Relaxation lagrangienne en programmation par contraintes appliquée au problème du commis voyageur

Raphaël Boudreault et Claude-Guy Quimper

12 mars 2021

Université Laval

Le problème du commis voyageur

Définition

- *Problème du commis voyageur*
- *Problème du voyageur de commerce*
- *Traveling salesman problem* (TSP)

Définition

- *Problème du commis voyageur*
- *Problème du voyageur de commerce*
- *Traveling salesman problem* (TSP)

Définition

Étant donné une liste de villes et les distances entre chacune d'elles, déterminer le chemin le plus court visitant chaque ville une seule fois et retournant à son point de départ.

Définition

- *Problème du commis voyageur*
- *Problème du voyageur de commerce*
- *Traveling salesman problem* (TSP)

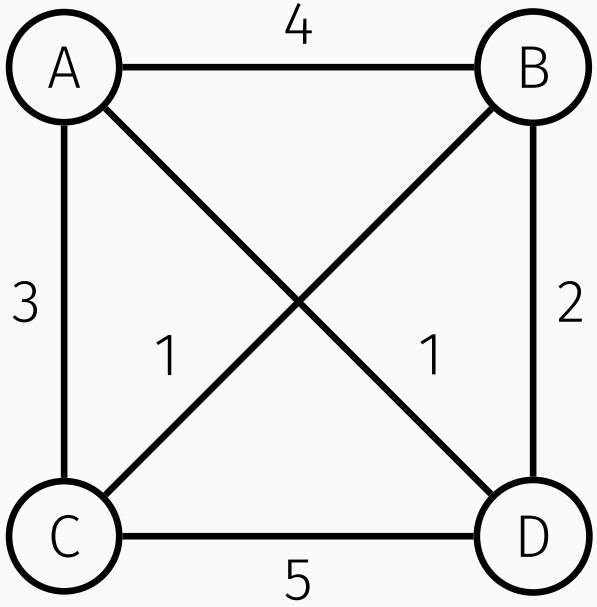
Définition

Étant donné une liste de villes et les distances entre chacune d'elles, déterminer le chemin le plus court visitant chaque ville une seule fois et retournant à son point de départ.

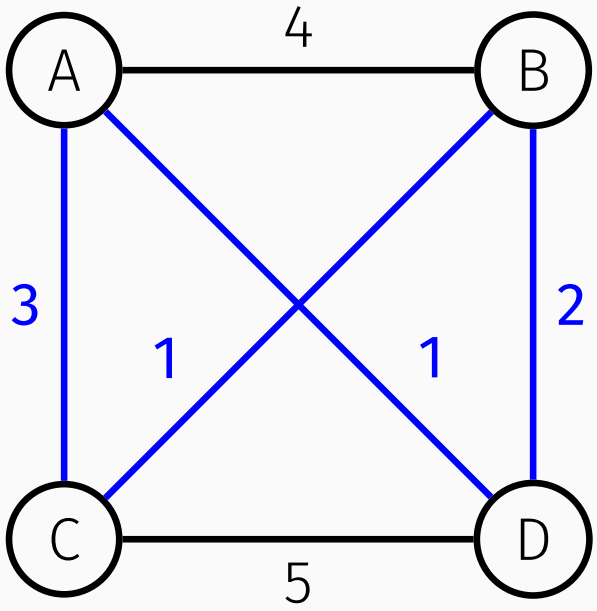
Définition (graphe)

Étant donné un graphe pondéré G, trouver un cycle hamiltonien dans G de poids minimal.

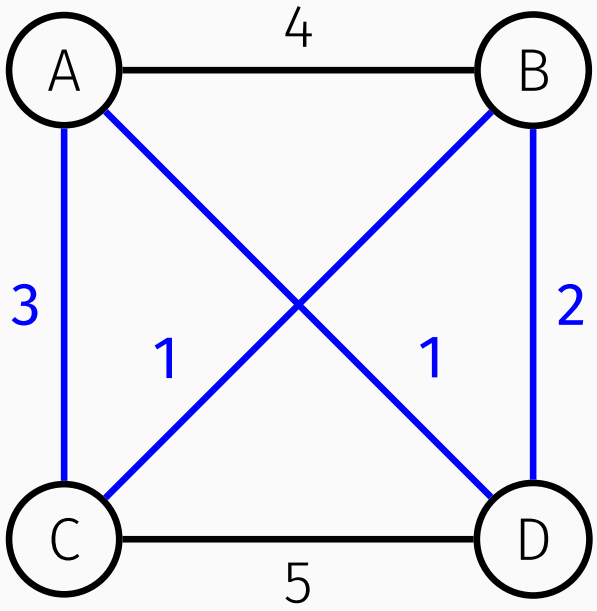
Exemple



Exemple



Exemple



Note : Problème symétrique (graphe non dirigé)

Explosion combinatoire

Pour n villes, il existe $\frac{1}{2}(n-1)!$ chemins candidats...

Explosion combinatoire

Pour n villes, il existe $\frac{1}{2}(n-1)!$ chemins candidats…

n	# chemins
4	3
6	60
10	$181\,440$
15	$43\,589\,145\,600$
20	$6{,}1 \times 10^{16}$
49	$6{,}2 \times 10^{60}$
61	$4{,}2 \times 10^{81}$

Explosion combinatoire

Pour n villes, il existe $\frac{1}{2}(n-1)!$ chemins candidats…

n	# chemins
4	3
6	60
10	$181\,440$
15	$43\,589\,145\,600$
20	$6{,}1 \times 10^{16}$
49	$6{,}2 \times 10^{60}$
61	$4{,}2 \times 10^{81}$

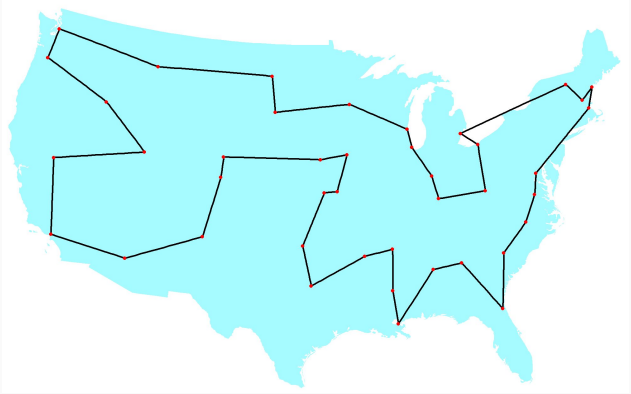
En général, c’est un problème NP-difficile

Bref historique

- Origines incertaines... 1800?
- 1930 : Première formulation mathématique

Bref historique

- Origines incertaines... 1800?
- 1930 : Première formulation mathématique
- 1954 : Dantzig, Fulkerson et Johnson résolvent un TSP à 49 villes (à la main!)

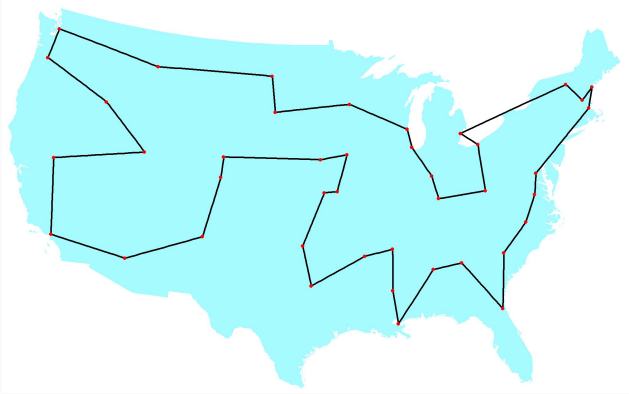


1

1. http://www.math.uwaterloo.ca/tsp/history/pictorial/dfj.html

Bref historique

- Origines incertaines... 1800?
- 1930 : Première formulation mathématique
- 1954 : Dantzig, Fulkerson et Johnson résolvent un TSP à 49 villes (à la main!)



1

- Années 90 : Applegate, Bixby, Chvátal et Cook développent le solveur exact *Concorde*
- 2000 à aujourd'hui : *Concorde* résout des problèmes difficiles allant jusqu'à 109 399 villes

1. `http://www.math.uwaterloo.ca/tsp/history/pictorial/dfj.html`

Formulation linéaire

V : ensemble des sommets (*villes*), $V \coloneqq \{1, \ldots, n\}$
E : ensemble des arêtes (*routes*)
$w(e)$: poids de l'arête $e \in E$ (*longueur de la route* e)

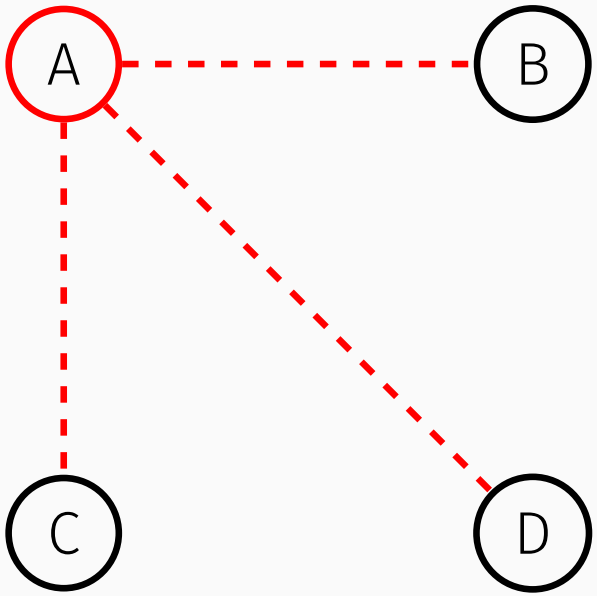
$$\min \quad Z = \sum_{e \in E} w(e) x_e$$

$$\text{s. à} \quad \sum_{e \in \delta(i)} x_e = 2 \qquad \forall i \in V \qquad (1)$$

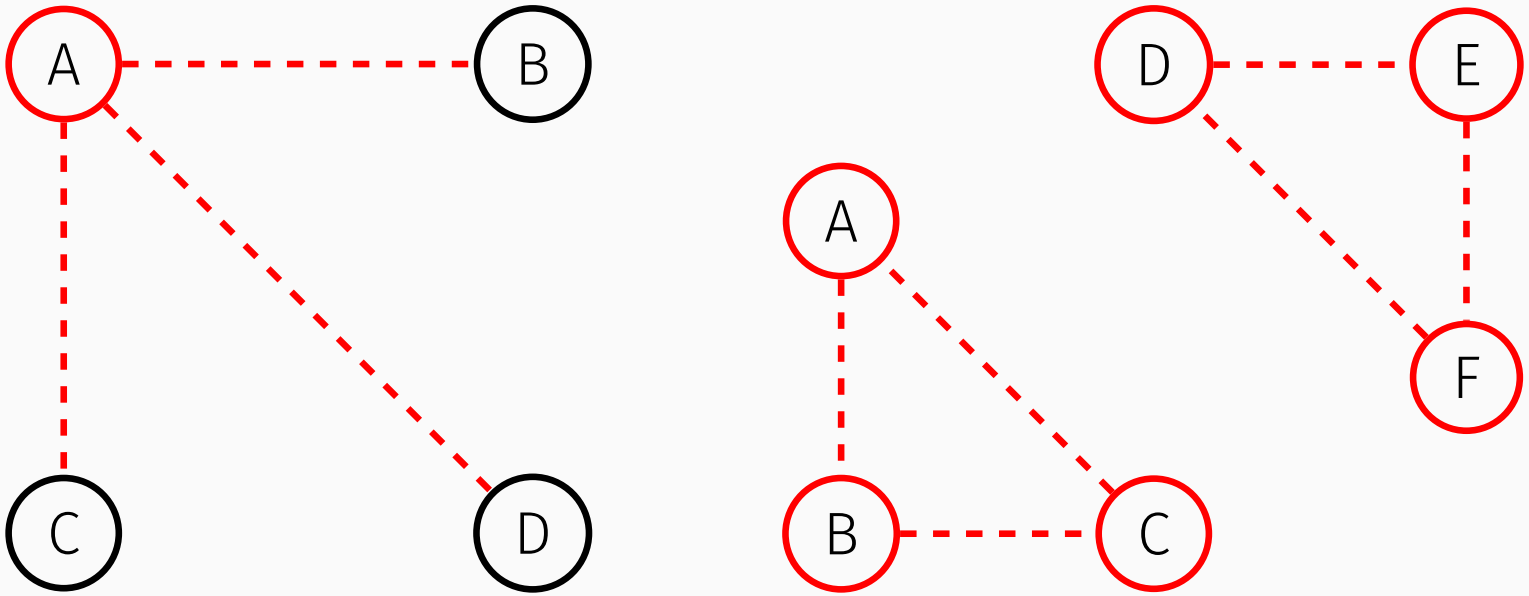
$$\sum_{i,j \in S, i<j} x_{\{i,j\}} \leq |S| - 1 \qquad \forall S \subset V, |S| \geq 3 \qquad (2)$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \qquad (3)$$

Formulation linéaire



Formulation linéaire



Approche générale de résolution

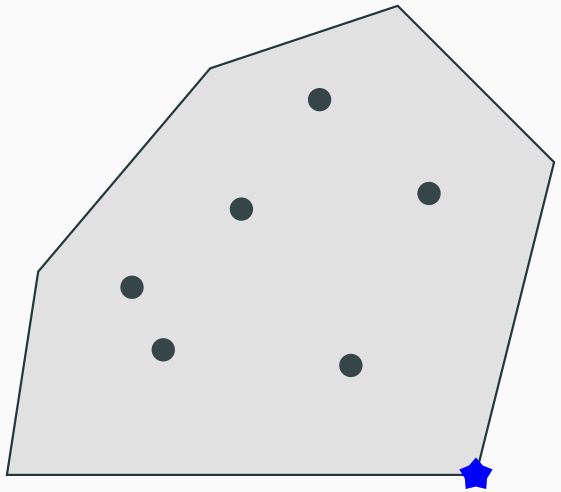
- *Problème d'optimisation linéaire en nombres entiers*

Approche générale de résolution

- *Problème d'optimisation linéaire en nombres entiers*
- Approche : deux séquences monotones de bornes
  - Bornes supérieures : solutions admissibles
  - Bornes inférieures : relaxations

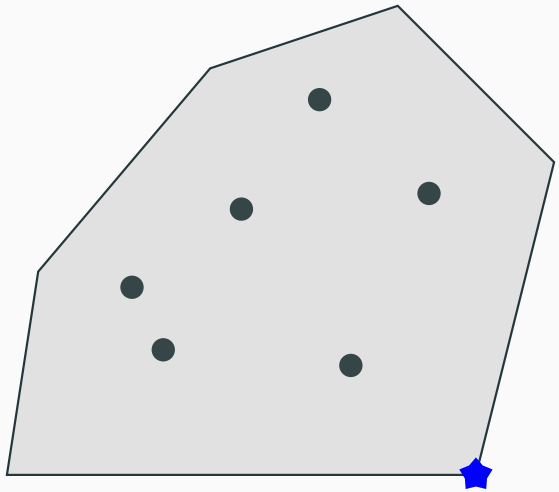
Approche générale de résolution

- *Problème d'optimisation linéaire en nombres entiers*
- Approche : deux séquences monotones de bornes
  - Bornes supérieures : solutions admissibles
  - Bornes inférieures : relaxations



Approche générale de résolution

- *Problème d'optimisation linéaire en nombres entiers*
- Approche : deux séquences monotones de bornes
  - Bornes supérieures : solutions admissibles
  - Bornes inférieures : relaxations



- Plans sécants, *branch-and-bound*…

Relaxation *1-tree* du TSP

- Introduite par Held et Karp en 1970
- Obtenue en relaxant les contraintes de degré

Relaxation *1-tree* du TSP

- Introduite par Held et Karp en 1970
- Obtenue en relaxant les contraintes de degré

$$\min \quad Z = \sum_{e \in E} w(e) x_e$$

$$\text{s. à} \quad \sum_{e \in \delta(1)} x_e = 2 \qquad \cancel{\forall i \in V} \qquad (1)$$

$$\sum_{e \in E} x_e = n \qquad (N)$$

$$\sum_{i,j \in S, i<j} x_{\{i,j\}} \leq |S| - 1 \qquad \forall S \subset V \backslash \{1\}, |S| \geq 3 \qquad (2)$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \qquad (3)$$

Relaxation *1-tree* du TSP

- Introduite par Held et Karp en 1970
- Obtenue en relaxant les contraintes de degré

$$\min \quad Z = \sum_{e \in E} w(e) x_e$$

$$\text{s. à} \quad \sum_{e \in \delta(1)} x_e = 2 \qquad \cancel{\forall i \in V} \qquad (1)$$

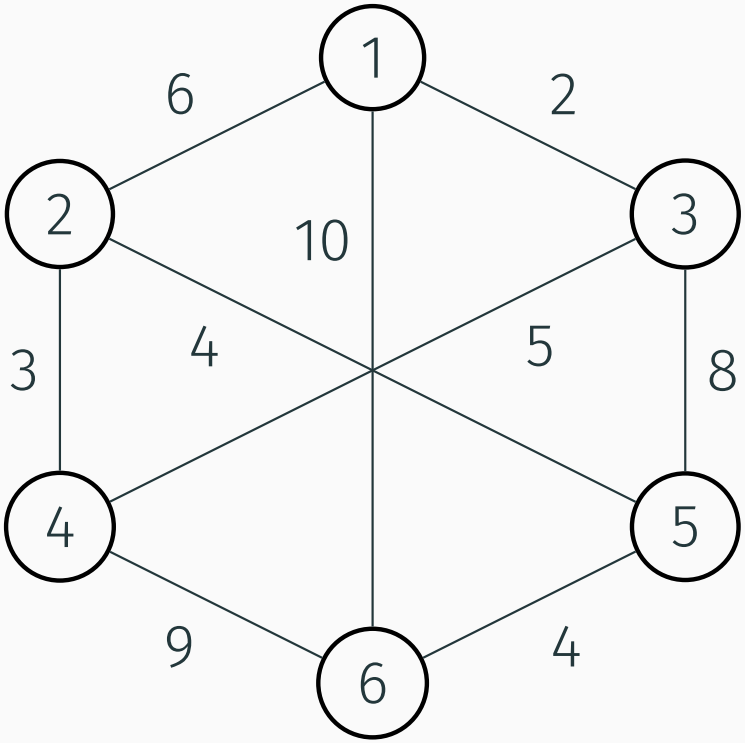
$$\sum_{e \in E} x_e = n \qquad (\mathsf{N})$$

$$\sum_{i,j \in S, i<j} x_{\{i,j\}} \leq |S| - 1 \qquad \forall S \subset V \backslash \{1\}, |S| \geq 3 \qquad (2)$$

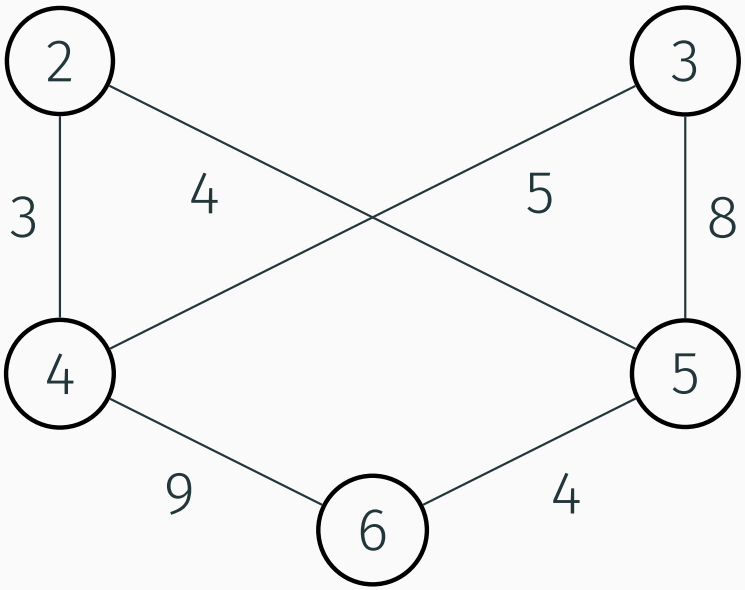
$$x_e \in \{0, 1\} \qquad \forall e \in E \qquad (3)$$

Contraintes (1), (N), (2) : structure d'un *1-tree*

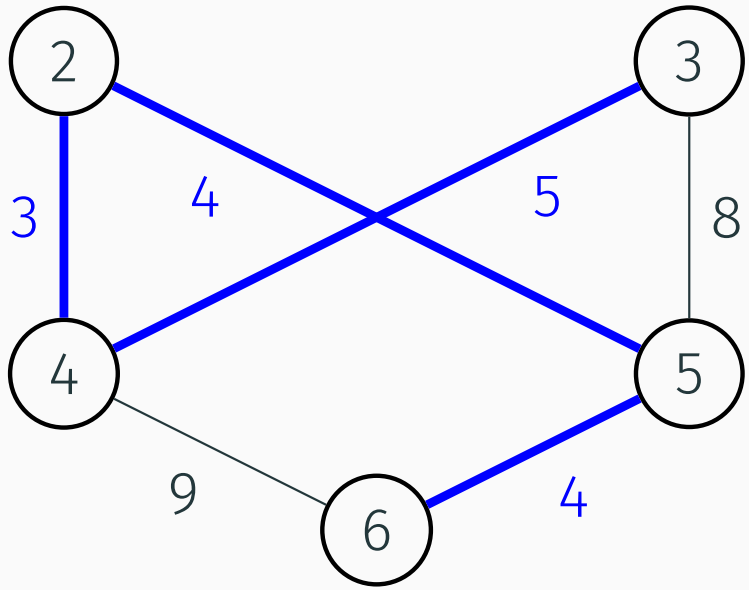
Exemple



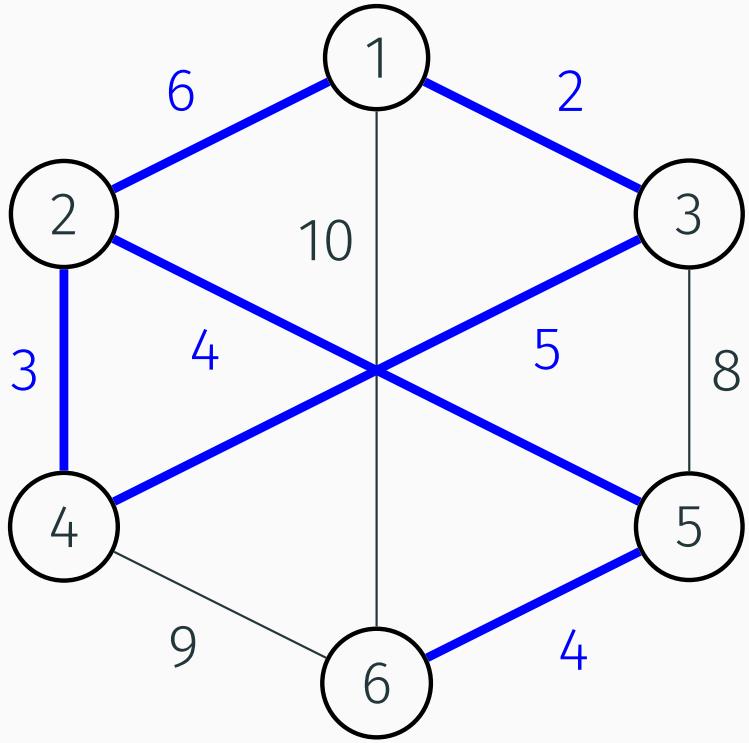
Exemple



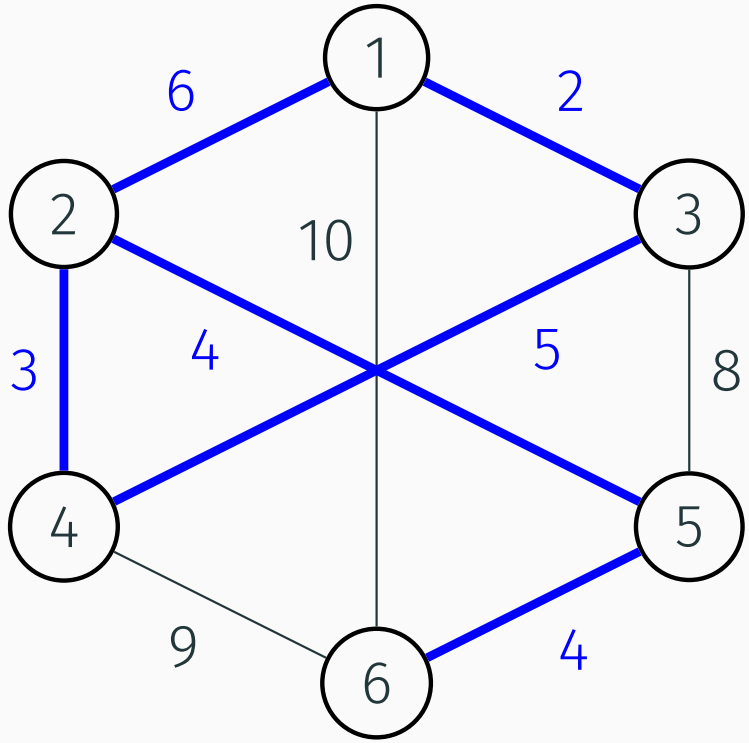
Exemple



Exemple



Exemple



Borne : **24**

Optimale : **28**

Amélioration : *relaxation lagrangienne*

Intégration des contraintes de degré dans la fonction objectif en ajoutant des multiplicateurs de Lagrange

$\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_n) \in \mathbb{R}^n, \lambda_1 = 0$

Amélioration : *relaxation lagrangienne*

Intégration des contraintes de degré dans la fonction objectif en ajoutant des multiplicateurs de Lagrange

$\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_n) \in \mathbb{R}^n, \lambda_1 = 0$

$$\min \quad Z_{RL}(\boldsymbol{\lambda}) = \sum_{e \in E} w(e) x_e + \sum_{i \in V} \lambda_i (\deg_T(i) - 2)$$

$$\text{s. à} \quad T = \{e : x_e = 1\} \text{ est un 1-tree}$$

Amélioration : *relaxation lagrangienne*

Intégration des contraintes de degré dans la fonction objectif en ajoutant des multiplicateurs de Lagrange

$\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_n) \in \mathbb{R}^n, \lambda_1 = 0$

$$\begin{aligned} \min \quad & Z_{RL}(\boldsymbol{\lambda}) = \sum_{e \in E} w(e) x_e + \sum_{i \in V} \lambda_i (\deg_T(i) - 2) \\ \text{s. à} \quad & T = \{e : x_e = 1\} \text{ est un 1-tree} \end{aligned}$$

Pour tout $\boldsymbol{\lambda}$, $Z_{RL}(\boldsymbol{\lambda})$ est une borne inférieure valide!

Amélioration : *relaxation lagrangienne*

Étant donné $\boldsymbol{\lambda}$...

Amélioration : *relaxation lagrangienne*

Étant donné $\boldsymbol{\lambda}$...

$$Z_{RL}(\boldsymbol{\lambda}) = \sum_{\{i,j\} \in E} (w(i,j) + \lambda_i + \lambda_j) x_{\{i,j\}} - 2 \sum_{i \in V} \lambda_i$$

Amélioration : *relaxation lagrangienne*

Étant donné $\boldsymbol{\lambda}$...

$$Z_{RL}(\boldsymbol{\lambda}) = \sum_{\{i,j\} \in E} (\underbrace{w(i,j) + \lambda_i + \lambda_j}_{\tilde{w}(i,j)}) x_{\{i,j\}} - 2 \sum_{i \in V} \lambda_i$$

Amélioration : *relaxation lagrangienne*

Étant donné $\boldsymbol{\lambda}$...

$$Z_{RL}(\boldsymbol{\lambda}) = \sum_{\{i,j\} \in E} (\underbrace{w(i,j) + \lambda_i + \lambda_j}_{\tilde{w}(i,j)}) x_{\{i,j\}} - 2 \sum_{i \in V} \lambda_i$$

Amélioration de la borne (problème des multiplicateurs) :

$$\max_{\boldsymbol{\lambda}} Z_{RL}(\boldsymbol{\lambda})$$

→Méthodes itératives (*sous-gradients*)

Et *Concorde* dans tout ça?

- Utilise les techniques mentionnées précédemment (*1-tree*, plans sécants, *branch-and-bound*...)
- Résout rapidement des instances de grande taille (1000+)
- Est l'état de l'art

Mais...

Et *Concorde* dans tout ça?

- Utilise les techniques mentionnées précédemment (*1-tree*, plans sécants, *branch-and-bound*...)
- Résout rapidement des instances de grande taille (1000+)
- Est l'état de l'art

Mais...

Il ne peut résoudre que le TSP *pur*, sans contraintes additionnelles (fenêtres de temps, capacités, trafic, etc.)

La programmation par contraintes

Définition

- Paradigme de programmation qui permet de résoudre des problèmes combinatoires
- Sépare la partie *modélisation* de la partie *résolution*

Définition

- Paradigme de programmation qui permet de résoudre des problèmes combinatoires
- Sépare la partie *modélisation* de la partie *résolution*

Modélisation

- Variables de décision
  - Type : Entier, booléen
  - Domaine : Énuméré ou intervalle

Définition

- Paradigme de programmation qui permet de résoudre des problèmes combinatoires
- Sépare la partie *modélisation* de la partie *résolution*

Modélisation

- Variables de décision
  - Type : Entier, booléen
  - Domaine : Énuméré ou intervalle
- Contraintes
  - Arithmétiques ($x \leq 8$, $x \neq y$, $z > 0$)
  - Logiques ($x \vee y$, $y \Rightarrow z$, $\neg x \wedge w$)
  - Globales ($\textsc{AllDifferent}(x_1, \ldots, x_n)$)

Définition

- Paradigme de programmation qui permet de résoudre des problèmes combinatoires
- Sépare la partie *modélisation* de la partie *résolution*

Modélisation

- Variables de décision
  - Type : Entier, booléen
  - Domaine : Énuméré ou intervalle
- Contraintes
  - Arithmétiques ($x \leq 8$, $x \neq y$, $z > 0$)
  - Logiques ($x \vee y$, $y \Rightarrow z$, $\neg x \wedge w$)
  - Globales (AllDifferent$(x_1, \ldots, x_n)$)
- Fonction objectif (si *problème d'optimisation*)

Arbre de recherche

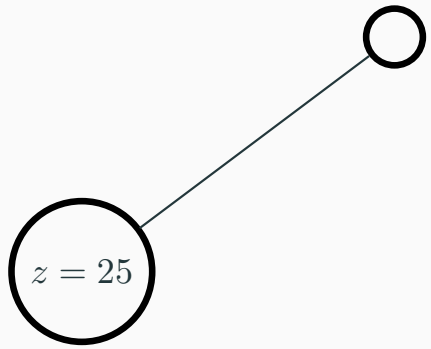
Variables : $x \in \{4, 5\}$, $y \in \{5, 6\}$ et $z \in \{4, 25\}$

Contraintes : $x > z$ et $x \neq y$

Arbre de recherche

Variables : $x \in \{4, 5\}$, $y \in \{5, 6\}$ et $z \in \{4, 25\}$

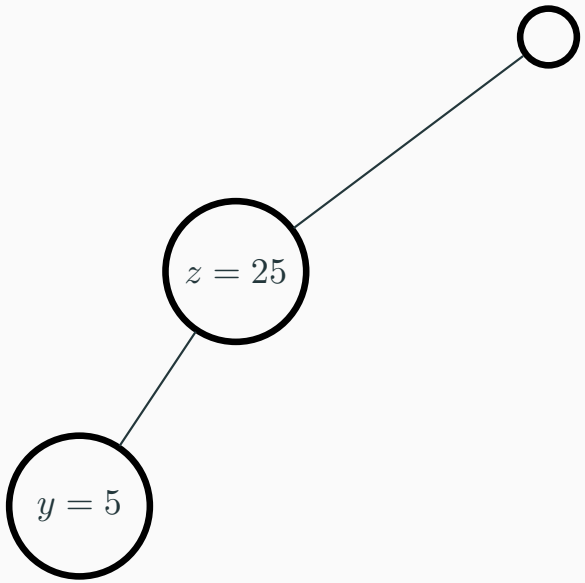
Contraintes : $x > z$ et $x \neq y$



Arbre de recherche

Variables : $x \in \{4, 5\}$, $y \in \{5, 6\}$ et $z \in \{4, 25\}$

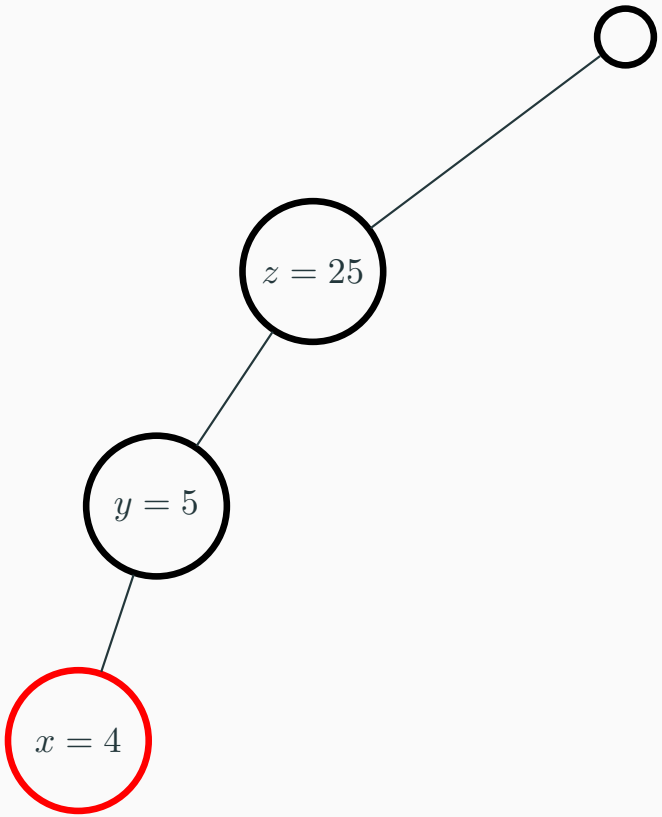
Contraintes : $x > z$ et $x \neq y$



Arbre de recherche

Variables : $x \in \{4, 5\}$, $y \in \{5, 6\}$ et $z \in \{4, 25\}$

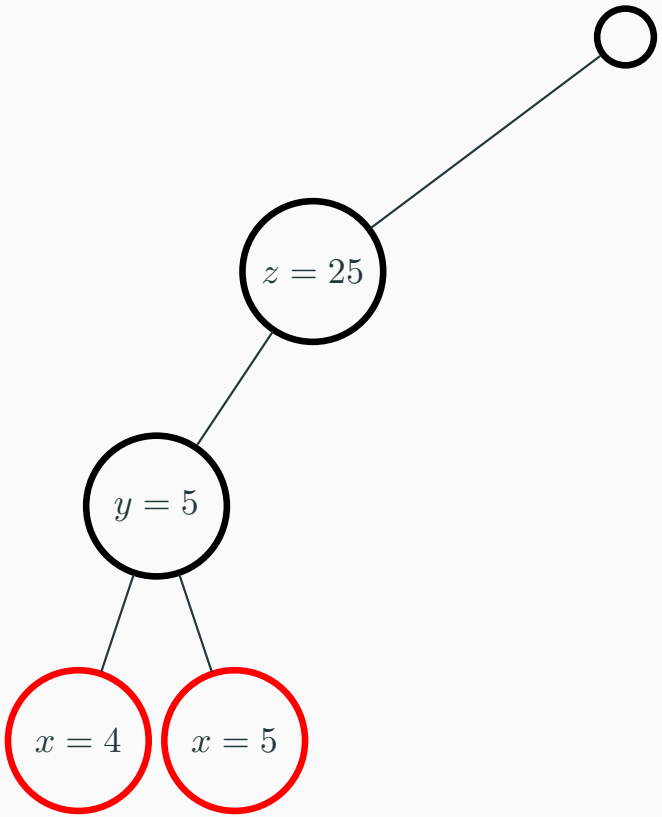
Contraintes : $x > z$ et $x \neq y$



Arbre de recherche

Variables : $x \in \{4, 5\}$, $y \in \{5, 6\}$ et $z \in \{4, 25\}$

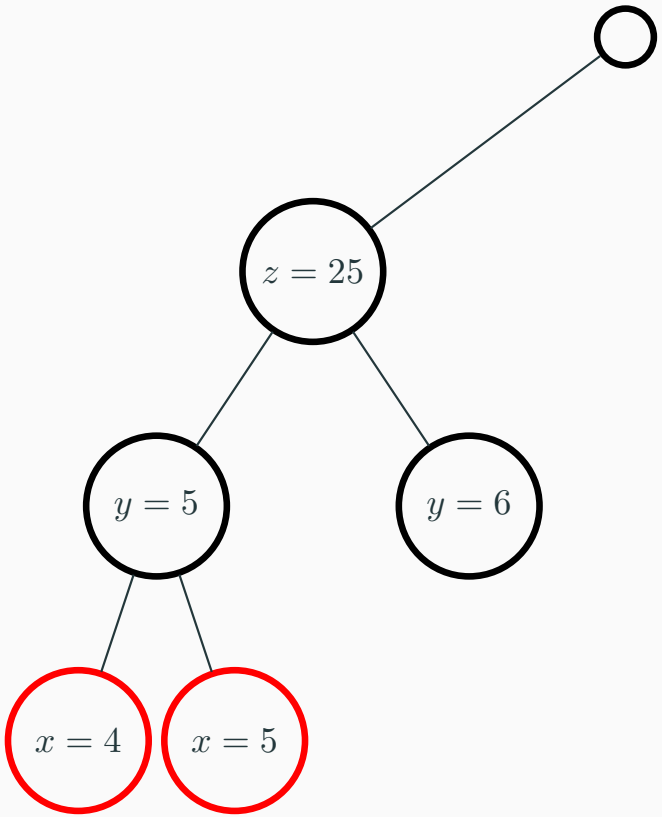
Contraintes : $x > z$ et $x \neq y$



Arbre de recherche

Variables : $x \in \{4, 5\}$, $y \in \{5, 6\}$ et $z \in \{4, 25\}$

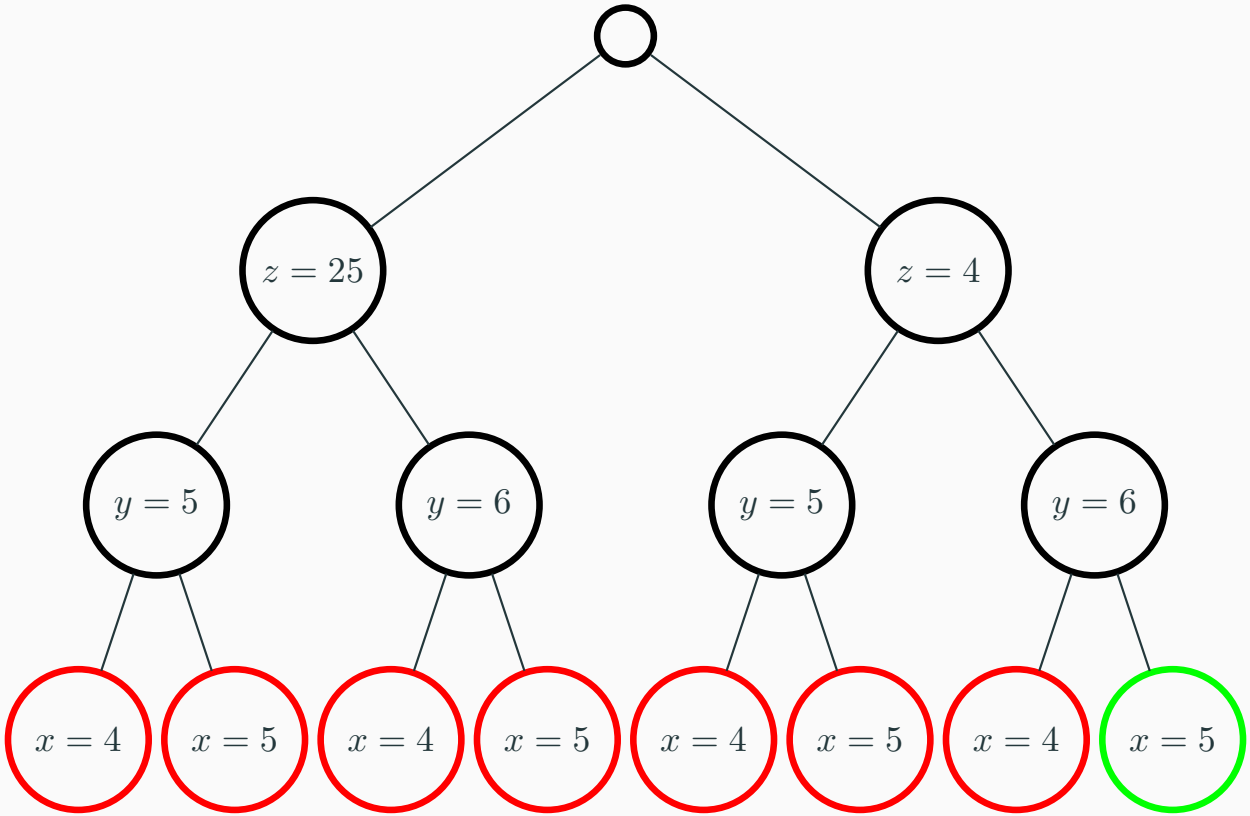
Contraintes : $x > z$ et $x \neq y$



Arbre de recherche

Variables : $x \in \{4, 5\}$, $y \in \{5, 6\}$ et $z \in \{4, 25\}$

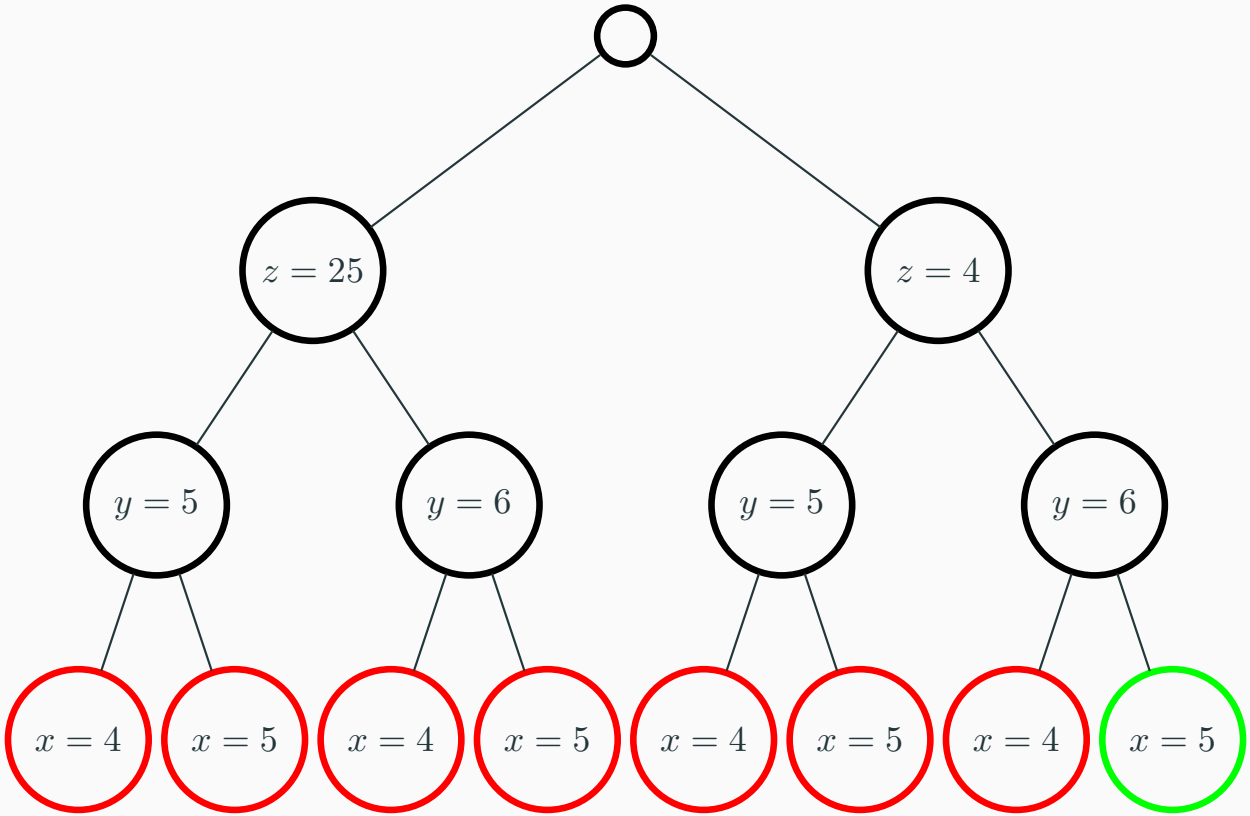
Contraintes : $x > z$ et $x \neq y$



Arbre de recherche

Variables : $x \in \{4, 5\}$, $y \in \{5, 6\}$ et $z \in \{4, 25\}$

Contraintes : $x > z$ et $x \neq y$



Pour le choix des variables et des valeurs : heuristique

Filtrage

- Couper des branches de l'arbre
- Chaque contrainte (arithmétique, logique ou **globale**) nécessite son propre algorithme de filtrage
- Prend en entrée les domaines des variables concernées et *fait le ménage*
- Généralement appelés à chaque nœud de l'arbre

Retour à l'exemple

Variables : $x \in \{4, 5\}$, $y \in \{5, 6\}$ et $z \in \{4, 25\}$

Contraintes : $\mathbf{x} > \mathbf{z}$ et $x \neq y$

Retour à l'exemple

Variables : $x \in \{\cancel{4}, 5\}$, $y \in \{5, 6\}$ et $z \in \{4, \cancel{25}\}$

Contraintes : $x > z$ et $\mathbf{x \neq y}$

Retour à l'exemple

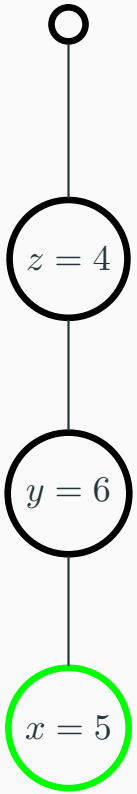
Variables : $x \in \{\cancel{4}, 5\}$, $y \in \{\cancel{5}, 6\}$ et $z \in \{4, \cancel{25}\}$

Contraintes : $x > z$ et $x \neq y$

Retour à l'exemple

Variables : $x \in \{\cancel{4}, 5\}$, $y \in \{\cancel{5}, 6\}$ et $z \in \{4, \cancel{25}\}$

Contraintes : $x > z$ et $x \neq y$



La contrainte WeightedCircuit (Benchimol *et al.*, 2012)

Étant donné $G = (V, E)$ et w...

- Variables binaires $\boldsymbol{x} = \left(x_{e_1}, \ldots, x_{e_{|E|}}\right)$
- Variable entière $z \in [0, U]$

La contrainte WeightedCircuit (Benchimol *et al.*, 2012)

Étant donné $G = (V, E)$ et w...

- Variables binaires $\boldsymbol{x} = \left(x_{e_1}, \ldots, x_{e_{|E|}}\right)$
- Variable entière $z \in [0, U]$

WeightedCircuit$(\boldsymbol{x}, z)$ est satisfaite ssi

- $T = \{e : x_e = 1\}$ est un cycle hamiltonien de G
- $\sum_{e \in E} w(e) x_e \leq z$

La contrainte WeightedCircuit (Benchimol *et al.*, 2012)

Étant donné $G = (V, E)$ et w...

- Variables binaires $\boldsymbol{x} = \left(x_{e_1}, \ldots, x_{e_{|E|}}\right)$
- Variable entière $z \in [0, U]$

WeightedCircuit$(\boldsymbol{x}, z)$ est satisfaite ssi

- $T = \{e : x_e = 1\}$ est un cycle hamiltonien de G
- $\sum_{e \in E} w(e) x_e \leq z$

Reformulation du TSP

$$\begin{aligned} \min \quad & z \\ \text{s. à} \quad & \text{WeightedCircuit}(\boldsymbol{x}, z) \end{aligned}$$

Le filtrage de WeightedCircuit

Propriétés

- Filtrage basé sur les coûts (Focacci *et al.*, 1999)
  - Relaxation L et borne supérieure U
  - Si $L > U$, *échec*

Propriétés

- Filtrage basé sur les coûts (Focacci *et al.*, 1999)
  - Relaxation L et borne supérieure U
  - Si $L > U$, *échec*
  - Sinon, si $L[x = \mu] > U$, μ est retirée du domaine de x

Propriétés

- Filtrage basé sur les coûts (Focacci *et al.*, 1999)
  - Relaxation L et borne supérieure U
  - Si $L > U$, *échec*
  - Sinon, si $L[x = \mu] > U$, μ est retirée du domaine de x
- Utilise les coûts de la relaxation *1-tree*
  - Identification d'arêtes **interdites**
  - Identification d'arêtes **obligatoires**

Propriétés

- Filtrage basé sur les coûts (Focacci *et al.*, 1999)
  - Relaxation L et borne supérieure U
  - Si $L > U$, *échec*
  - Sinon, si $L[x = \mu] > U$, μ est retirée du domaine de x
- Utilise les coûts de la relaxation *1-tree*
  - Identification d'arêtes **interdites**
  - Identification d'arêtes **obligatoires**
- Intégré au processus itératif de la RL
  - *CP-based Lagrangian relaxation*, CP-LR (Sellmann, 2004)

Arêtes *interdites*

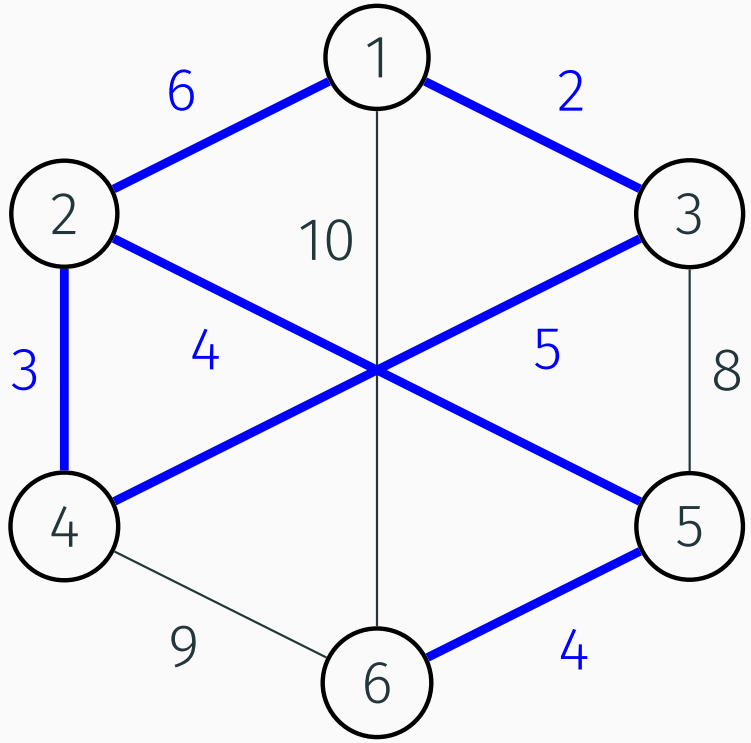
Soit $e \in E$ **pas** dans le *1-tree* T ($x_e = 0$)...

Arêtes *interdites*

Soit $e \in E$ **pas** dans le *1-tree* T ($x_e = 0$)...

- À enlever : arête de support $s \in T$
- Coût réduit : $\bar{c}(e) = \tilde{w}(e) - \tilde{w}(s)$
- e est *interdite* si $Z_{RL}(\boldsymbol{\lambda})[x_e = 1] = Z_{RL}(\boldsymbol{\lambda}) + \bar{c}(e) > U$

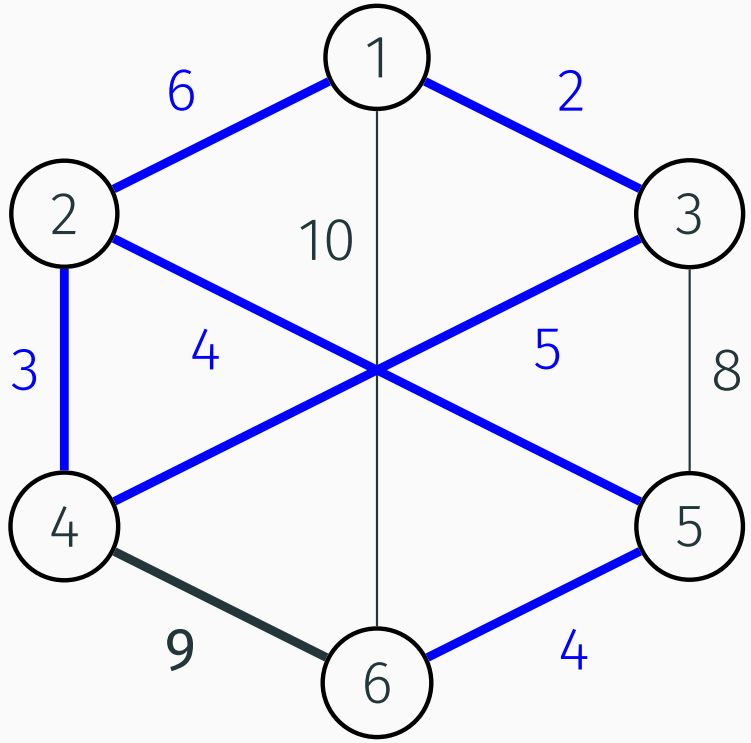
Arêtes *interdites*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 28$

$U = 28$

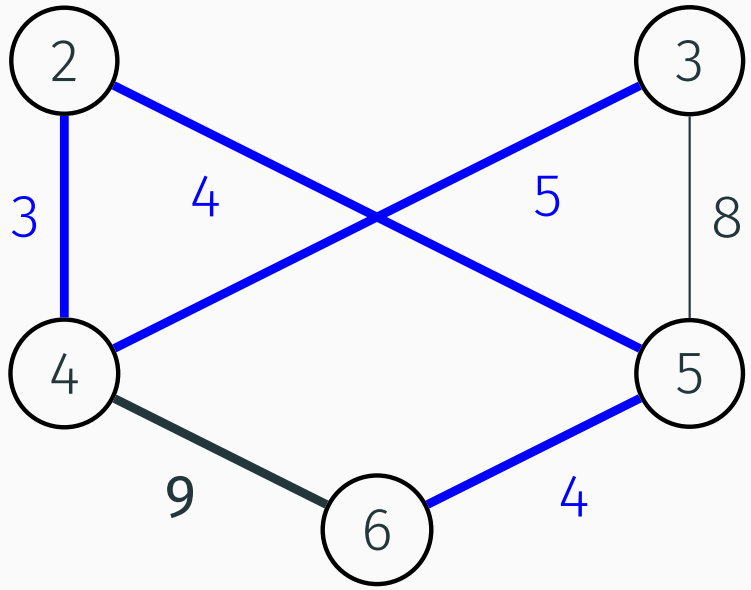
Arêtes *interdites*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 28$

$U = 28$

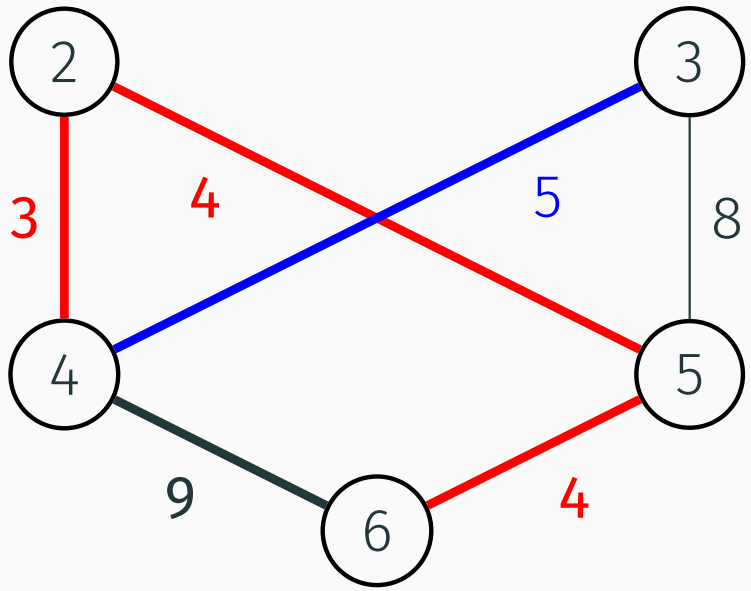
Arêtes *interdites*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 24$

$U = 28$

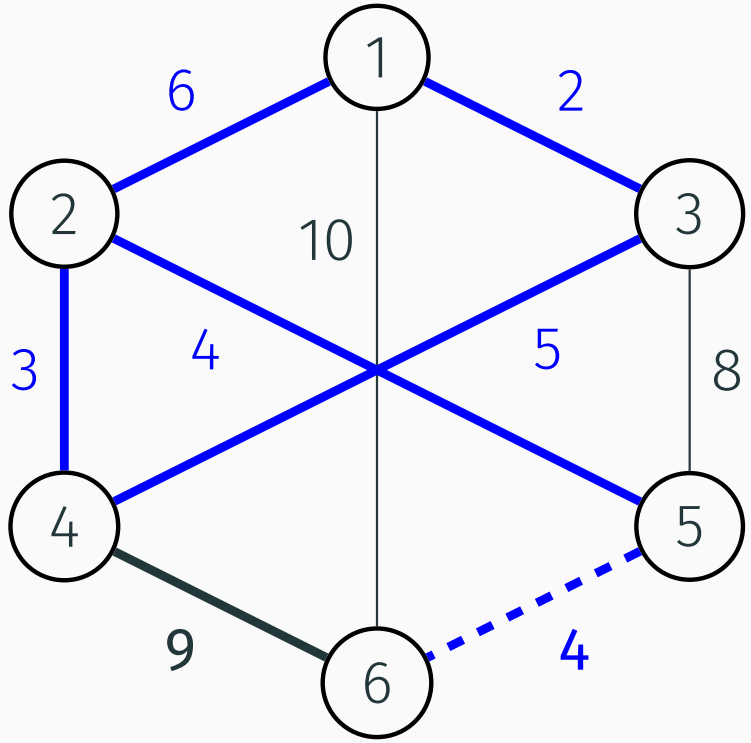
Arêtes *interdites*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 24$

$U = 28$

Arêtes *interdites*

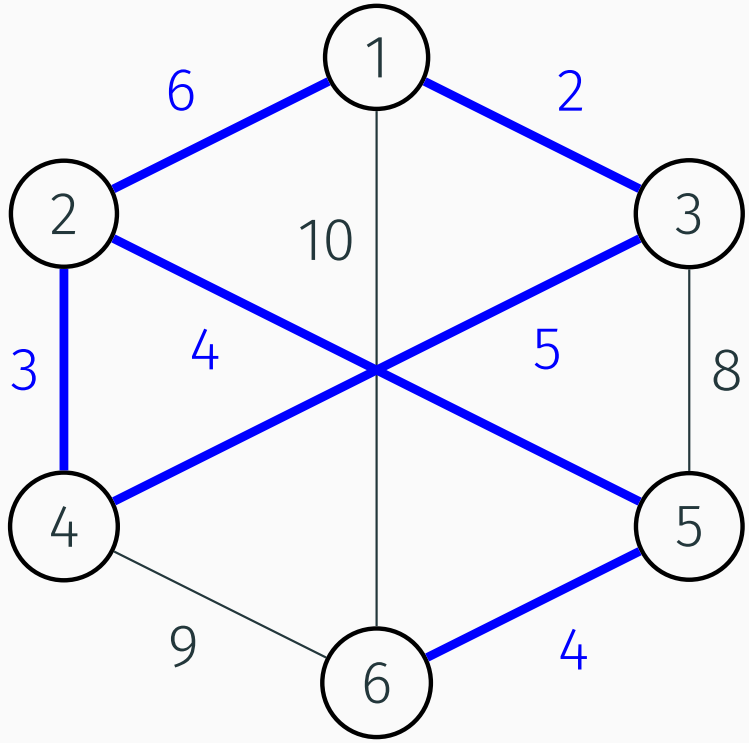


$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 24$

$U = 28$

$\bar{c}(\{4,6\}) = \underbrace{\tilde{w}(4,6)}_{9} - \underbrace{\tilde{w}(5,6)}_{4} = 5$

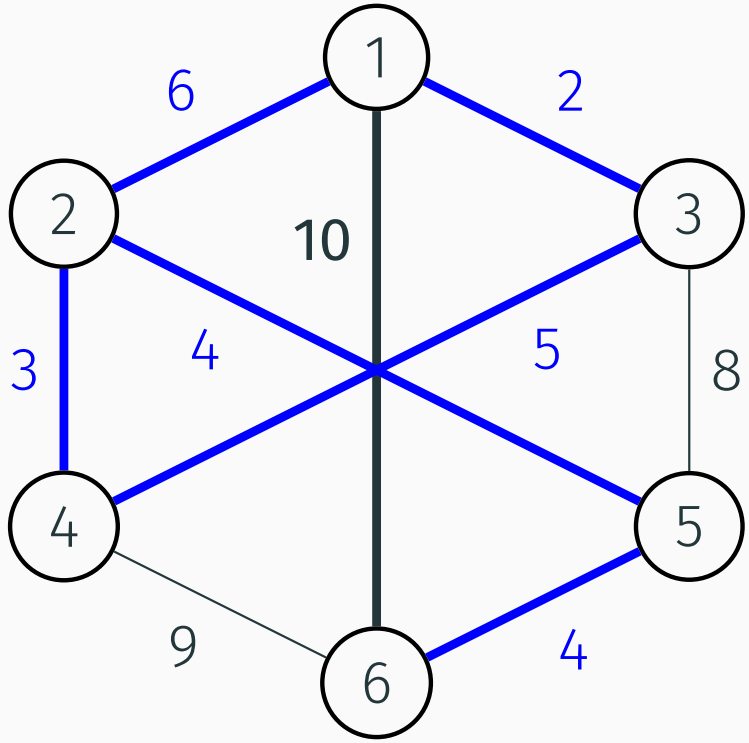
Arêtes *interdites*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 28$

$U = 28$

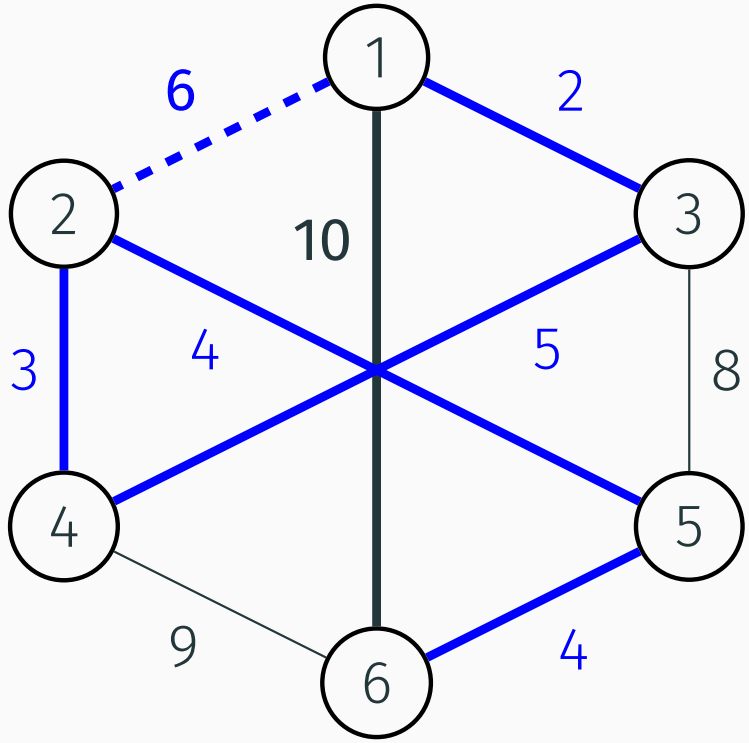
Arêtes *interdites*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 28$

$U = 28$

Arêtes *interdites*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 24$

$U = 28$

$\bar{c}(\{1,6\}) = \underbrace{\tilde{w}(1,6)}_{10} - \underbrace{\tilde{w}(1,2)}_{6} = 4$

Arêtes *obligatoires*

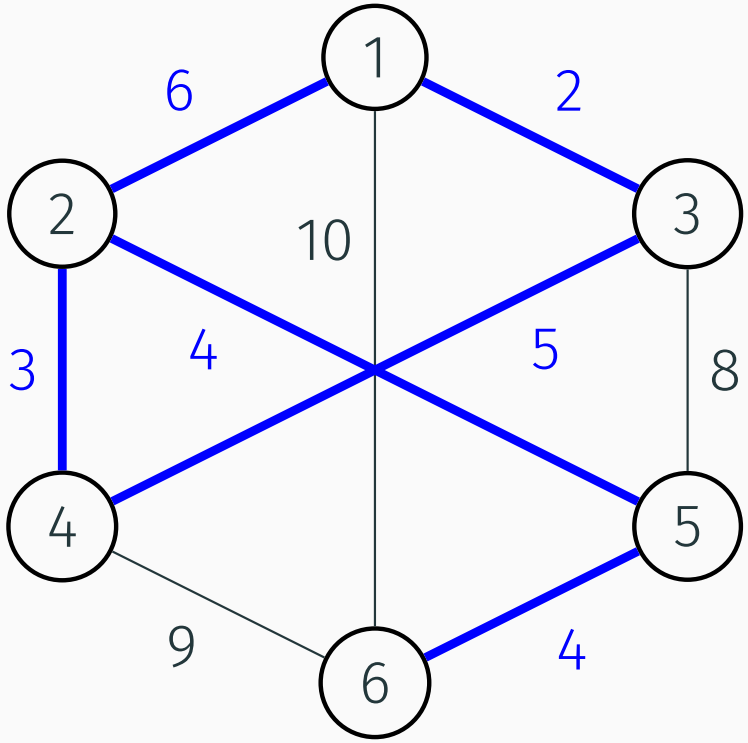
Soit $e \in E$ **dans** le *1-tree* T ($x_e = 1$)...

Arêtes *obligatoires*

Soit $e \in E$ **dans** le *1-tree* T ($x_e = 1$)...

- À ajouter : arête de remplacement $r \in T$
- Coût de remplacement : $\hat{c}(e) = \tilde{w}(r) - \tilde{w}(e)$
- e est *obligatoire* si $Z_{RL}(\boldsymbol{\lambda})[x_e = 0] = Z_{RL}(\boldsymbol{\lambda}) + \hat{c}(e) > U$

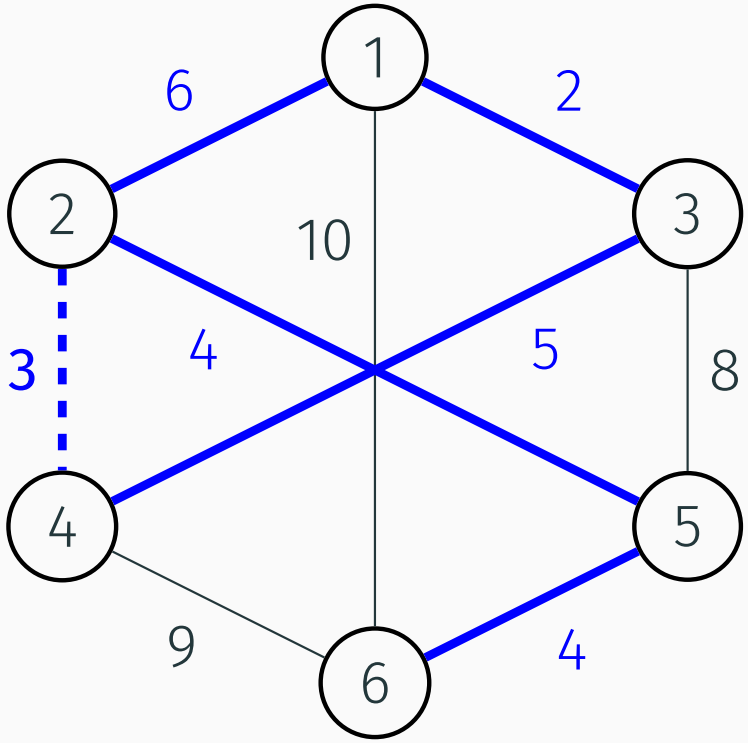
Arêtes *obligatoires*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 28$

$U = 28$

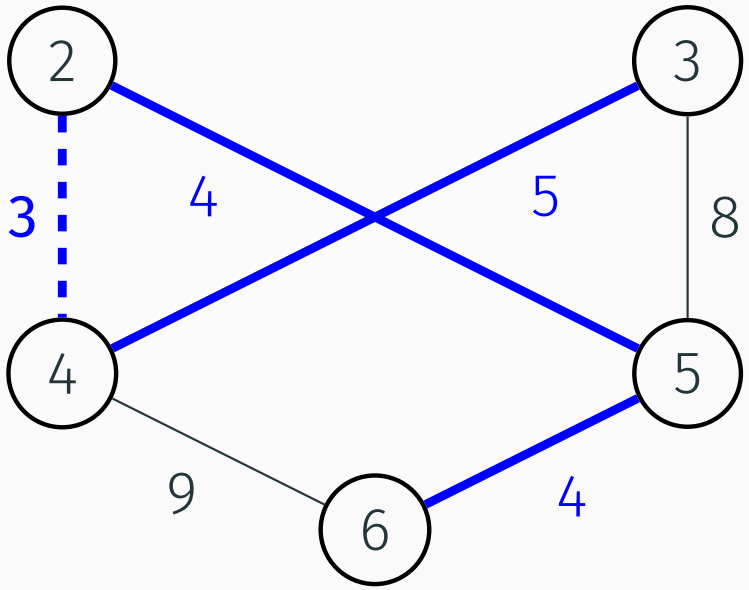
Arêtes *obligatoires*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 28$

$U = 28$

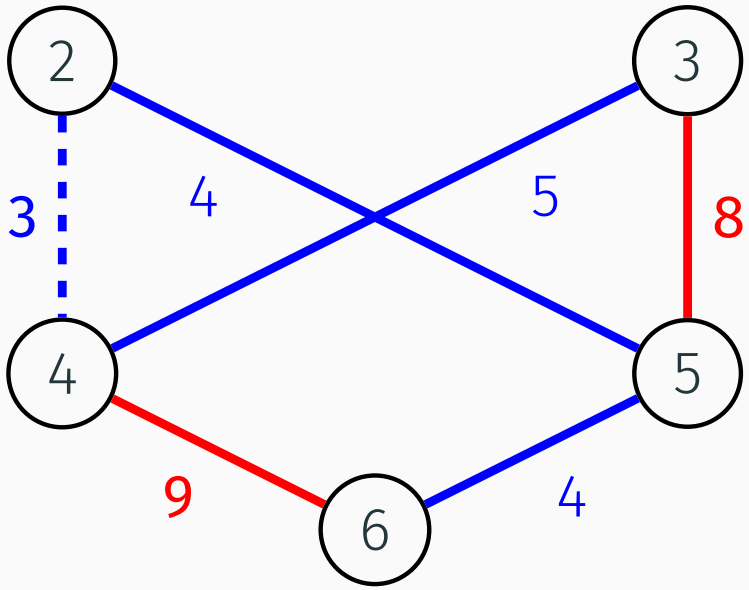
Arêtes *obligatoires*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 24$

$U = 28$

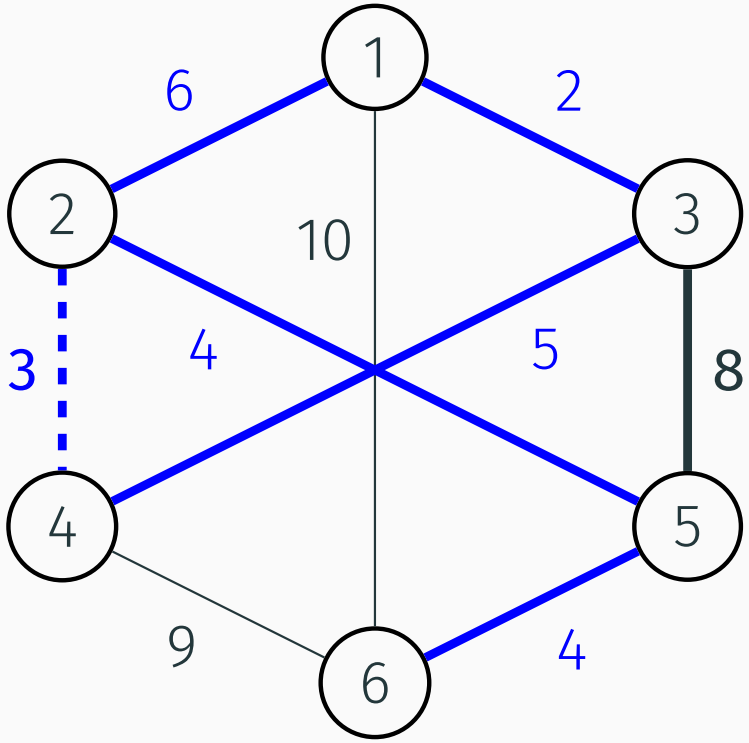
Arêtes *obligatoires*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 24$

$U = 28$

Arêtes *obligatoires*

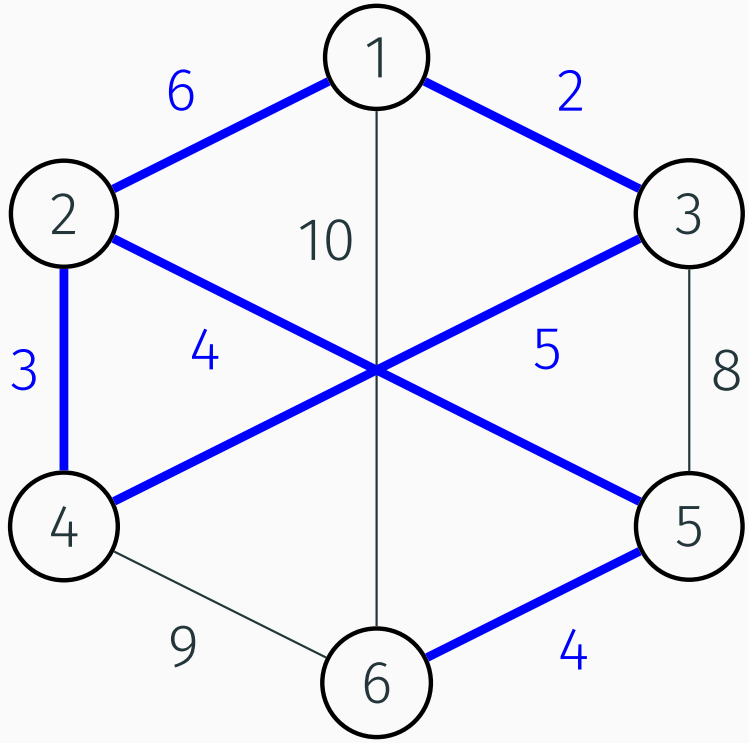


$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 24$

$U = 28$

$\hat{c}(\{2,4\}) = \underbrace{\tilde{w}(3,5)}_{8} - \underbrace{\tilde{w}(2,4)}_{3} = 5$

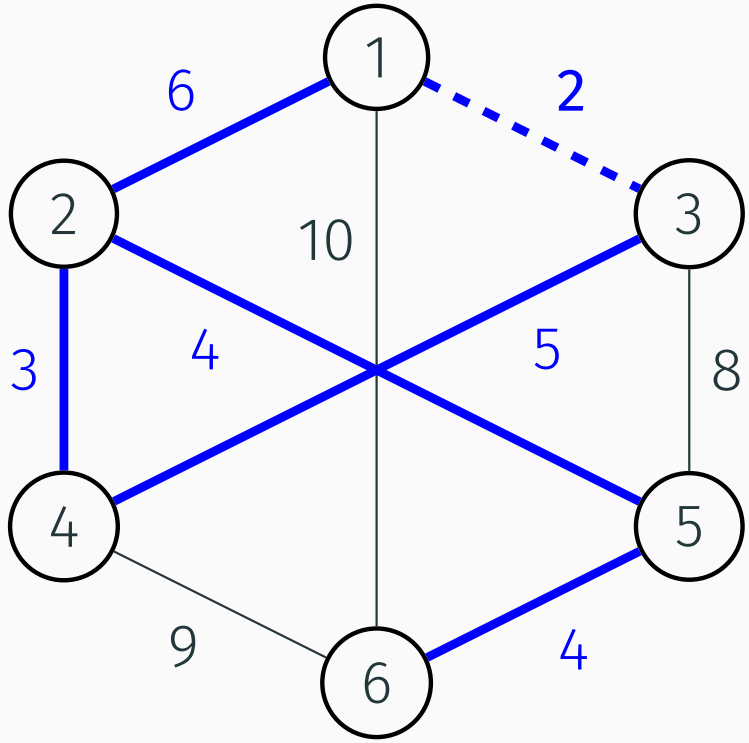
Arêtes *obligatoires*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 28$

$U = 28$

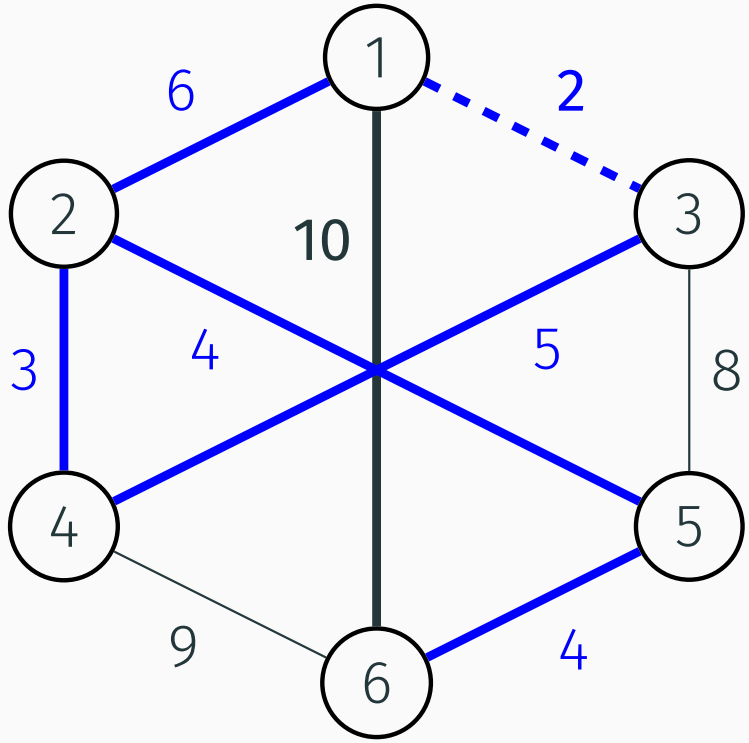
Arêtes *obligatoires*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 28$

$U = 28$

Arêtes *obligatoires*



$Z_{RL}(\boldsymbol{\lambda}) = 24 - 2(0) = 28$

$U = 28$

$\hat{c}(\{1,3\}) = \underbrace{\tilde{w}(1,6)}_{10} - \underbrace{\tilde{w}(1,3)}_{2} = 8$

Approche CP-LR améliorée

Principe

Question

Étant donné une variable x et une valeur μ de son domaine, est-il possible de modifier *temporairement* les multiplicateurs de Lagrange $\boldsymbol{\lambda}$ pour que cette valeur soit filtrée grâce aux coûts?

Principe

Question

Étant donné une variable x et une valeur μ de son domaine, est-il possible de modifier *temporairement* les multiplicateurs de Lagrange $\boldsymbol{\lambda}$ pour que cette valeur soit filtrée grâce aux coûts?

Objectif : Trouver $\boldsymbol{\lambda}'$ tel que $Z_{RL}(\boldsymbol{\lambda}')[x = \mu] > U$

Principe

Question

Étant donné une variable x et une valeur μ de son domaine, est-il possible de modifier *temporairement* les multiplicateurs de Lagrange $\boldsymbol{\lambda}$ pour que cette valeur soit filtrée grâce aux coûts?

Objectif : Trouver $\boldsymbol{\lambda}'$ tel que $Z_{RL}(\boldsymbol{\lambda}')[x = \mu] > U$

Lemme magique

Soit $e \in E$ et T le *1-tree* minimal sous $\boldsymbol{\lambda}$. En modifiant λ_i, si T reste inchangé et e garde la même arête de support (remplacement), $Z_{RL}(\boldsymbol{\lambda}')[x_e = b]$ est donné par un calcul arithmétique simple.

Algorithme Simple

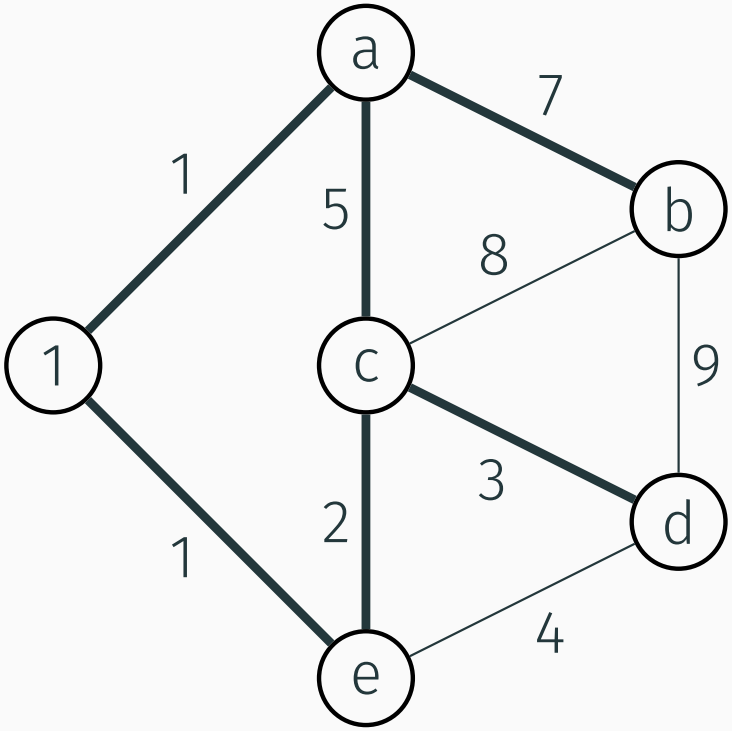
Soit $\{i, j\} \in E$...

Algorithme Simple

Soit $\{i, j\} \in E$...

- Considère des cas particuliers du *lemme magique*
- Modifie simultanément λ_i et λ_j selon les conditions
- Ne peut qu'augmenter $Z_{RL}(\boldsymbol{\lambda})[x_{\{i,j\}} = b]$

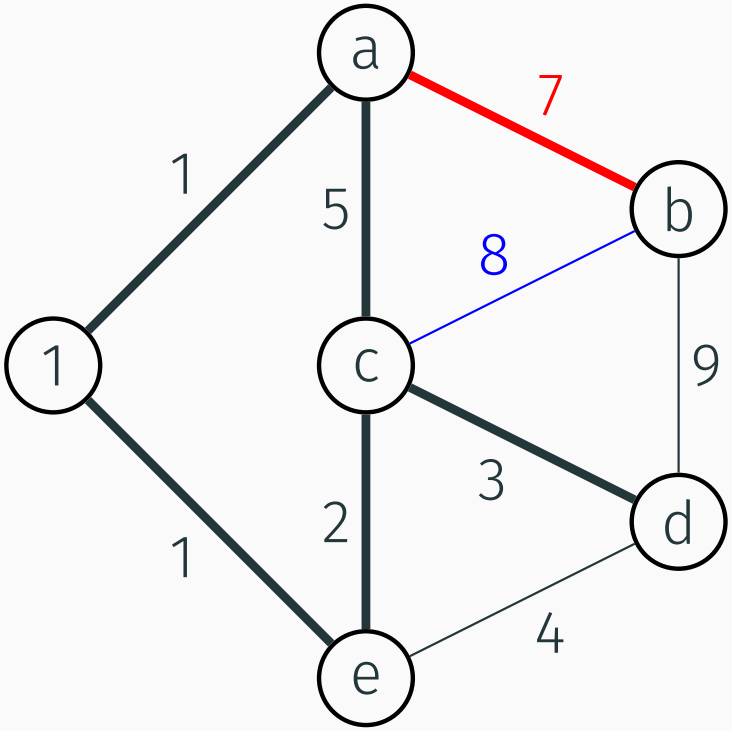
Algorithme Simple



$Z_{RL}(\boldsymbol{\lambda}) = 19 - 2(0) = 19$

$U = 23$

Algorithme Simple

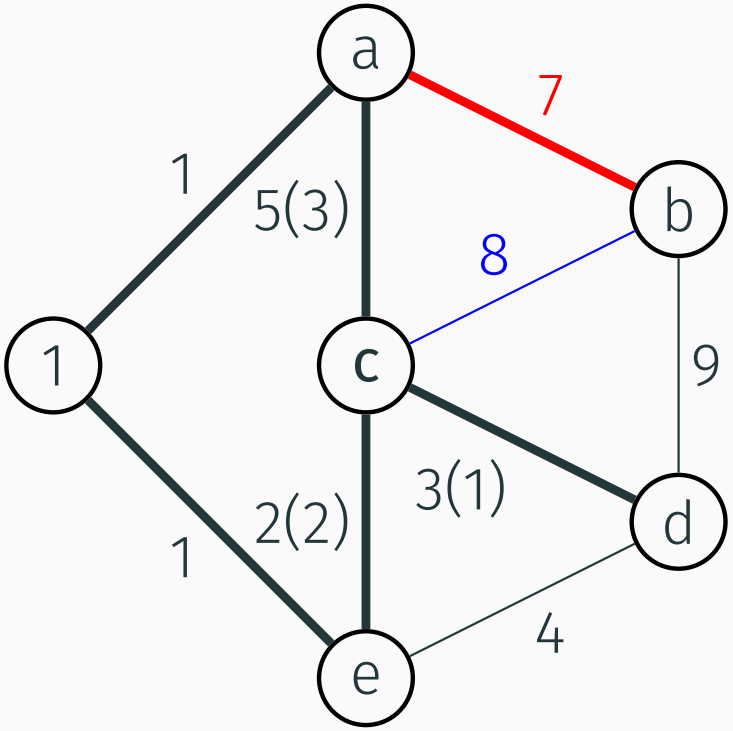


$$Z_{RL}(\boldsymbol{\lambda}) = 19 - 2(0) = 19$$

$$U = 23$$

$$\bar{c}(\{b,c\}) = \underbrace{\tilde{w}(b,c)}_{8} - \underbrace{\tilde{w}(a,b)}_{7} = 1$$

Algorithme Simple

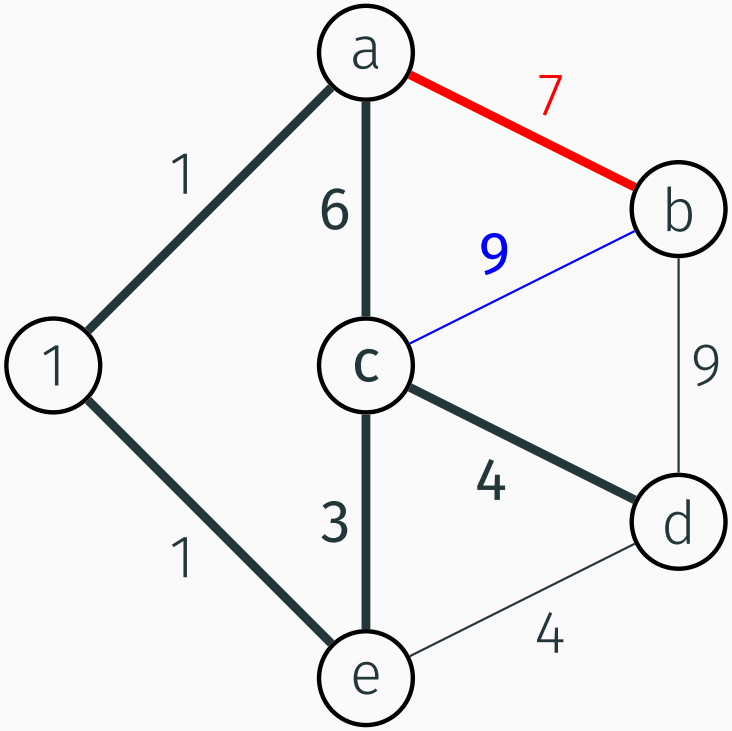


$$Z_{RL}(\boldsymbol{\lambda}) = 19 - 2(0) = 19$$

$$U = 23$$

$$\bar{c}(\{b,c\}) = \underbrace{\tilde{w}(b,c)}_{8} - \underbrace{\tilde{w}(a,b)}_{7} = 1$$

Algorithme Simple



$Z_{RL}(\boldsymbol{\lambda}) = \mathbf{22} - 2(\mathbf{1}) = 20$

$U = 23$

$\bar{c}(\{b,c\}) = \underbrace{\tilde{w}(b,c)}_{\mathbf{9}} - \underbrace{\tilde{w}(a,b)}_{7} = 2$

Algorithme Simple

- Si possible, on modifie deux nœuds sans recalcul
- Technique analogue pour les arêtes dans T

Algorithme Simple

- Si possible, on modifie deux nœuds sans recalcul
- Technique analogue pour les arêtes dans T
- 2 versions : *Relaxed* et *Complete*

Algorithme Simple

- Si possible, on modifie deux nœuds sans recalcul
- Technique analogue pour les arêtes dans T
- 2 versions : *Relaxed* et *Complete*
- Complexité en temps (pire cas) : $O(|V|)$

Algorithme α-Sets

Soit $\{i, j\} \in E$. On peut formuler les contraintes sur $\boldsymbol{\lambda}'$...

$$\lambda'_a + \lambda'_b - \lambda'_c - \lambda'_d \leq w(c, d) - w(a, b)$$

Algorithme α-Sets

Soit $\{i, j\} \in E$. On peut formuler les contraintes sur $\boldsymbol{\lambda}'$...

$$\lambda'_a + \lambda'_b - \lambda'_c - \lambda'_d \leq w(c, d) - w(a, b)$$

$O(|V|^4)$ contraintes ☹

Algorithme α-Sets

Soit $\{i, j\} \in E$. On peut formuler les contraintes sur $\boldsymbol{\lambda}'$...

$$\lambda'_a + \lambda'_b - \lambda'_c - \lambda'_d \leq w(c, d) - w(a, b)$$

$O(|V|^4)$ contraintes ☹

- Considère un ensemble incrémental de contraintes Ω
- Cherche un ensemble de nœuds A et $\alpha \geq 0$ tels que $\lambda'_u = \lambda_u + \sigma_u \cdot \alpha$ avec $\sigma_u \in \{+1, -1\}$, pour tout $u \in A$

Algorithme α-Sets

Soit $\{i, j\} \in E$. On peut formuler les contraintes sur $\boldsymbol{\lambda}'$...

$$\lambda'_a + \lambda'_b - \lambda'_c - \lambda'_d \leq w(c, d) - w(a, b)$$

$O(|V|^4)$ contraintes ☹

- Considère un ensemble incrémental de contraintes Ω
- Cherche un ensemble de nœuds A et $\alpha \geq 0$ tels que $\lambda'_u = \lambda_u + \sigma_u \cdot \alpha$ avec $\sigma_u \in \{+1, -1\}$, pour tout $u \in A$
- Chaque contrainte $\omega \in \Omega$ s'écrit sous la forme $c_\omega \cdot \alpha \leq m_\omega$
- En maximisant α, on trouve $\alpha^* \geq 0$:

Algorithme α-Sets

Soit $\{i, j\} \in E$. On peut formuler les contraintes sur $\boldsymbol{\lambda}'$...

$$\lambda'_a + \lambda'_b - \lambda'_c - \lambda'_d \leq w(c, d) - w(a, b)$$

$O(|V|^4)$ contraintes ☹

- Considère un ensemble incrémental de contraintes Ω
- Cherche un ensemble de nœuds A et $\alpha \geq 0$ tels que $\lambda'_u = \lambda_u + \sigma_u \cdot \alpha$ avec $\sigma_u \in \{+1, -1\}$, pour tout $u \in A$
- Chaque contrainte $\omega \in \Omega$ s'écrit sous la forme $c_\omega \cdot \alpha \leq m_\omega$
- En maximisant α, on trouve $\alpha^* \geq 0$:
  - Si $\alpha^* > 0$, on augmente $Z_{RL}(\boldsymbol{\lambda})[x_{\{i,j\}} = b]$

Algorithme α-Sets

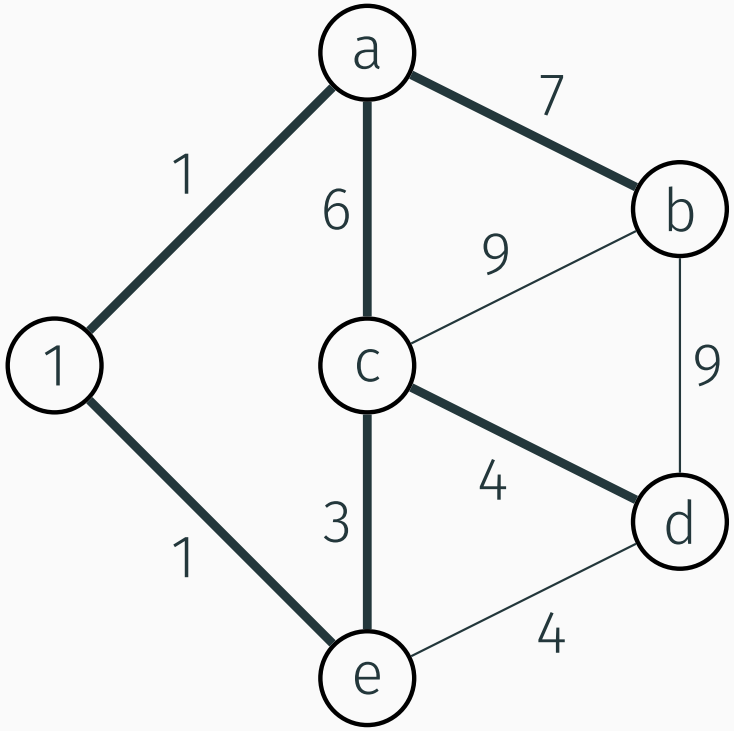
Soit $\{i, j\} \in E$. On peut formuler les contraintes sur $\boldsymbol{\lambda}'$...

$$\lambda'_a + \lambda'_b - \lambda'_c - \lambda'_d \leq w(c, d) - w(a, b)$$

$O(|V|^4)$ contraintes ☹

- Considère un ensemble incrémental de contraintes Ω
- Cherche un ensemble de nœuds A et $\alpha \geq 0$ tels que $\lambda'_u = \lambda_u + \sigma_u \cdot \alpha$ avec $\sigma_u \in \{+1, -1\}$, pour tout $u \in A$
- Chaque contrainte $\omega \in \Omega$ s'écrit sous la forme $c_\omega \cdot \alpha \leq m_\omega$
- En maximisant α, on trouve $\alpha^* \geq 0$:
  - Si $\alpha^* > 0$, on augmente $Z_{RL}(\boldsymbol{\lambda})[x_{\{i,j\}} = b]$
  - Si $\alpha^* = 0$, faire un choix de nœud pour débloquer!

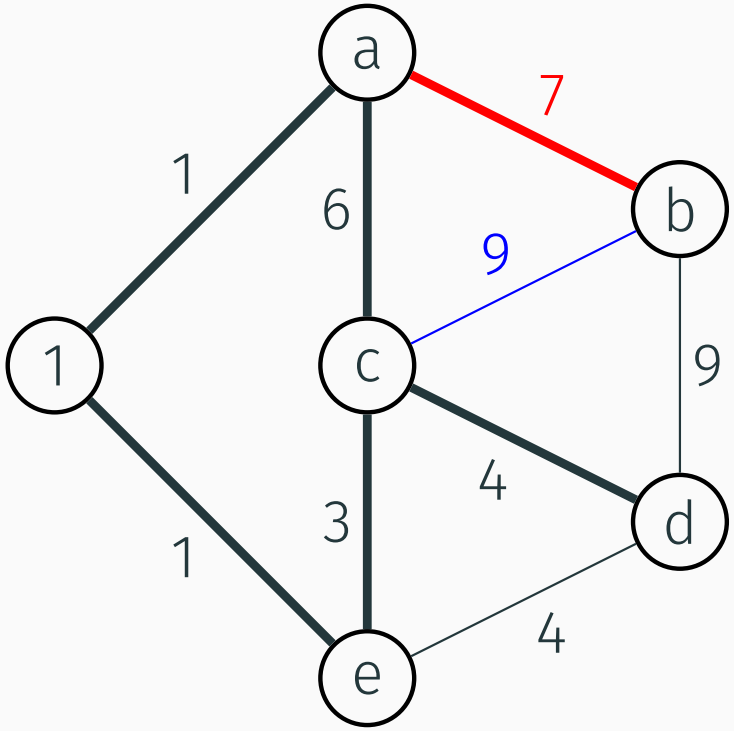
Algorithme α-Sets



$Z_{RL}(\boldsymbol{\lambda}) = 22 - 2(1) = 20$

$U = 23$

Algorithme α-Sets

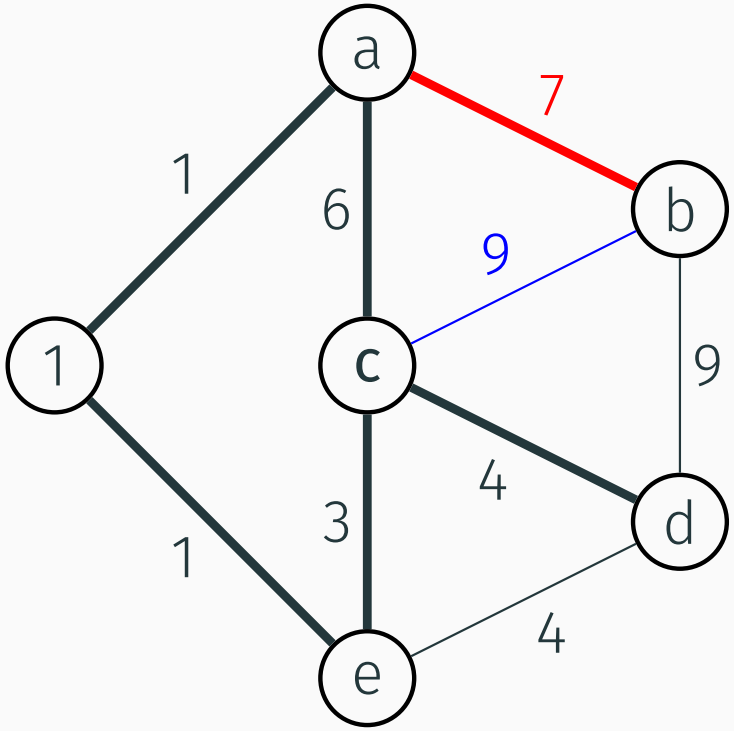


$$Z_{RL}(\boldsymbol{\lambda}) = 22 - 2(1) = 20$$

$$U = 23$$

$$\bar{c}(\{b,c\}) = \underbrace{\tilde{w}(b,c)}_{9} - \underbrace{\tilde{w}(a,b)}_{7} = 2$$

Algorithme α-Sets

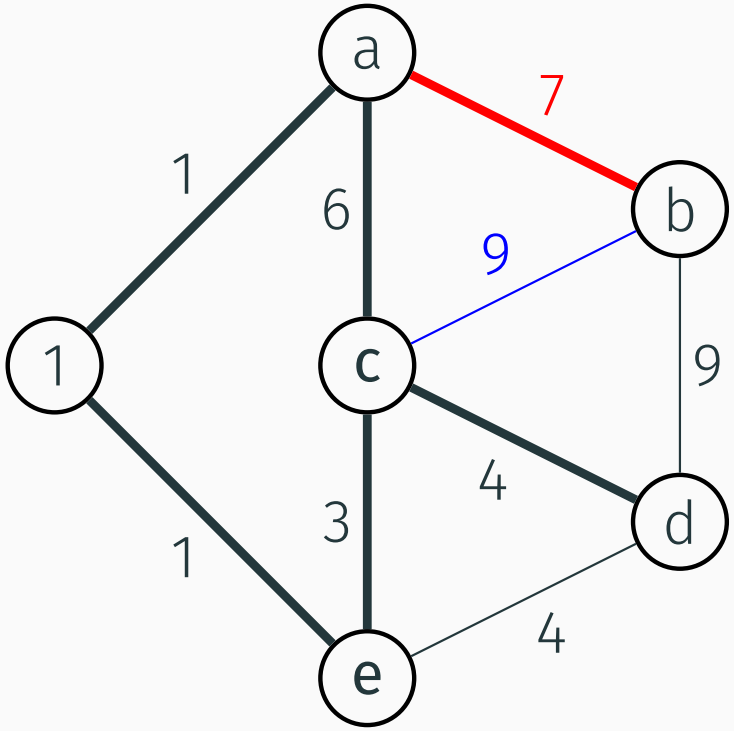


Contraintes

$$\alpha \leq \tilde{w}(a,b) - \tilde{w}(a,c) = 1$$
$$\alpha \leq \tilde{w}(b,d) - \tilde{w}(a,c) = 3$$
$$\alpha \leq \tilde{w}(b,d) - \tilde{w}(c,d) = 5$$
$$\alpha \leq \tilde{w}(d,e) - \tilde{w}(c,d) = \mathbf{0}$$
$$\alpha \leq \tilde{w}(d,e) - \tilde{w}(c,e) = 1$$

$$Z_{RL}(\boldsymbol{\lambda}) = 22 - 2(1) = 20$$
$$U = 23$$
$$\bar{c}(\{b,c\}) = \underbrace{\tilde{w}(b,c)}_{9} - \underbrace{\tilde{w}(a,b)}_{7} = 2$$

Algorithme α-Sets

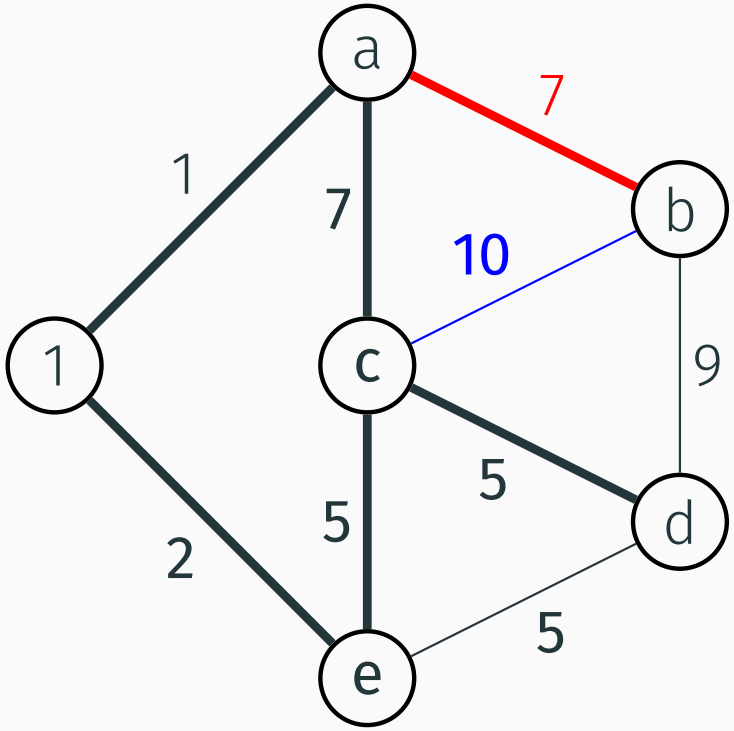


Contraintes

$$\alpha \leq \tilde{w}(a,b) - \tilde{w}(a,c) = \mathbf{1}$$
$$\alpha \leq \tilde{w}(b,d) - \tilde{w}(a,c) = 3$$
$$\alpha \leq \tilde{w}(b,d) - \tilde{w}(c,d) = 5$$
$$\alpha \leq \tilde{w}(d,e) - \tilde{w}(c,d) = 0$$
$$\alpha \leq \tilde{w}(d,e) - \tilde{w}(c,e) = \mathbf{1}$$

$$Z_{RL}(\boldsymbol{\lambda}) = 22 - 2(1) = 20$$
$$U = 23$$
$$\bar{c}(\{b,c\}) = \underbrace{\tilde{w}(b,c)}_{9} - \underbrace{\tilde{w}(a,b)}_{7} = 2$$

Algorithme α-Sets



$Z_{RL}(\boldsymbol{\lambda}) = \mathbf{27} - 2(\mathbf{3}) = 21$

$U = 23$

$\bar{c}(\{b,c\}) = \underbrace{\tilde{w}(b,c)}_{\mathbf{10}} - \underbrace{\tilde{w}(a,b)}_{7} = 3$ ☺

Algorithme α-Sets

- Itératif : Tant qu'un ensemble A valide existe
- Pour limiter le nombre de contraintes considérées : cardinalité maximale C_m
- Implémentation : Recherche en profondeur itérative

Algorithme α-Sets

- Itératif : Tant qu'un ensemble A valide existe
- Pour limiter le nombre de contraintes considérées : cardinalité maximale C_m
- Implémentation : Recherche en profondeur itérative
- Complexité en temps (pire cas) : $O(C_m|V||E|4^{C_m})$

Algorithme α-Sets

- Itératif : Tant qu'un ensemble A valide existe
- Pour limiter le nombre de contraintes considérées : cardinalité maximale C_m
- Implémentation : Recherche en profondeur itérative
- Complexité en temps (pire cas) : $O(C_m|V||E|4^{C_m})$
- Algorithme Hybrid : Simple *Complete* d'abord

Résultats et conclusion

Méthodologie

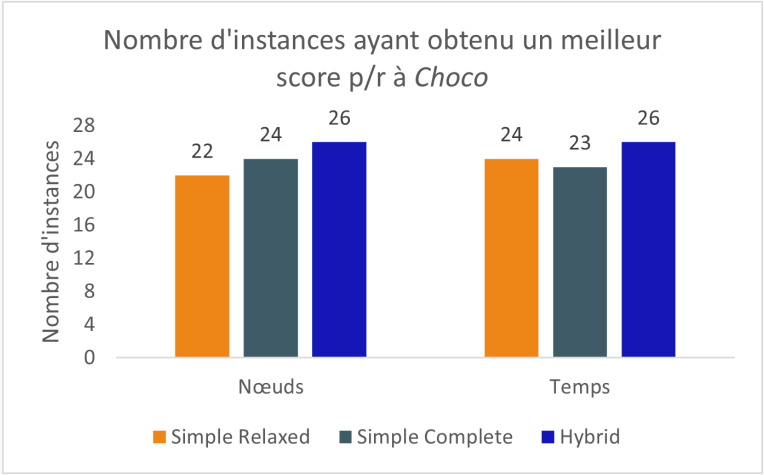
- Implémentés dans *Choco* / *Choco Graph*
- Banc d'essai : TSPLIB (28 instances, taille 96 à 431)
- Comparaison de Simple (2 versions) et Hybrid VS *Choco* (état de l'art pour l'implémentation de WeightedCircuit)
- Appelés à la dernière itération du sous-gradient
- α-Sets :
  - Nombre maximal d'itérations : 10
  - $C_m = 2$
  - Condition d'appel : $|E| \leq 2|V|$

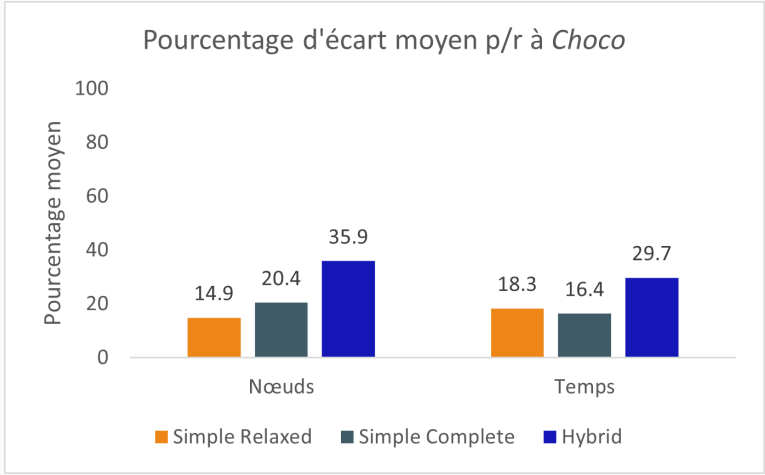
Résultats (en bref)

Mesure	*Choco*		Simple *Relaxed*		Simple *Complete*		Hybrid	
	Nœuds	Temps	Nœuds	Temps	Nœuds	Temps	Nœuds	Temps
Moy. arithmétique	89031	1779.8	65906	1393.3	63276	1339.5	**50592**	**1130.8**
Moy. géométrique	4084	53.1	3293	41.8	3150	43.3	**2484**	**35.6**
Écart-type	242149	4878	176157	3955.5	176451	3760.8	**138628**	**3317.6**

Résultats (en bref)

Mesure	*Choco*		Simple *Relaxed*		Simple *Complete*		Hybrid	
	Nœuds	Temps	Nœuds	Temps	Nœuds	Temps	Nœuds	Temps
Moy. arithmétique	89031	1779.8	65906	1393.3	63276	1339.5	**50592**	**1130.8**
Moy. géométrique	4084	53.1	3293	41.8	3150	43.3	**2484**	**35.6**
Écart-type	242149	4878	176157	3955.5	176451	3760.8	**138628**	**3317.6**





Conclusion

Contributions

- Introduction d'une approche améliorée pour les problèmes de type CP-LR
- Application de cette approche sur le filtrage de la contrainte WeightedCircuit
  1. Algorithme Simple
  2. Algorithme α-Sets
- Amélioration significative du temps de résolution comparativement à l'état de l'art implémenté dans *Choco*

Conclusion

Contributions

- Introduction d'une approche améliorée pour les problèmes de type CP-LR
- Application de cette approche sur le filtrage de la contrainte WeightedCircuit
  1. Algorithme Simple
  2. Algorithme α-Sets
- Amélioration significative du temps de résolution comparativement à l'état de l'art implémenté dans *Choco*

Suite des choses

- Intégration avec les travaux d'Isoart et Régin (2019, 2020)
- Expérimentations sur d'autres problèmes de type CP-LR

Merci!
Questions?