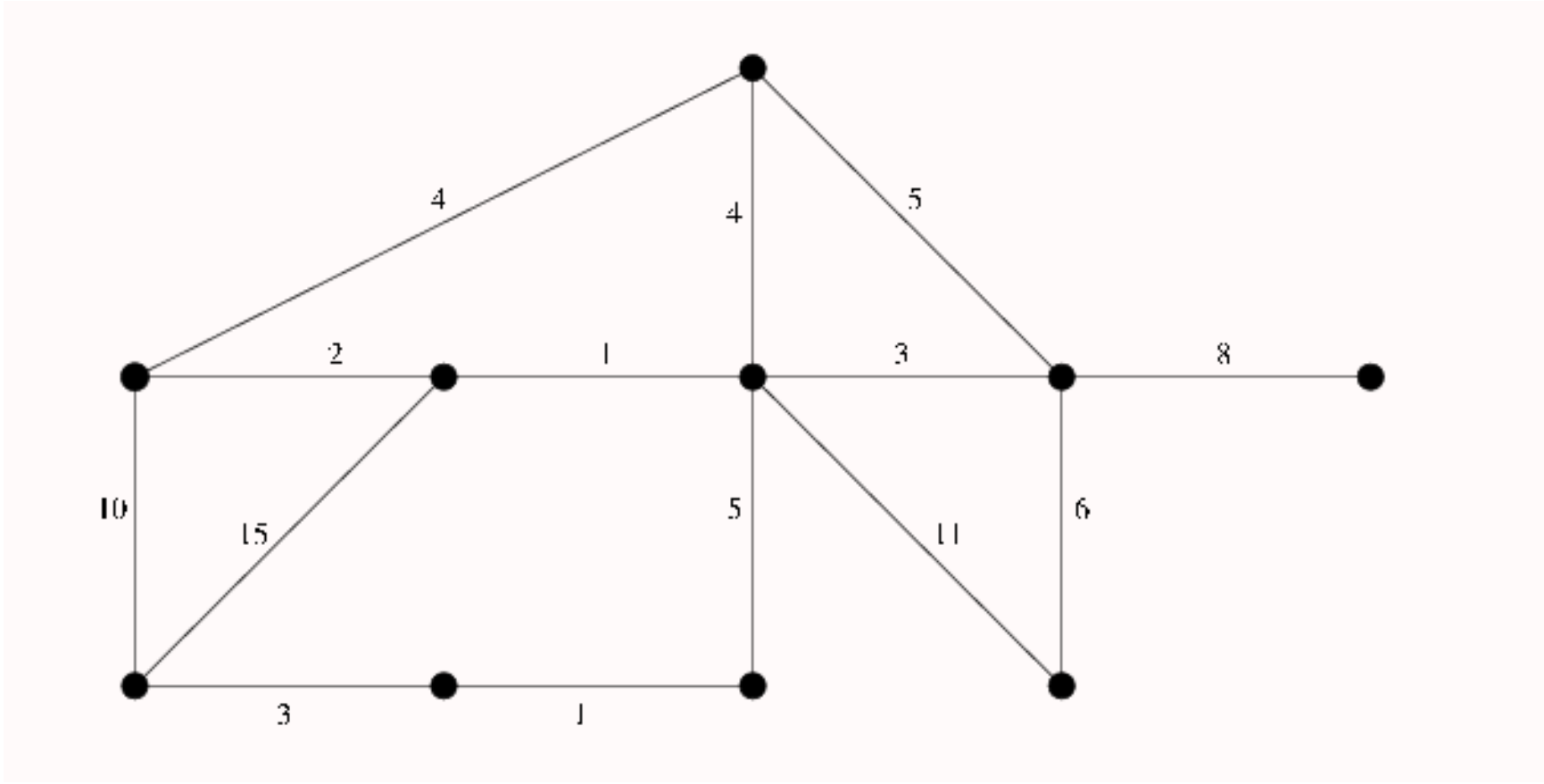
Algorithme de Kruskal

Algorithme $Kruskal(G, w)$

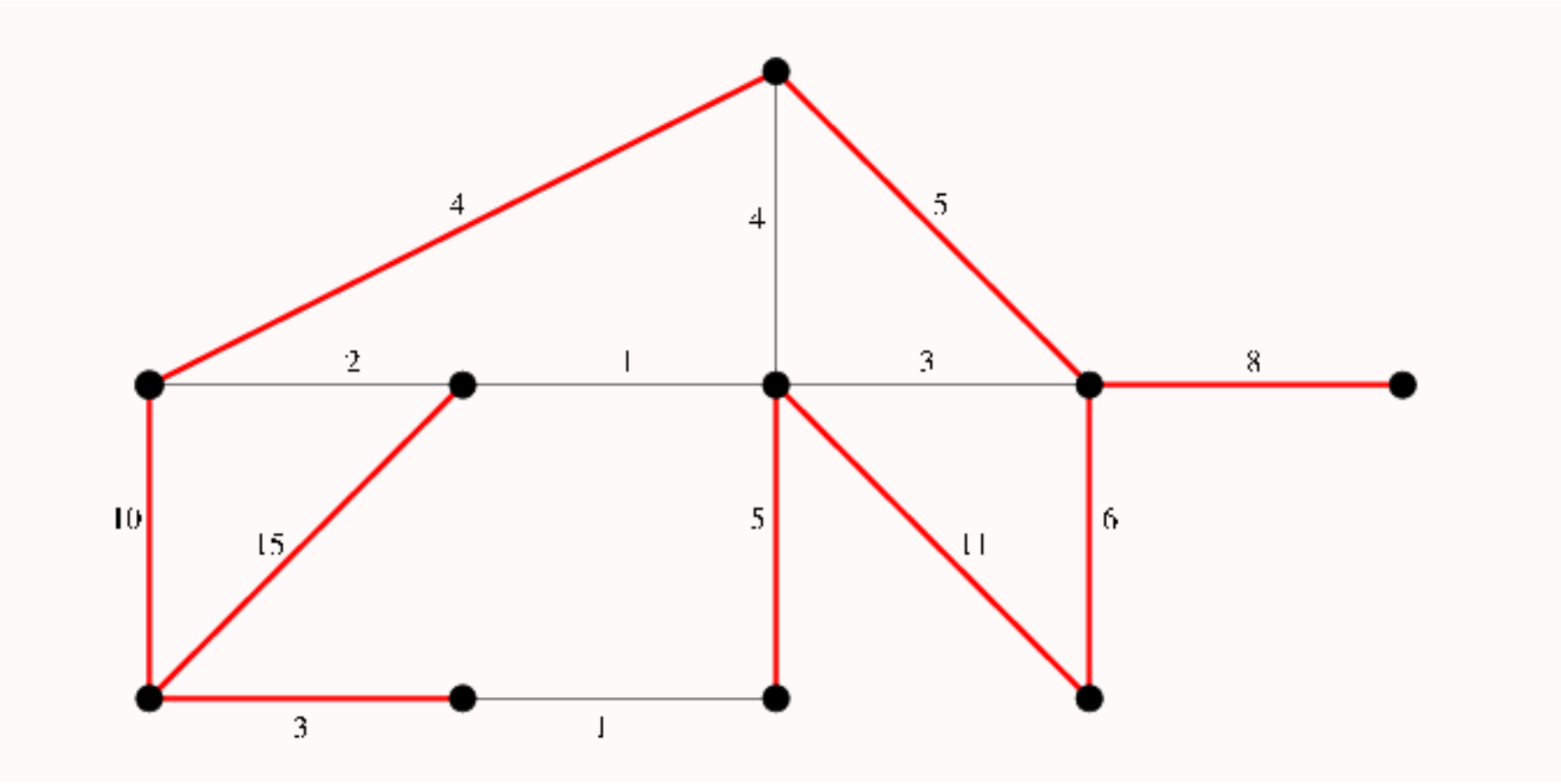
- Trier les éléments de E par ordre de poids décroissants
  $w(e_1) \geq w(e_2) \geq \ldots \geq w(e_a)$
- $T := \emptyset$
- Pour $i := 1$ à $|E|$
  - si $T \cup \{e_i\}$ est acyclique alors
    - $T := T \cup \{e_i\}$
- Retourner T.

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'arêtes 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

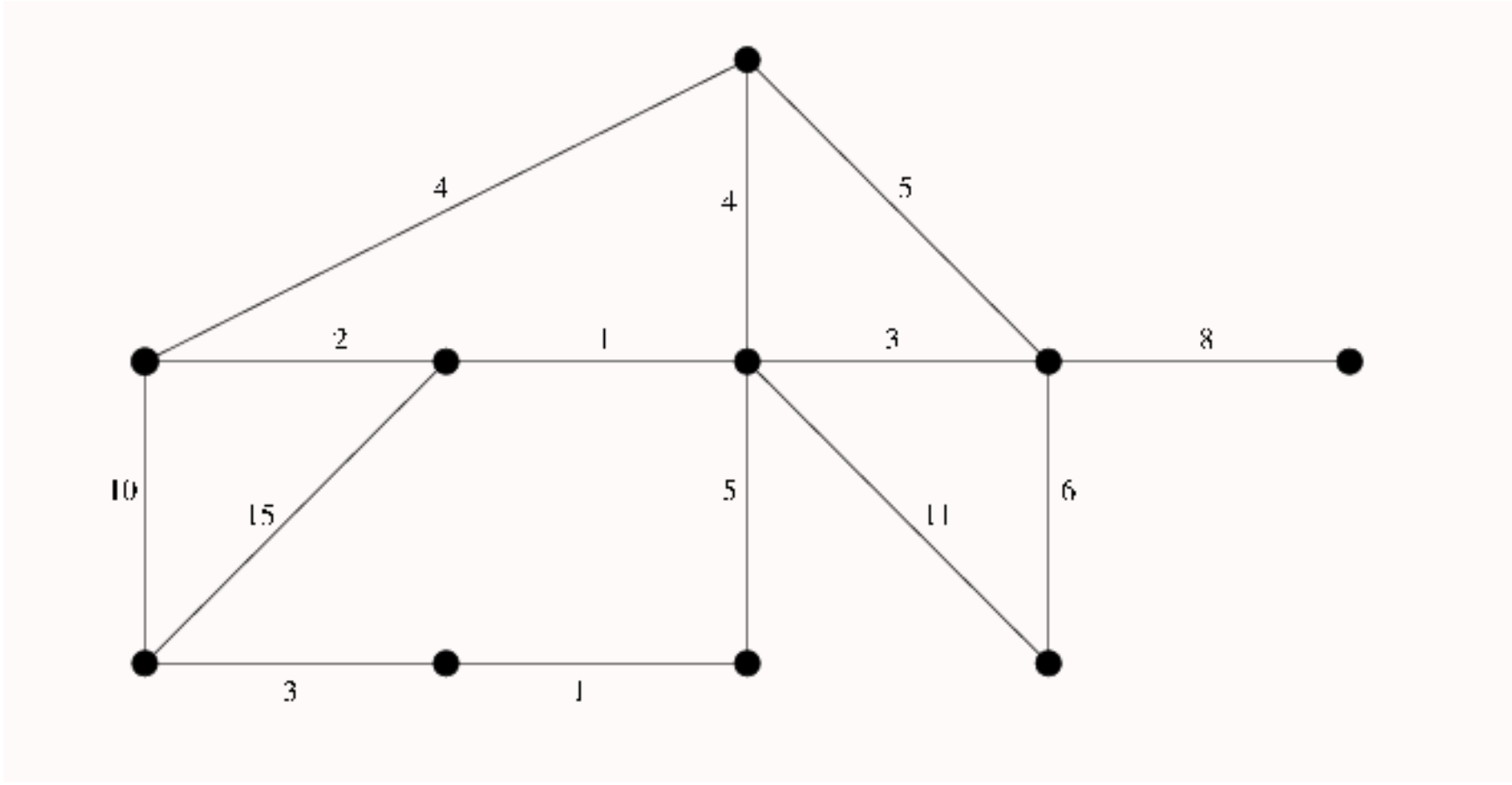
- L'algorithme fonctionne si G n'est pas connexe.
- On peut aussi obtenir l'arbre de poids MINIMUM: considérer $w'(e) = M - w(e)$ pour $M = \max\{w(e)|e \in A\}$.

Complexité: $O(e \log v)$.

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: $O(v \log e)$, comme Kruskal.

Exemple



Résultat