Le problème StackMST

Olivier Marty
Sous la direction de Jose Correa, Universidad de Chile

École Normale Supérieure de Cachan

Lundi 7 septembre 2015

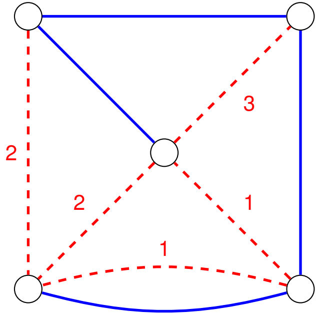
Universidad de Chile

Universidad de Chile

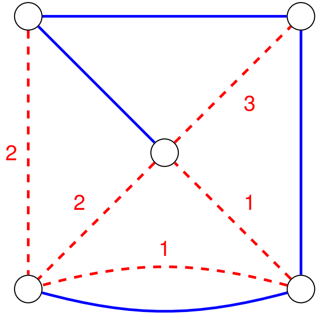
- 2 séminaires hebdomadaires
- 1 rencontre hebdomadaire
- 1 séminaire de mes travaux

Plan

Exemple d'instance



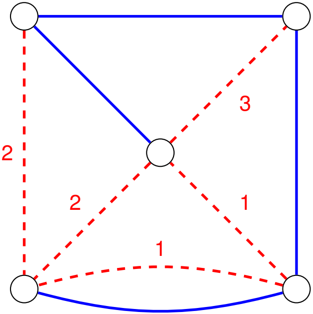
Exemple d'instance



Définition (StackMST)

1. Leader : choisit les prix bleus;
2. Follower : choisit un MST;

Exemple d'instance

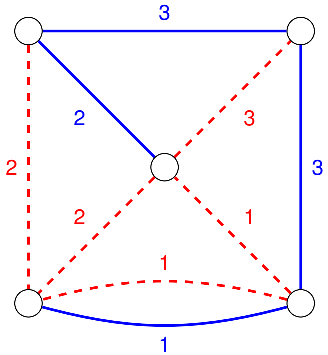


Définition (StackMST)

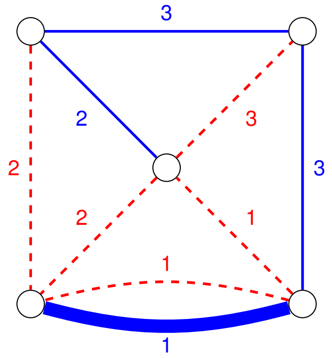
1. Leader : choisit les prix bleus;
2. Follower : choisit un MST;

Quels prix appliquer ?

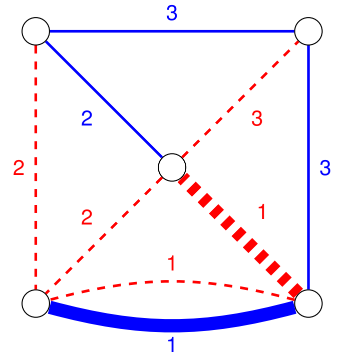
Exemple d'instance



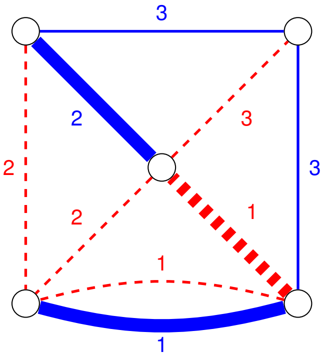
Exemple d'instance



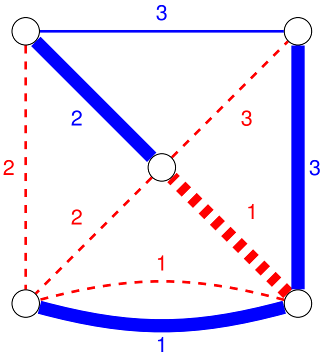
Exemple d'instance



Exemple d'instance



Exemple d'instance



Algorithmes gloutons

Définition (Algorithme de Kruskal)

1. Trier les arêtes;
2. Les selectionner dans l'ordre, sans créer de cycle.

Algorithmes gloutons

Définition (Algorithme de Kruskal)

1. Trier les arêtes;
2. Les selectionner dans l'ordre, sans créer de cycle.

$\Rightarrow$ Dans un cycle l'arête la plus chère n'est jamais selectionnée.

Algorithmes gloutons

Définition (Algorithme de Kruskal)

1. Trier les arêtes;
2. Les selectionner dans l'ordre, sans créer de cycle.

$\Rightarrow$ Dans un cycle l'arête la plus chère n'est jamais selectionnée.

Définition (Algorithme de Prim)

1. Partir d'un arbre-sommet;
2. Ajouter l'arête minimal de la coupe.

Algorithmes gloutons

Définition (Algorithme de Kruskal)

1. Trier les arêtes;
2. Les selectionner dans l'ordre, sans créer de cycle.

$\Rightarrow$ Dans un cycle l'arête la plus chère n'est jamais selectionnée.

Définition (Algorithme de Prim)

1. Partir d'un arbre-sommet;
2. Ajouter l'arête minimal de la coupe.

$\Rightarrow$ Tout MST doit contenir l'une des arêtes les moins chères d'une coupe.

Simplifications

- On peut utiliser les même valeurs
  $\rightarrow$ sinon on peut faire mieux;

Simplifications

- On peut utiliser les même valeurs
  $\rightarrow$ sinon on peut faire mieux;
- Les arêtes bleues sont preférées
  $\rightarrow$ sinon pareil avec des ε;

Simplifications

- On peut utiliser les même valeurs
  $\rightarrow$ sinon on peut faire mieux;
- Les arêtes bleues sont preférées
  $\rightarrow$ sinon pareil avec des ε;
- *R* doit contenir un arbre couvrant
  $\rightarrow$ sinon revenu infini;

Simplifications

- On peut utiliser les même valeurs
  $\rightarrow$ sinon on peut faire mieux;
- Les arêtes bleues sont preférées
  $\rightarrow$ sinon pareil avec des ε;
- *R* doit contenir un arbre couvrant
  $\rightarrow$ sinon revenu infini;
- *R* peut être un arbre
  $\rightarrow$ sinon arêtes rouges inutiles;

Simplifications

- On peut utiliser les même valeurs
  $\rightarrow$ sinon on peut faire mieux;
- Les arêtes bleues sont preférées
  $\rightarrow$ sinon pareil avec des ε;
- *R* doit contenir un arbre couvrant
  $\rightarrow$ sinon revenu infini;
- *R* peut être un arbre
  $\rightarrow$ sinon arêtes rouges inutiles;
- *R* peut être un chemin
  $\rightarrow$ par transformation en un problème équivalent;

Simplifications

- On peut utiliser les même valeurs
  $\rightarrow$ sinon on peut faire mieux;
- Les arêtes bleues sont preférées
  $\rightarrow$ sinon pareil avec des ε;
- R doit contenir un arbre couvrant
  $\rightarrow$ sinon revenu infini;
- R peut être un arbre
  $\rightarrow$ sinon arêtes rouges inutiles;
- R peut être un chemin
  $\rightarrow$ par transformation en un problème équivalent;
- B peut contenir un arbre couvrant
  $\rightarrow$ sinon arêtes rouges inutiles.

Simplifications

- On peut utiliser les même valeurs
  $\rightarrow$ sinon on peut faire mieux;
- Les arêtes bleues sont preférées
  $\rightarrow$ sinon pareil avec des ε;
- *R* doit contenir un arbre couvrant
  $\rightarrow$ sinon revenu infini;
- *R* peut être un arbre
  $\rightarrow$ sinon arêtes rouges inutiles;
- *R* peut être un chemin
  $\rightarrow$ par transformation en un problème équivalent;
- *B* peut contenir un arbre couvrant
  $\rightarrow$ sinon arêtes rouges inutiles.

NP-complet

Théorème

StackMST est NP-complet

- même avec les prix $\{1, 2\}$;

NP-complet

Théorème

StackMST est NP-complet

- même avec les prix $\{1, 2\}$;
- même avec un graphe planaire;

NP-complet

Théorème

StackMST est NP-complet

- même avec les prix $\{1, 2\}$;
- même avec un graphe planaire;
- même en fixant les arêtes rouges;

NP-complet

Théorème

StackMST est NP-complet

- même avec les prix $\{1, 2\}$;
- même avec un graphe planaire;
- même en fixant les arêtes rouges;
- même avec deux chemins et trois prix;

NP-complet

Théorème

StackMST est NP-complet

- même avec les prix $\{1, 2\}$;
- même avec un graphe planaire;
- même en fixant les arêtes rouges;
- même avec deux chemins et trois prix;

Best-out-of-k

1. Pour $1 \leq j \leq k$, affecter tous les prix à c_i et calculer le revenu;
2. Renvoyer la meilleure valeur.

Best-out-of-k

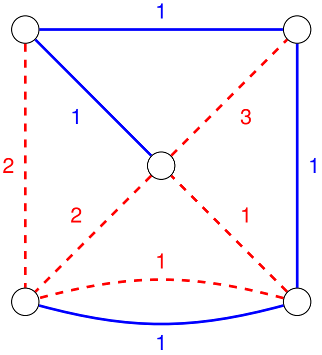
1. Pour $1 \leq j \leq k$, affecter tous les prix à c_i et calculer le revenu;
2. Renvoyer la meilleure valeur.

Théorème

Facteur d'approximation :

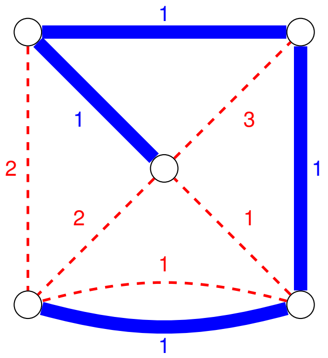
$$\min\left\{k, 1 + \ln\left(\frac{c_1}{c_k}\right), 1 + \ln(|B|)\right\}$$

Best-out-of-k : exemple



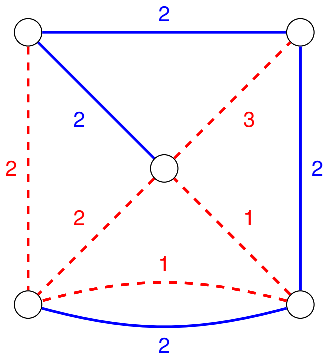
Avec 1 :

Best-out-of-k : exemple



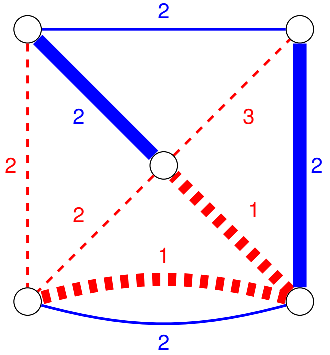
Avec 1 : revenu = 4

Best-out-of-k : exemple



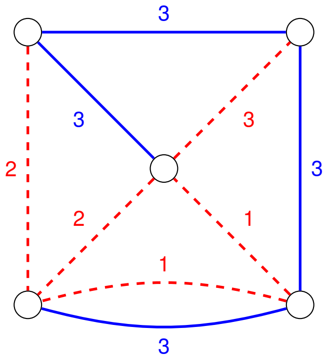
Avec 1 : revenu = 4
Avec 2 :

Best-out-of-k : exemple



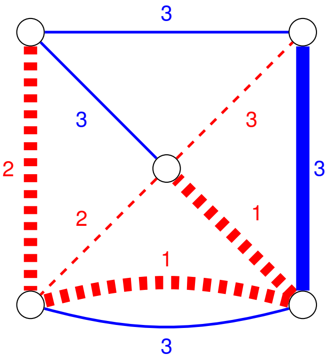
Avec 1 : revenu = 4
Avec 2 : revenu = 4

Best-out-of-k : exemple



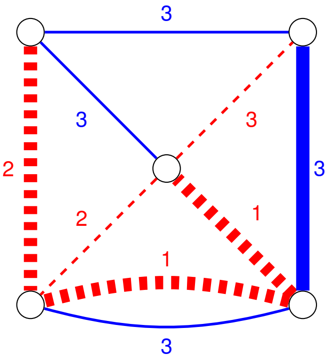
Avec 1 : revenu = 4
Avec 2 : revenu = 4
Avec 3 :

Best-out-of-k : exemple



Avec 1 : revenu = 4
Avec 2 : revenu = 4
Avec 3 : revenu = 3

Best-out-of-k : exemple



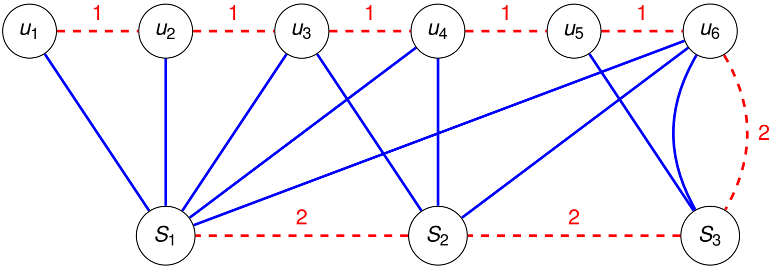
$$\left.\begin{array}{l}\text{Avec 1 : revenu} = 4\\ \text{Avec 2 : revenu} = 4\\ \text{Avec 3 : revenu} = 3\end{array}\right\}\ \text{retourne 4}$$

Plan

1. Problème StackMST
2. Réductions
3. Programme entier

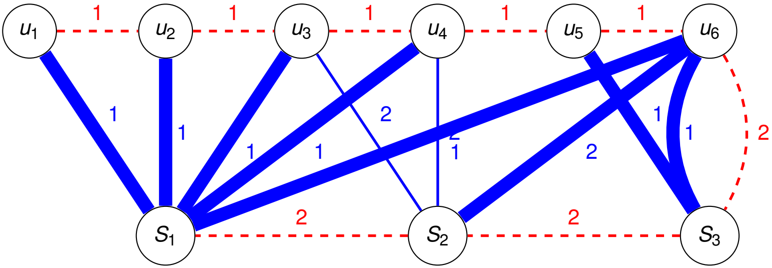
Réduction : ensemble rouge fixé

$$S_1 = \{u_1, u_2, u_3, u_4, u_6\},\ S_2 = \{u_3, u_4, u_6\},\ S_3 = \{u_5, u_6\}$$



Réduction : ensemble rouge fixé

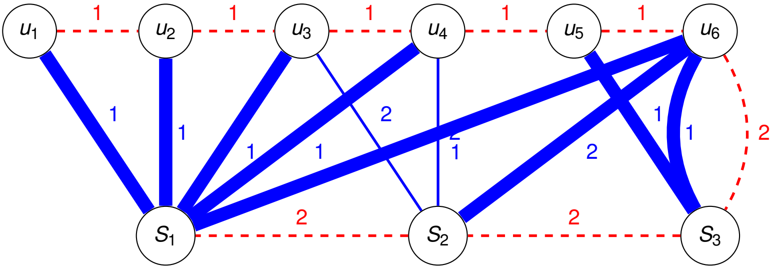
$$S_1 = \{u_1, u_2, u_3, u_4, u_6\},\ S_2 = \{u_3, u_4, u_6\},\ S_3 = \{u_5, u_6\}$$



Couverture $\{S_1, S_3\}$

Réduction : ensemble rouge fixé

$S_1 = \{u_1, u_2, u_3, u_4, u_6\}$, $S_2 = \{u_3, u_4, u_6\}$, $S_3 = \{u_5, u_6\}$



Couverture $\{S_1, S_3\}$

Équivalence

Revenu $n + 2m - t - 1$ si et seulement si couverture de taille t.

Réduction : ensemble rouge fixé

Théorème

StackMST est NP-difficile même si on veut un ensemble d'arêtes rouges.

Réduction : ensemble rouge fixé

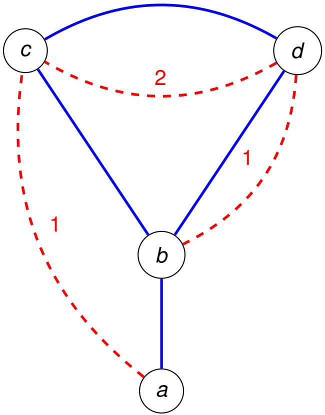
Théorème

StackMST est NP-difficile même si on veut un ensemble d'arêtes rouges.

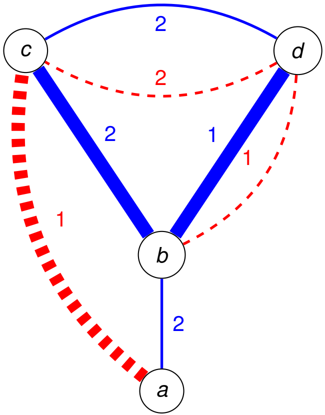
Proof.

Même réduction :
$\Rightarrow$ construction sans arêtes rouges;
$\Leftarrow$ déjà vrai sans l'hypothèse supplémentaire. $\square$

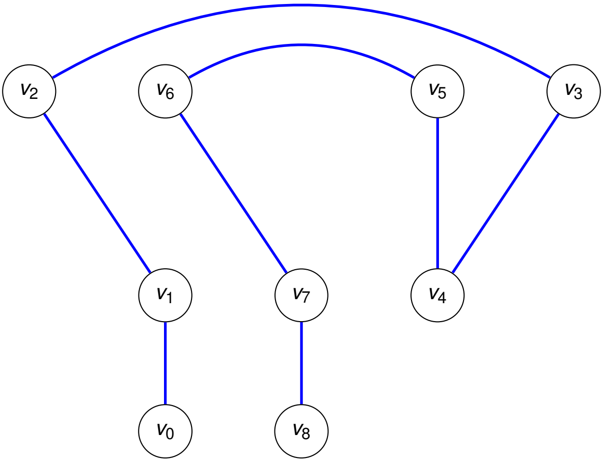
Réduction : deux chemins



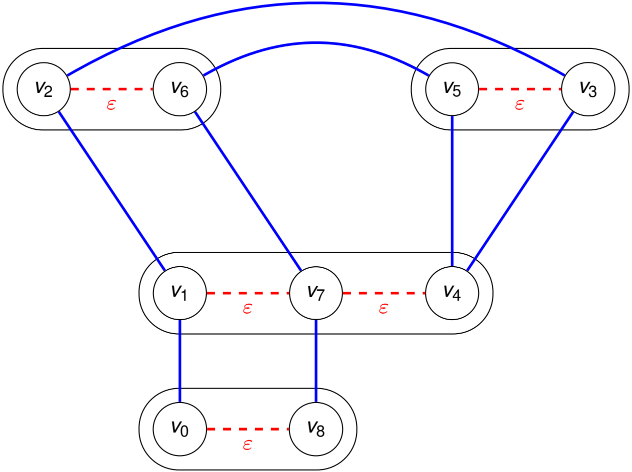
Réduction : deux chemins



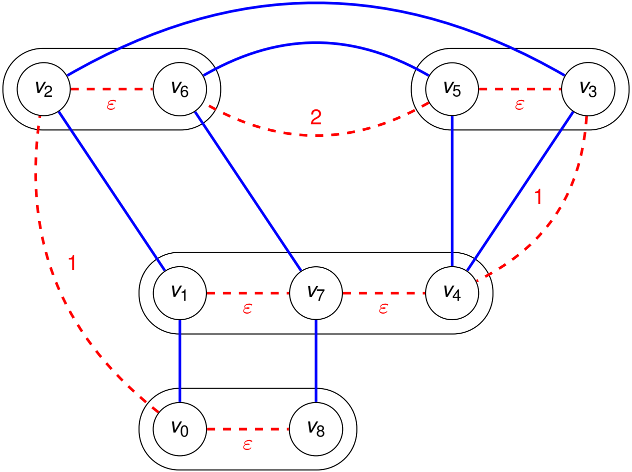
Réduction : deux chemins



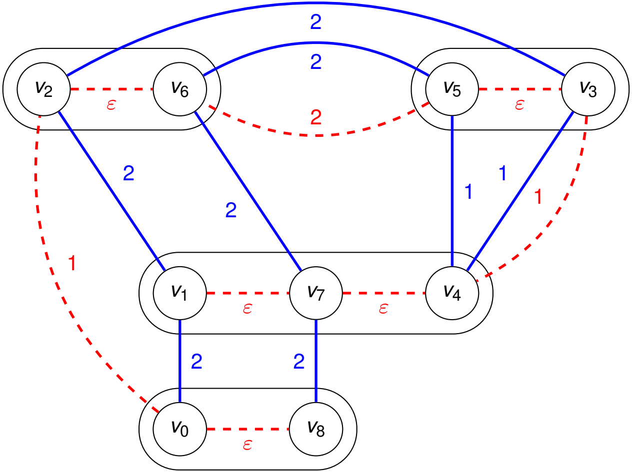
Réduction : deux chemins



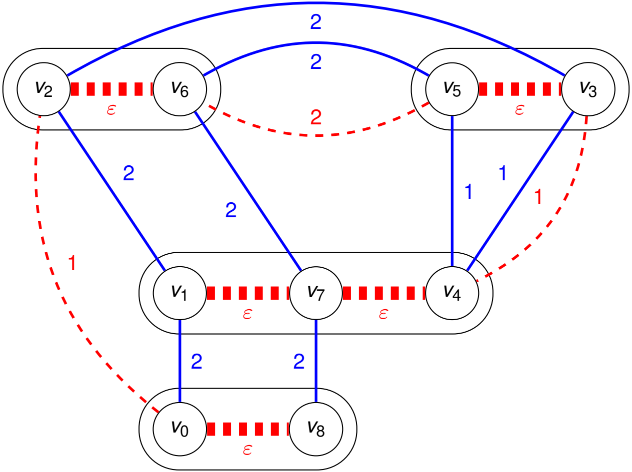
Réduction : deux chemins



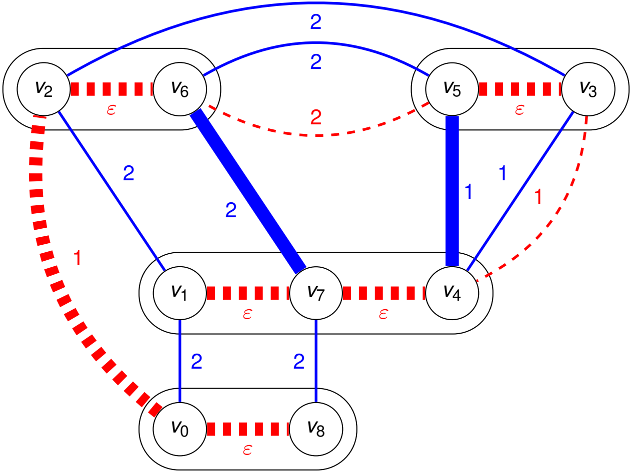
Réduction : deux chemins



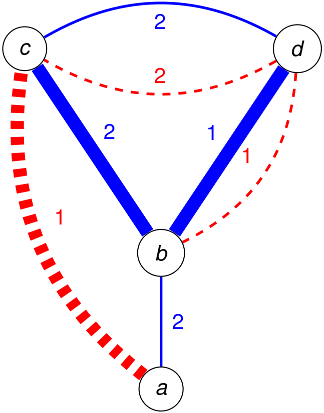
Réduction : deux chemins



Réduction : deux chemins



Réduction : deux chemins



Réduction : deux chemins, réciproque

Proof.

Réciproquement :

- supprimer les ε bleus (perte de revenu < 1);

Réduction : deux chemins, réciproque

Proof.

Réciproquement :

- supprimer les ε bleus (perte de revenu < 1);
- garder le prix minimum de deux arêtes bleus parallèles;

Réduction : deux chemins, réciproque

Proof.

Réciproquement :

- supprimer les ε bleus (perte de revenu < 1);
- garder le prix minimum de deux arêtes bleus parallèles;
- après que les arêtes rouges ε sont sélectionnées les deux problèmes sont équivalents.

$\square$

Plan

Programme entier

- x_e : 1 si l'arête e est dans le MST.
- $B_1, \ldots, B_k$: copies de B.

Programme entier

- x_e : 1 si l'arête e est dans le MST.
- $B_1, \ldots, B_k$: copies de B.

But :

$$\max \sum_{e \in B_1 \cup \cdots \cup B_k} x_e p_e$$

Programme entier

Tel que :

- Il y a $|V| - 1$ arêtes dans un arbre couvrant :

$$x(E) = |V| - 1$$

Programme entier

Tel que :

- Il y a $|V| - 1$ arêtes dans un arbre couvrant :

$$x(E) = |V| - 1$$

- Dans une coupe, l'une des arêtes les moins chère est selectionnée (Prim) :
  $\forall S \subsetneq V, S \neq \emptyset,$

$$x(\bar{\delta}(S)) \geq 1$$

Programme entier

Tel que :

- Il y a $|V|-1$ arêtes dans un arbre couvrant :
$$x(E) = |V| - 1$$
- Dans une coupe, l'une des arêtes les moins chère est selectionnée (Prim) :
$\forall S \subsetneq V, S \neq \emptyset,$
$$x(\bar{\delta}(S)) \geq 1$$
- Les embouts des arêtes de coût $< c_j$ sont déjà connectés (Kruskal) :
$\forall j \in \{1, \ldots, k\}, \forall S \subset V, S \neq \emptyset,$
$$|R_{\leq j-1}(S)| + x(E_{\geq j}(S)) \leq |S| - 1$$

Programme entier

$$\max \sum_{e \in B_1 \cup \cdots \cup B_k} x_e p_e$$

s.t.

$$x(E) = |V| - 1; \tag{1}$$

$$x(\bar{\delta}(S)) \geq 1 \qquad \forall S \subsetneq V, S \neq \emptyset; \tag{2}$$

$$|R_{\leq j-1}(S)| + x(E_{\geq j}(S)) \leq |S| - 1 \quad \forall j \in \{1, \ldots, k\}, \forall S \subset V, S \neq \emptyset; \tag{3}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E. \tag{4}$$

Programme entier : séparation

But : résoudre le PL de taille exponentielle en temps polynomial.

Programme entier : séparation

But : résoudre le PL de taille exponentielle en temps polynomial.

- $\forall S \subsetneq V, S \neq \emptyset, x(\bar{\delta}(S)) \geq 1$:
  pour l'arête $e = (a, b)$ rouge : trouver la coupe $a - b$ minimal dans le graphe $(V, E_{\leq p_e})$.

Programme entier : séparation

- $\forall j \in \{1, \ldots, k\}, \forall S \subset V, S \neq \emptyset, |R_{\leq j-1}(S)| + x(E_{\geq j}(S)) \leq |S| - 1$:
  équivalent à
  $\forall j \in \{1, \ldots, k\}, \forall S \subset V,$

$$|R_{\leq j-1}(S)| + x(E_{\geq j}(S)) \leq \max\{|S| - 1, 0\}$$

Programme entier : séparation

- $\forall j \in \{1, \dots, k\}, \forall S \subset V, S \neq \emptyset, |R_{\leq j-1}(S)| + x(E_{\geq j}(S)) \leq |S| - 1$:
  équivalent à
  $\forall j \in \{1, \dots, k\}, \forall S \subset V,$
  $$|R_{\leq j-1}(S)| + x(E_{\geq j}(S)) \leq \max\{|S| - 1, 0\}$$
  Pour j fixé, la fonction
  $$S \mapsto |R_{\leq j-1}(S)| + x(E_{\geq j}(S)) - \max\{|S| - 1, 0\}$$
  est supermodulaire.

Programme entier : encore à faire

- Vraiment meilleur ?
- Integrality gap ?
- Algorithme ?

Conclusion

- Des résultats de compléxité bienvenus;
- Encore beaucoup à faire pour un algorithme d'approximation à facteur constant.

References I

Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti.
Specializations and generalizations of the stackelberg minimum spanning tree game.
In Amin Saberi, editor, *Internet and Network Economics*, volume 6484 of *Lecture Notes in Computer Science*, pages 75–86. Springer Berlin Heidelberg, 2010.

Jean Cardinal, ErikD. Demaine, Samuel Fiorini, Gwenaël Joret, Stefan Langerman, Ilan Newman, and Oren Weimann.
The stackelberg minimum spanning tree game.
Algorithmica, 59(2):129–144, 2011.

References II

Jean Cardinal, ErikD. Demaine, Samuel Fiorini, Gwenaël Joret, Ilan Newman, and Oren Weimann.
The stackelberg minimum spanning tree game on planar and bounded-treewidth graphs.
In Stefano Leonardi, editor, *Internet and Network Economics*, volume 5929 of *Lecture Notes in Computer Science*, pages 125–136. Springer Berlin Heidelberg, 2009.

Backup I

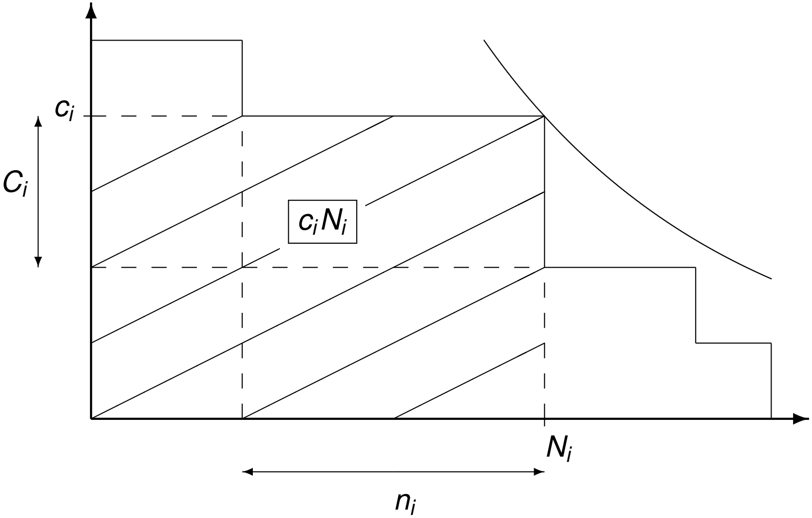


Figure: Exemple de distribution des coûts

Backup II

$(a_i)_{1 \leq i \leq k}$
On a $\frac{a_i - a_{i-1}}{a_i} \leq \frac{1}{a_{i-1}+1} + \cdots + \frac{1}{a_i}$ donc

$$\sum_{i=1}^{k} \frac{a_i - a_{i-1}}{a_i} \leq 1 + \sum_{j=a_1+1}^{a_k} \frac{1}{j}$$

Et $\frac{1}{n} \leq \int_{n-1}^{n} \frac{dt}{t}$ d'où :

$$1 + \sum_{j=a_1+1}^{a_k} \frac{1}{j} = 1 + \int_{a_1}^{a_k} \frac{dt}{t} \leq 1 + \ln\left(\frac{a_k}{a_1}\right)$$