Optimisation en nombres entiers

Michel Bierlaire

michel.bierlaire@epfl.ch

EPFL - Laboratoire Transport et Mobilité - ENAC

Définitions

Optimisation en nombres entiers

Un problème d'optimisation en nombres entiers est un problème d'optimisation dont toutes les variables sont contraintes à ne prendre que des valeurs entières.

- Variables discrètes : nombre d'objets à considérer, nombre d'actions à effectuer, etc.
  - Nombres de vélos à installer sur le campus.
  - Nombres d'ouvriers à affecter à un chantier.
- Variables binaires (0/1) : oui/non, allumer/éteindre, etc.
  - Utiliser la voiture ou pas.
  - Construire un pont ou pas.
  - Allumer la climatisation ou pas.

Définitions

Optimisation mixte en nombres entiers

Un problème d'optimisation mixte en nombres entiers est un problème d'optimisation dont certaines *variables sont contraintes à ne prendre que des valeurs entières.*

- Mobilité :
  - Décision binaire : acheter une seconde voiture ou non.
  - Décision continue : nombre de kilomètres à effectuer.
- Energie :
  - Décision binaire : installer une nouvelle chaudière électricité/gaz.
  - Décision continue : quantité de gaz à brûler.

Introduction

- Problème d'optimisation linéaire en nombres entiers

$$\min_{x \in \mathbb{R}^n} c^T x$$

sous contraintes

$$Ax = b$$

$$x \geq 0$$

$$x \in \mathbb{N}$$

Introduction

- Problème d'optimisation linéaire binaire

$$\min_{x \in \mathbb{R}^n} c^T x$$

sous contraintes

$$Ax = b$$

$$x \geq 0$$

$$x \in \{0, 1\}^n$$

Introduction

Approche intuitive immédiate :

- Ignorer les contraintes d'intégralité.
- Résoudre le problème linéaire.
- Si la solution n'est pas entière, arrondir à l'entier le plus proche.

En général, cela ne fonctionne pas !

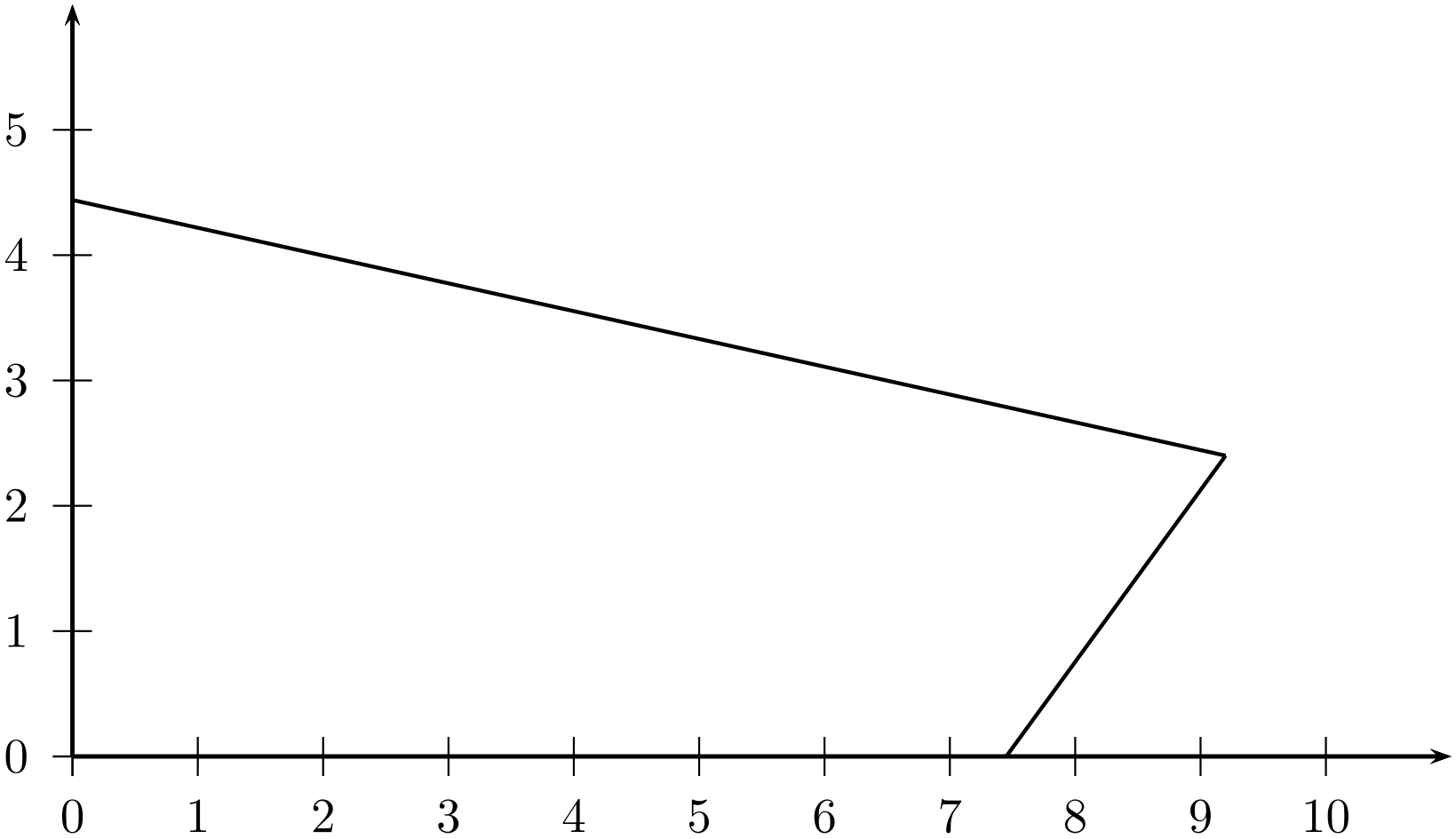
Exemple

$$\min_{x \in \mathbb{R}^2} -3x_1 - 13x_2$$

sous contraintes

$$\begin{aligned} 2x_1 + 9x_2 &\leq 40 \\ 11x_1 - 8x_2 &\leq 82 \\ x_1, x_2 &\geq 0 \\ x_1, x_2 &\text{ entiers} \end{aligned}$$

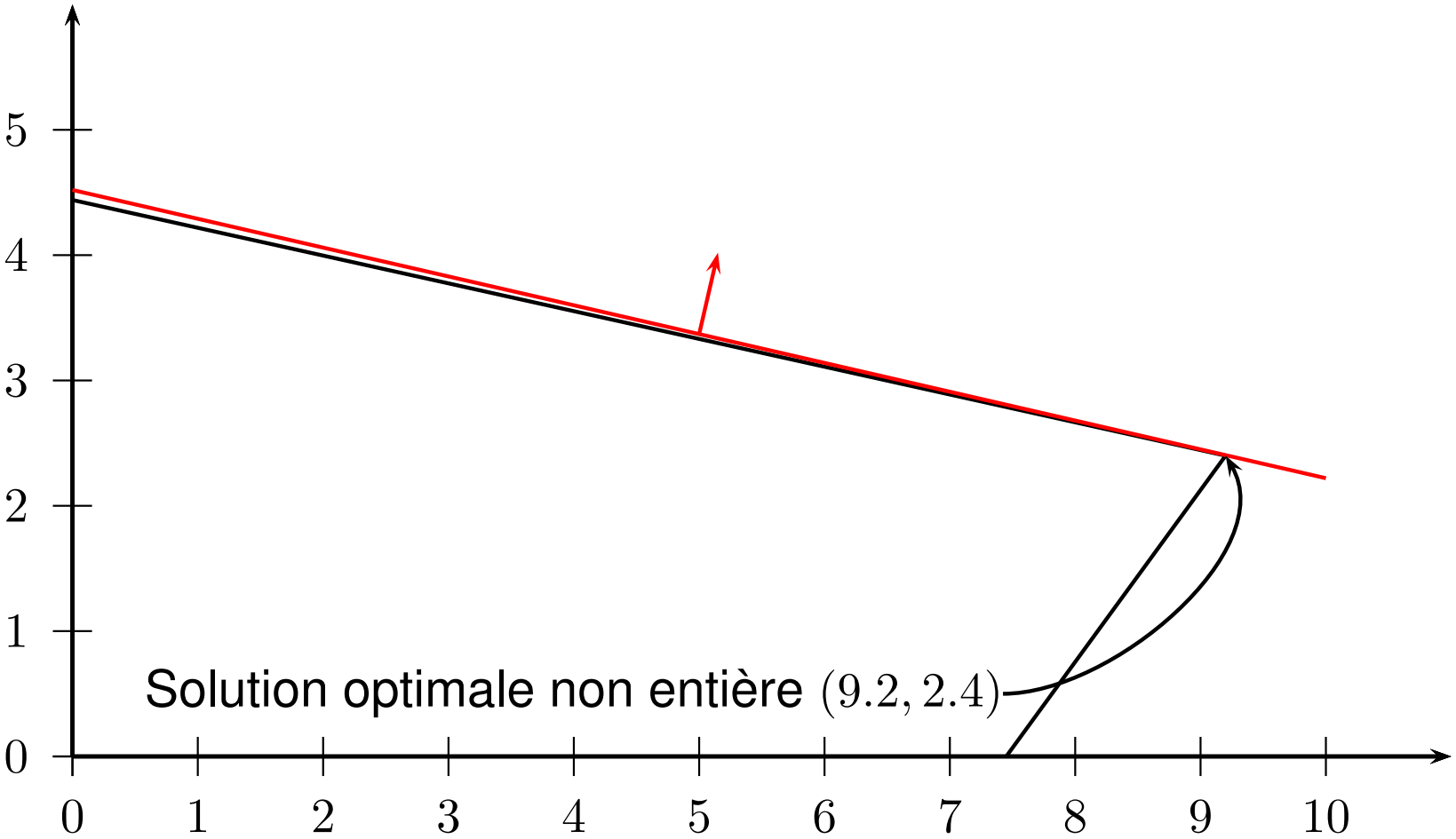
Exemple : polytope des contraintes



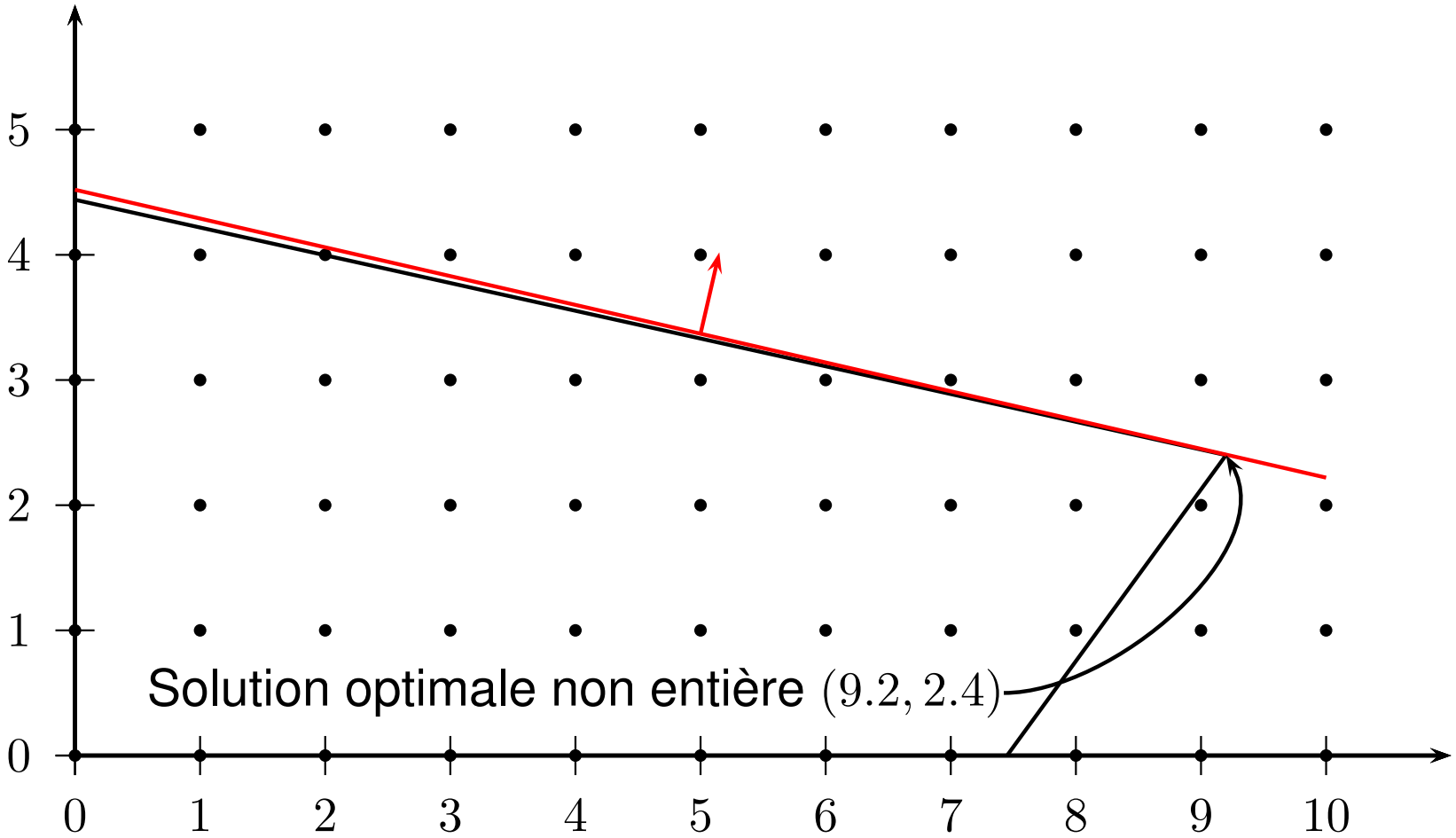
TRANSP-OR

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

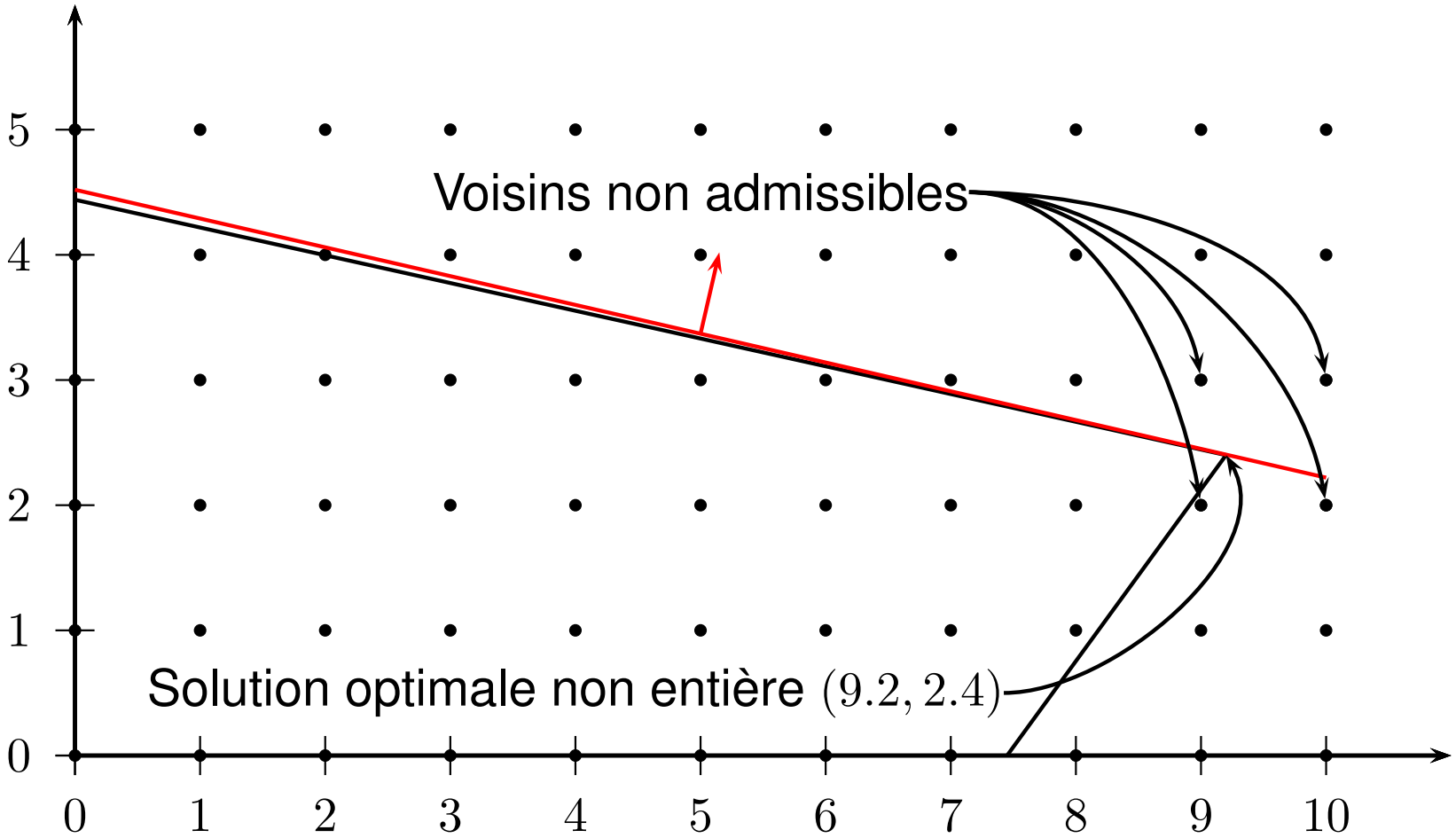
Exemple : solution du problème continu



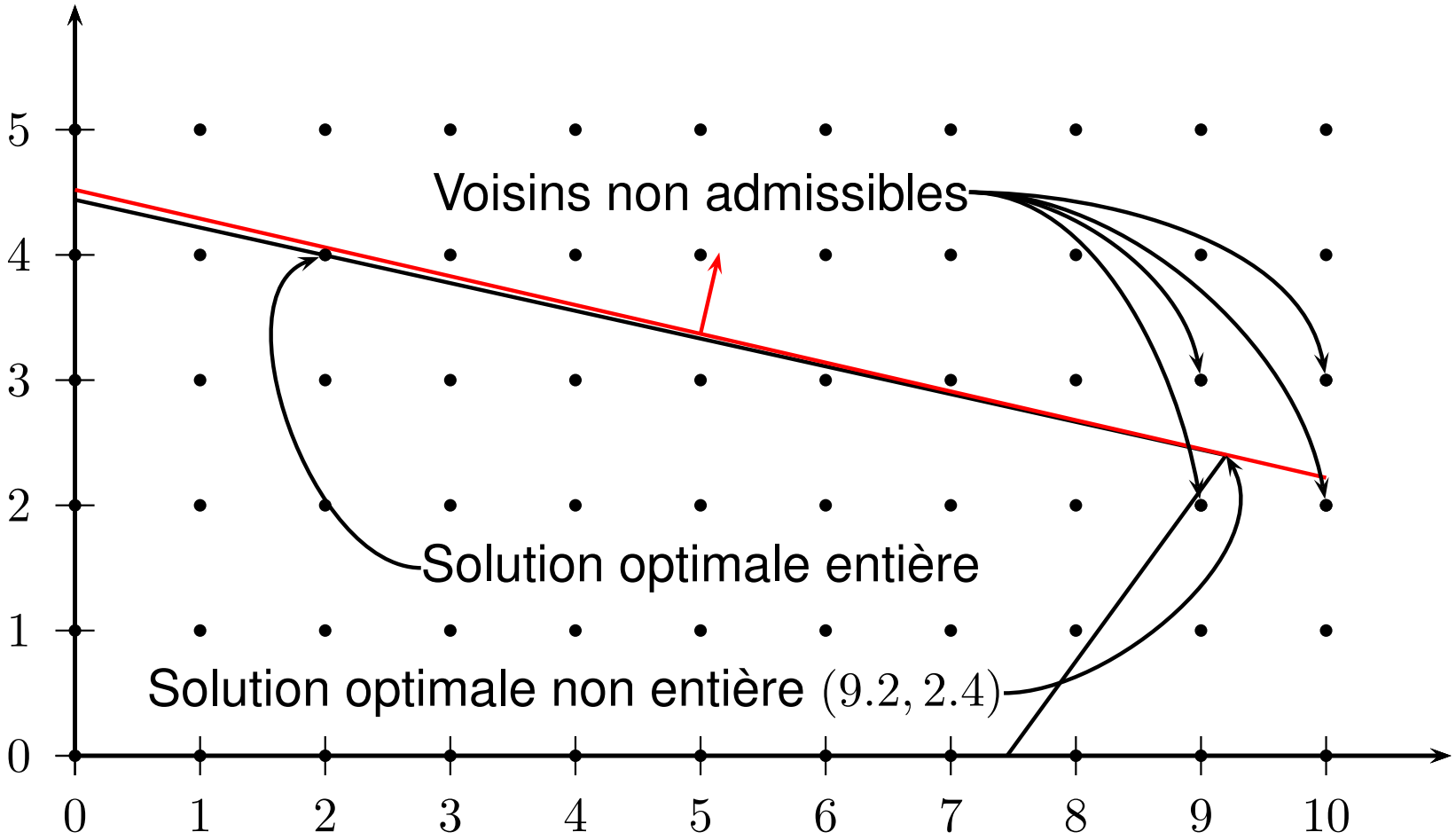
Exemple : contraintes d'intégralité



Exemple : voisins non admissibles



Exemple : solution du problème discret



Problèmes

- Il y a 2^n façons d'arrondir une solution non entière. Laquelle choisir ?
- Arrondir une solution non entière peut générer une solution non admissible.
- La solution arrondie peut se trouver très loin de la solution optimale.

Problème d'investissement

- Une société désire investir dans l'énergie hydro-électrique.
- Les ingénieurs ont identifié 4 sites potentiels pour la construction de barrages.
- Pour chaque site, ils ont évalué les coûts d'investissement, et le bénéfice attendu sur le long terme.
- La société a une capacité d'investissement de 1400 kCHF.
- Quels barrages doit-elle construire ?

Barrage	Coût	Bénéfice	Rendement
1	500	1600	3.20
2	700	2200	3.14
3	400	1200	3.00
4	300	800	2.67

Problème d'investissement

Modélisation :

- Variables de décision: x_1, x_2, x_3, x_4.

$$x_i = \begin{cases} 1 & \text{si le barrage } i \text{ est choisi,} \\ 0 & \text{sinon.} \end{cases}$$

- Fonction objectif : maximiser le bénéfice

$$\max 16x_1 + 22x_2 + 12x_3 + 8x_4$$

- Contrainte : capacité d'investissement

$$5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14$$

Problème d'investissement

Problème linéaire continu :

$$\max_{x \in \mathbb{R}^4} 16x_1 + 22x_2 + 12x_3 + 8x_4$$

sous contraintes

$$5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14$$

$$x_1, x_2, x_3, x_4 \geq 0.$$

Solution optimale :

$$x_1 = 2.8, x_2 = 0, x_3 = 0, x_4 = 0.$$

Problème d'investissement

Solution optimale :

$$x_1 = 2.8, x_2 = 0, x_3 = 0, x_4 = 0.$$

- Décision : ne construire que le barrage 1.
- Coût : 500 kCHF
- Somme non investie : 900 kCHF
- Bénéfice : 1600 kCHF

Problème d'investissement

Problème linéaire continu 2 :

$$\max_{x \in \mathbb{R}^4} 16x_1 + 22x_2 + 12x_3 + 8x_4$$

sous contraintes

$$5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14$$

$$0 \leq x_1, x_2, x_3, x_4 \leq 1.$$

Solution optimale :

$$x_1 = 1, x_2 = 1, x_3 = 0.5, x_4 = 0.$$

Problème d'investissement

Solution optimale :

$$x_1 = 1, x_2 = 1, x_3 = 0.5, x_4 = 0.$$

- Décision : construire les barrages 1 et 2.
- Barrage 3 : plus assez d'argent pour le construire.
- Coût : 1200 kCHF
- Somme non investie : 200 kCHF
- Bénéfice : 3800 kCHF

Problème d'investissement

Problème linéaire discret :

$$\max_{x \in \mathbb{R}^4} 16x_1 + 22x_2 + 12x_3 + 8x_4$$

sous contraintes

$$5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14$$

$$x_1, x_2, x_3, x_4 \in \{0, 1\}$$

Solution optimale :

$$x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1.$$

Problème d'investissement

Solution optimale :

$$x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1.$$

- Décision : construire les barrages 2, 3 et 4.
- Coût : 1400 kCHF
- Somme non investie : 0 kCHF
- Bénéfice : 4200 kCHF
- Le barrage correspondant au plus haut rendement n'est pas sélectionné.

Conclusion : l'approche "intuitive" ne fonctionne pas.

Le problème du sac à dos

- Le problème d'investissement est connu dans la littérature sous le nom de "problème du sac à dos".
- Soient n objets.
- Chacun a une valeur, une utilité u_i, $i = 1, \ldots, n$.
- Chacun a un poids, une masse w_i, $i = 1, \ldots, n$.
- Quels sont les objets qu'il faut choisir pour maximiser l'utilité en respectant une contrainte de capacité ?

$$\max_{x \in \{0,1\}^n} u^T x$$

sous contraintes

$$w^T x \leq W.$$

Enumération des solutions

- Pour chaque objet, on peut soit le prendre, soit non.
- Il y a donc 2^n possibilités.
- Pour chacune d'elles, il faut vérifier si elle est admissible et, si oui, calculer l'utilité.
- Cela fait environ $2n$ opérations "floating point" par possibilité.
- Supposons que l'on dispose d'un processor de 1 Teraflops, c'est-à-dire 10^{12} opérations floating point par seconde.
- Si $n = 34$, cela prend environ 1 seconde à résoudre.
- Si $n = 40$, cela prend environ 1 minute.
- Si $n = 45$, cela prend environ 1 heure.
- Si $n = 50$, cela prend environ 1 jour.

Enumération des solutions

- Si $n = 58$, cela prend environ 1 an.
- Si $n = 69$, cela prend environ 2583 ans, plus que la durée de l'ère chrétienne.
- Si $n = 78$, cela prend 1 500 000 ans, temps écoulé depuis l'arrivée de l'homo erectus sur terre.
- Si $n = 91$, cela prend 10^{10} ans, environ l'âge de l'univers.

Conditions d'optimalité

- Il n'est pas possible de caractériser la solution optimale d'un problème d'optimisation en nombres entiers.
- Autrement dit, il n'y a pas de conditions d'optimalité pour l'optimisation discrète.
- Cela complique considérablement la résolution du problème.
- Il y a essentiellement deux manières d'aborder le problème.

Algorithmes

1. Méthodes exactes :
   - la solution optimale est fournie,
   - mais le temps nécessaire pour la trouver est une fonction exponentielle de la taille du problème.
2. Méthodes heuristiques :
   - une “bonne” solution est fournie,
   - aucune garantie d’optimalité,
   - aucune mesure de qualité de la solution,
   - performances évaluées empiriquement sur des problèmes connus.

Branch & bound

Idées :

- Parcours systématique de l'ensemble admissible.
- Diviser pour conquérir.
- Utilisation de bornes sur le coût optimal pour éliminer des régions admissibles sans les explorer.

Branch

Soit le problème d'optimisation P

$$\min f(x)$$

sous contraintes

$$x \in F$$

- F est l'ensemble des solutions admissibles.
- Soit une partition de F

$$F = F_1 \cup \ldots \cup F_K.$$

- Soit x_k^* la solution optimale du problème P_k

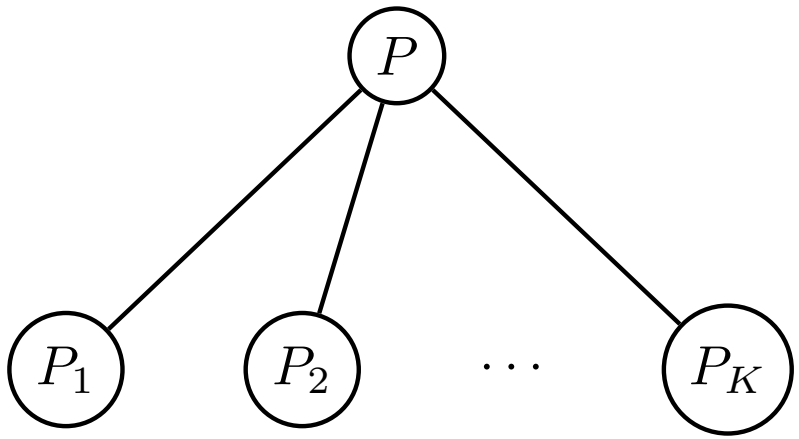
$$\min f(x)$$

sous contraintes

$$x \in F_k$$

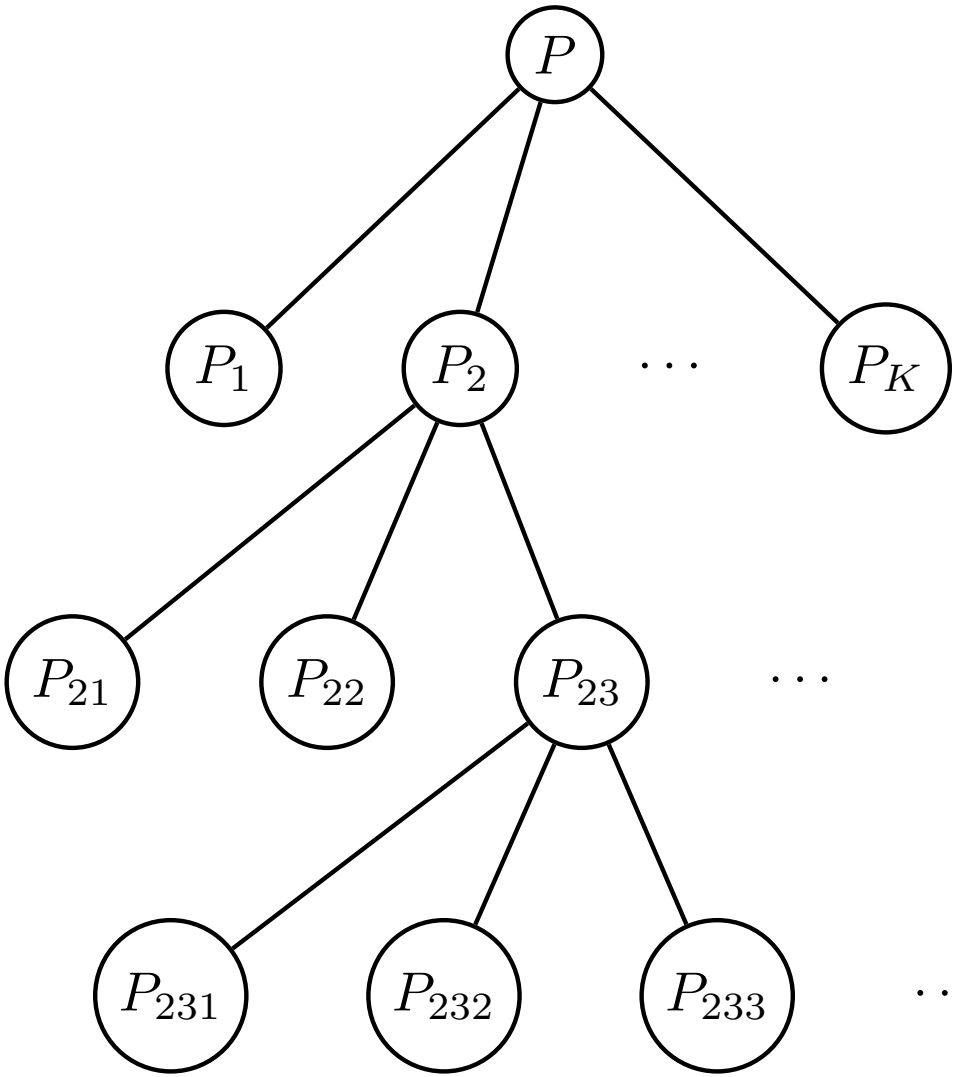
Branch

- Pour tout k, x_k^* est admissible pour le problème P.
- Soit i tel que
$$f(x_i^*) \leq f(x_k^*),\ \forall k = 1, \ldots, K.$$
- Alors, x_i^* est la solution optimale du problème P.
- Motivation : chaque sous-problème est plus simple que le problème initial.



Branch

Et chaque problème peut à nouveau être partitionné.



Branch

- Le nombre de sous-problèmes devient vite très grand.
- Il faut éviter de passer toutes les combinaisons possibles en revue.
- Idée : utiliser des bornes.

Bound

Soit le sous-problème P_k

$$\min f(x)$$

sous contraintes

$$x \in F_k$$

- On suppose que l'on peut calculer facilement une borne inférieure b_k

$$b_k \leq f(x),\ \forall x \in F_k.$$

- Soit $y \in F$ une solution admissible du problème P.
- Si

$$f(y) \leq b_k \leq f(x),\ \forall x \in F_k,$$

alors cela ne vaut pas la peine de résoudre le problème P_k.

Algorithme de branch & bound

A chaque instant on maintient

- une liste de sous-problèmes actifs $\{P_1, P_2, \ldots\}$,
- la valeur $U = f(y)$ de la meilleure solution admissible obtenue jusque là.
- Initialisation :
  - soit $U = +\infty$,
  - soit $U = f(x)$ avec x admissible connu.

Algorithme de branch & bound

Itération:

- Soit P_k un sous-problème actif.
- Si P_k est non admissible, le supprimer de la liste.
- Sinon, calculer la borne b_k.
- Si $U \leq b_k$, supprimer P_k de la liste.
- Sinon,
  - soit résoudre P_k directement,
  - soit partitionner F_k et créer de nouveaux sous-problèmes, qui sont ajoutés à la liste.

Exemple : problème d'affectation

- Sur un chantier, il faut affecter 4 ouvriers à 4 tâches.
- L'ouvrier i effectue la tâche j en c_{ij} heures :

Ouvrier	Tâche 1	Tâche 2	Tâche 3	Tâche 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

- Comment répartir les tâches pour que le nombre total d'heures soit le plus petit possible ?

Exemple : problème d'affectation

Modélisation :

- Variables de décision :

$$x_{ij} = \begin{cases} 1 & \text{si l'ouvrier } i \text{ effectue la tâche } j, \\ 0 & \text{sinon.} \end{cases}$$

- Fonction objectif :

$$\min \sum_{ij} c_{ij} x_{ij}$$

Exemple : problème d'affectation

- Contraintes:
  1. Chaque ouvrier doit effectuer exactement une tâche :

$$\sum_j x_{ij} = 1 \; \forall i.$$

  2. Chaque tâche doit être effectuée par exactement un ouvrier :

$$\sum_i x_{ij} = 1 \; \forall j.$$

Exemple : problème d'affectation

Ouvrier	Tâche 1	Tâche 2	Tâche 3	Tâche 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Calcul de la borne :

- La tâche la plus rapide pour A prend 2 heures.
- La tâche la plus rapide pour B prend 3 heures.
- La tâche la plus rapide pour C prend 1 heure.
- La tâche la plus rapide pour D prend 4 heures.

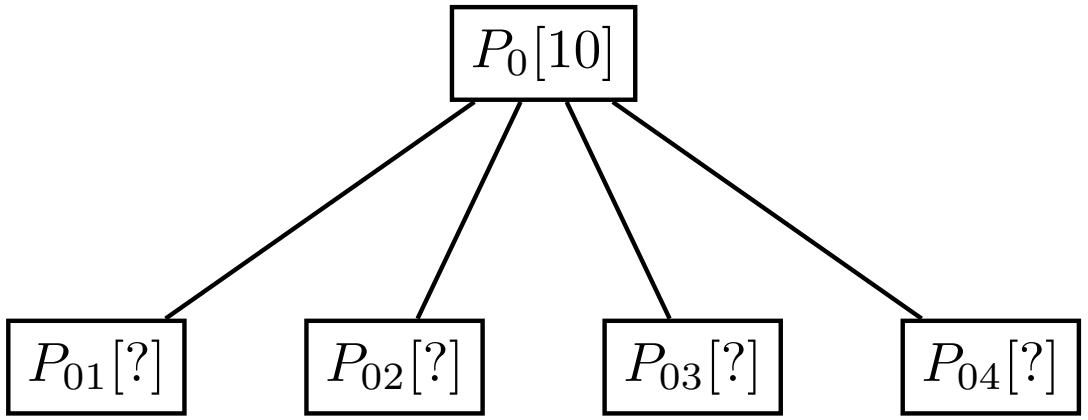
Impossible de faire mieux que 10 heures.

Exemple : problème d'affectation

- Problèmes actifs : $\{P_0\}$
- $U = +\infty$
- $b_0 = 10 < U$: on partitionne P_0.
  - P_{01} : on décide que A effectue la tâche 1,
  - P_{02} : on décide que A effectue la tâche 2,
  - P_{03} : on décide que A effectue la tâche 3,
  - P_{04} : on décide que A effectue la tâche 4.
- Chacun de ces problèmes revient à affecter 3 ouvriers à 3 tâches.
- Ils sont chacun plus simples que P_0.
- Notation : $P_k[b_k]$

Exemple : problème d'affectation

$U = +\infty$



Exemple : problème d'affectation

Calcul des bornes.

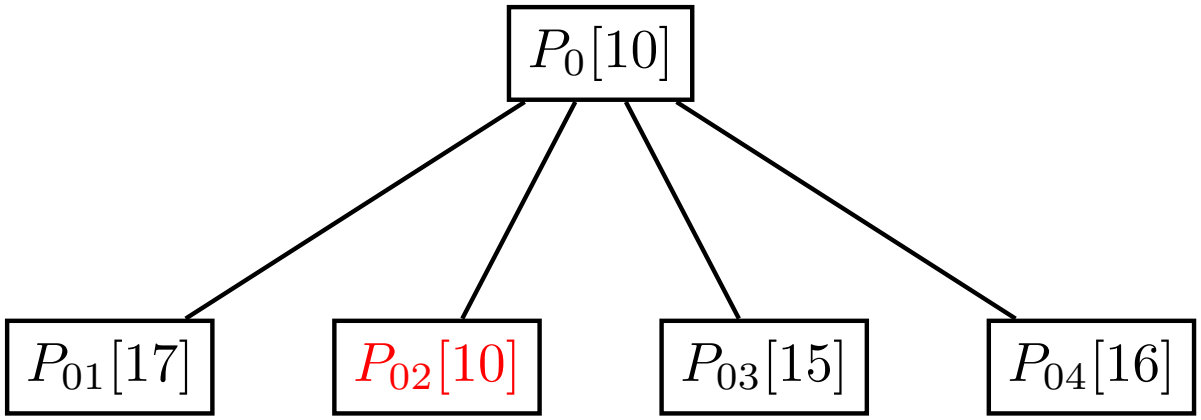
Ouvrier	Tâche 1	Tâche 2	Tâche 3	Tâche 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

- La tâche la plus rapide pour B prend 3 heures.
- La tâche la plus rapide pour C prend 1 heure.
- La tâche la plus rapide pour D prend 4 heures.

$$
\begin{aligned}
b_{01} &= 9+8 &= 17 \\
b_{02} &= 2+8 &= 10 \\
b_{03} &= 7+8 &= 15 \\
b_{04} &= 8+8 &= 16
\end{aligned}
$$

Exemple : problème d'affectation

$U = +\infty$

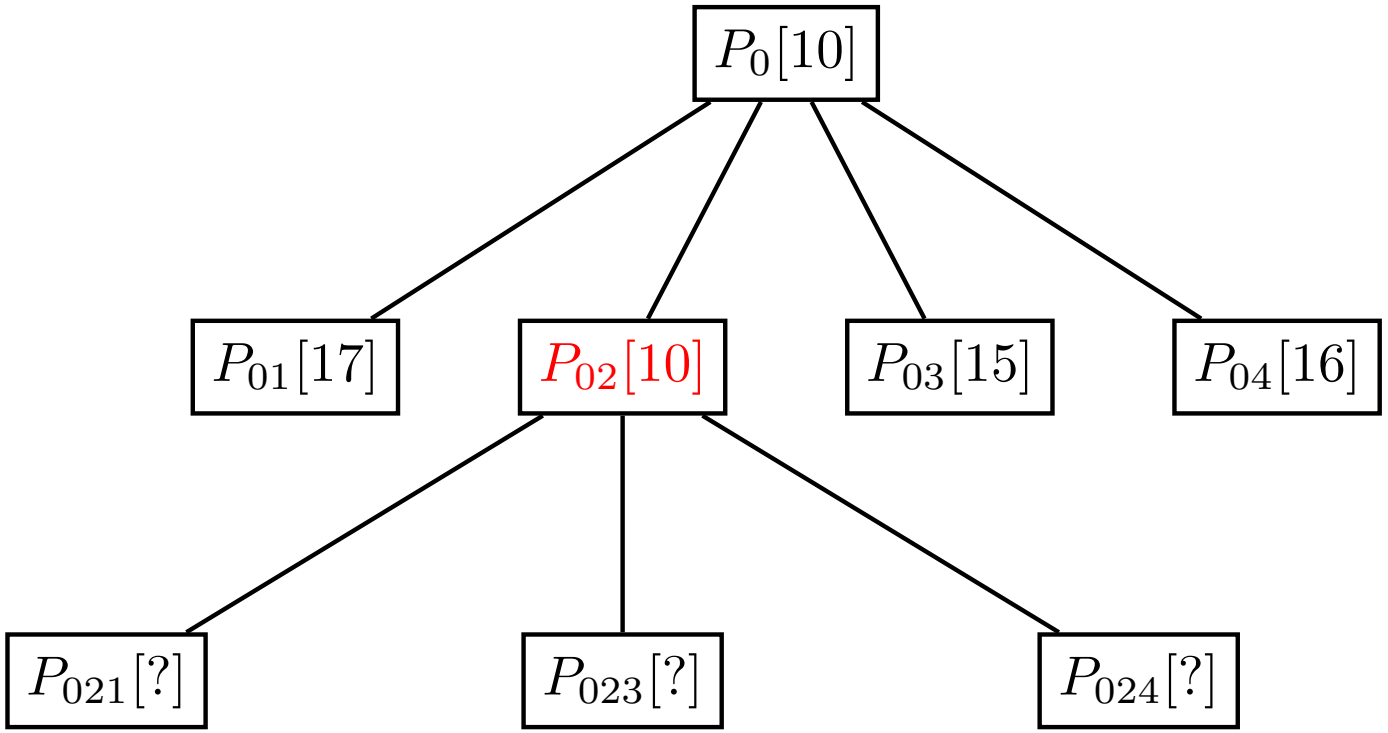


Exemple : problème d'affectation

- Problèmes actifs : $\{P_0, P_{01}, P_{02}, P_{03}, P_{04}\}$
- $U = +\infty$
- P_{02} est le plus prometteur car associé à la meilleure borne.
- On partitionne P_{02}
  - P_{021} : on décide que B effectue la tâche 1,
  - P_{022} : on décide que B effectue la tâche 2,
  - P_{023} : on décide que B effectue la tâche 3,
  - P_{024} : on décide que B effectue la tâche 4.
- P_{022} est non admissible.
- Chacun des autres problèmes revient à affecter 2 ouvriers à 2 tâches.

Exemple : problème d'affectation

$U = +\infty$



Exemple : problème d'affectation

Calcul des bornes.

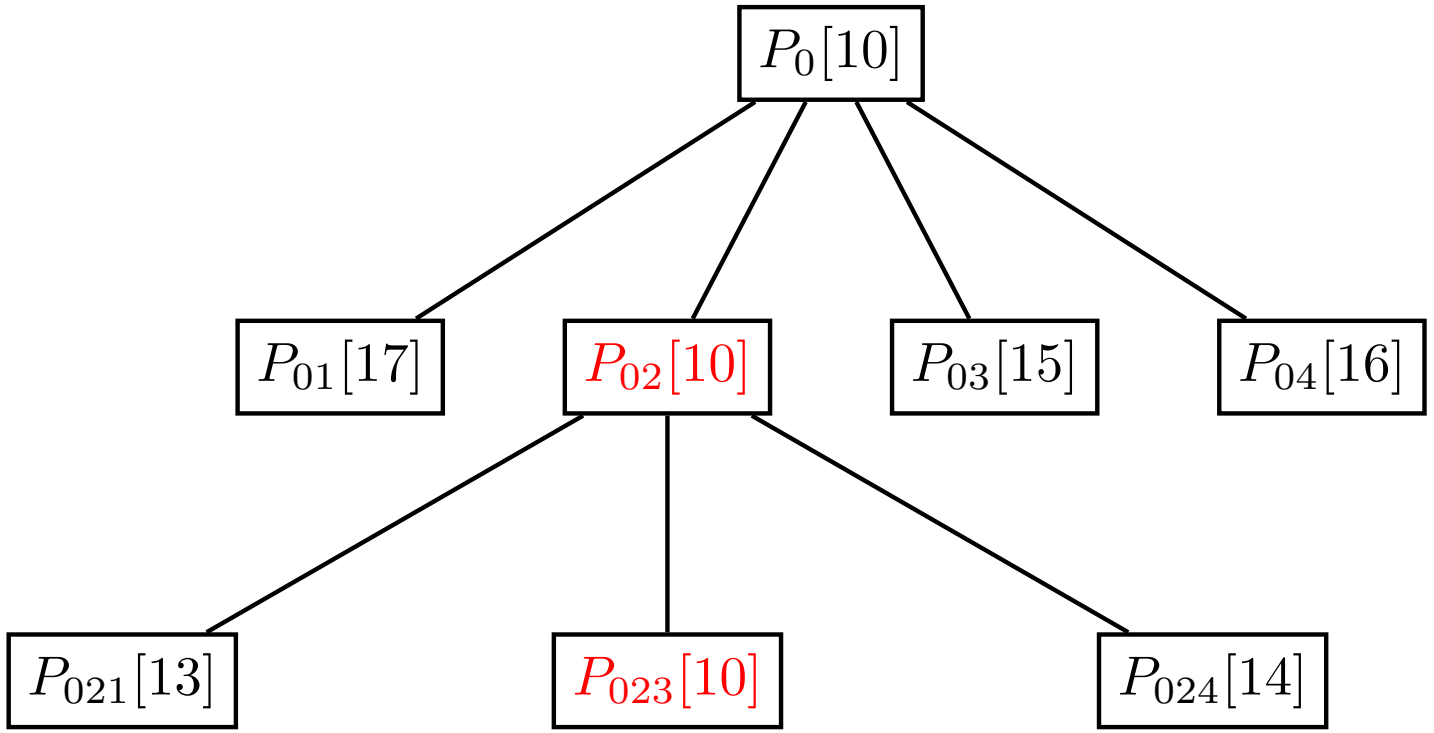
Ouvrier	Tâche 1	Tâche 2	Tâche 3	Tâche 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

- La tâche la plus rapide pour C prend 1 heure.
- La tâche la plus rapide pour D prend 4 heures.

$$
\begin{aligned}
b_{021} &= 2+6+5 = 13 \\
b_{023} &= 2+3+5 = 10 \\
b_{024} &= 2+7+5 = 14
\end{aligned}
$$

Exemple : problème d'affectation

$U = +\infty$



Exemple : problème d'affectation

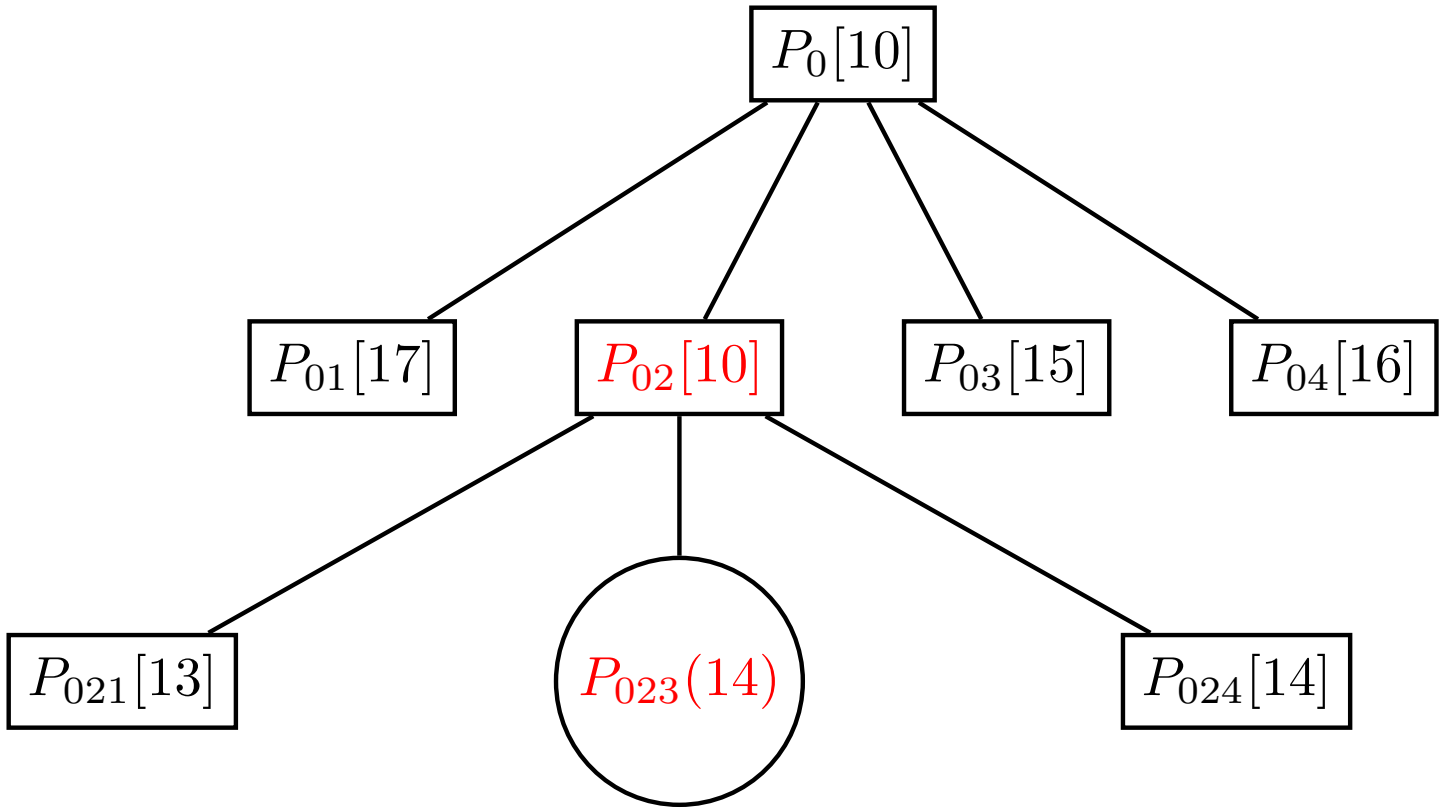
- Problèmes actifs : $\{P_0, P_{01}, P_{02}, P_{03}, P_{04}, P_{021}, P_{023}, P_{024}\}$
- $U = +\infty$
- P_{023} est le plus prometteur car associé à la meilleure borne.
  - P_{0231} : on décide que C effectue la tâche 1,
  - P_{0232} : on décide que C effectue la tâche 2,
  - P_{0233} : on décide que C effectue la tâche 3,
  - P_{0234} : on décide que C effectue la tâche 4.
- P_{0232} et P_{0233} sont non admissible.
- Chacun des autres problèmes est trivial à résoudre.
- P_{0231} : A(2), B(3), C(1), D(4).
- Temps total : 2 + 3 + 5 + 4 = 14.
- P_{0234} : A(2), B(3), C(4), D(1).
- Temps total : 2 + 3 + 8 + 7 = 20.

Exemple : problème d'affectation

- Solution optimale de P_{023} trouvée. Valeur : 14
- Problèmes actifs : $\{P_0, P_{01}, P_{02}, P_{03}, P_{04}, P_{021}, P_{024}\}$
- $U = 14$
- Notation : $P_k(f(x_k^*))$

Exemple : problème d'affectation

$U = 14$

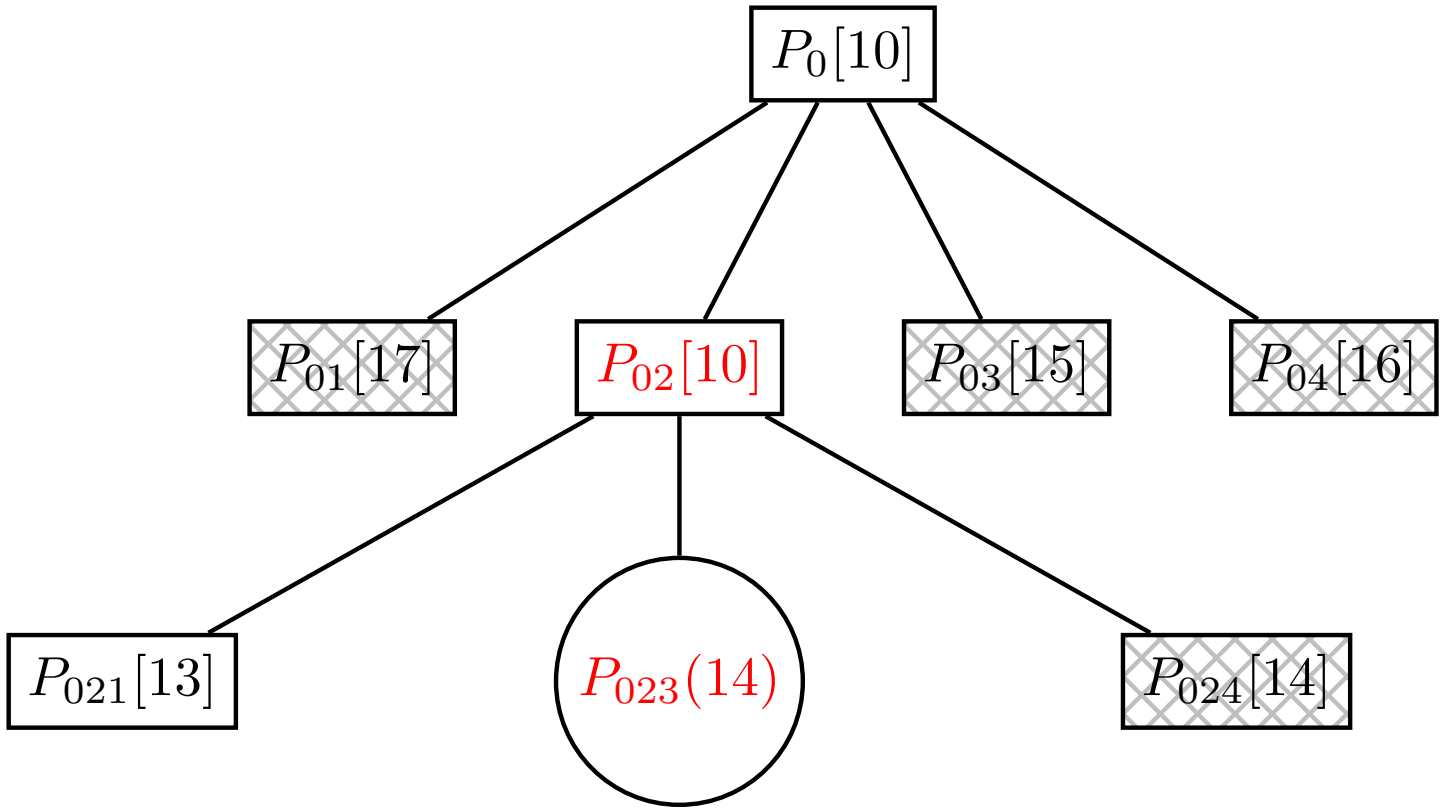


Exemple : problème d'affectation

- On peut maintenant supprimer les sous-problèmes dont la borne est plus grande ou égale à U.

Exemple : problème d'affectation

$U = 14$

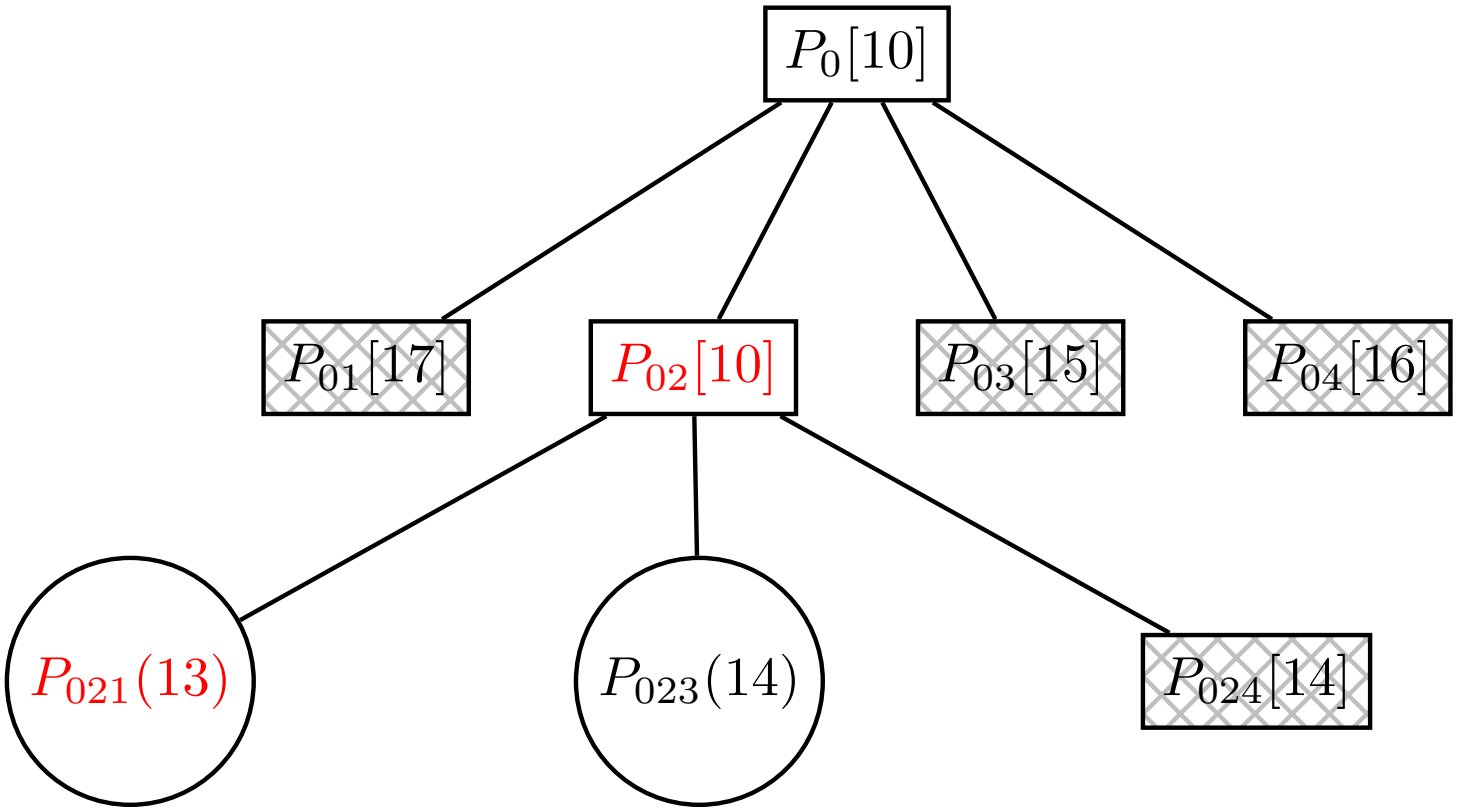


Exemple : problème d'affectation

- Problèmes actifs : $\{P_0, P_{02}, P_{021}\}$
- $U = 14$
- On partitionne P_{021}
  - P_{0211} : on décide que C effectue la tâche 1,
  - P_{0212} : on décide que C effectue la tâche 2,
  - P_{0213} : on décide que C effectue la tâche 3,
  - P_{0214} : on décide que C effectue la tâche 4.
- P_{0211} et P_{0212} sont non admissibles.
- Chacun des autres problèmes est trivial à résoudre.
- P_{0213} : A(2), B(1), C(3), D(4).
- Temps total : 2 + 6 + 1 + 4 = 13.
- P_{0214} : A(2), B(1), C(4), D(3).
- Temps total : 2 + 6 + 8 + 9 = 25.

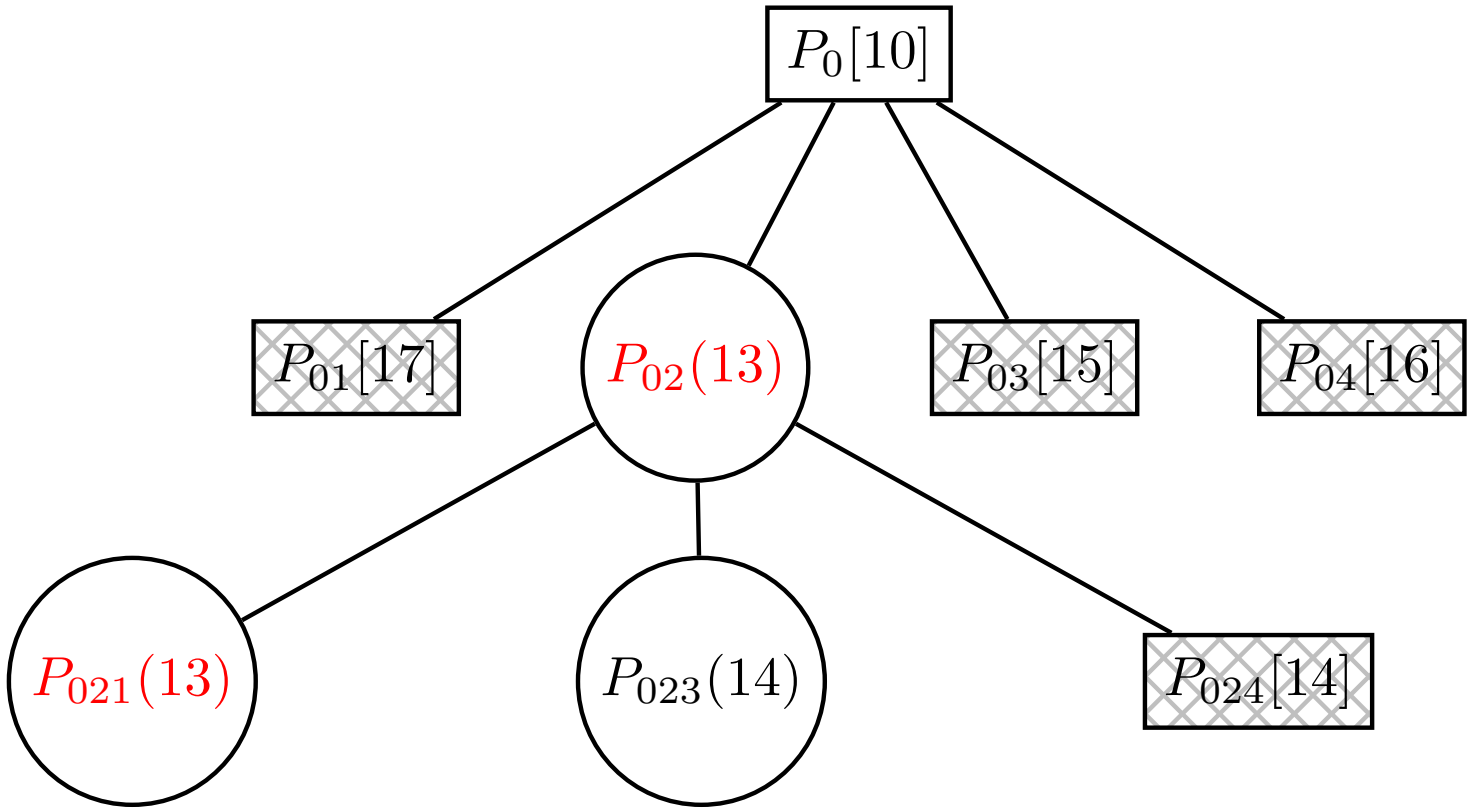
Exemple : problème d'affectation

$U = 13$



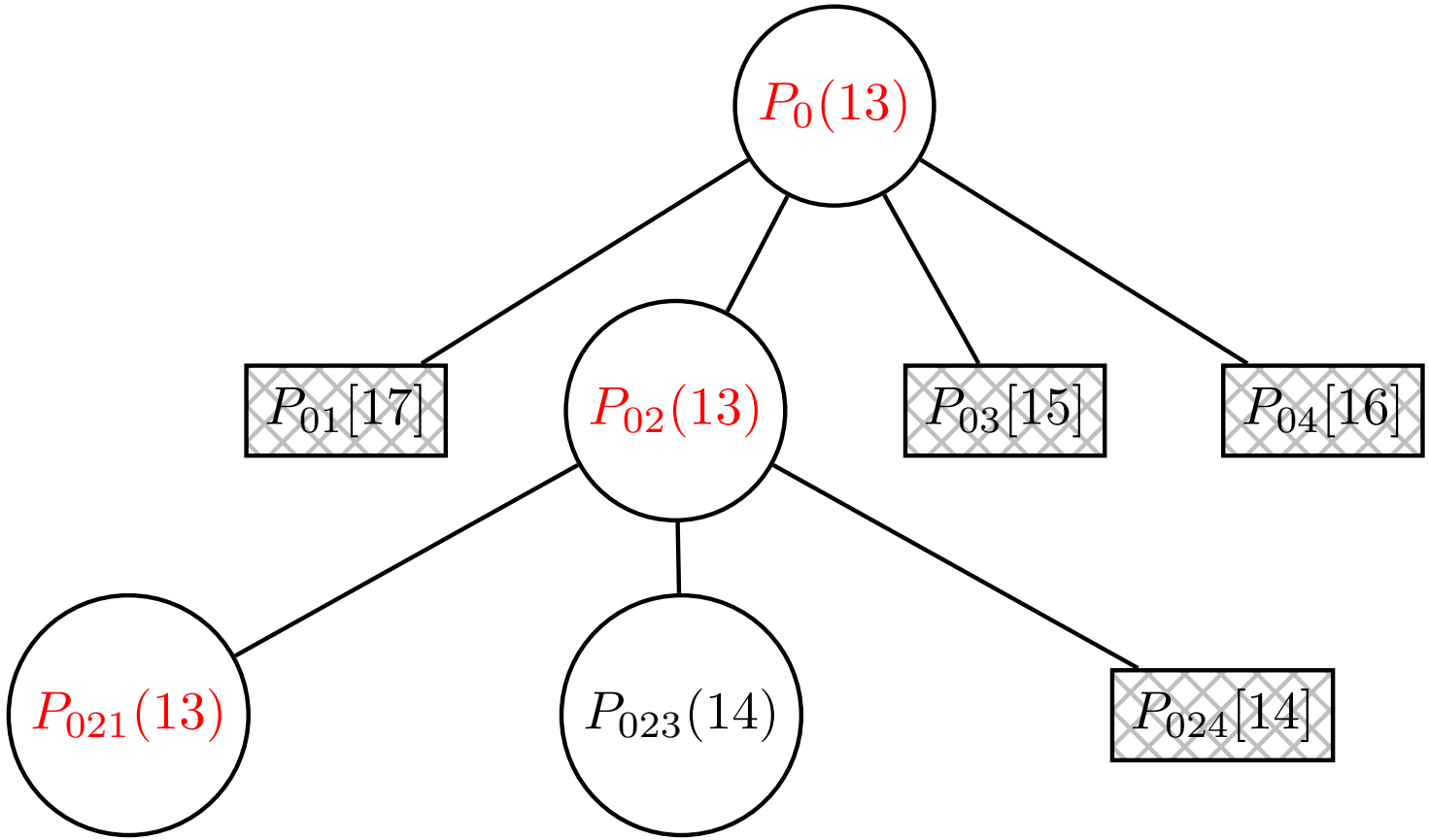
Exemple : problème d'affectation

$U = 13$



Exemple : problème d'affectation

$U = 13$



Branch & Bound

Classe d'algorithmes avec

- différentes méthodes pour partitionner,
- différentes méthodes pour choisir le sous-problème à traiter,
- différentes méthodes pour calculer les bornes.

Relaxation

Soit le problème d'optimisation P

$$\min f(x)$$

sous contraintes

$$\begin{aligned} g(x) &\leq 0 \\ h(x) &= 0 \\ x &\in \mathbb{Z} \end{aligned}$$

Le problème relaxé $R(P)$ est obtenu en ignorant les contraintes d'intégralité

$$\min f(x)$$

sous contraintes

$$\begin{aligned} g(x) &\leq 0 \\ h(x) &= 0 \end{aligned}$$

Branch

- Soit x_R^* la solution optimale de $R(P)$.
- Si toutes les composantes sont entières, alors x_R^* est aussi solution optimale de P.
- Sinon, il existe au moins une composante i non entière $(x_R^*)_i$.
- Le problème P est alors partitionné :

P_ℓ	P_r
$\min f(x)$	$\min f(x)$
sous contraintes	sous contraintes
$g(x) \leq 0$	$g(x) \leq 0$
$h(x) = 0$	$h(x) = 0$
$x \in \mathbb{Z}$	$x \in \mathbb{Z}$
$x_i \leq \lfloor (x_R^*)_i \rfloor$	$x_i \geq \lceil (x_R^*)_i \rceil$

Branch

- Toute solution admissible de P est solution admissible soit de P_ℓ, soit de P_r. Il s'agit donc bien d'une partition de l'ensemble admissible.
- La solution x_R^* n'est pas admissible pour les relaxations des nouveaux sous-problèmes $R(P_\ell)$ et $R(P_r)$.
- En effet, comme $(x_R^*)_i$ est non entier, les contraintes

$$x_i \leq \lfloor (x_R^*)_i \rfloor \text{ et } x_i \geq \lceil (x_R^*)_i \rceil$$

  sont violées par $(x_R^*)_i$.
- La solution optimale des problèmes relaxés sera donc différente de x_R^*.

Bound

- Soit x^* la solution optimale de P.
- Soit x_R^* la solution optimale de $R(P)$.
- On a toujours
$$f(x_R^*) \leq f(x^*).$$
- On obtient donc une borne inférieure en résolvant le problème relaxé.
- Attention : cela ne fonctionne que si on peut trouver l'optimum global de $R(P)$.
- C'est le cas en particulier si le problème d'optimisation est linéaire.

Exemple

Soit le problème P_0

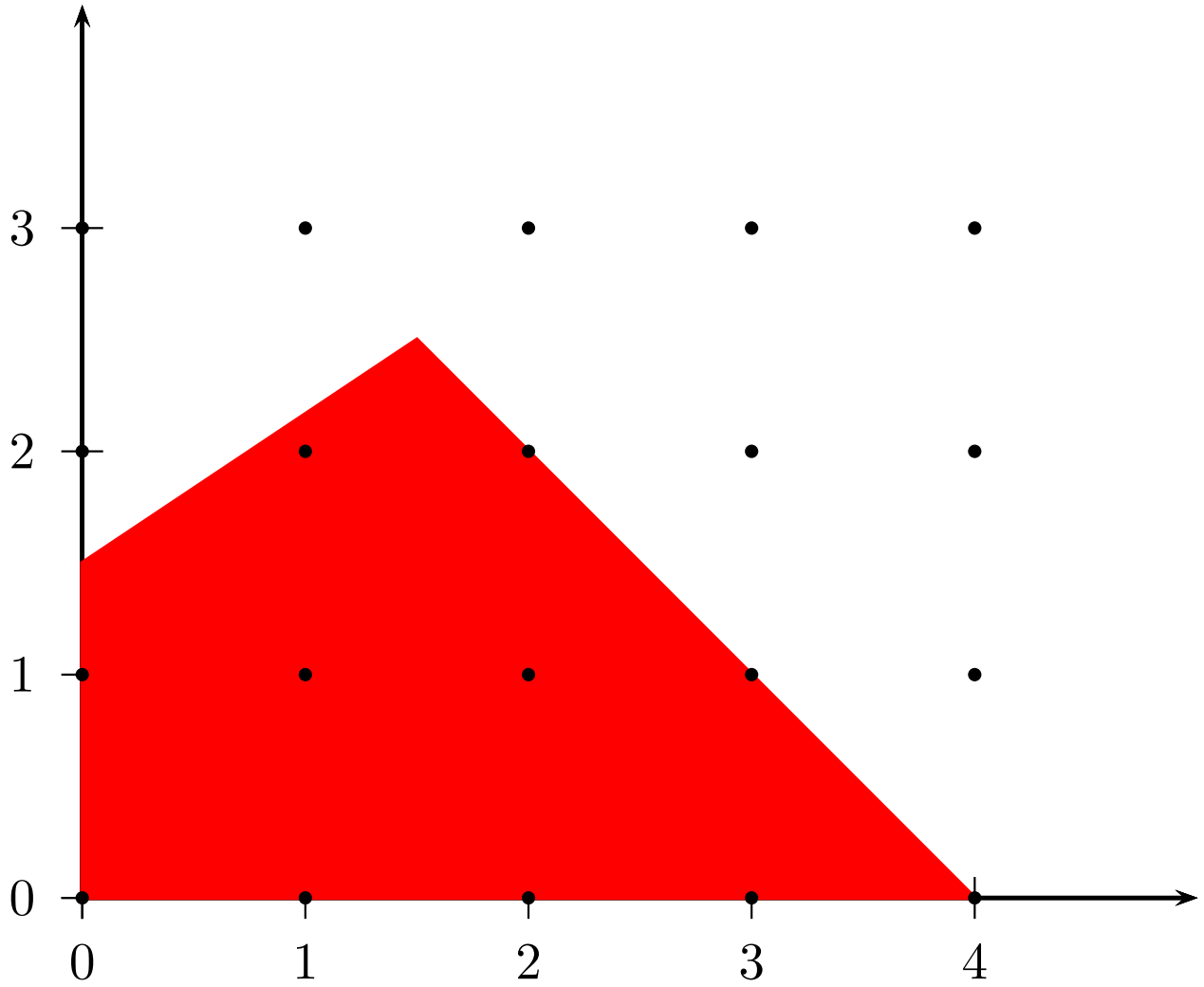
$$\min x_1 - 2x_2$$

sous contraintes

$$\begin{aligned} -4x_1 + 6x_2 &\leq 9 \\ x_1 + x_2 &\leq 4 \\ x_1, x_2 &\geq 0 \\ x_1, x_2 &\in \mathbb{N}. \end{aligned}$$

Note : $(0, 0)$ est admissible. Donc $U = 0$.

Exemple



Exemple

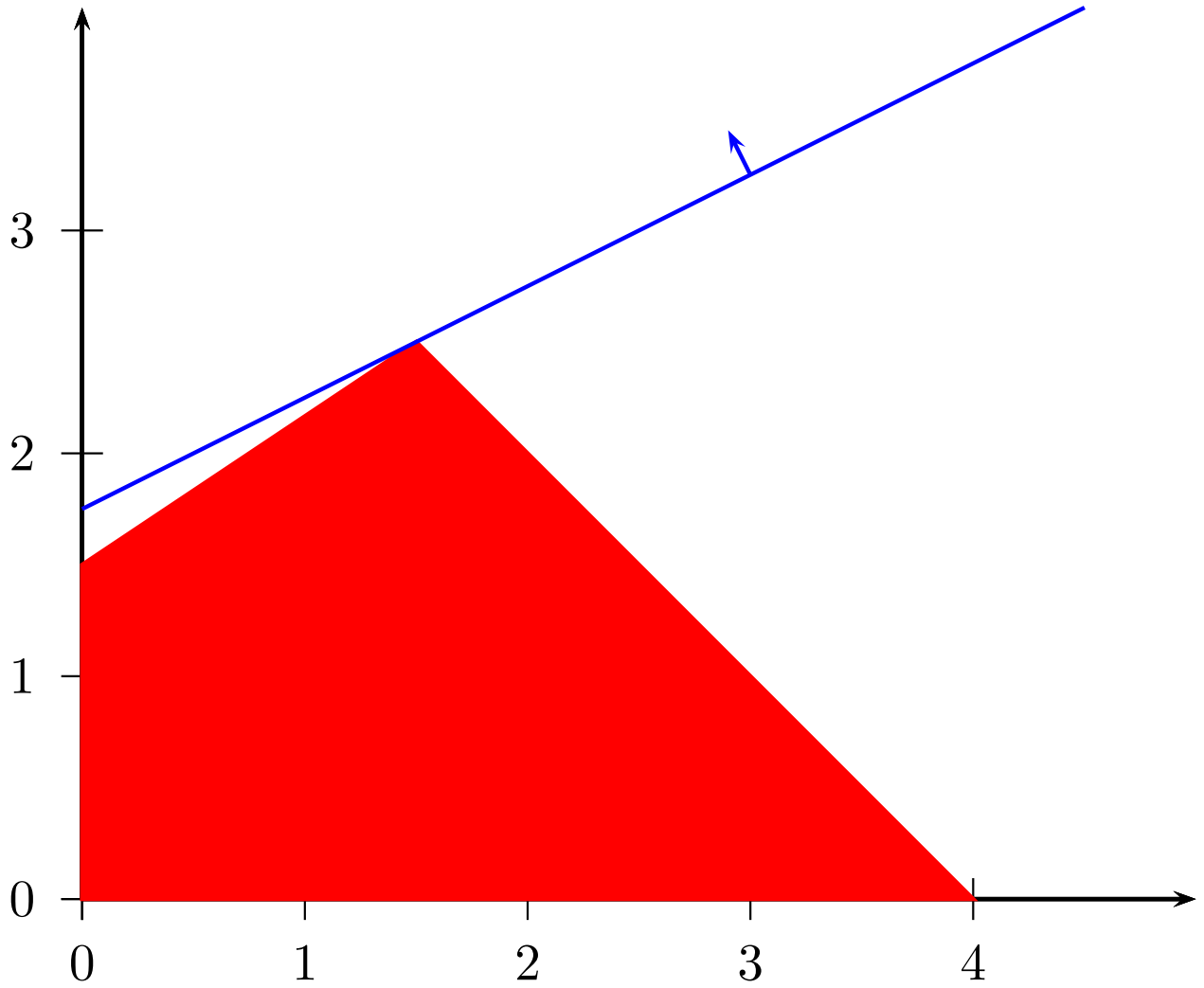
Problème relaxé $R(P_0)$.

$$\min x_1 - 2x_2$$

sous contraintes

$$\begin{aligned} -4x_1 + 6x_2 &\leq 9 \\ x_1 + x_2 &\leq 4 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Exemple



TRANSP-OR

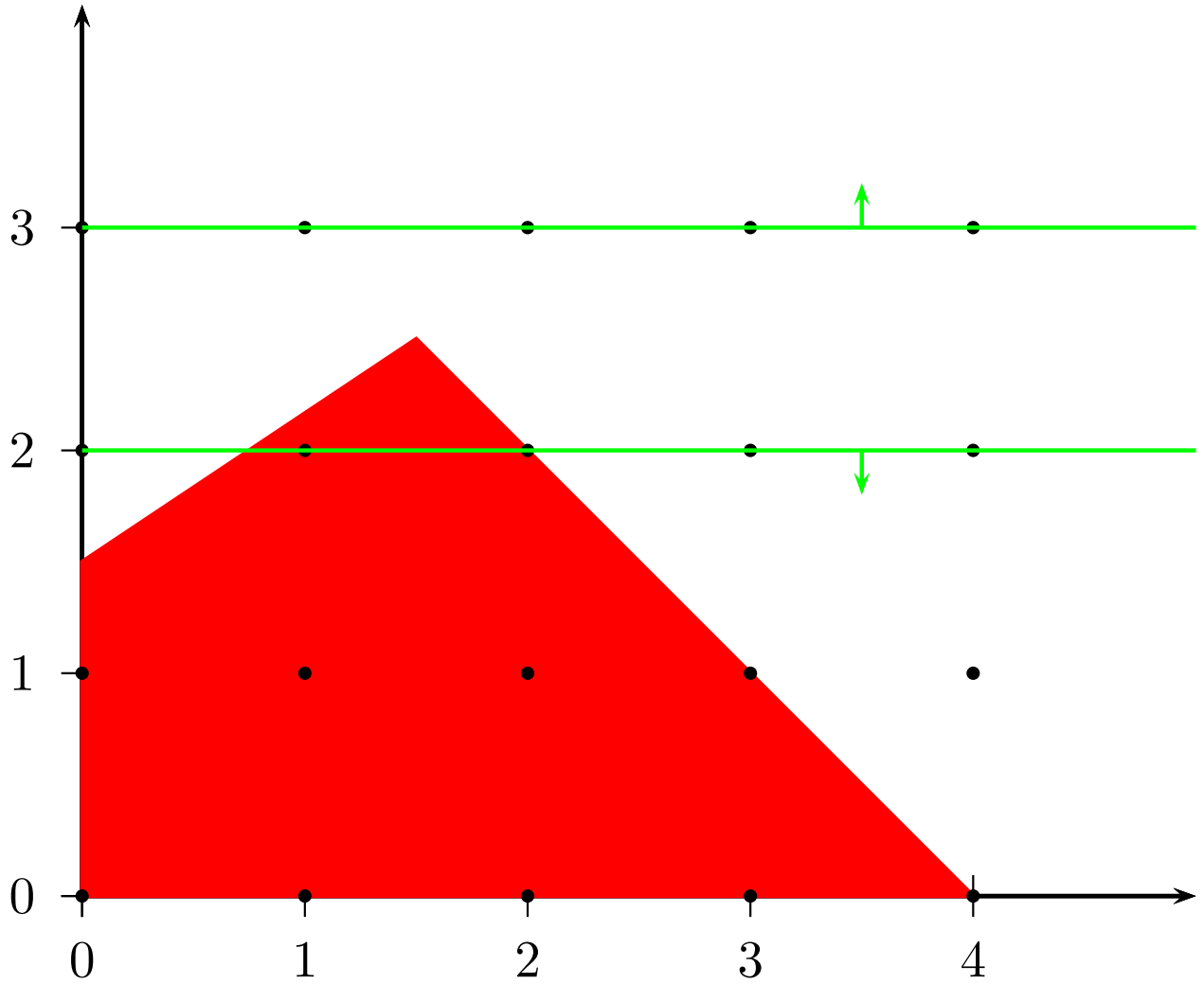
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Exemple

- Solution optimale de $R(P_0)$: $(1.5, 2.5)$
- Borne pour P_0 : $b_0 = -3.5$
- $x_2 = 2.5$ est non entier. Partition :

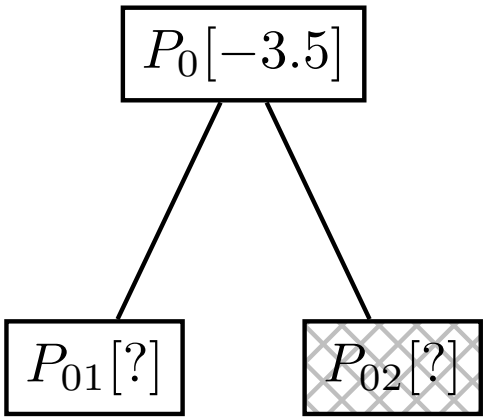
P_{01}			P_{02}		
$\min x_1 - 2x_2$			$\min x_1 - 2x_2$		
s.c.			s.c.		
$-4x_1 + 6x_2$	$\leq$	9	$-4x_1 + 6x_2$	$\leq$	9
$x_1 + x_2$	$\leq$	4	$x_1 + x_2$	$\leq$	4
x_1, x_2	$\geq$	0	x_1, x_2	$\geq$	0
x_1, x_2	$\in$	$\mathbb{N}$	x_1, x_2	$\in$	$\mathbb{N}$
x_2	$\leq$	2	x_2	$\geq$	3

Exemple



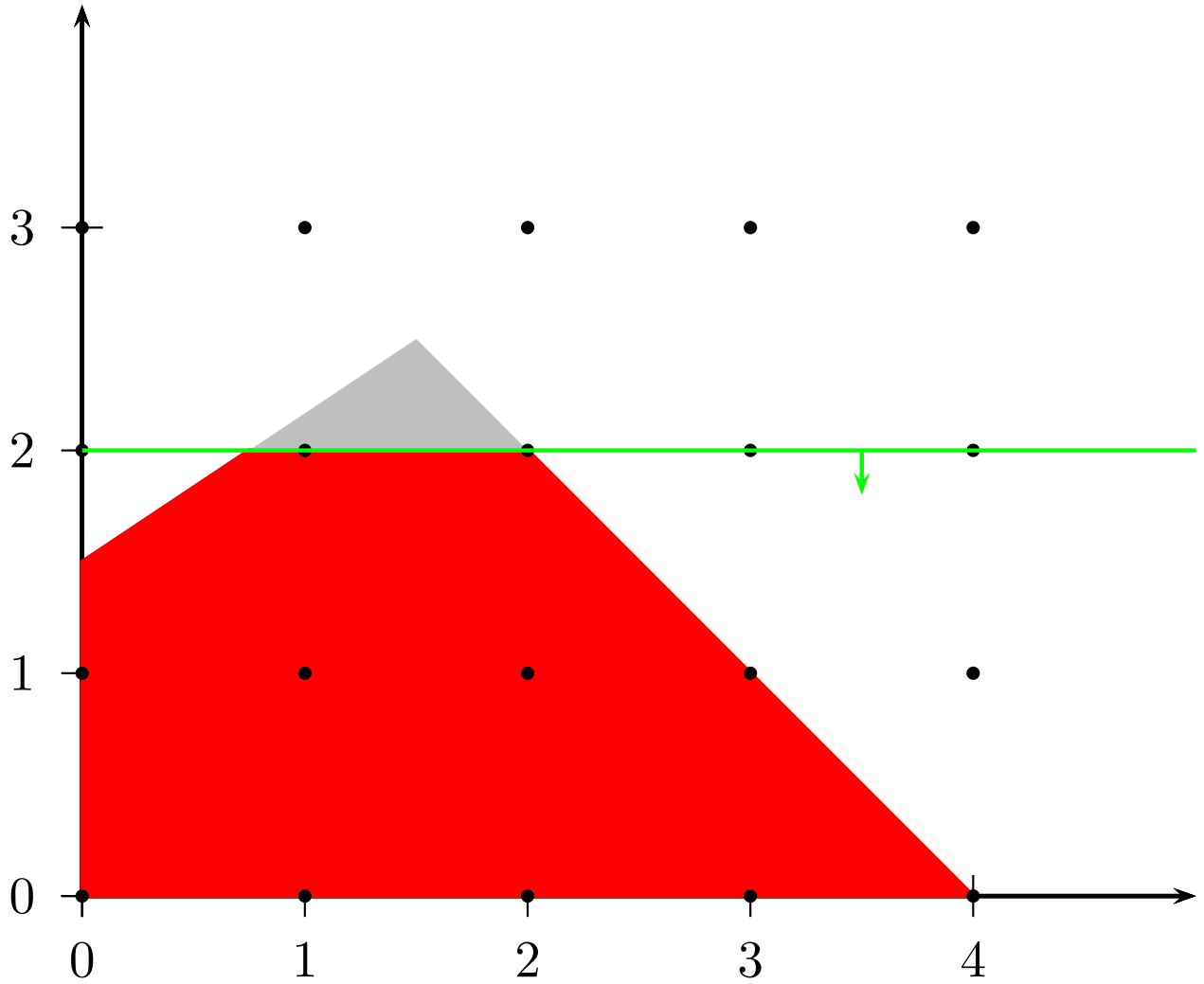
Exemple

$U = 0$



P_{02} est non admissible.

Exemple



Exemple

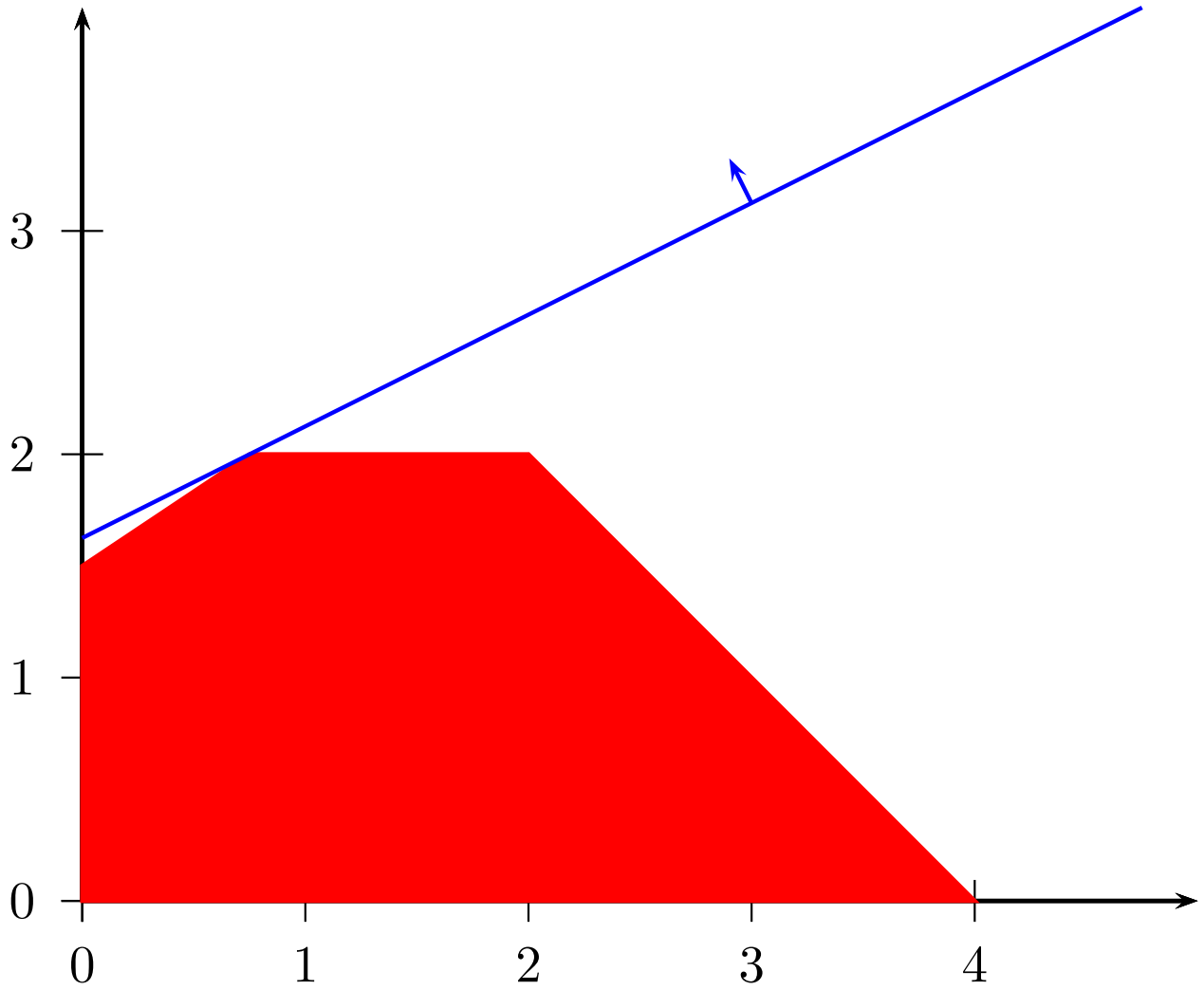
- Problème P_{01} : $\min x_1 - 2x_2$ sous contraintes

$$\begin{aligned} -4x_1 + 6x_2 &\leq 9 \\ x_1 + x_2 &\leq 4 \\ x_1, x_2 &\geq 0 \\ x_2 &\leq 2 \\ x_1, x_2 &\in \mathbb{N}. \end{aligned}$$

- Problème relaxé $R(P_{01})$: $\min x_1 - 2x_2$ sous contraintes

$$\begin{aligned} -4x_1 + 6x_2 &\leq 9 \\ x_1 + x_2 &\leq 4 \\ x_1, x_2 &\geq 0 \\ x_2 &\leq 2 \end{aligned}$$

Exemple



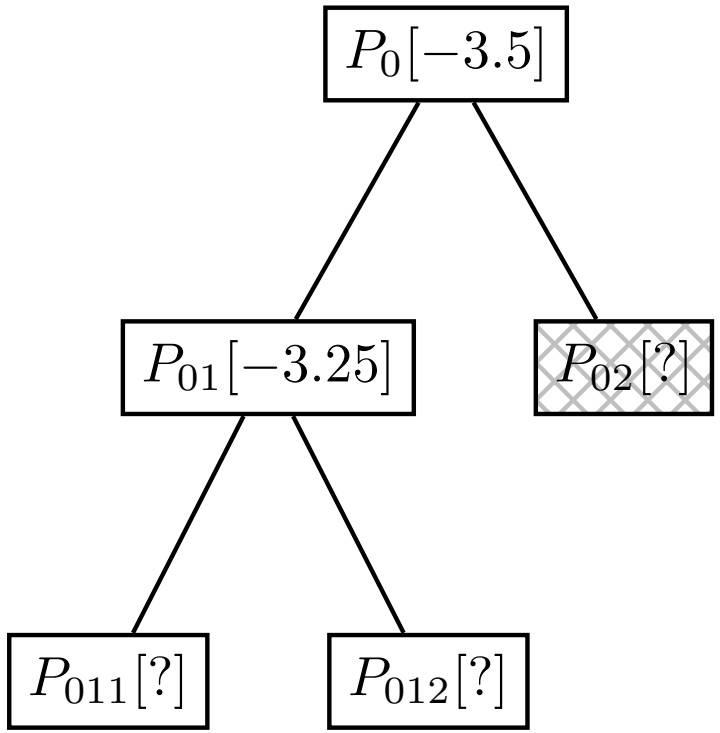
Exemple

- Solution optimale de $R(P_{01})$: $(0.75, 2)$
- Borne pour P_{01} : $b_{01} = -3.25$
- $x_1 = 0.75$ est non entier. Partition :

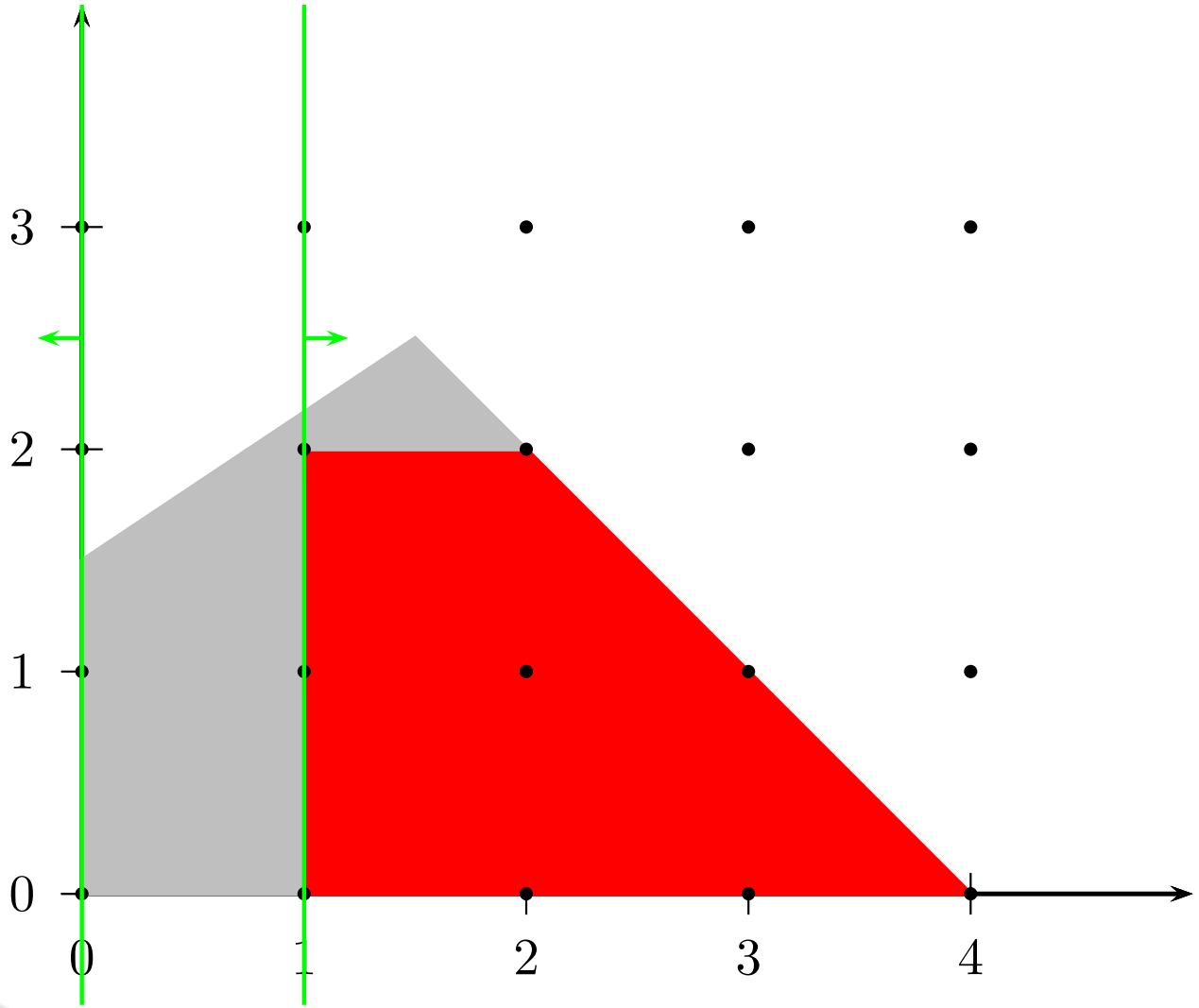
P_{011}			P_{012}		
$\min x_1 - 2x_2$			$\min x_1 - 2x_2$		
s.c.			s.c.		
$-4x_1 + 6x_2$	$\leq$	9	$-4x_1 + 6x_2$	$\leq$	9
$x_1 + x_2$	$\leq$	4	$x_1 + x_2$	$\leq$	4
x_1, x_2	$\geq$	0	x_1, x_2	$\geq$	0
x_1, x_2	$\in$	$\mathbb{N}$	x_1, x_2	$\in$	$\mathbb{N}$
x_2	$\leq$	2	x_2	$\leq$	2
x_1	$\leq$	0	x_1	$\geq$	1

Exemple

$U = 0$



Exemple



Exemple

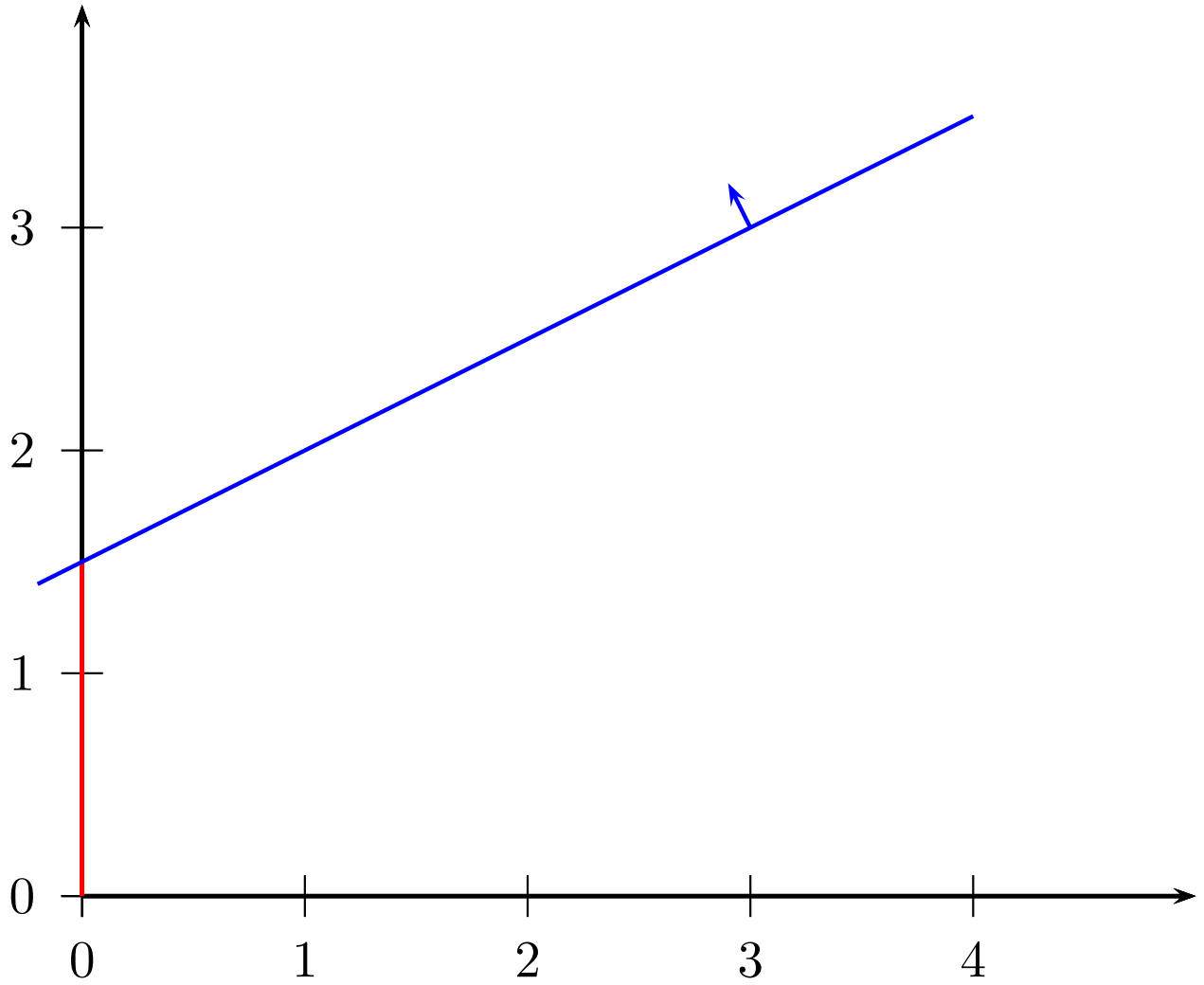
- Problème P_{011} : $\min x_1 - 2x_2$ sous contraintes

$$\begin{aligned} -4x_1 + 6x_2 &\leq 9 \\ x_1 + x_2 &\leq 4 \\ x_1, x_2 &\geq 0 \\ x_2 &\leq 2 \\ x_1 &\leq 0 \\ x_1, x_2 &\in \mathbb{N}. \end{aligned}$$

- Problème relaxé $R(P_{01})$: $\min x_1 - 2x_2$ sous contraintes

$$\begin{aligned} -4x_1 + 6x_2 &\leq 9 \\ x_1 + x_2 &\leq 4 \\ x_1, x_2 &\geq 0 \\ x_2 &\leq 2 \\ x_1 &\leq 0 \end{aligned}$$

Exemple



Exemple

- Solution optimale de $R(P_{011})$: $(0, 1.5)$
- Borne pour P_{011} : $b_{011} = -3$

Exemple

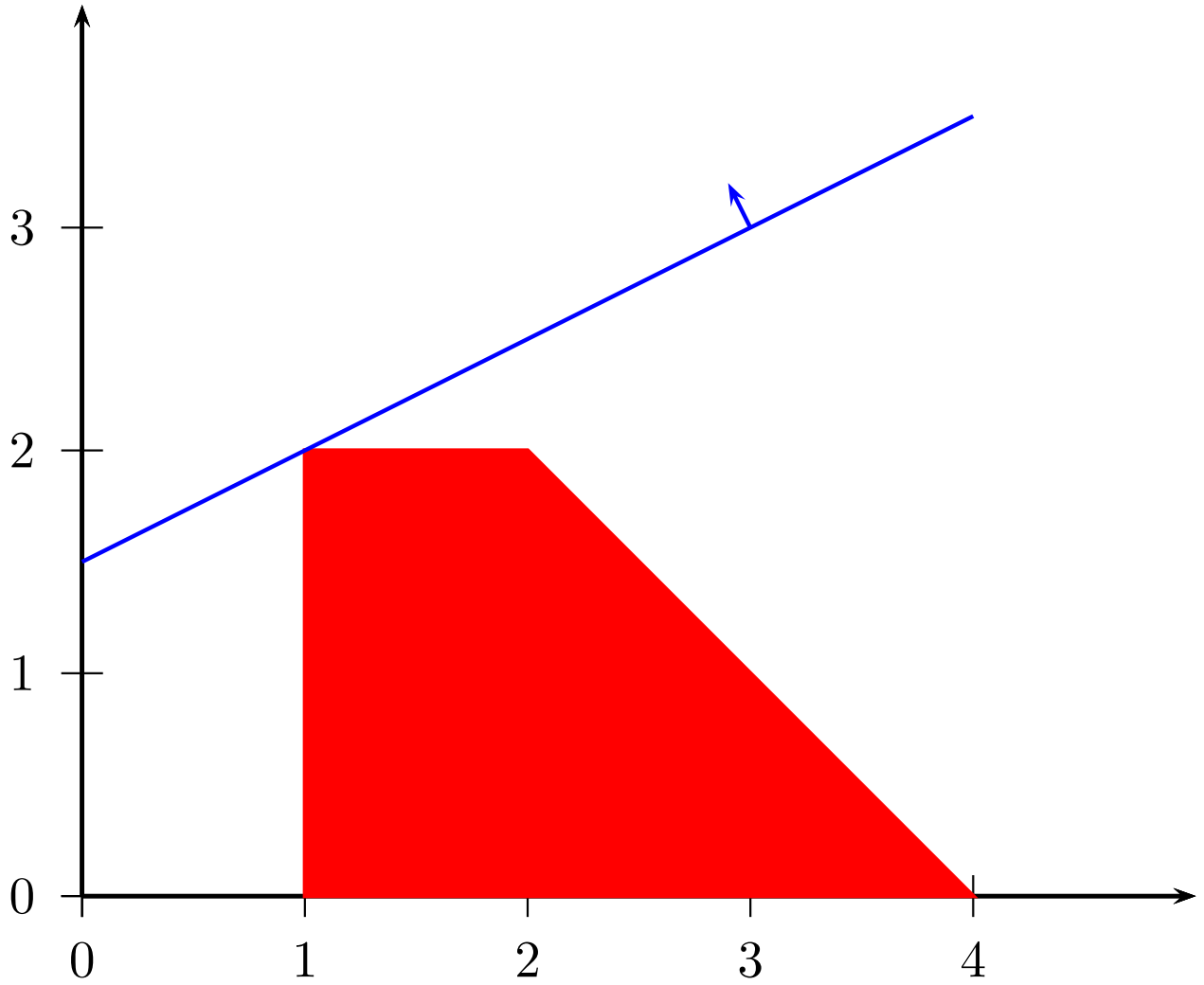
- Problème P_{012} : $\min x_1 - 2x_2$ sous contraintes

$$\begin{aligned} -4x_1 + 6x_2 &\leq 9 \\ x_1 + x_2 &\leq 4 \\ x_1, x_2 &\geq 0 \\ x_2 &\leq 2 \\ x_1 &\geq 1 \\ x_1, x_2 &\in \mathbb{N}. \end{aligned}$$

- Problème relaxé $R(P_{01})$: $\min x_1 - 2x_2$ sous contraintes

$$\begin{aligned} -4x_1 + 6x_2 &\leq 9 \\ x_1 + x_2 &\leq 4 \\ x_1, x_2 &\geq 0 \\ x_2 &\leq 2 \\ x_1 &\geq 1 \end{aligned}$$

Exemple

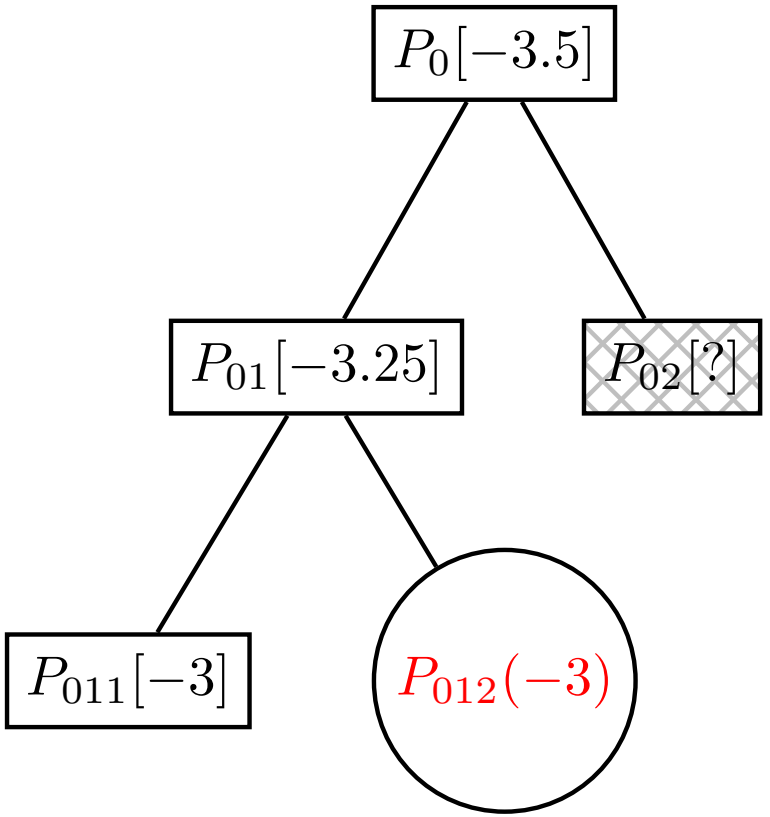


Exemple

- Solution optimale de $R(P_{012})$: $(1, 2)$
- Solution entière.
- C'est donc la solution optimale pour P_{012}.
- $U = -3$.

Exemple

$U = -3$

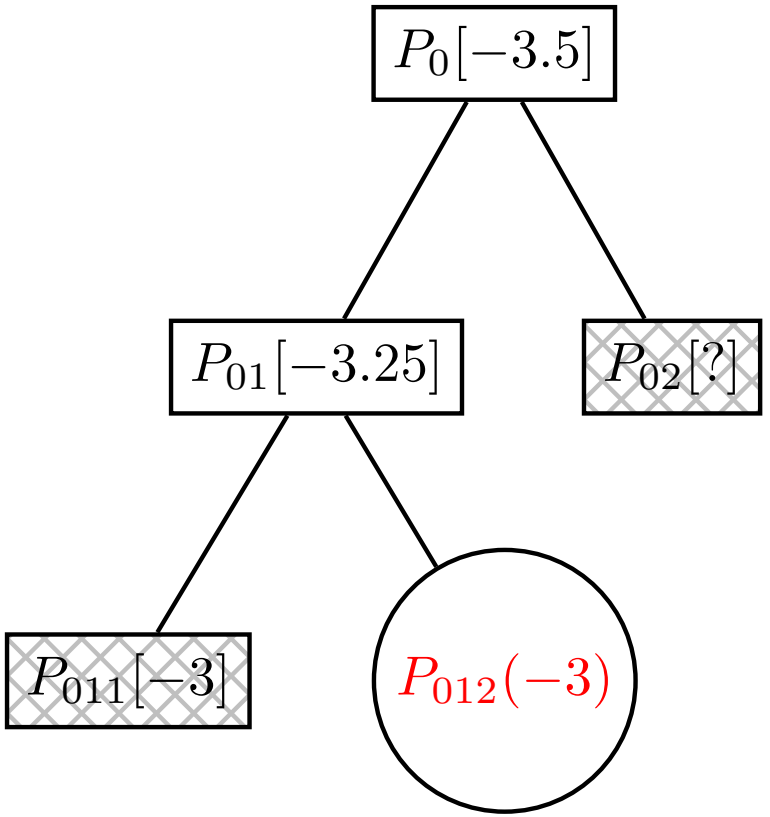


Exemple

- U a été modifié.
- Les sous-problèmes dont la borne est plus grande ou égale à U peuvent être supprimés.

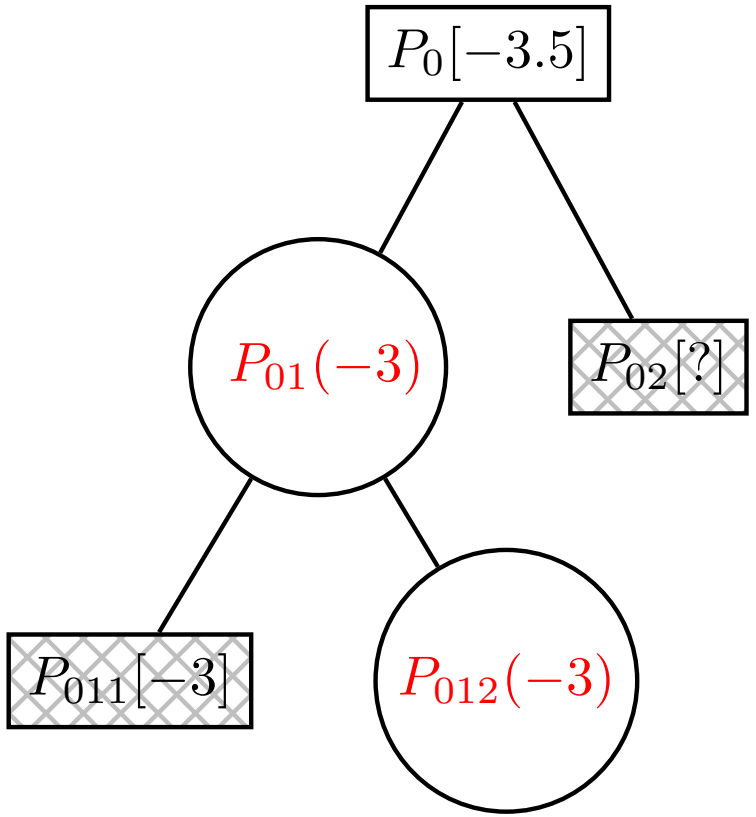
Exemple

$U = -3$



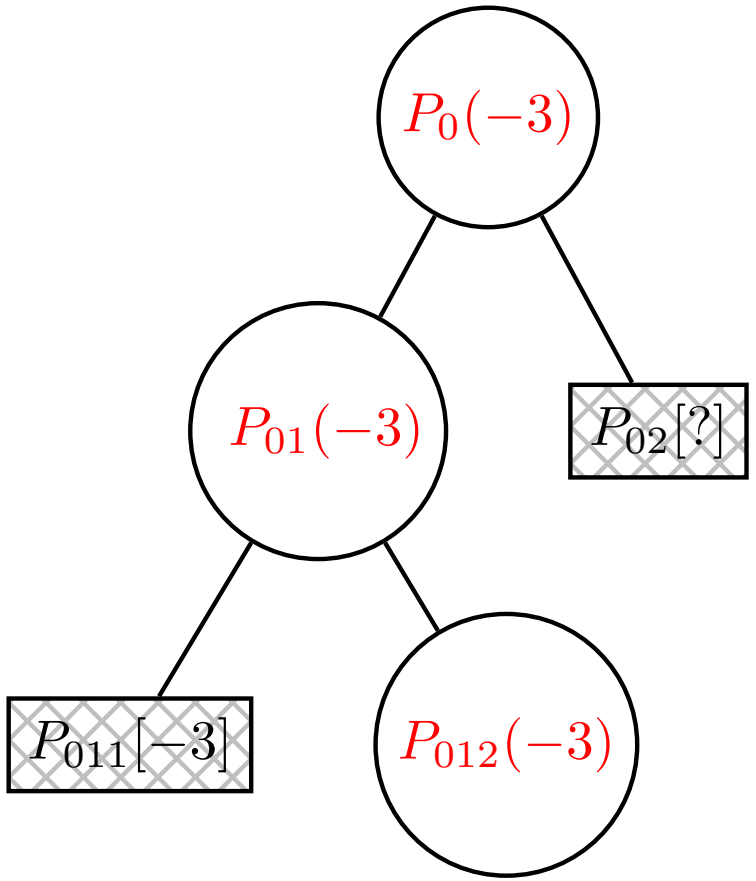
Exemple

$U = -3$



Exemple

$U = -3$



Résumé

- Optimisation en nombres entiers = problème difficile.
- Utiliser l'algorithme du simplexe et arrondir les solutions ne fonctionne en général pas.
- Méthode exacte : branch & bound
- Branch :
  - Diviser pour conquérir.
  - Partitionner l'ensemble admissible.
  - On obtient une série de problèmes plus simples.
- Bound :
  - Calculer une borne inférieure pour un problème avant de le résoudre.
  - Si cette borne est moins bonne que la meilleure solution trouvée jusque là, pas besoin de résoudre le problème.

Résumé

- Utilisation de la relaxation. C'est un problème continu.
  - Branch : éliminer les solutions non entières = "couper" le polytope.
  - Bound : solution du problème relaxé = borne pour le problème non relaxé.
- Les variantes sont nombreuses.
- Exemple : utilisation de la dualité (relaxation lagrangienne).

Heuristiques

- Certains problèmes sont trop complexes pour être résolus exactement.
- Cependant, il sont importants en pratiques.
- On utilise alors une *méthode heuristique*.

Méthode heustique

Une méthode heuristique est une procédure visant à résoudre un problème d'optimisation complexe en utilisant une approche intuitive dans laquelle la structure du problème peu être interprétée et exploitée intelligemment pour obtenir une solution raisonnable.

Heuristiques : concepts principaux

- Solution initiale
- Voisinages
- Recherche locale
- Diversification

Heuristiques : exemples

- Optimisation en nombres entiers
- Le problème du sac à dos
- Le problème du voyageur de commerce

Heuristiques : optimisation en nombres entiers

$$\min_{x \in \mathbb{R}^n} c^T x$$

sous contraintes

$$Ax = b$$

$$x \geq 0$$

$$x \in \mathbb{N}$$

- Solution initiale
- Voisinages
- Recherche locale
- Diversification

TRANSP-OR

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Optimisation en nombres entiers : voisinage

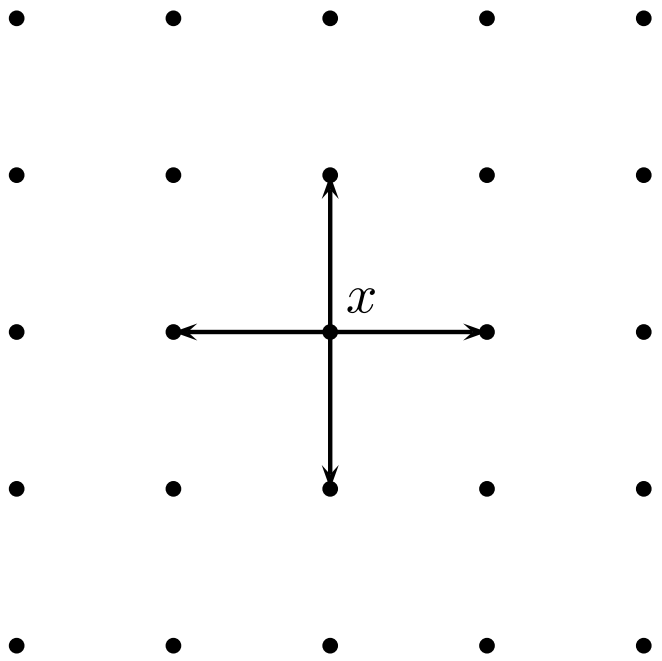
- Considérons un problème d'optimisation avec des variables entières.
- Soit un vecteur $x \in \mathbb{Z}^n$.
- Pour chaque indice k, on définit 2 voisins en augmentant et en diminuant de 1 la valeur de x_k.
- Les voisins y^{k+} et y^{k-} sont définis par

$$y_i^{k+} = y_i^{k-} = x_i, \forall i \neq k, \quad y_k^{k+} = x_k + 1, \quad y_k^{k+} = x_k - 1.$$

- Exemple

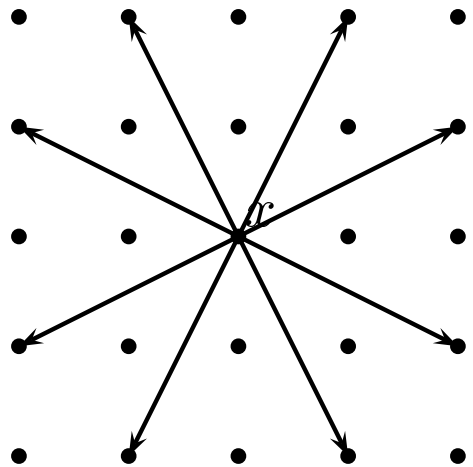
$$x = (3, 5, 2, 8) \quad y_2^+ = (3, 6, 2, 8) \quad y_2^- = (3, 4, 2, 8)$$

Optimisation en nombres entiers : voisinage



Optimisation en nombres entiers : voisinage

- La notion de voisinage est très générale.
- Le voisinage doit être défini en fonction de la structure du problème.
- Il faut être créatif.



Recherche locale

- Soit le problème d'optimisation

$$\min_{x \in \mathbb{Z}^n} f(x)$$

sous contraintes

$$x \in \mathcal{F}.$$

- Soit la structure de voisinage $V(x)$, ou $V(x)$ est l'ensemble des voisins de x.
- Soit $x_0 \in \mathcal{F}$ un point admissible. $k = 0$.
- Itération k:
  - Pour chaque $y \in V(x_k)$, si $y \in \mathcal{F}$ et $f(y) < f(x_k)$, alors $x_{k+1} = y$.
  - Si aucun voisin ne vérifie ces deux critères, on arrête.

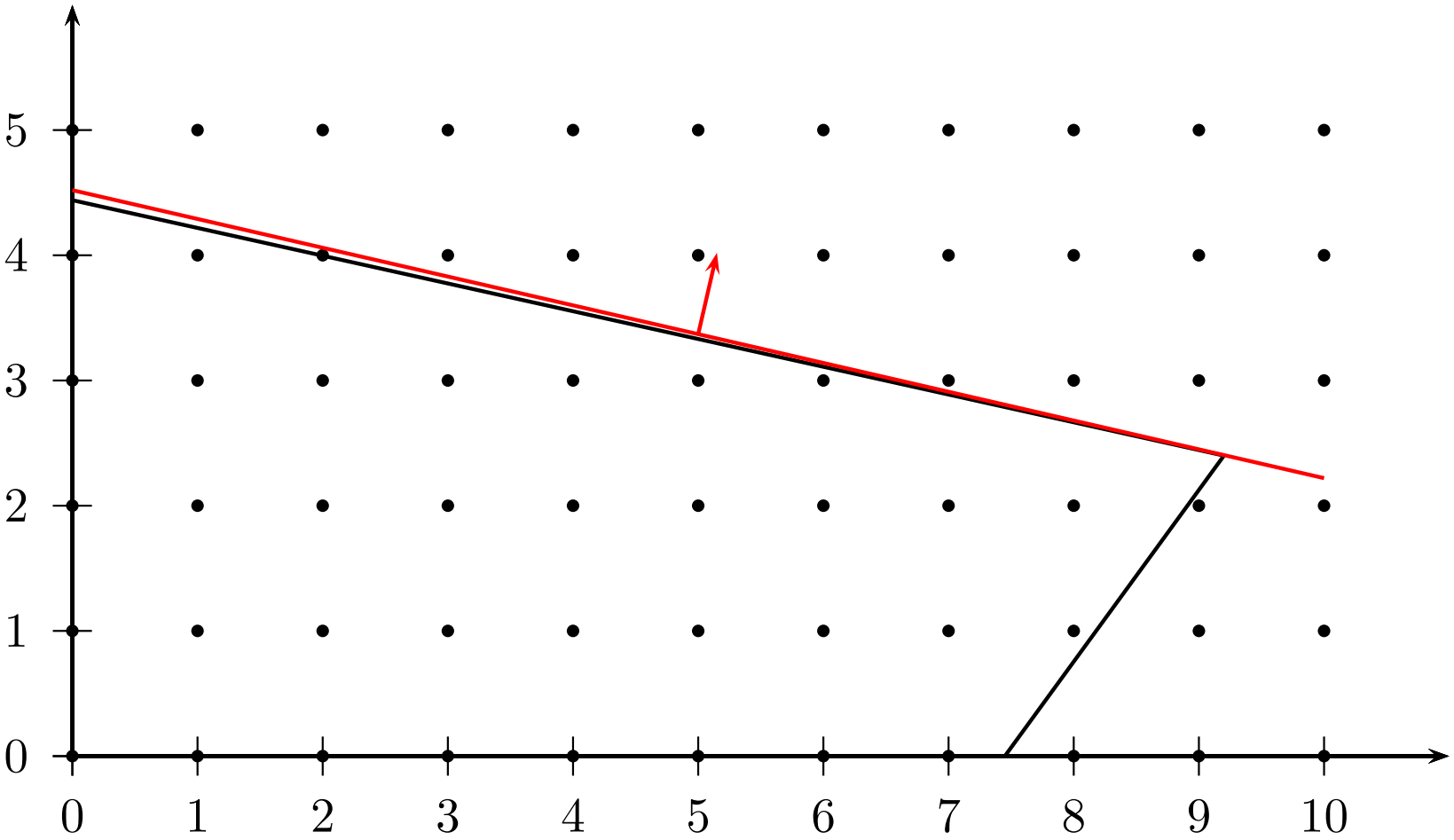
Recherche locale : exemple

$$\min_{x \in \mathbb{R}^2} -3x_1 - 13x_2$$

sous contraintes

$$\begin{aligned} 2x_1 + 9x_2 &\leq 40 \\ 11x_1 - 8x_2 &\leq 82 \\ x_1, x_2 &\geq 0 \\ x_1, x_2 &\text{entiers} \end{aligned}$$

Recherche locale : exemple

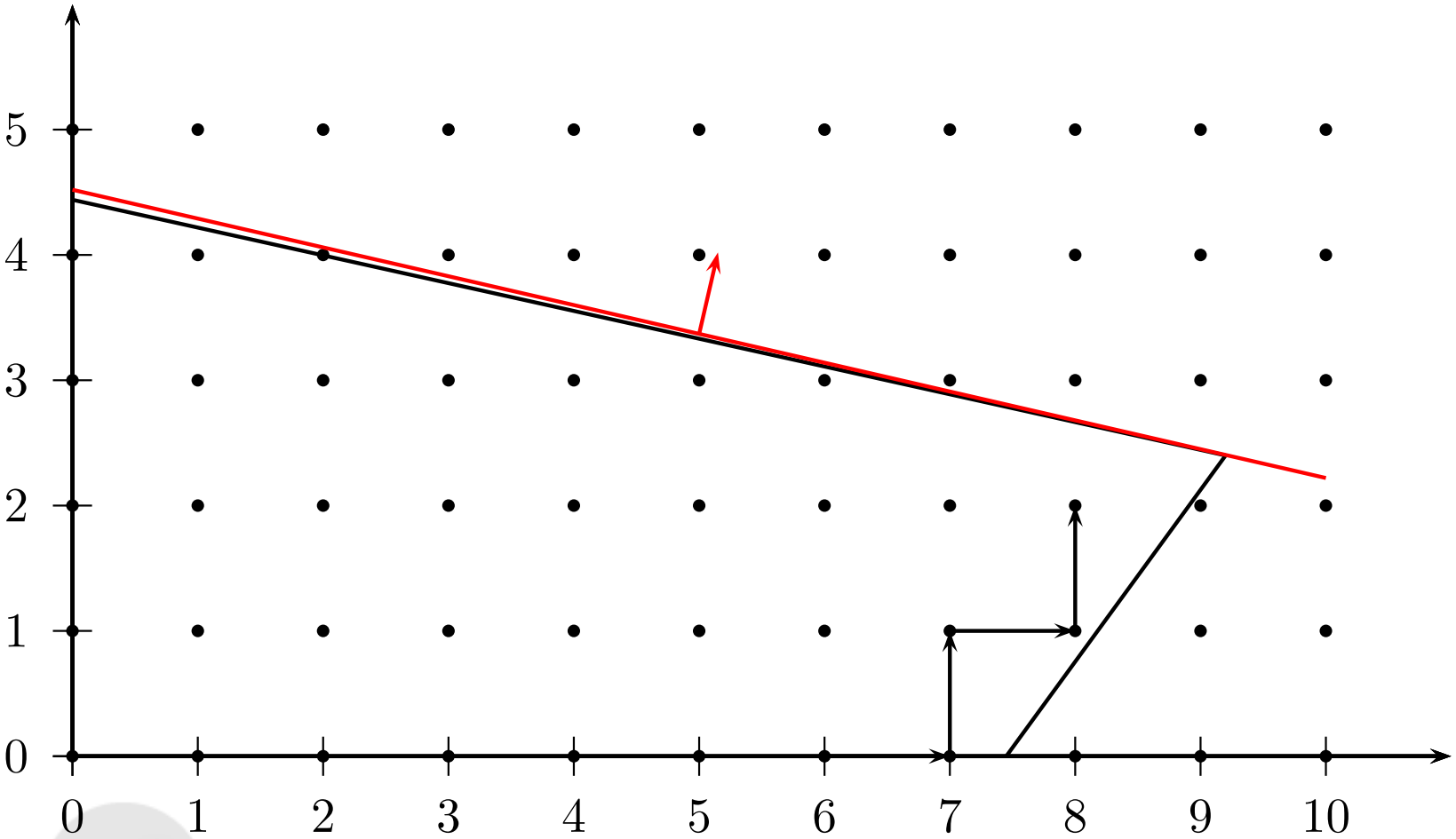


TRANSP-OR

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Recherche locale : exemple

$x_0 = (6, 0)$ - Voisinage : E - N - O - S



Recherche locale : exemple

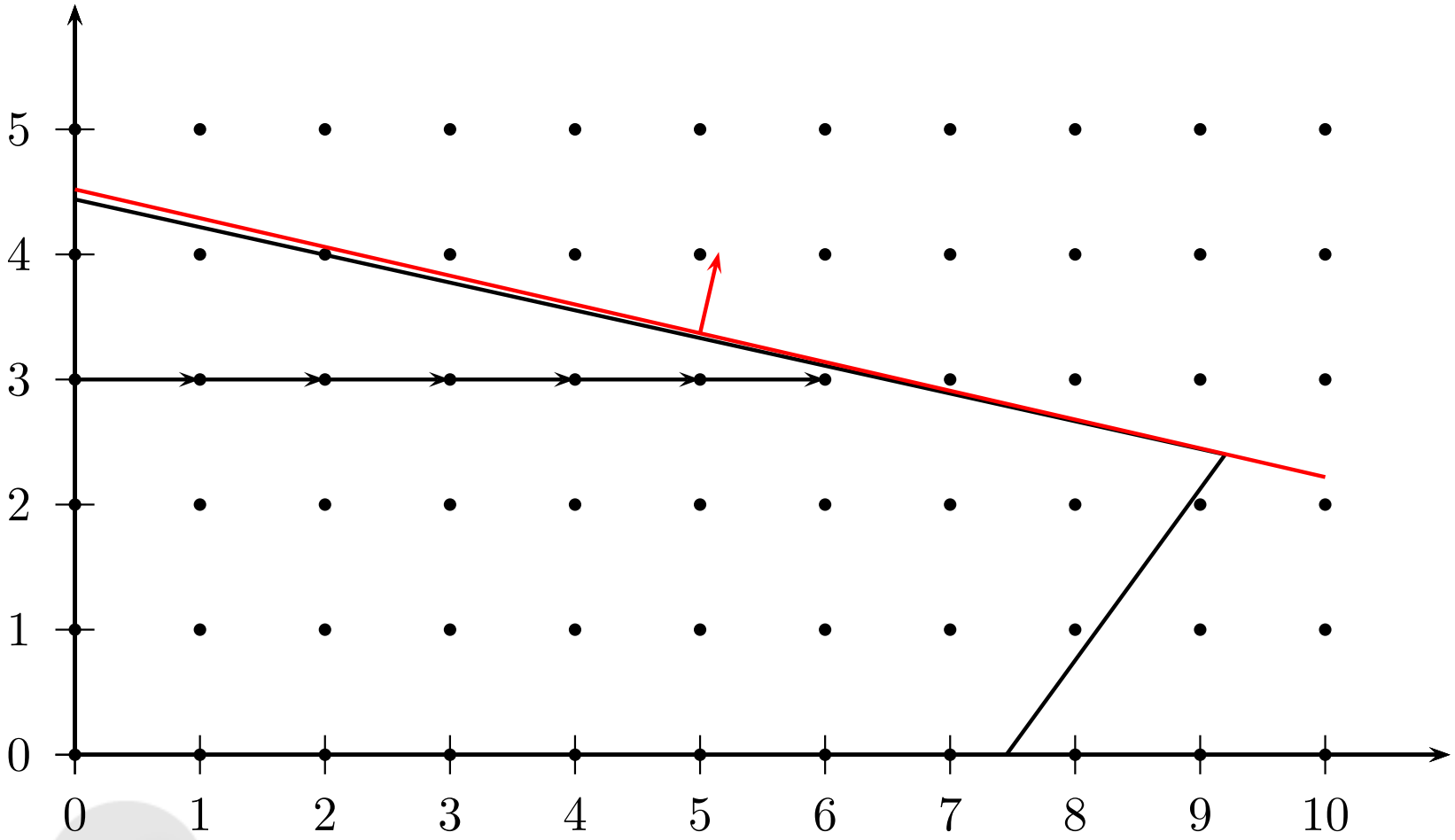
- Voisinage : Est–Nord–Ouest–Sud.
- Point de départ : $x_0 = (6, 0)$, $f(x_0) = -18$.
- Itération 1 :
  - Voisin “est” de x_k : $y = (7, 0)$
  - Admissible : oui.
  - $f(y) = -21 < f(x_0) = -18$.
  - Accepté : $x_1 = (7, 0)$

Recherche locale : exemple

- Itération 2 :
  - Voisin “est” de x_k : $y = (8, 0)$
  - Admissible : non. Rejeté
  - Voisin “nord” de x_k : $y = (7, 1)$
  - Admissible : oui.
  - $f(y) = -34 < f(x_1) = -21$.
  - Accepté. $x_2 = (7, 1)$
- Itération 3 :
  - Voisin “est” de x_k : $y = (8, 1)$
  - Admissible : oui.
  - $f(y) = -37 < f(x_2) = -34$.
  - Accepté. $x_3 = (8, 1)$
- etc.

Recherche locale : exemple

$x_0 = (0, 3)$ - Voisinage : E - N - O - S

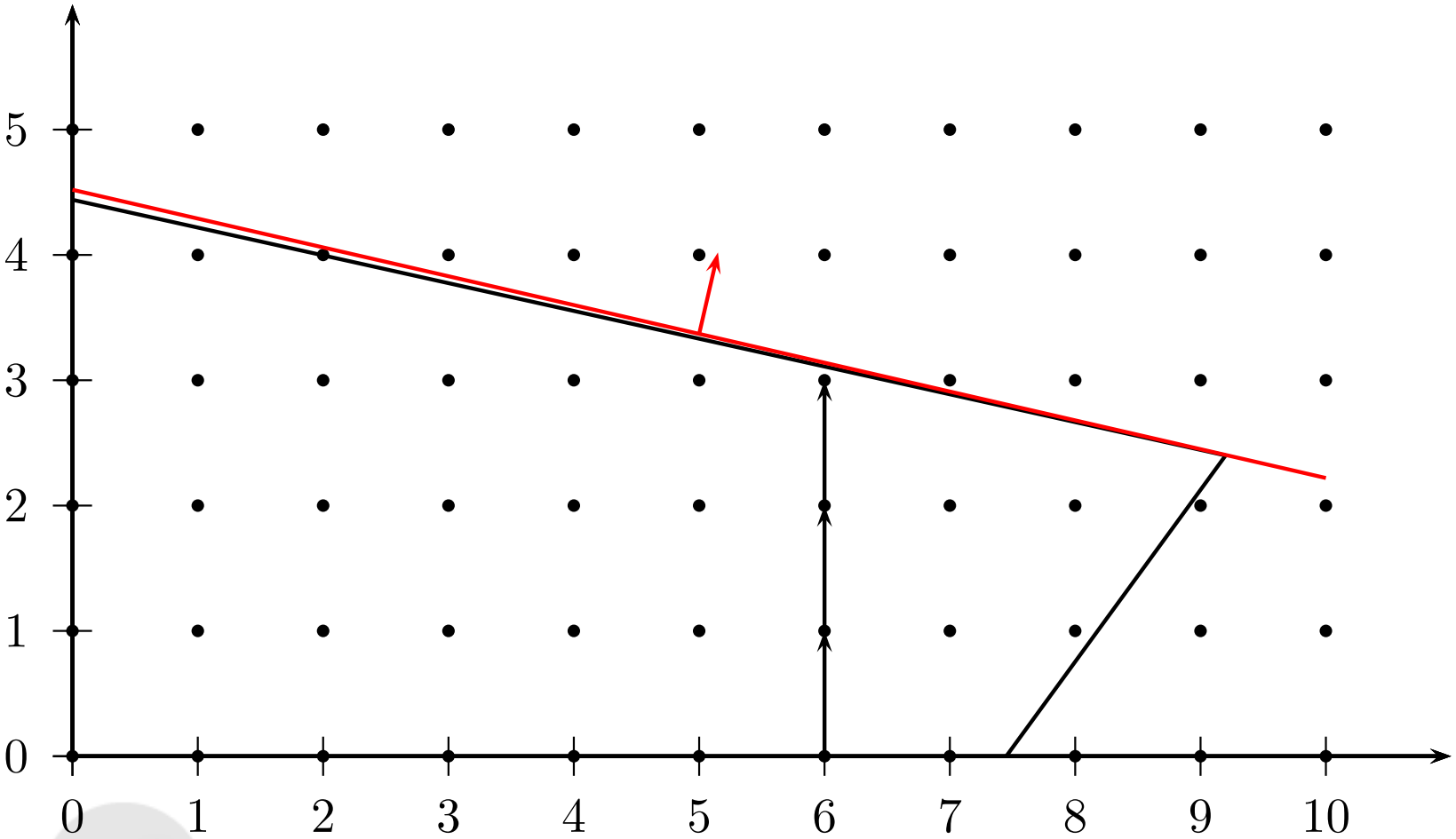


TRANSP-OR

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Recherche locale : exemple

$x_0 = (6, 0)$ - Voisinage : N - O - S - E



TRANSP-OR

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Recherche locale : commentaires

- L'algorithme s'arrête lorsque qu'un minimum local est trouvé.
- Un minimum local est un point qui est meilleur que tous ses voisins.
- Le comportement de la méthode dépend beaucoup de la structure du voisinage et du point de départ.
- Le voisinage doit être suffisamment grand pour avoir plus de chances de trouver un meilleur point, et suffisamment petit pour éviter une énumération trop longue.
- Exemple d'un voisinage trop petit : un seul voisin à l'ouest.
- Exemple d'un voisinage trop grand : chaque point admissible est un voisin.
- Il est conseillé d'utiliser des voisinages symétriques :

$$y \in V(x) \Longleftrightarrow x \in V(y).$$

Le problème du sac à dos

$$\max_{x \in \{0,1\}^n} u^T x$$

sous contraintes

$$w^T x \leq W.$$

- Solution initiale –> trivial : $x = 0$ (sac vide)
- Voisinages
- Recherche locale
- Diversification

Le problème du sac à dos : voisinage

- Itéré courant : pour chaque objet i, $x_i = 0$ ou $x_i = 1$.
- Solution voisine : on choisit un objet j, et on change la décision le concernant : $x_j \leftarrow 1 - x_j$.
- Attention : la solution doit être admissible pour être dans le voisinage.
- Généralisation : voisinage de taille k: on choisit k objets, et on change la décision les concernant, en vérifiant l'admissibilité.

Le problème du sac à dos : voisinage

- Un voisinage de taille k consiste à modifier k variables.
- Il y a autant de voisins que de combinaisons de k objets parmi n

$$\frac{n!}{k!(n-k)!}$$

- Pour $k = 1$, il y a n voisins.
- Pour $k = n$, il y a 1 voisin.

Exemple : capacité du sac à dos : 300.

i	1	2	3	4	5	6	7	8	9	10	11	12
U	80	31	48	17	27	84	34	39	46	58	23	67
W	84	27	47	22	21	96	42	46	54	53	32	78

Point de départ : sac vide.

Le problème du sac à dos : recherche locale

k	1	2	3	4	5	6	7	8	9	10	11	12	$w^T x_c$	$u^T x_c$	$u^T x^*$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	84	80	0

Le problème du sac à dos : recherche locale

k	1	2	3	4	5	6	7	8	9	10	11	12	$w^T x_c$	$u^T x_c$	$u^T x^*$
1	1	0	0	0	0	0	0	0	0	0	0	0	84		80
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80
2	1	1	0	0	0	0	0	0	0	0	0	0	111	111	80

Le problème du sac à dos : recherche locale

k	1	2	3	4	5	6	7	8	9	10	11	12	$w^T x_c$	$u^T x_c$	$u^T x^*$
2	1	1	0	0	0	0	0	0	0	0	0	0	111		111
	0	1	0	0	0	0	0	0	0	0	0	0	27	31	111
	1	0	0	0	0	0	0	0	0	0	0	0	84	80	111
3	1	1	1	0	0	0	0	0	0	0	0	0	158	159	111

Le problème du sac à dos : recherche locale

k	1	2	3	4	5	6	7	8	9	10	11	12	$w^T x_c$	$u^T x_c$	$u^T x^*$
3	1	1	1	0	0	0	0	0	0	0	0	0	158		159
	0	1	1	0	0	0	0	0	0	0	0	0	74	79	159
	1	0	1	0	0	0	0	0	0	0	0	0	131	128	159
	1	1	0	0	0	0	0	0	0	0	0	0	111	111	159
4	1	1	1	1	0	0	0	0	0	0	0	0	180	176	159

Le problème du sac à dos : recherche locale

k	1	2	3	4	5	6	7	8	9	10	11	12	$w^T x_c$	$u^T x_c$	$u^T x^*$
4	1	1	1	1	0	0	0	0	0	0	0	0	180		176
	0	1	1	1	0	0	0	0	0	0	0	0	96	96	176
	1	0	1	1	0	0	0	0	0	0	0	0	153	145	176
	1	1	0	1	0	0	0	0	0	0	0	0	133	128	176
	1	1	1	0	0	0	0	0	0	0	0	0	158	159	176
5	1	1	1	1	1	0	0	0	0	0	0	0	201	203	176

Le problème du sac à dos : recherche locale

k	1	2	3	4	5	6	7	8	9	10	11	12	$w^T x_c$	$u^T x_c$	$u^T x^*$
5	1	1	1	1	1	0	0	0	0	0	0	0	201		203
	0	1	1	1	1	0	0	0	0	0	0	0	117	123	203
	1	0	1	1	1	0	0	0	0	0	0	0	174	172	203
	1	1	0	1	1	0	0	0	0	0	0	0	154	155	203
	1	1	1	0	1	0	0	0	0	0	0	0	179	186	203
	1	1	1	1	0	0	0	0	0	0	0	0	180	176	203
6	1	1	1	1	1	1	0	0	0	0	0	0	297	287	203

Le problème du sac à dos : recherche locale

k	1	2	3	4	5	6	7	8	9	10	11	12	$w^T x_c$	$u^T x_c$	$u^T x^*$
6	1	1	1	1	1	1	0	0	0	0	0	0	297		287
	0	1	1	1	1	1	0	0	0	0	0	0	213	207	287
	1	0	1	1	1	1	0	0	0	0	0	0	270	256	287
	1	1	0	1	1	1	0	0	0	0	0	0	250	239	287
	1	1	1	0	1	1	0	0	0	0	0	0	275	270	287
	1	1	1	1	0	1	0	0	0	0	0	0	276	260	287
	1	1	1	1	1	0	0	0	0	0	0	0	201	203	287
	1	1	1	1	1	1	1	0	0	0	0	0	339	321	287
	1	1	1	1	1	1	0	1	0	0	0	0	343	326	287
	1	1	1	1	1	1	0	0	1	0	0	0	351	333	287
	1	1	1	1	1	1	0	0	0	1	0	0	350	345	287
	1	1	1	1	1	1	0	0	0	0	1	0	329	310	287
	1	1	1	1	1	1	0	0	0	0	0	1	375	354	287

Diversification : Recherche à voisinages variables

- En anglais, “Variable Neighborhood Search” ou VNS.
- Se base sur plusieurs structures de voisinage.
- Idée principale : lorsqu’un optimum local a été trouvé pour une structure de voisinage, on passe à une autre.

VNS : méthode

Input
- $V_1, V_2, \ldots, V_K$ des structures de voisinages.
- Point initial x_0.

Initialisation
- $x^* \leftarrow x_0$
- $k \leftarrow 1$

Itérations Repéter
- Appliquer une recherche locale à partir de x_c avec le voisinage V_k

$$x^+ \leftarrow LS(x^*, V_k)$$

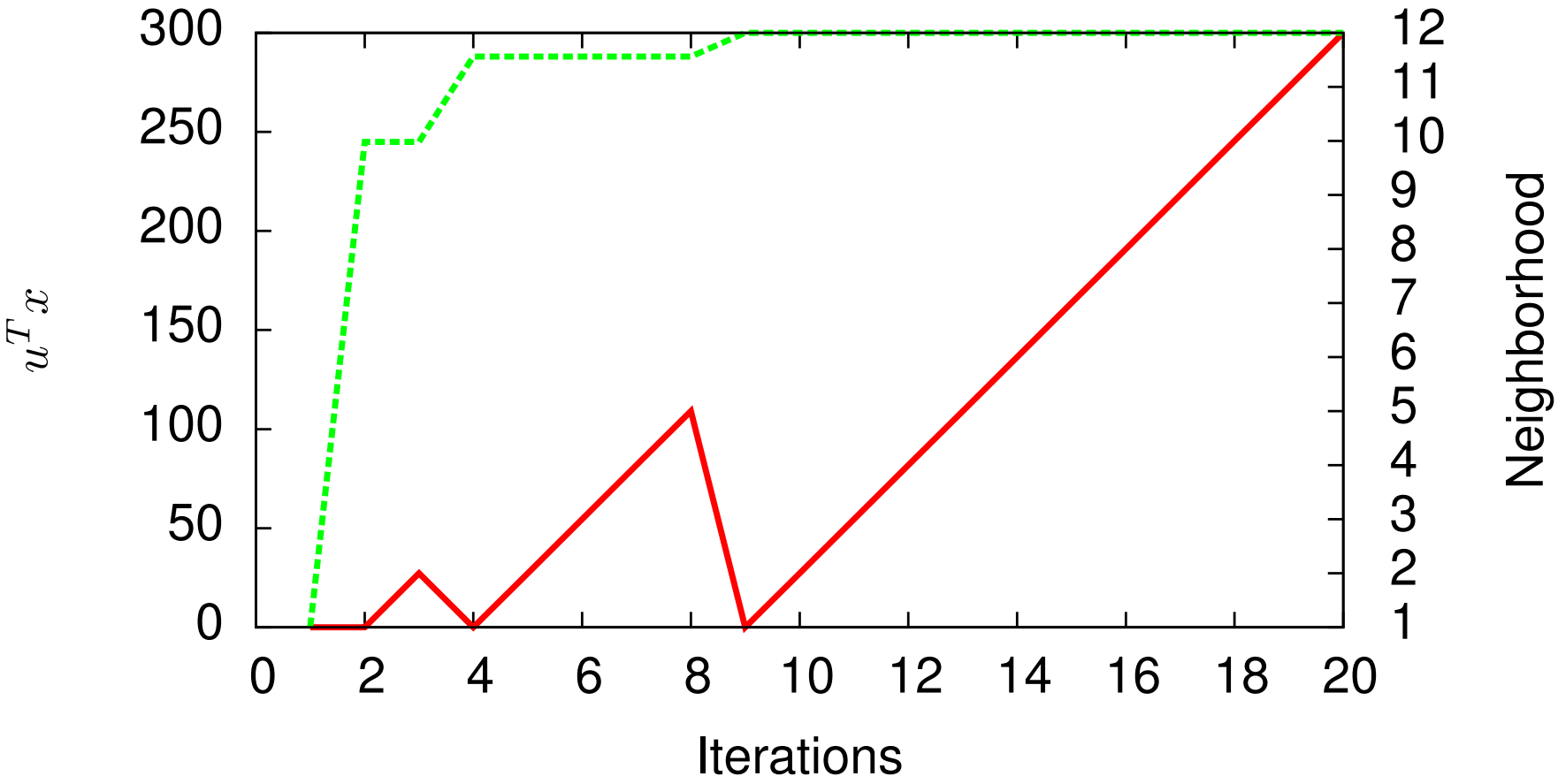
- Si $f(x^+) < f(x^*)$, alors $x^* \leftarrow x^+$, $k \leftarrow 1$.
- Sinon $k \leftarrow k + 1$.

Jusqu'à ce que $k = K$

VNS : exemple

- Structure k de voisinage = modifier k variables.
- Recherche locale : itéré courant : x_c
  - choisir aléatoirement un voisin x^+
  - si $w^T x^+ \leq W$ et $u^T x^+ > u^T x_c$, alors $x_c \leftarrow x^+$
- Effectuer cela 1000 fois, quelque soit k.
- Cela permet d'avoir une complexité indépendante de k.

VNS : exemple


$u^T x$
300
250
200
150
100
50
0
0
2
4
6
8
10
12
14
16
18
20
Iterations
12
11
10
9
8
7
6
5
4
3
2
1
Neighborhood

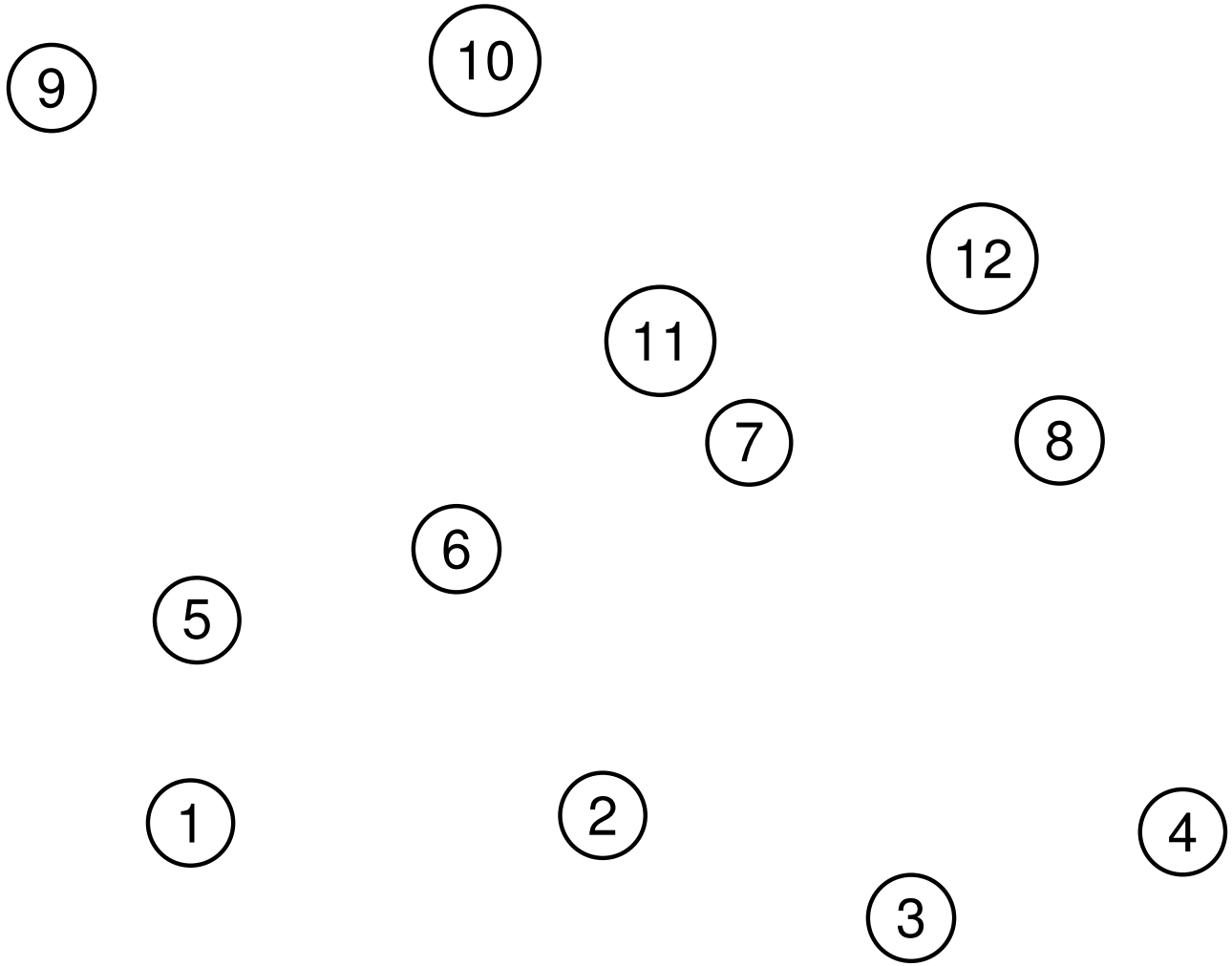
Le problème du voyageur de commerce

- Soit n villes.
- La ville i est distante de la ville j de d_{ij} kilomètres.
- Trouver un itinéraire qui permet au voyageur de commerce de visiter toutes les villes (chemin hamiltonien) et de revenir à son point de départ, en effectuant la plus petite distance possible.

Démarche :

- Solution initiale
- Voisinages
- Recherche locale
- Diversification

Exemple : 12 villes (dist. euclidienne)

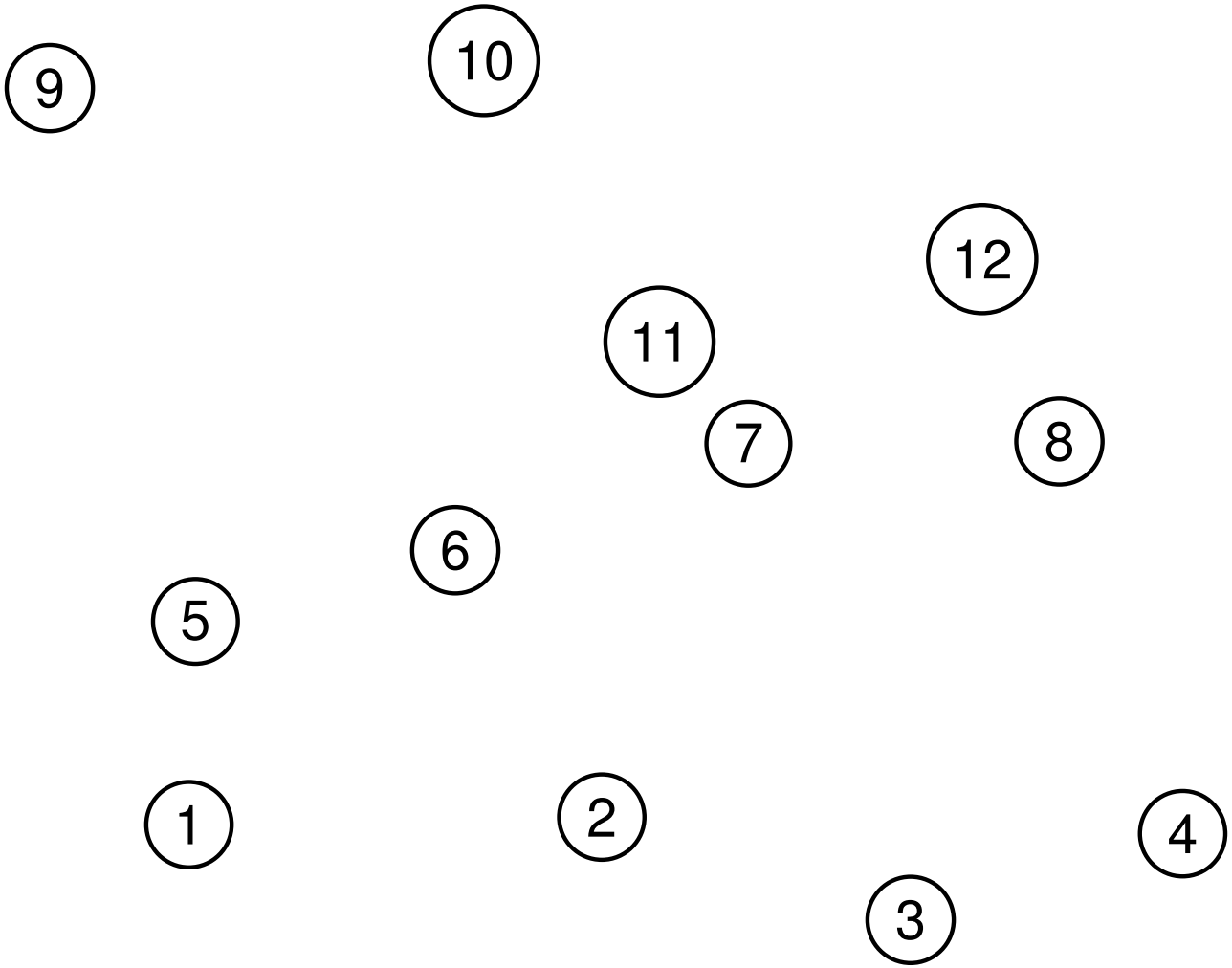


12 villes : solution initiale

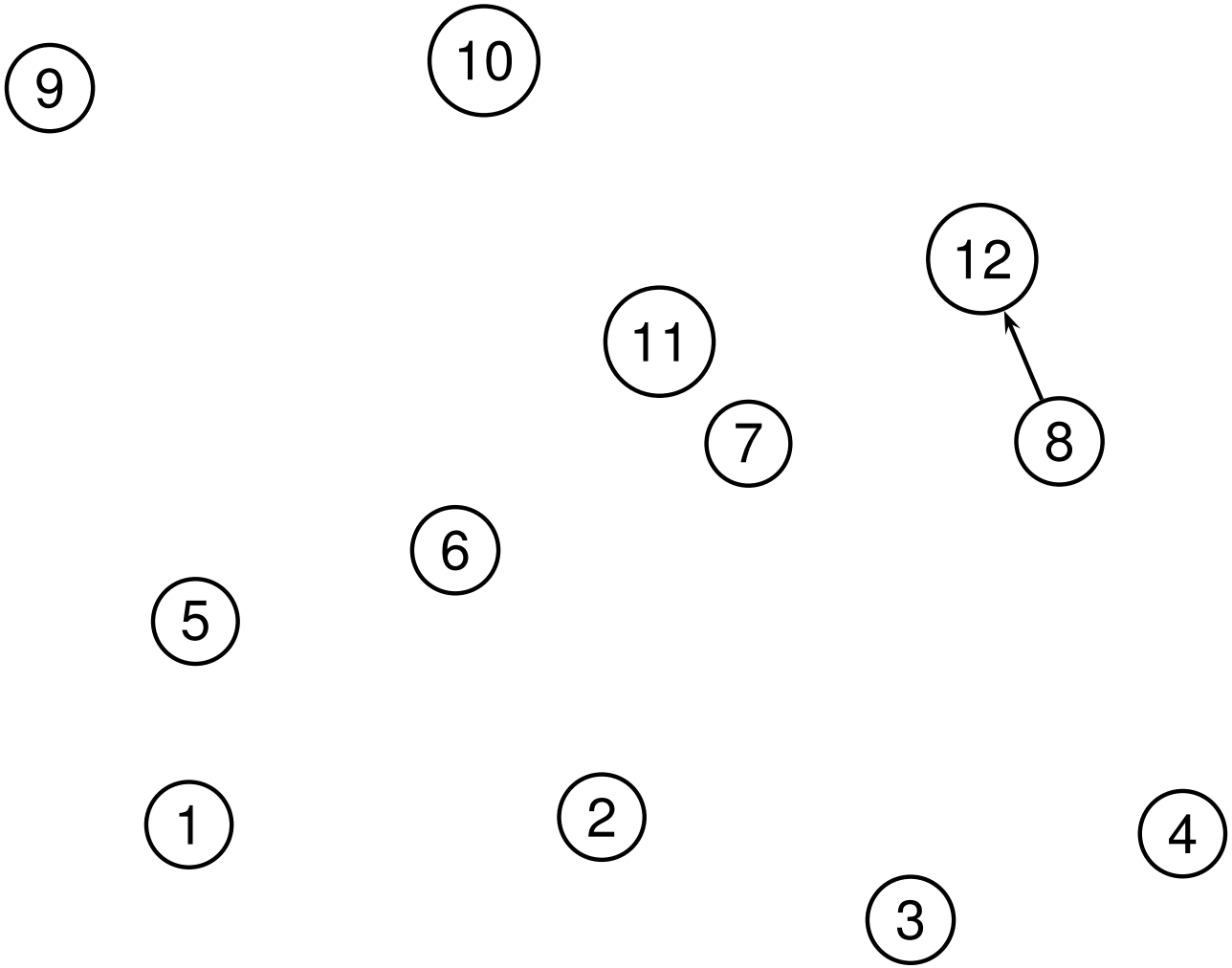
Algorithme “glouton” : stratégie du plus proche voisin.

- Choisir une ville au hasard comme point de départ.
- A chaque étape, se rendre à la ville qui, parmi celles qui n’ont pas été visitées, est la plus proche.

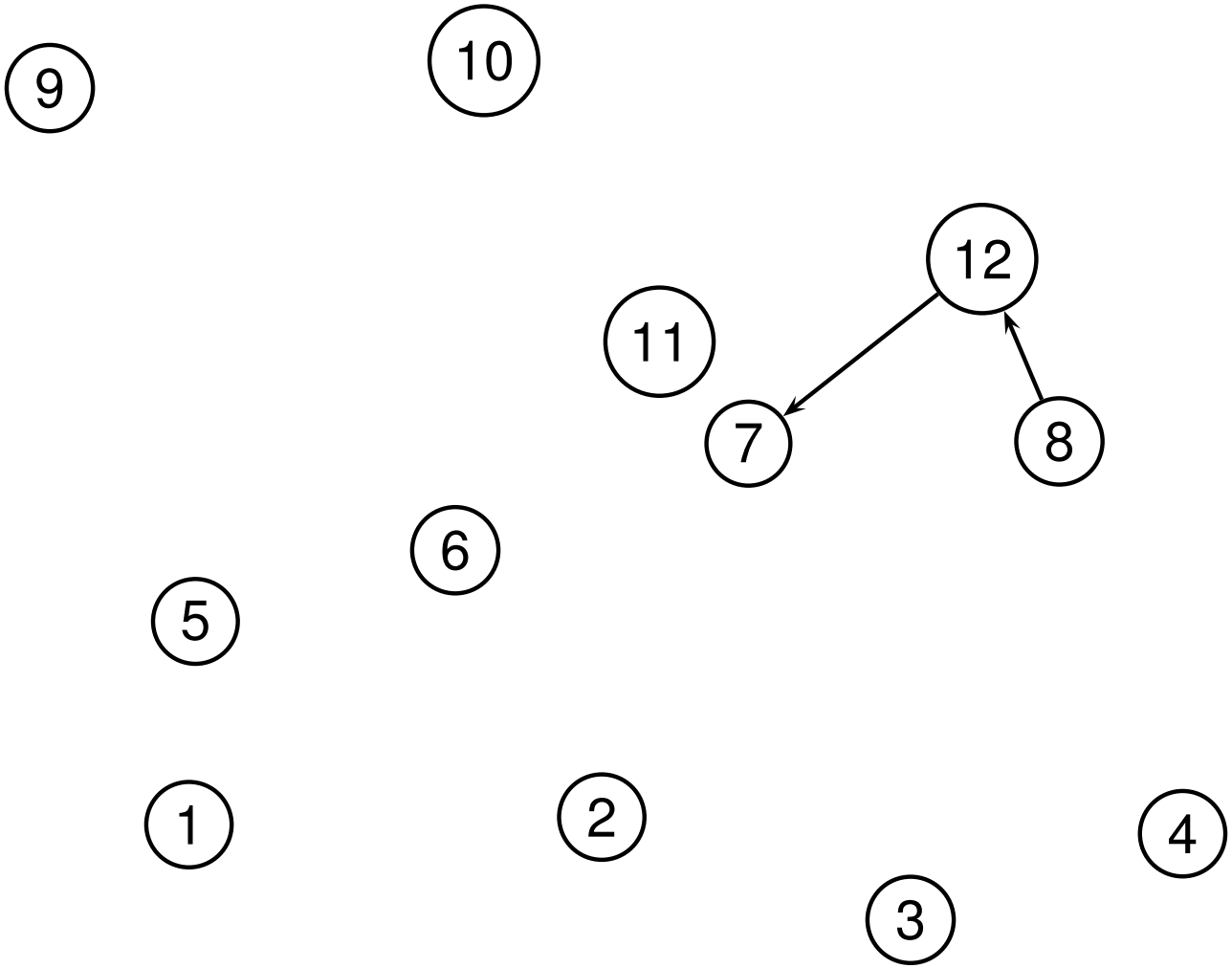
12 villes : solution initiale



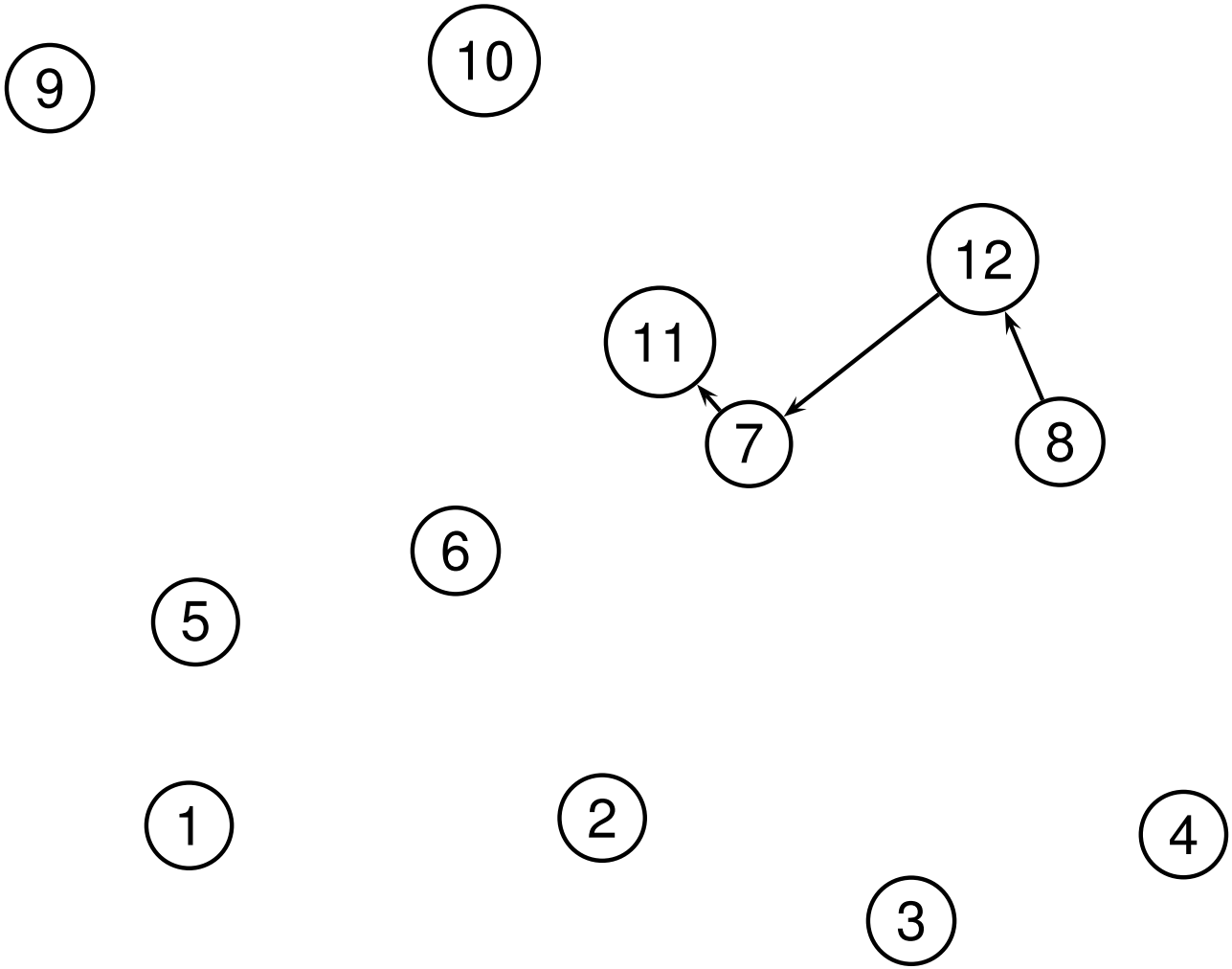
12 villes : solution initiale



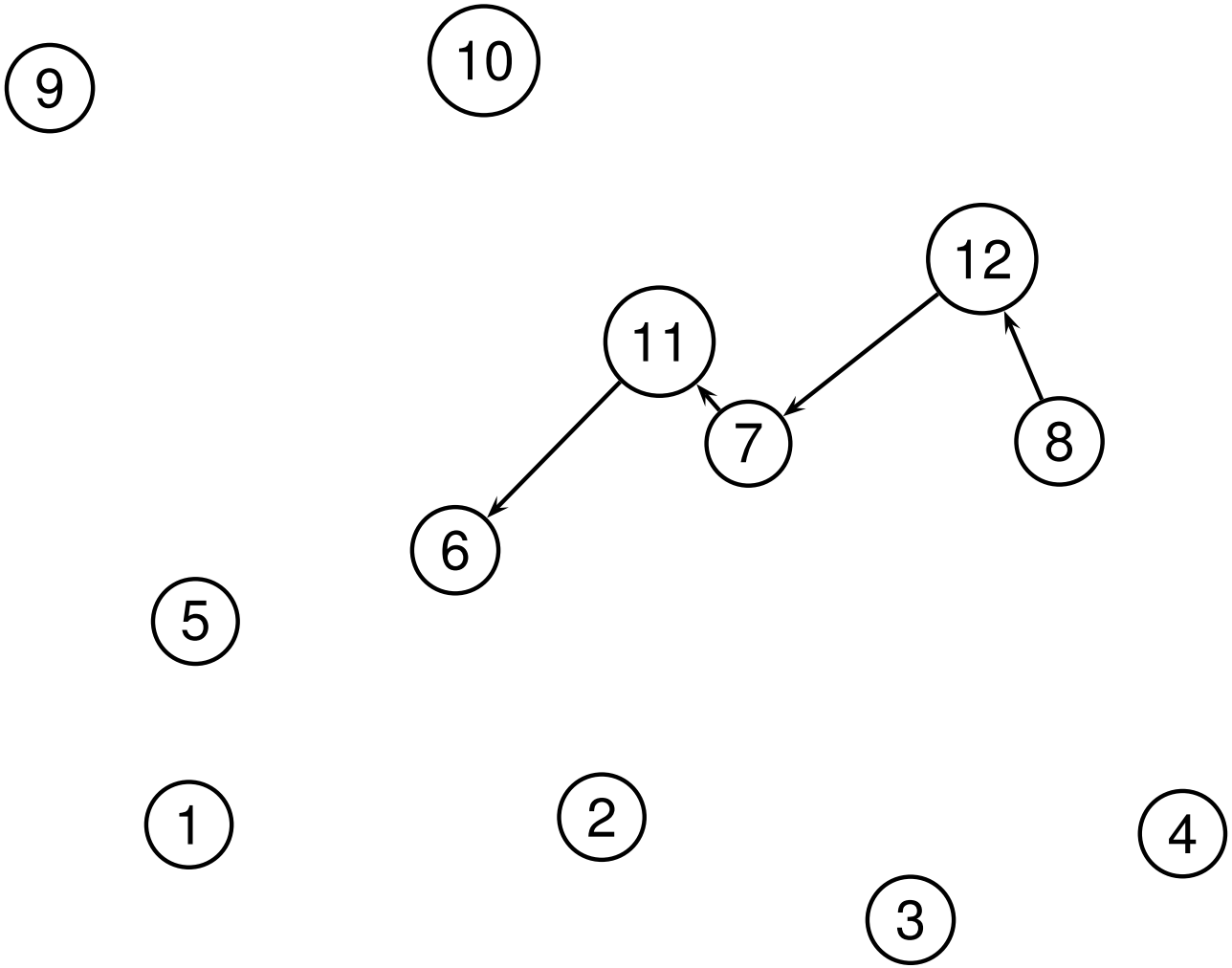
12 villes : solution initiale



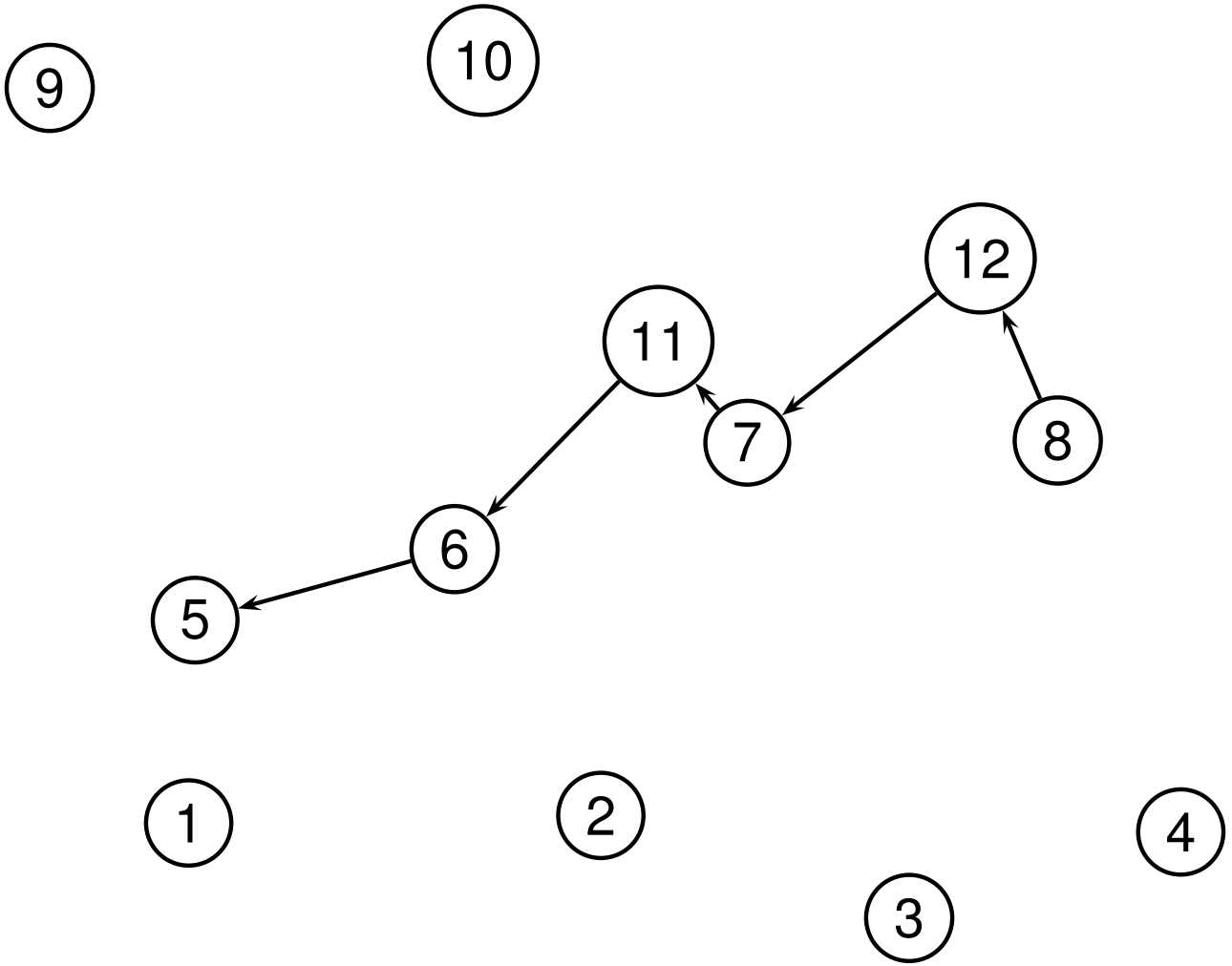
12 villes : solution initiale



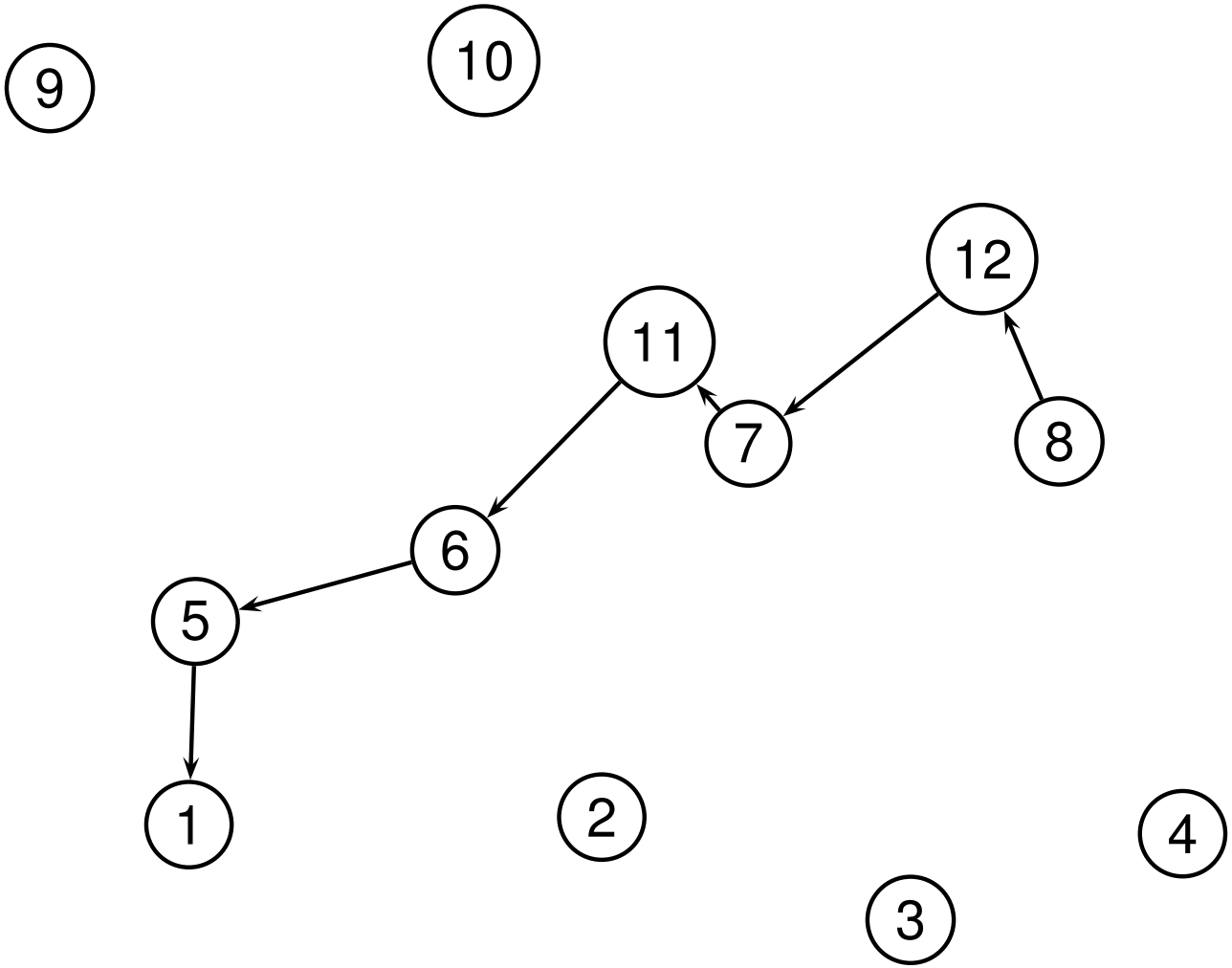
12 villes : solution initiale



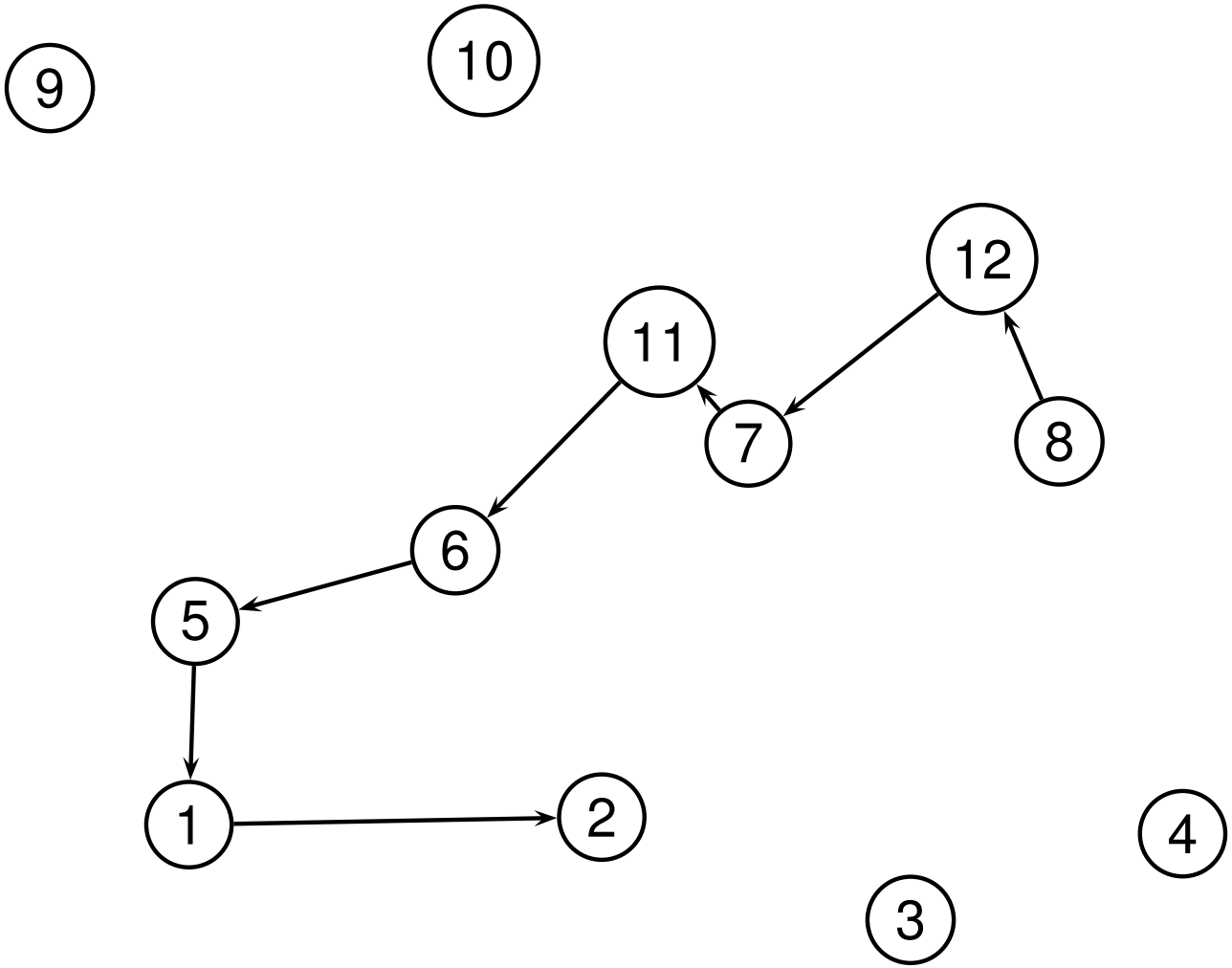
12 villes : solution initiale



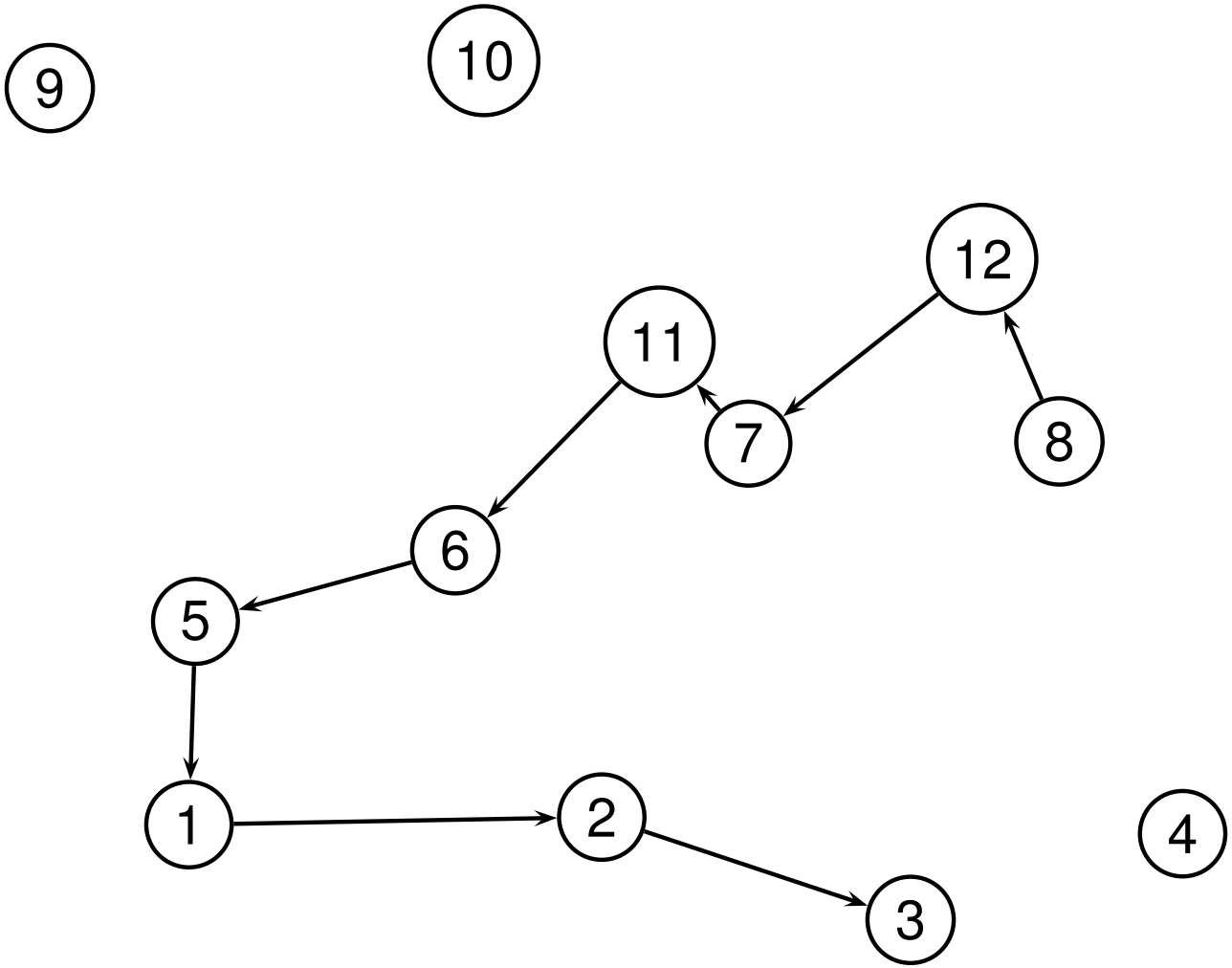
12 villes : solution initiale



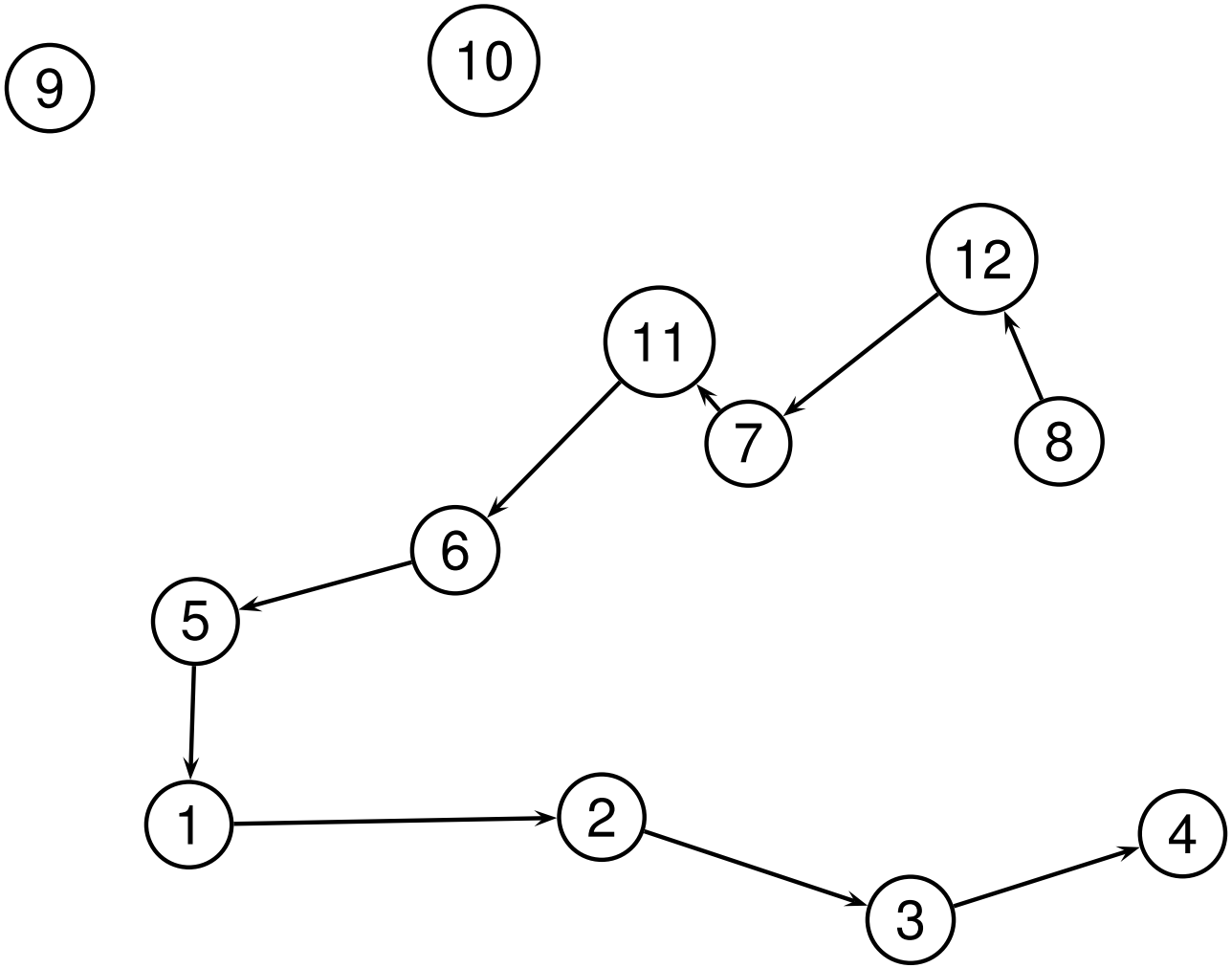
12 villes : solution initiale



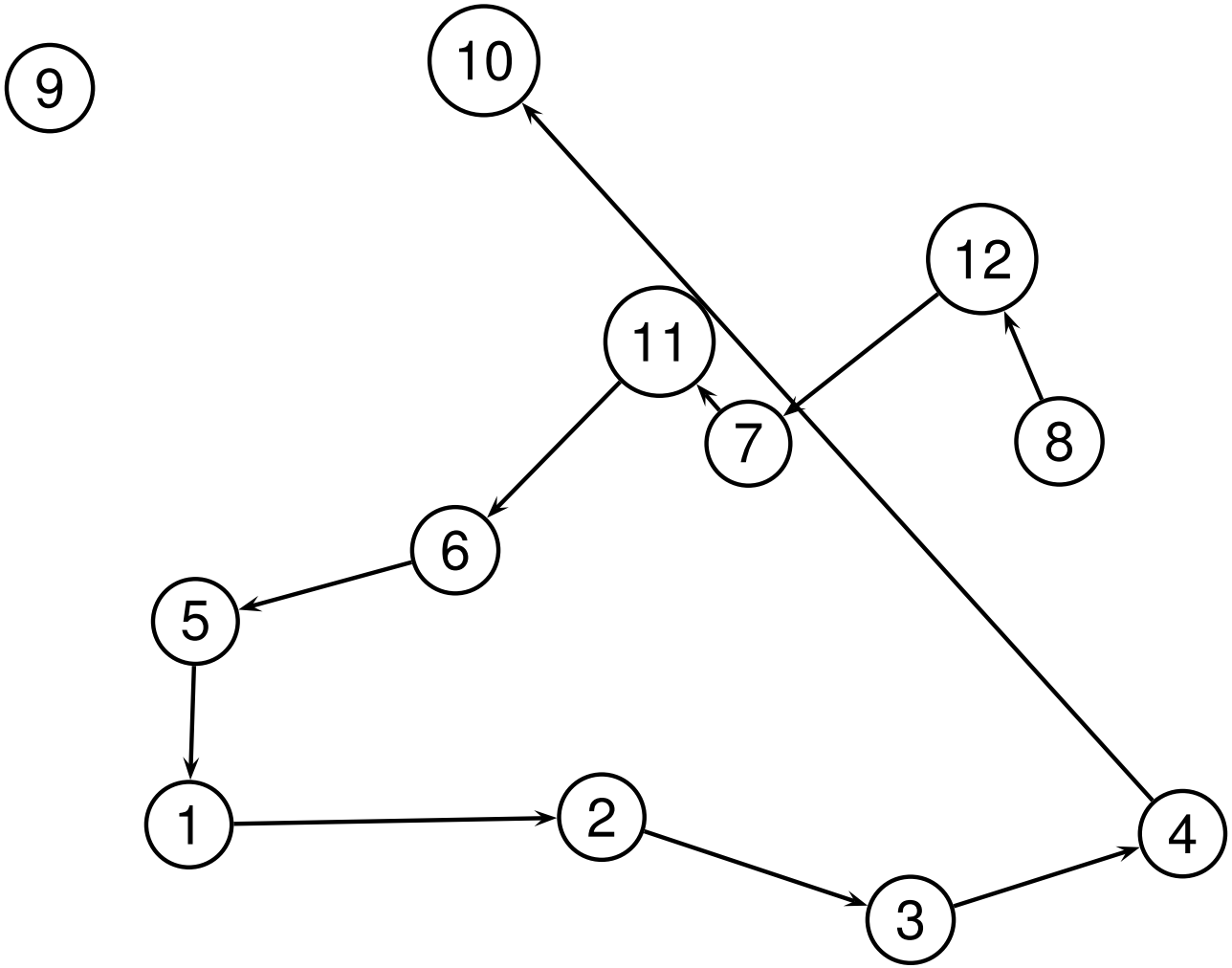
12 villes : solution initiale



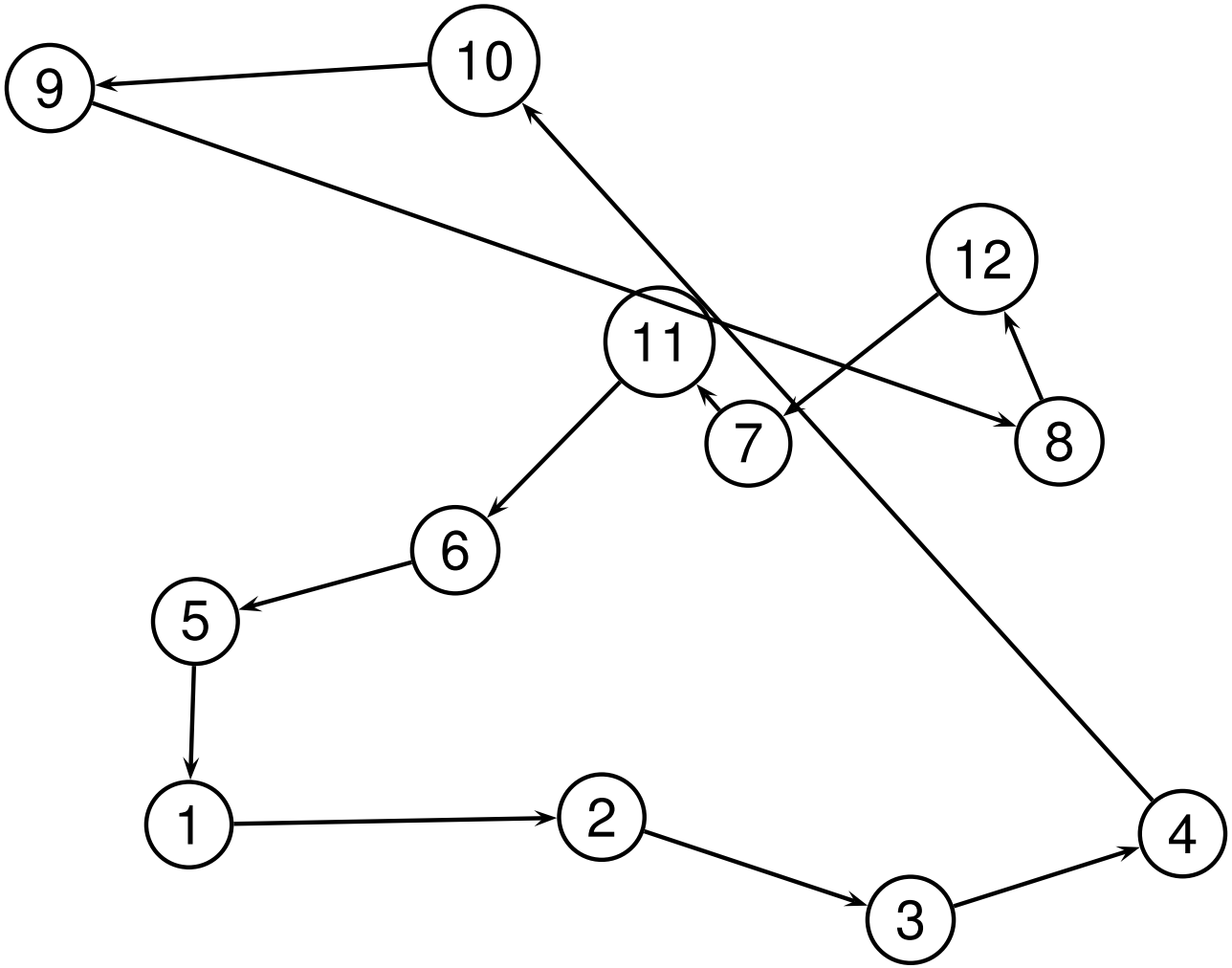
12 villes : solution initiale



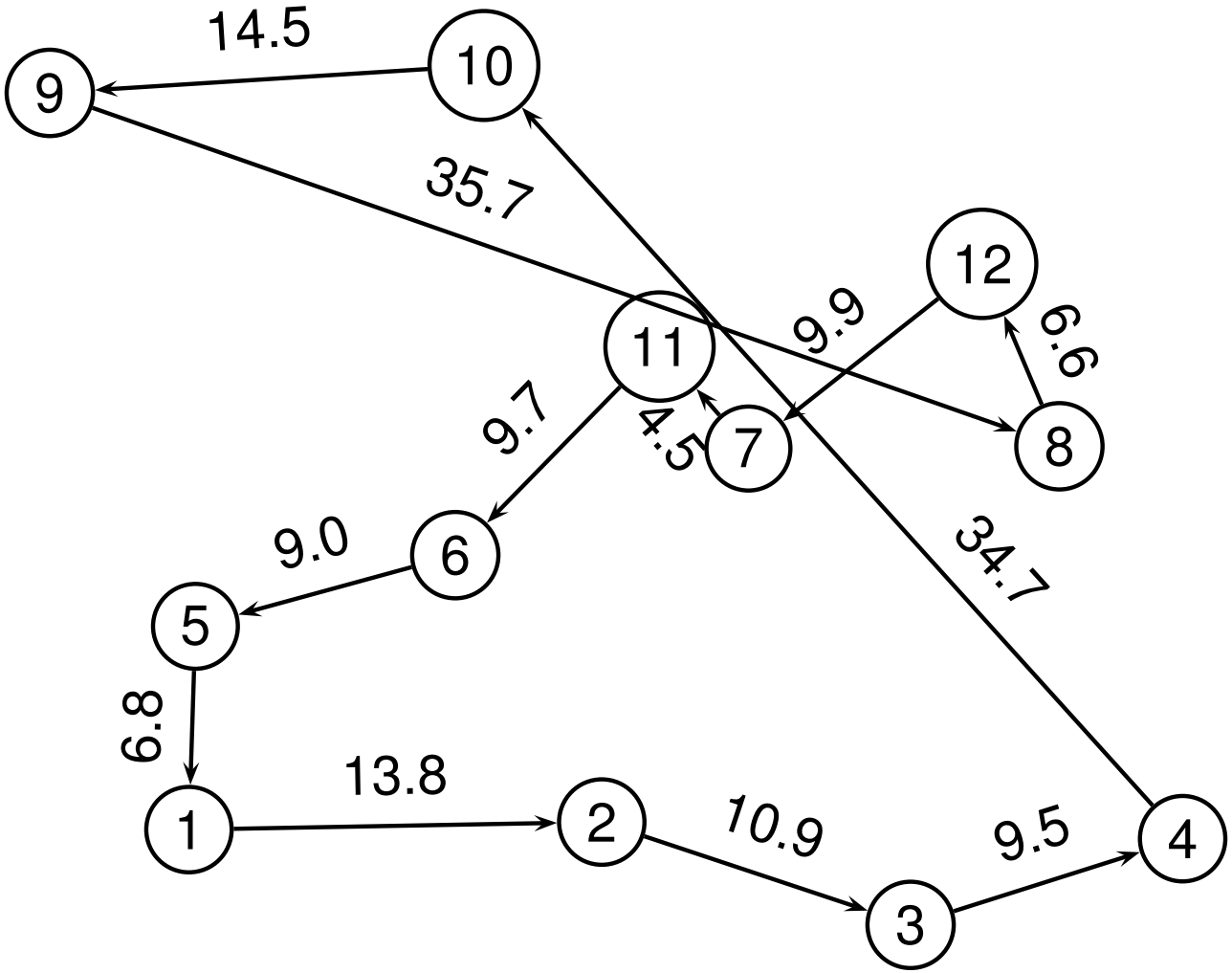
12 villes : solution initiale



12 villes : solution initiale



12 villes : solution initiale



Longueur : 165.6

Voisinage : 2-OPT

Le voisinage 2-OPT consiste à échanger deux villes, et à parcourir les villes intermédiaires dans l'autre sens.
Soit l'itinéraire :

A–B–C–D–E–F–G–H–A

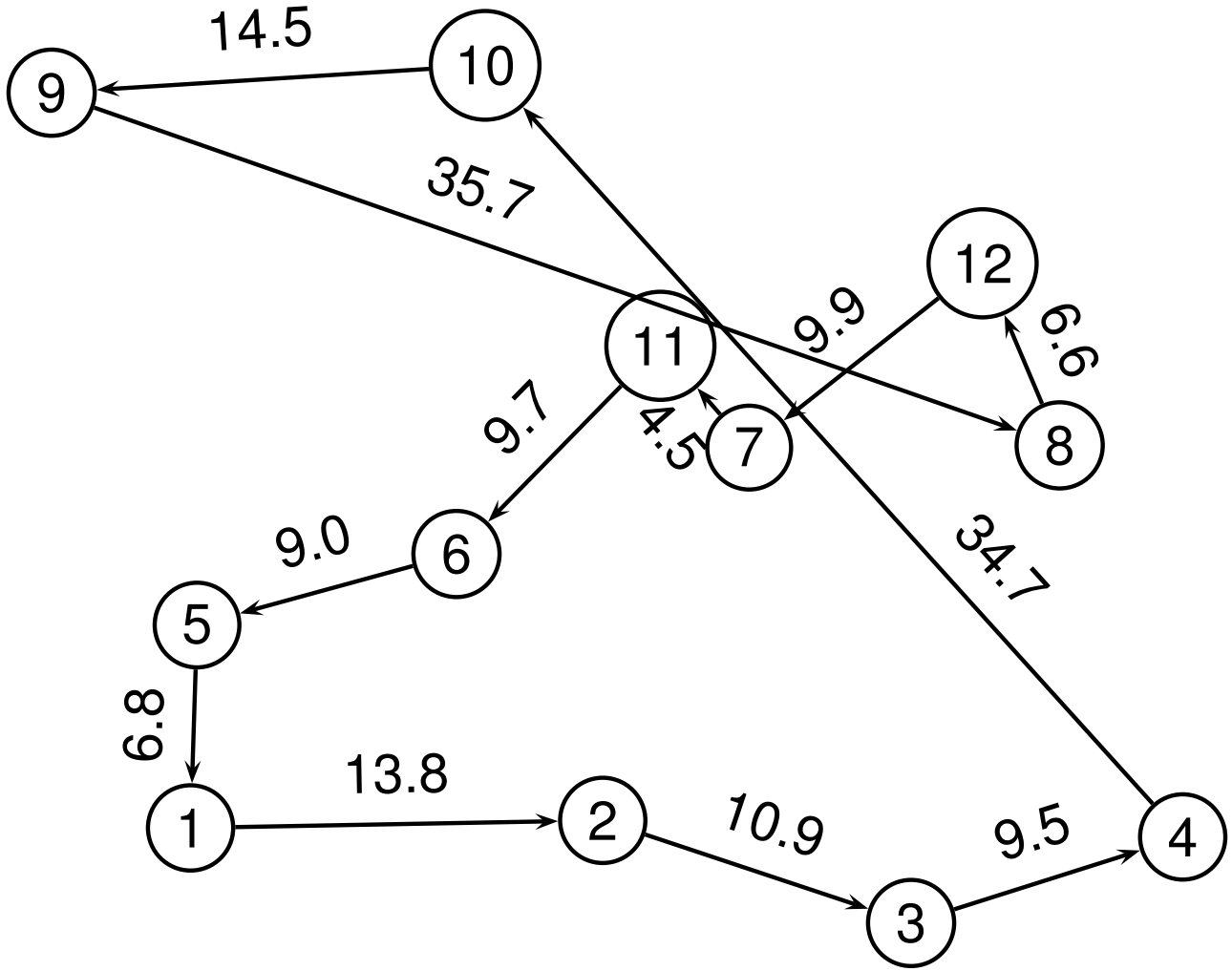
On échange C et G pour obtenir

A–B–G–F–E–D–C–H–A.

Voisinage : 2-OPT(1,9)

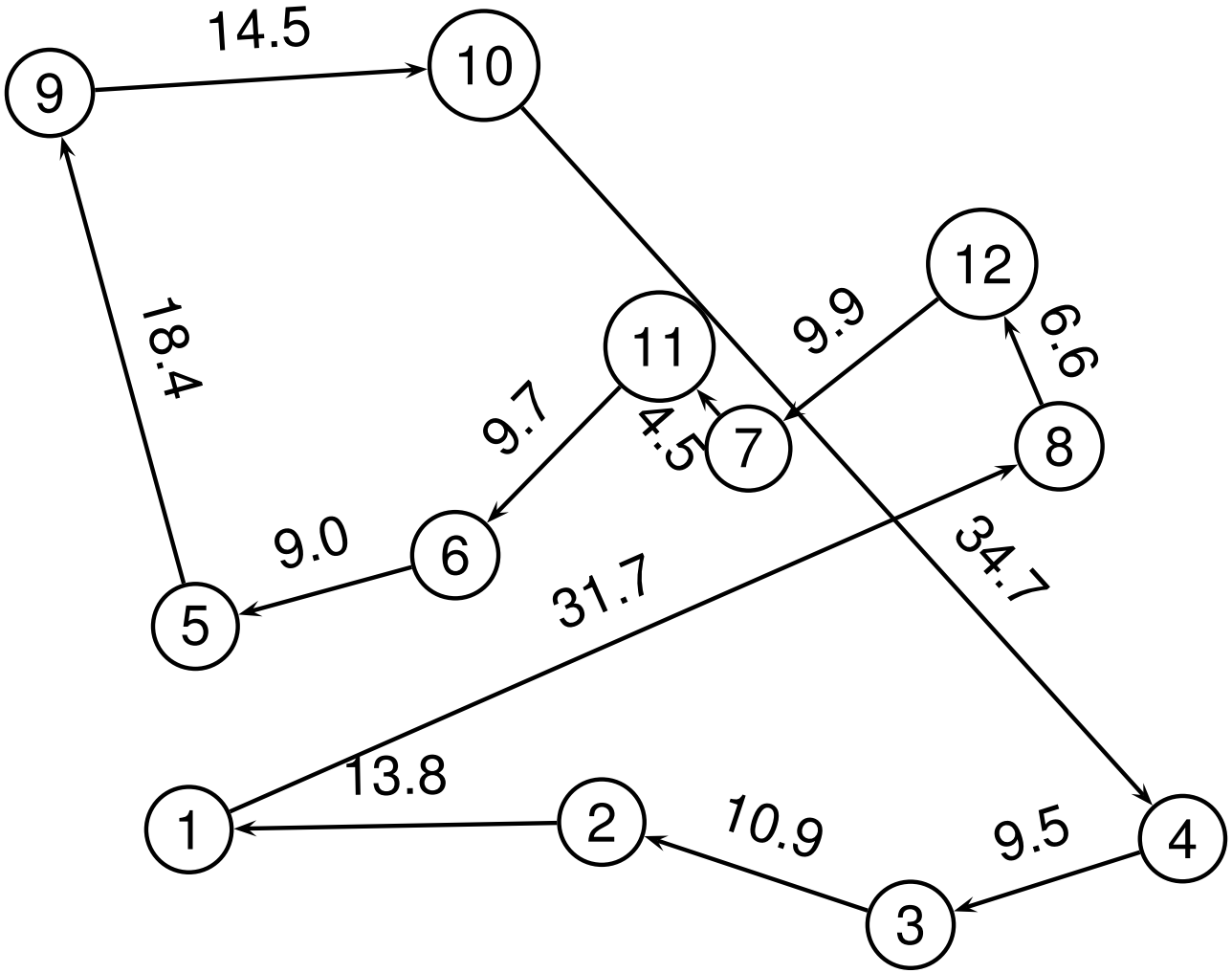
- On essaie d'améliorer en effectuant un 2-OPT en échangeant 1 et 9.
- Avant : 8–12–7–11–6–5–1–2–3–4–10–9–8 (longueur : 165.6)
- Après : 8–12–7–11–6–5–9–10–4–3–2–1–8 (longueur : 173.3)
- Pas d'amélioration.

Voisinage : 2-OPT(1,9)



Longueur : 165.6

Voisinage : 2-OPT(1,9)

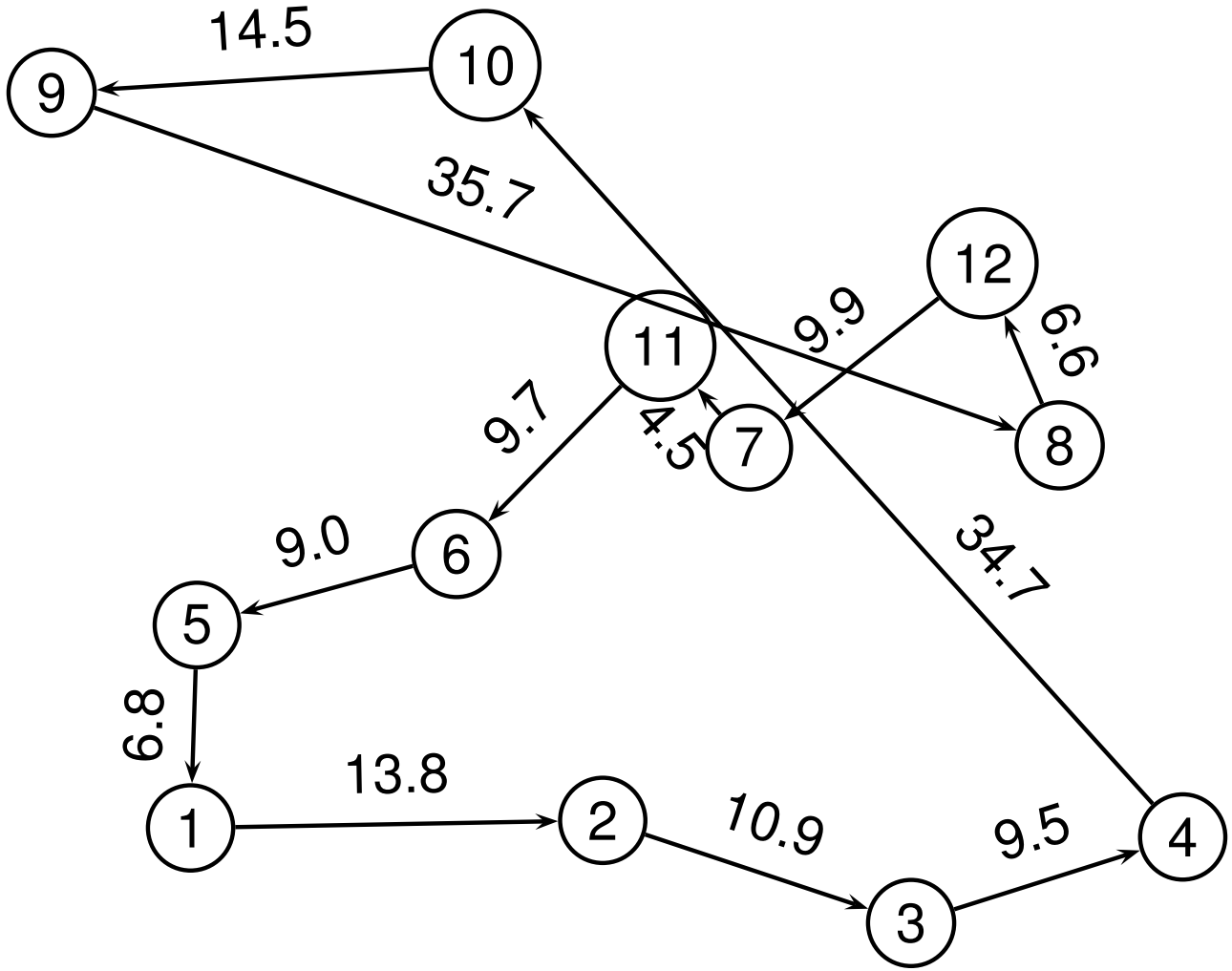


Longueur : 173.3

Recherche locale

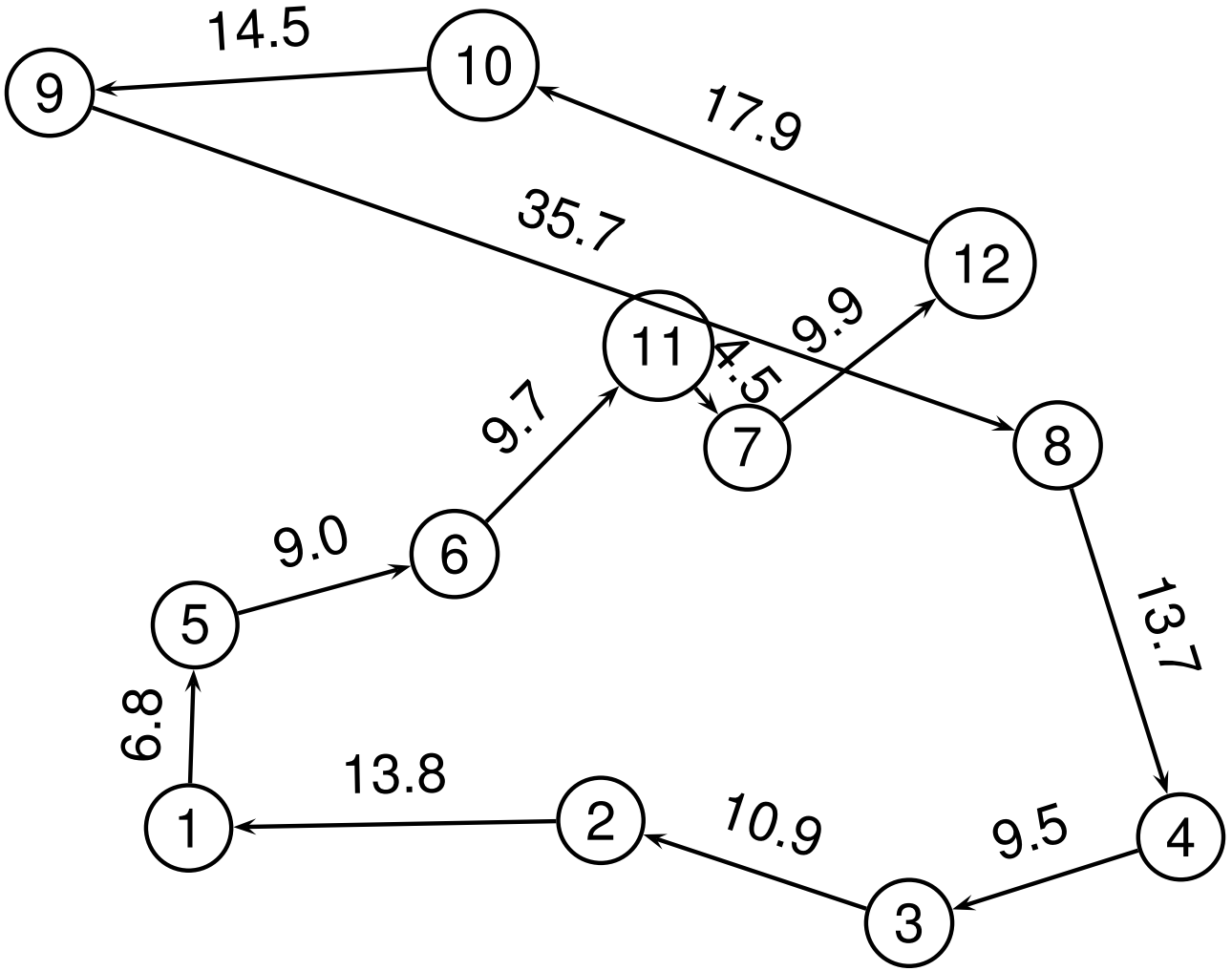
- Passer en revue chaque paire de noeud.
- Effectuer une modification 2-OPT.
- Choisir le résultat correspondant à la tournée la plus courte.

Solution actuelle



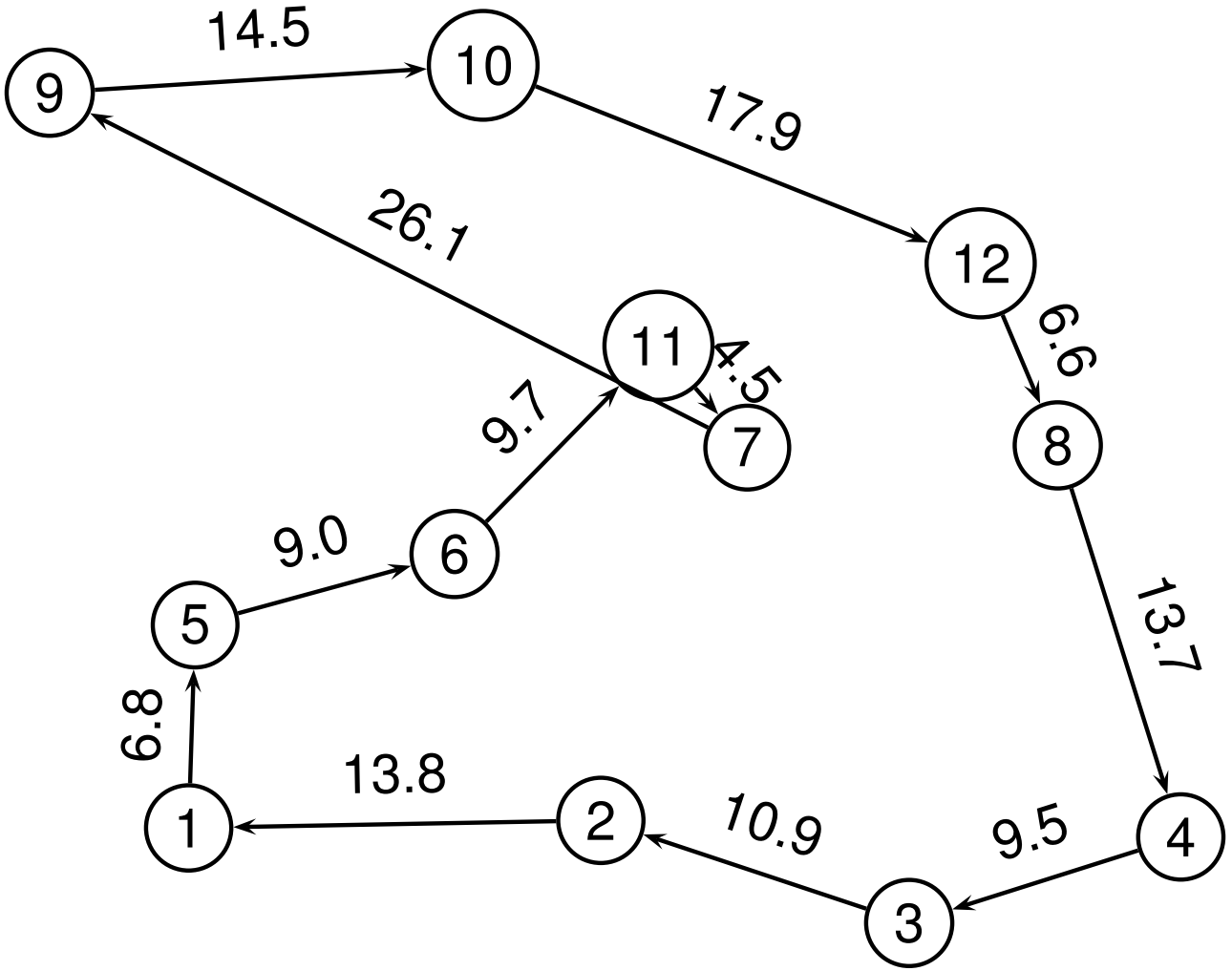
Longueur : 165.6

Meilleur voisin : 2-OPT(12,4)



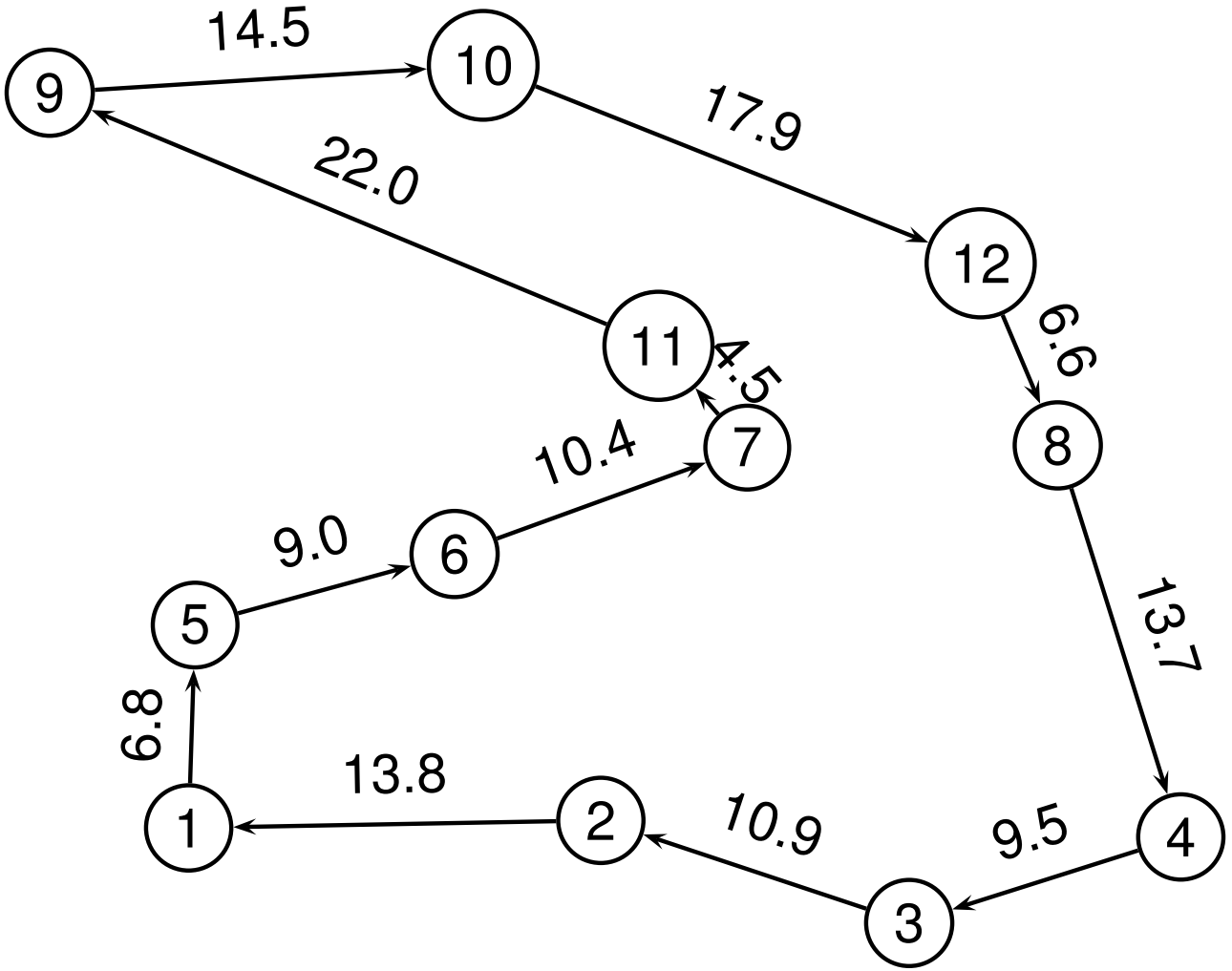
Longueur : 155.8

Meilleur voisin : 2-OPT(12,9)



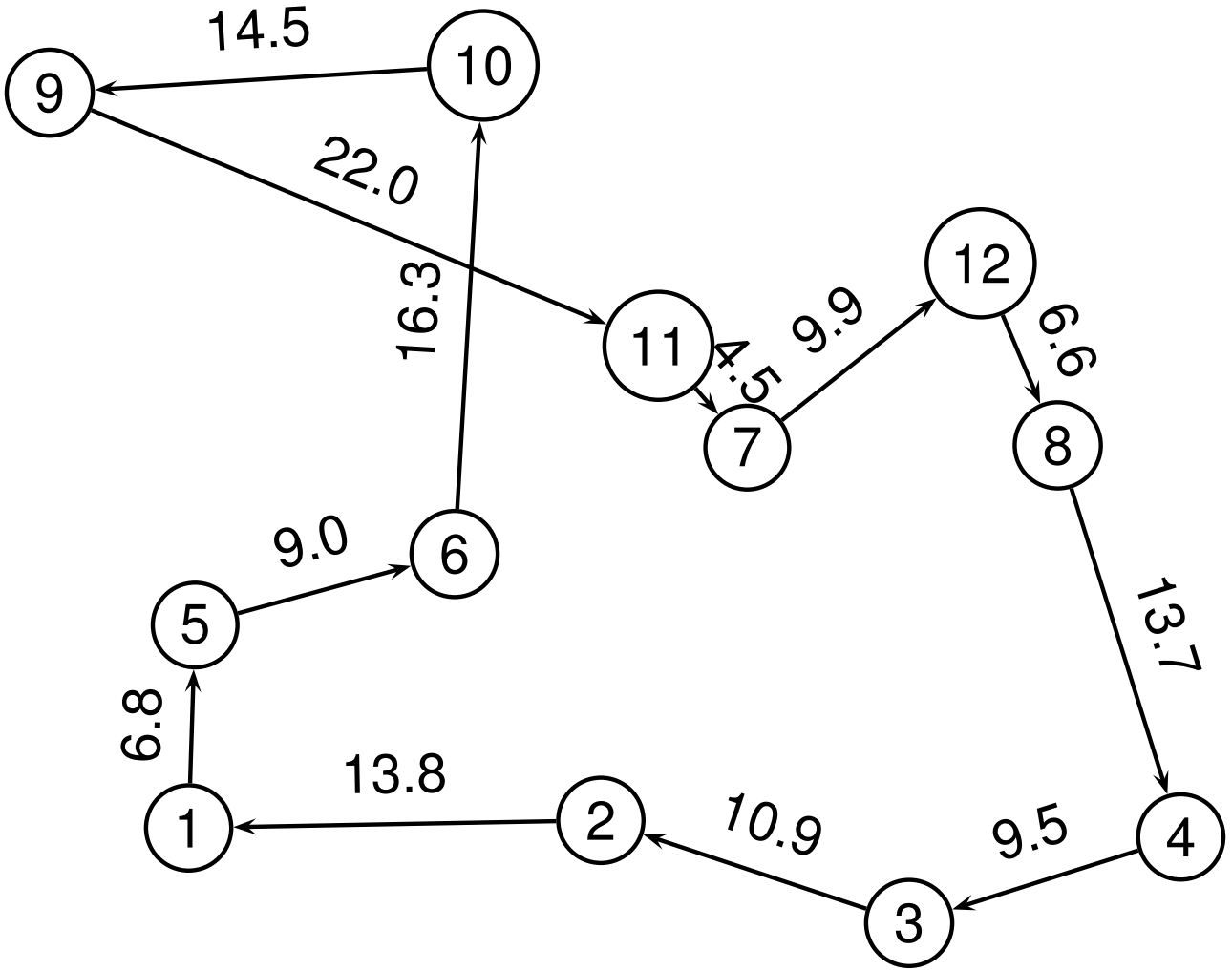
Longueur : 143.0

Meilleur voisin : 2-OPT(11,7)



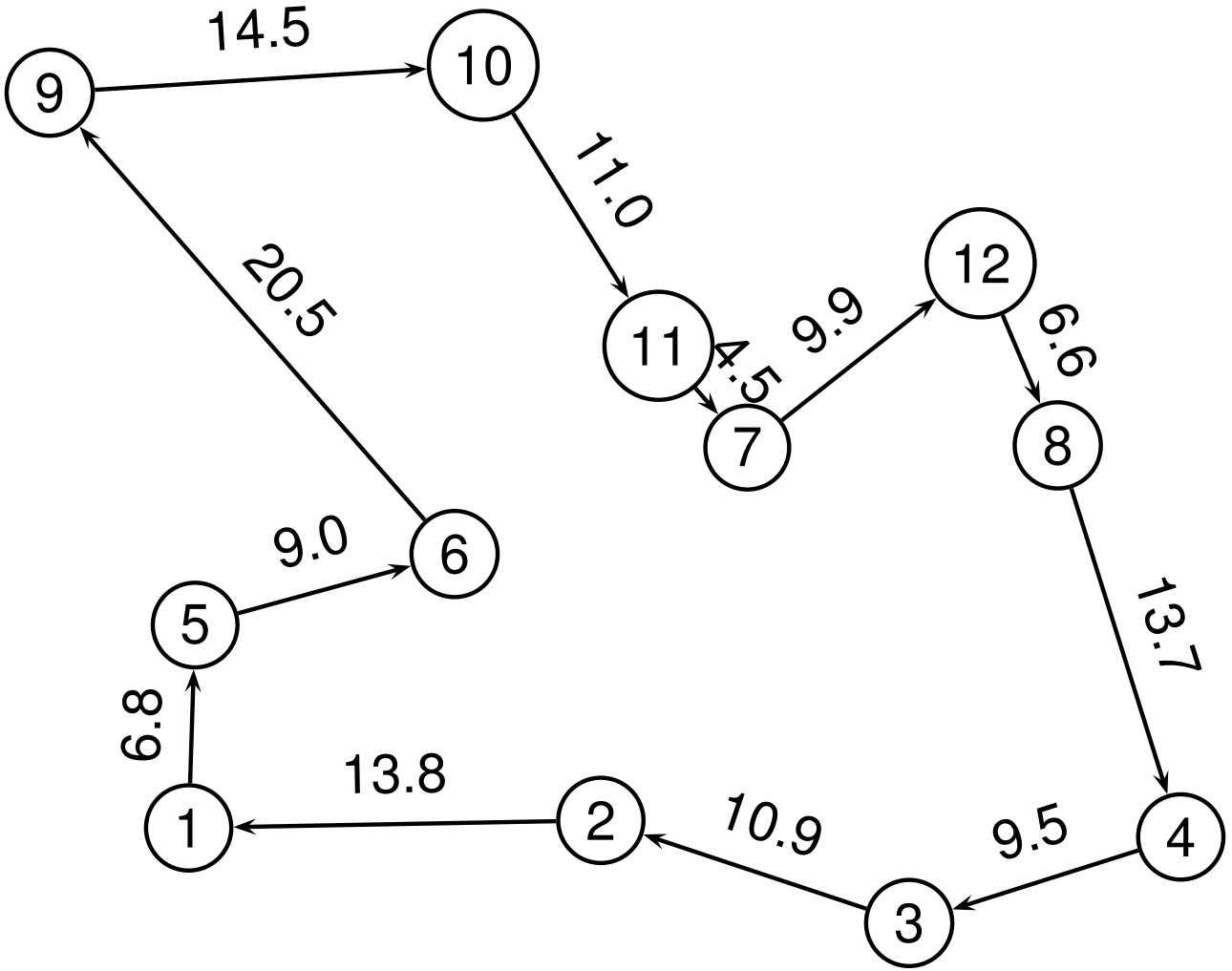
Longueur : 139.5

Meilleur voisin : 2-OPT(7,10)



Longueur : 137.5

Meilleur voisin : 2-OPT(10,9)



Longueur : 130.7

Diversification : recuit simulé

Analogie avec la physique des matériaux

- Pour rendre un métal stable, il faut ralentir son refroidissement.
- Pour ce faire, on le “recuit” en apportant de la chaleur externe.
- Ses atomes prennent alors une configuration plus solide.

En optimisation

- Recherche locale $\approx$ refroidissement.
- Si on descend “trop vite”, on est bloqué dans une configuration sous optimale.
- On simule un “recuit” en permettant à l’algorithme de “remonter” la pente de temps en temps.

Recuit simulé

Modification de la recherche locale
Pour simplifier : considérons une structure de voisinage ne contenant que des points admissibles
Soit x_k l'itéré courant.

- Choisir $y \in V(x_k)$.
- Si $f(y) \leq f(x_k)$, alors $x_{k+1} = y$.
- Sinon, $x_{k+1} = y$ avec probabilité

$$e^{-\frac{f(y)-f(x)}{T}}$$

avec $T > 0$.

Concrètement, tirer un nombre aléatoire r entre 0 et 1.
Accepter y comme itéré courant si

$$e^{-\frac{f(y)-f(x)}{T}} > r$$

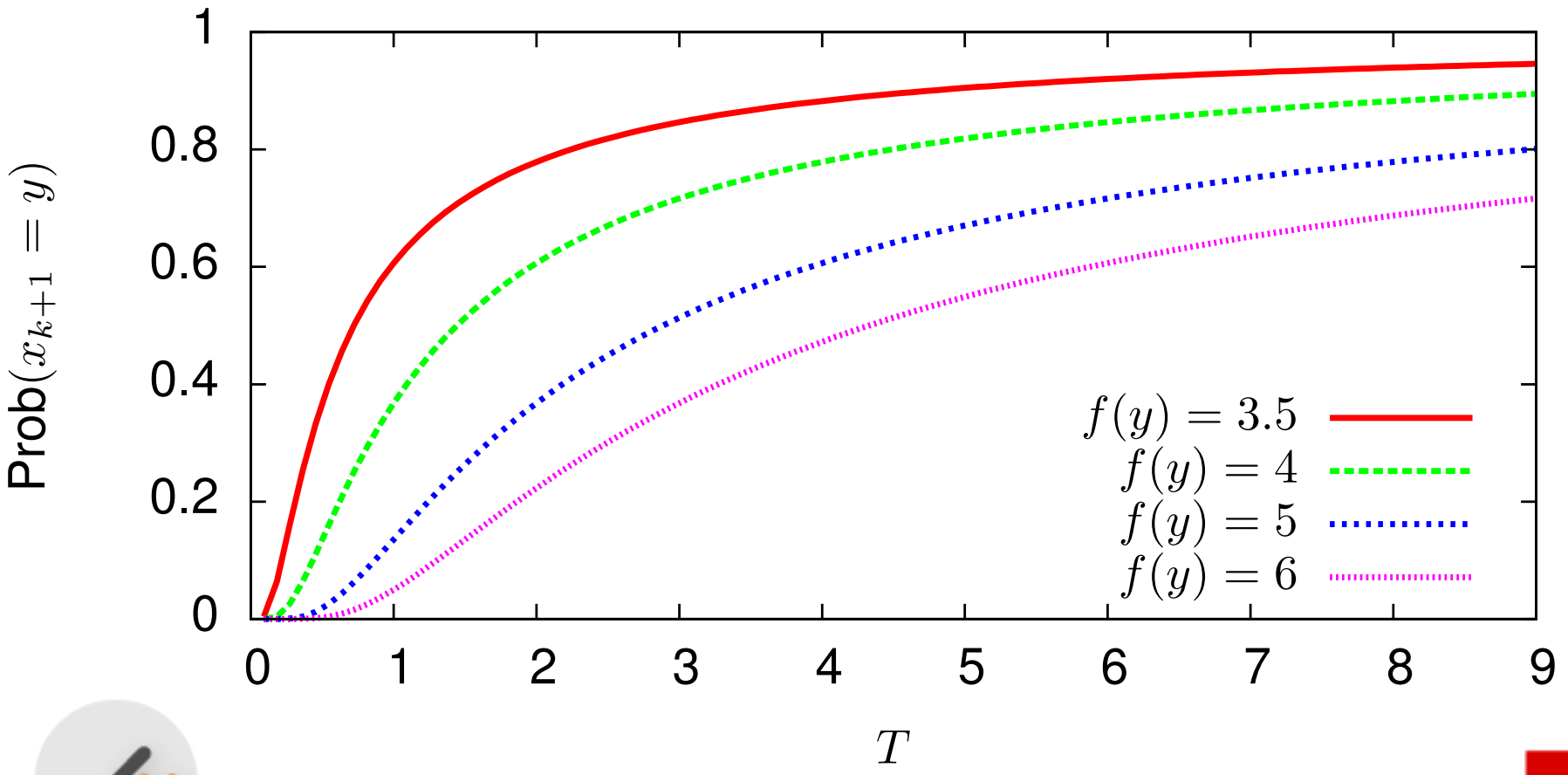
Recuit simulé

$$\text{Prob}(x_{k+1} = y) = \begin{cases} 1 & \text{si } f(y) \leq f(x_k) \\ e^{-\frac{f(y)-f(x)}{T}} & \text{si } f(y) > f(x_k) \end{cases}$$

- Si T est grand (température “chaude”), forte probabilité de monter.
- Si T est proche de zéro, quasiment uniquement de la descente.

Recuit simulé

Exemple : $f(x_k) = 3$



Recuit simulé

- En pratique, on commence avec une température élevée pour donner de la flexibilité à l'algorithme.
- On diminue progressivement la valeur de T.

Recuit simulé

Input
- Point de départ x_0
- Température initiale T_0, température minimale T_f
- Structure de voisinage $V(x)$
- Nombre maximum d'itérations K

Initialize $x_c \leftarrow x_0$, $x^* \leftarrow x_0$, $T \leftarrow T_0$

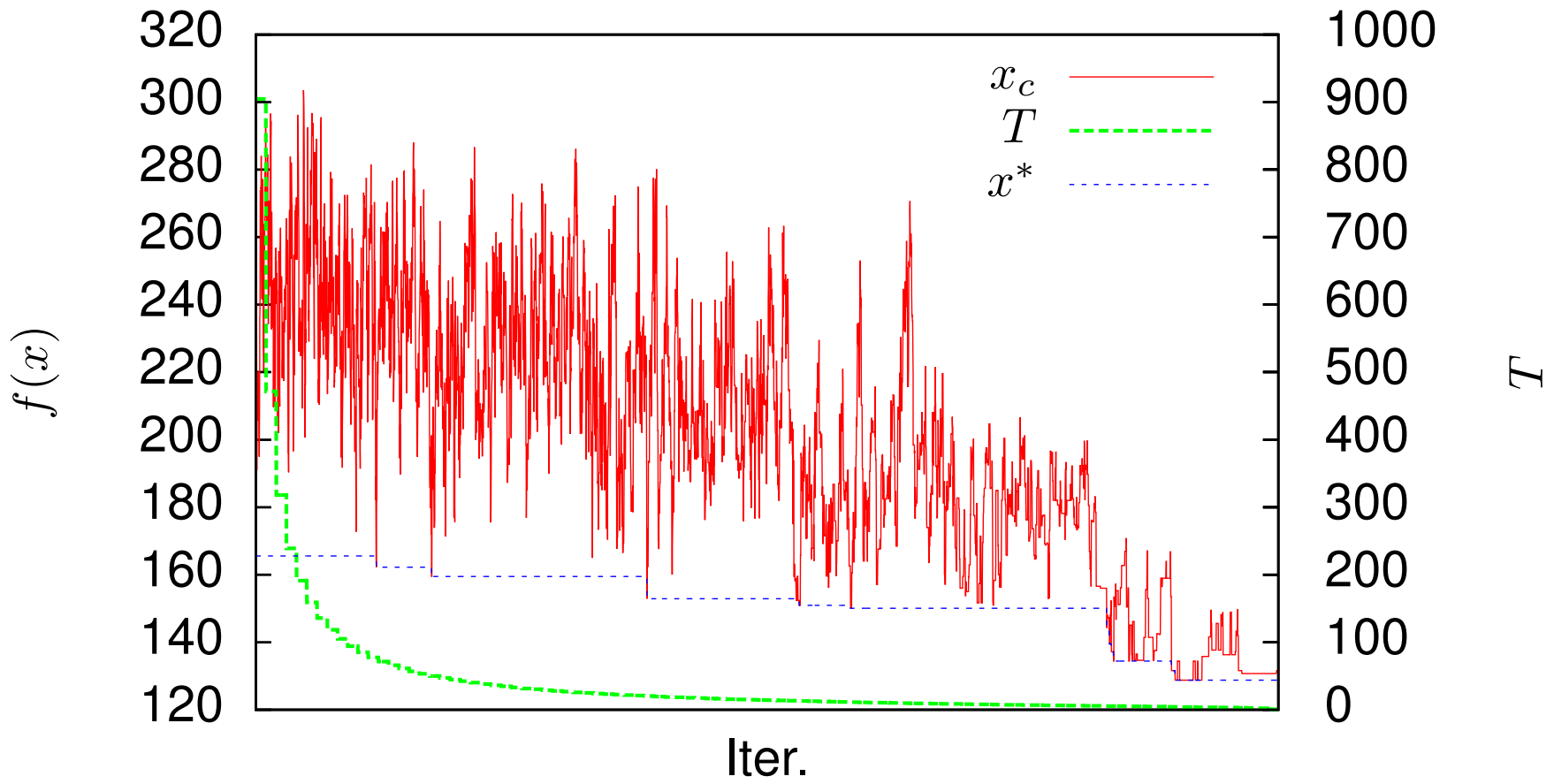
Recuit simulé

Répéter $k \leftarrow 1$

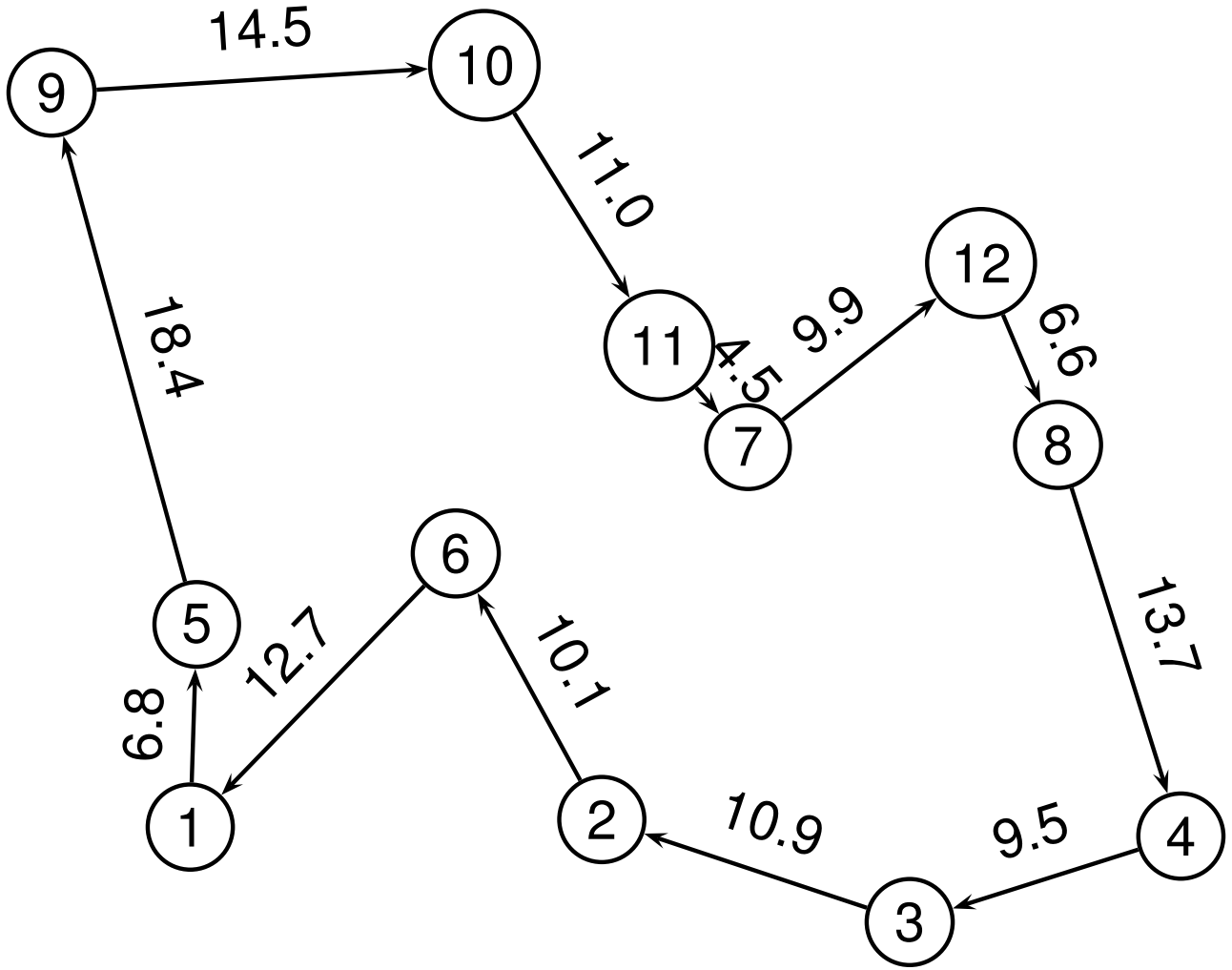
- Tant que $k < K$
  - Choisir au hasard un voisin $y \in V(x_c)$
  - $\delta \leftarrow f(y) - f(x_c)$
  - Si $\delta < 0$, $x_c = y$.
  - Sinon choisir r aléatoirement entre 0 et 1
    - Si $r < exp(-\delta/T)$, alors $x_c = y$
  - Si $f(x_c) < f(x^*)$, $x^* = x_c$.
  - $k \leftarrow k + 1$
- Réduire T

Jusqu'à ce que $T \leq T_f$

Exemple sur le voyageur de commerce



Meilleure solution trouvée



Longueur : 128.8

Remarques

- Les paramètres de la méthode doivent être ajustés.
- En particulier, la réduction de la température doit être spécifiée.
  - Soit δ_t une augmentation tyique de la fonction objectif.
  - Au début, on voudrait que cette augmentation soit acceptée avec probabilitié p_0 (typiquement, $p_0 = 0.999$)
  - A la fin, on voudrait que cette augmentation soit acceptée avec une probabilité p_f (typiquement, $p_f = 0.00001$)
  - Si on se permet de faire M mises à jour de la température, alors pour $m = 0, \ldots, M$,

$$T = -\frac{\delta_t}{\ln(p_0 + \frac{p_f - p_0}{M} m)}$$

Diversification : commentaires

Comment éviter de rester bloqué à un minimum local ?

- Appliquer l'algorithme à partir de plusieurs points de départ.
  - Comment choisir les points de départ ?
  - Comment éviter de "tirer dans le noir" ?
- Permettre à l'algorithme de remonter la pente : recuit simulé
  - Grimper sur la montagne pour trouver une autre vallée.
  - Comment décider quand on monte et quand on descend ?
- Changer la structure du voisinage : recherche à voisinages variables
  - Quels voisinages choisir ?

Heuristiques : remarques

- Les méthodes cherchant à échapper aux optimums locaux sont parfois appelées “meta-heuristiques”.
- De très nombreuses variantes existent dans la litérature.
- En général, leur succès dépend de la manière dont les propriétés du problème sont exploitées dans la structure de voisinage.
- VNS est l’une de plus simples à programmer.
- De nombreuses méthodes inspirées de la biologie ont également été suggérées et appliquées : algorithmes génétiques, algorithmes fourmis, etc.