

# Méta-Heuristiques

M.-J. Huguet

<https://homepages.laas.fr/huguet>

2019-2020

# Plan

---

1. **Contexte**
2. **Méthodes approchées**
  1. Heuristiques gloutonnes
  2. Méthodes de recherche locale
  3. Méthodes à population
3. **Conclusions et ouvertures**

# 1. Contexte :

## Résolution de problèmes combinatoires

---

- **Types de problèmes**

- **Décision**

- Existe-t-il une solution satisfaisant une certaine propriété ?
    - Résultat : OUI / NON

- **Optimisation**

- Parmi les solutions satisfaisant une certaine propriété, quelle est celle qui optimise une fonction objectif (fonction de coût)
    - Résultat : une solution de coût optimal

- Résoudre un problème d'optimisation

- Au moins aussi difficile que résoudre le problème de décision associé

# 1. Contexte :

## Résolution de problèmes combinatoires

---

- **Problèmes d'optimisation combinatoire NP-Difficile**

- Espace de recherche : espace exploré pour trouver la solution (optimale)
  - Solutions candidates ou réalisables
- Comment faire face à l'explosion combinatoire ie. la taille de l'espace de recherche ?

- **Méthodes génériques**

- Contenir l'explosion combinatoire en structurant et en élaguant l'exploration de l'espace de recherche → **méthodes exactes**
- Contourner l'explosion combinatoire en faisant des impasses (sans explorer tout l'espace de recherche) → **méthodes approchées**

# 1. Contexte :

## Résolution de problèmes combinatoires

---

- **Formalismes de modélisation**

- Prog. Linéaire en Nombres Entiers / Mixtes : ILP / MILP
- Programmation par Contraintes : CP
- Satisfiabilité Booléenne : SAT

- **Méthodes exactes**

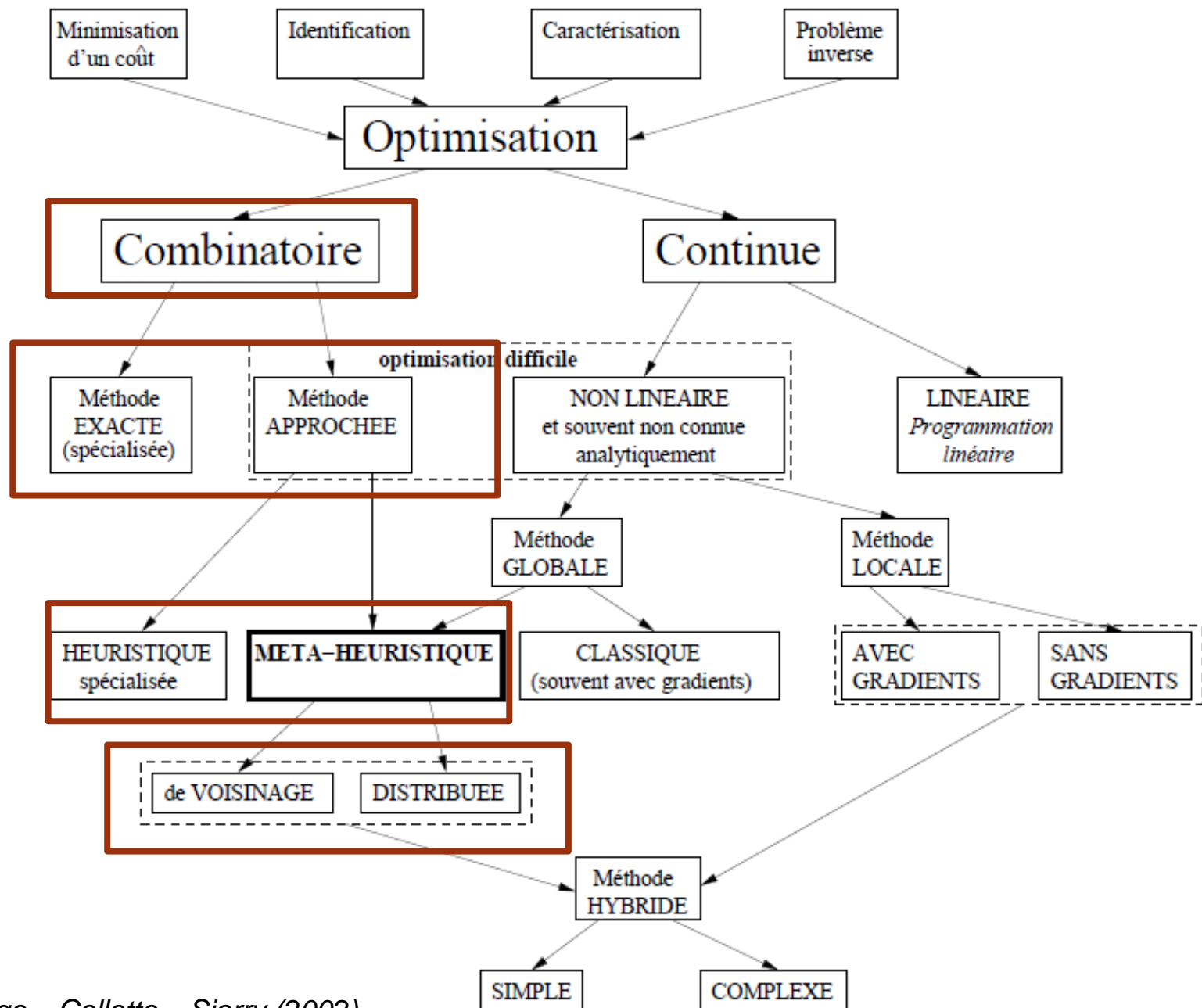
- Recherche arborescente
  - Séparation & élagage
    - Branch and Bound; Branch and Cut / Décomposition
    - Propagation, Redémarrage, Décomposition et reformulation,
  - Stratégie d'exploration
    - Heuristiques de séparation; ordre d'exploration des sommets

# 1. Contexte :

## Résolution de problèmes combinatoires

---

- **Caractéristiques des méthodes exactes**
  - Approches correctes et complètes
    - Fournissent la solution optimale (s'il existe des solutions)
  - Complexité exponentielle
    - Sont limitées à des problèmes de taille raisonnable
  - Différentes variantes des méthodes arborescentes
    - En liaison avec les formalismes de modélisation
- **Méthodes approchées ....**
  - en optimisation combinatoire
  - en optimisation continue
  - en apprentissage automatique



# Vocabulaire ...

---

- **Méthode exacte** : explore l'espace de recherche dans sa totalité (complète)
  - Optimum global
- **Méthode approchée** : explore une partie de l'espace de recherche
  - Optimum local
- **Méthode complète** : explore l'ensemble des solutions de façon exhaustive et systématique en structurant l'espace de recherche
- **Méthode incomplète** : explore seulement une sous-partie de l'ensemble des solutions en utilisant des heuristiques pour se guider vers les zones qui semblent plus prometteuses.
- **Méthode déterministe** : réalise toujours la même suite d'actions
- **Méthode stochastique** : effectue des choix aléatoires (probabilistes)



# Vocabulaire ...

---

- **Heuristique**
  - méthode de résolution spécialisée pour un problème
- **Méta-heuristique**
  - principe générique de résolution
- **Matheuristique**
  - Combinaison de méthodes heuristiques et de programmation mathématique
    - Ex : explorer un espace de recherche avec un algorithme de programmation dynamique
- **Hyper-heuristique**
  - Heuristique pour déterminer quelle heuristique appliquer
- **Nombreux travaux en cours :**
  - Intégration apprentissage automatique au sein de méthodes approchées pour « guider » l'exploration de l'espace de recherche

## 2. Méthodes approchées

---

- **Principe :**
  - Ne pas explorer tout l'espace de recherche
    - Contourner l'explosion combinatoire en faisant des impasses
- **Méthodes avec garantie de qualité / solution obtenue**
  - Algorithmes d'approximation
  - Trouver des solution avec garantie / optimum
- **Méthodes guidées par des heuristiques**
  - Trouver des « bonnes » solutions mais sans garantie / optimum
  - Complexité raisonnable

# Algorithmes d'approximation : exemple (1)

---

- **Algorithme de  $\rho$ -approximation :**

- Algorithme polynomial qui renvoie une solution approchée garantie au pire cas à un facteur  $\rho$  de l'optimum :
  - $Opt \leq Alg \leq \rho Opt$  (avec  $\rho > 1$ )
  - avec  $Opt$  la solution optimale et  $Alg$  la solution obtenue

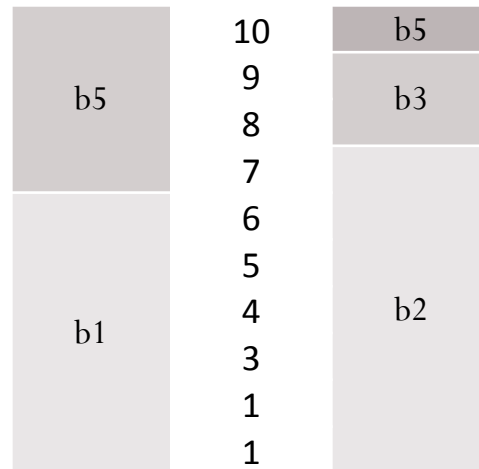
- **Problème de Bin Packing**

- un ensemble  $B$  de  $n$  objets (items) de « taille »  $w_i$ ,  $\forall b_i \in B$
- un entier  $C$  (capacité des sacs/bin)
- Un entier  $k$
- **Problème de décision** : existe-il un rangement des objets de  $B$  dans  $k$  sacs de capacité  $C$  ?  $\rightarrow$  problème NP-Complet
- **Problème d'optimisation** : quel est le nombre minimum de sacs de capacité  $C$  pour ranger les objets de  $B$  ?  $\rightarrow$  problème NP-difficile

# Algorithmes d'approximation : exemple (2)

- **Exemple**

- $B = \{b_1, b_2, b_3, b_4, b_5\}; w = \{6, 7, 2, 4, 1\}; c = 10$
- Existe-t-il un rangement de ces objets dans 2 sacs de capacité 10 ?
- Réponse :



# Algorithmes d'approximation : exemple (3)

---

- **Algorithme Next-Fit**

- Entrées : ensemble  $B = \{b_1, \dots, b_n\}$  d'objets, fonction  $w : B \rightarrow \mathbb{N}$ , entier  $c$
- Sortie : un entier  $k$

1.  $j \leftarrow 1$     // *indice des sacs*

2. Pour  $i$  de 1 à  $n$  faire

- si  $b_i$  peut être rangé dans sac  $S_j$  alors  $S_j \leftarrow S_j \cup \{b_i\}$
- sinon
  - $j \leftarrow j + 1$
  - $S_j \leftarrow \{b_i\}$

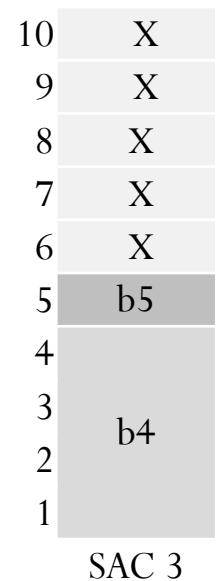
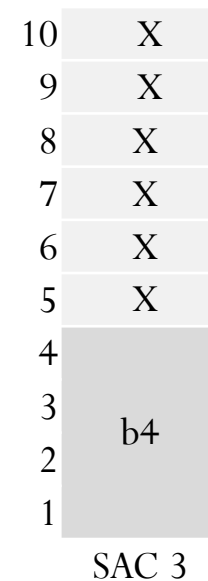
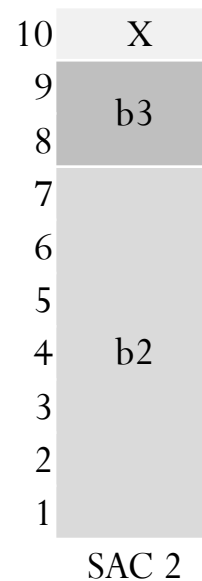
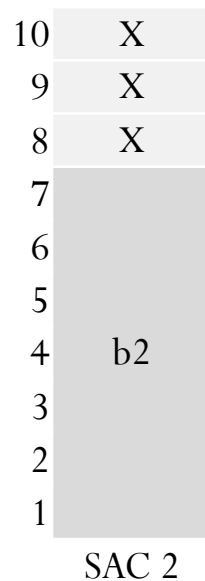
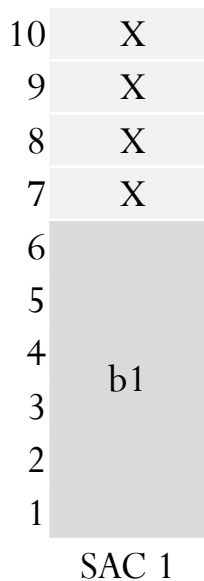
3. Retourner  $j$

- Complexité : linéaire dans le nombre d'objets

# Application Algorithme Next-Fit

- Exemple

- $B = \{b_1, b_2, b_3, b_4, b_5\}; w = \{6, 7, 2, 4, 1\}; c = 10$
- Minimiser le nombre de sacs ?



# Facteur d'approximation pour Next-Fit (1)

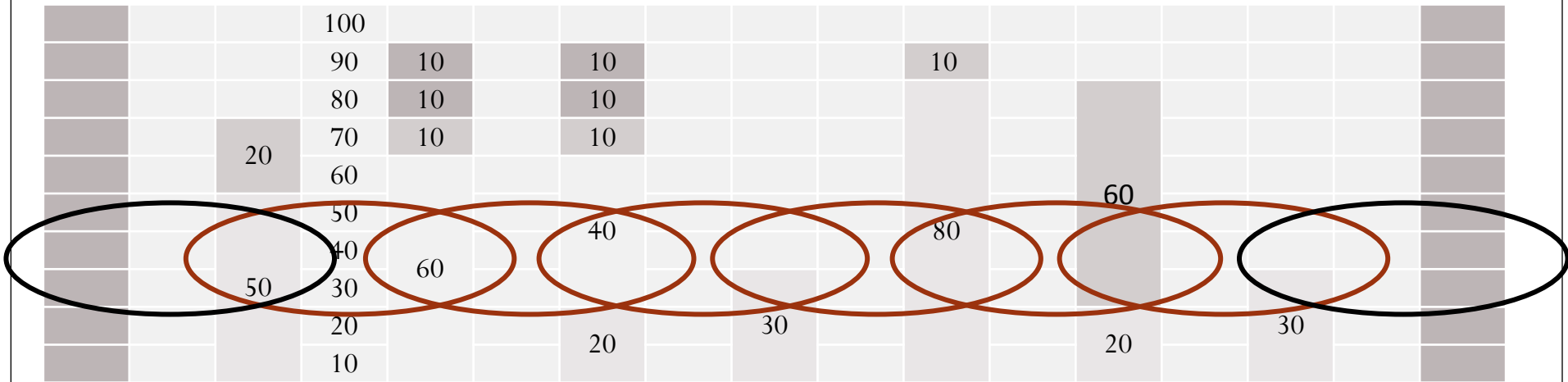
- Soit  $w(b_i)$  les tailles des objets,  $c$  la capacité des sacs et  $k$  la valeur retournée par l'algorithme Next-Fit
- Quel est le nombre minimal théorique de sacs ?  $\frac{\sum_{i=1}^n w(b_i)}{c}$
- Donc :  $Opt \geq \frac{\sum_{i=1}^n w(b_i)}{c}$  Borne inférieure
- Si on considère le fonctionnement de l'algorithme et regardant les sacs 2 par 2 :
  - $w = \{50, 20, 60, 10, 10, 10, 20, 40, 10, 10, 30, 80, 10, 20, 60, 30\}; c = 100$



- Somme des taille des objets dans 2 sacs consécutifs  $>$  capacité d'un sac

## Facteur d'approximation pour Next-Fit (2)

- Somme des tailles des objets sur toutes les paires de sacs



- Chaque paire a une taille  $>$  capacité (100) et il y a  $k - 1$  paires
  - Chaque élément est compté 2 fois (sauf 1<sup>ère</sup> et dernier sacs)
  - Ajouter des sacs fictifs en début et à la fin (supposés plein) : 2 paires de plus ( $k + 1$ ) et la taille de chaque objet est compté 2 fois
    - $2 \times \sum_{i=1}^n w(b_i) + 200 > 100 \times (k + 1)$
    - $2 \times \sum_{i=1}^n w(b_i) > 100 \times (k - 1)$
    - $\sum_{i=1}^n w(b_i) > \frac{100 \times (k-1)}{2}$



# Facteur d'approximation pour Next-Fit (3)

---

- On regroupe les inégalités :
  - $Opt \geq \frac{\sum_{i=1}^n w(b_i)}{100}$  et  $\sum_{i=1}^n w(b_i) \geq \frac{100 \times (k-1)}{2}$
- C'est à dire :
  - $Opt \geq \frac{\sum_{i=1}^n w(b_i)}{100} > \frac{k-1}{2}$
- D'où
  - $k < 2 \times Opt + 1$  et  $k \leq 2 \times Opt$
  - Facteur d'approximation : 2

# Pour aller plus loin sur le Bin Packing

---

- Autres algorithmes d'approximation pour le Bin Packing
  - First Fit Decreasing / Best-Fit Decreasing
    - Tri des objets dans l'ordre décroissant
    - Placer chacun dans le premier/meilleur sac qui peut le contenir
- Méthodes exactes

- **Objets avec plusieurs dimensions**

- **En pratique**

- Problèmes de rangement/remplissage (de caisses dans des camions, de fichiers sur des supports, ...)
- Problèmes de découpe

# Méthodes approchées

---

- **Principe :**
  - Ne pas explorer tout l'espace de recherche
  - *Méthodes avec garantie de qualité / solution obtenue*
    - *Algorithmes d'approximation*
    - *Trouver des solution avec garantie / optimum*
  - **Méthodes guidées par des heuristiques**
    - Trouver des « bonnes » solutions mais sans garantie / optimum
    - Complexité raisonnable

# Méthodes basées sur des heuristiques (1)

---

- **But : trouver une « bonne » solution mais sans garantie sur l'optimalité**
  - Avec une méthode de complexité raisonnable
  - Qui est robuste : fournit souvent une bonne solution
  - Qui est simple en œuvre
- **Grands principes :**
  - **Intensifier** l'exploration de zones prometteuses de l'espace de recherche
  - **Diversifier** pour découvrir de nouvelles zones

# Heuristique versus Métaheuristique (1)

---

- Pas de consensus sur des définitions précises. Le plus souvent :

- **Méthode heuristique**

- Détermine de bonnes solutions (c'est-à-dire presque optimales)
  - avec un coût de calcul raisonnable sans pouvoir garantir ni la faisabilité ni l'optimalité,
- Reflète une stratégie par rapport à une connaissance du problème
- Une heuristique est une méthode de résolution spécialisée pour un problème

- **Métaheuristique**

- processus de génération qui guide une heuristique
  - en combinant des concepts différents pour explorer l'espace de recherche afin de trouver efficacement des solutions quasi optimales
- Une métaheuristique est un principe générique à adapter pour chaque problème

# Heuristique versus Métaheuristique (2)

---

	Heuristique	Métaheuristique
<i>Domaine</i>	Problème d'optimisation	Optimisation combinatoire
<i>Entrée</i>	Données	Problème d'optimisation
<i>Coeur</i>	Algorithme	Ensemble de principes
<i>Sortie</i>	Solution	Algorithme

Classer les ratios poids/coût par ordre décroissant et sélectionner les variables correspondantes tant que la contrainte n'est pas saturée; ...

Mémoire des solutions visitées;  
Choisir parmi les a bonnes solutions;  
Ajuster la liste t des mouvements interdits dynamiquement;  
etc...

# Exploration de l'espace de recherche

---

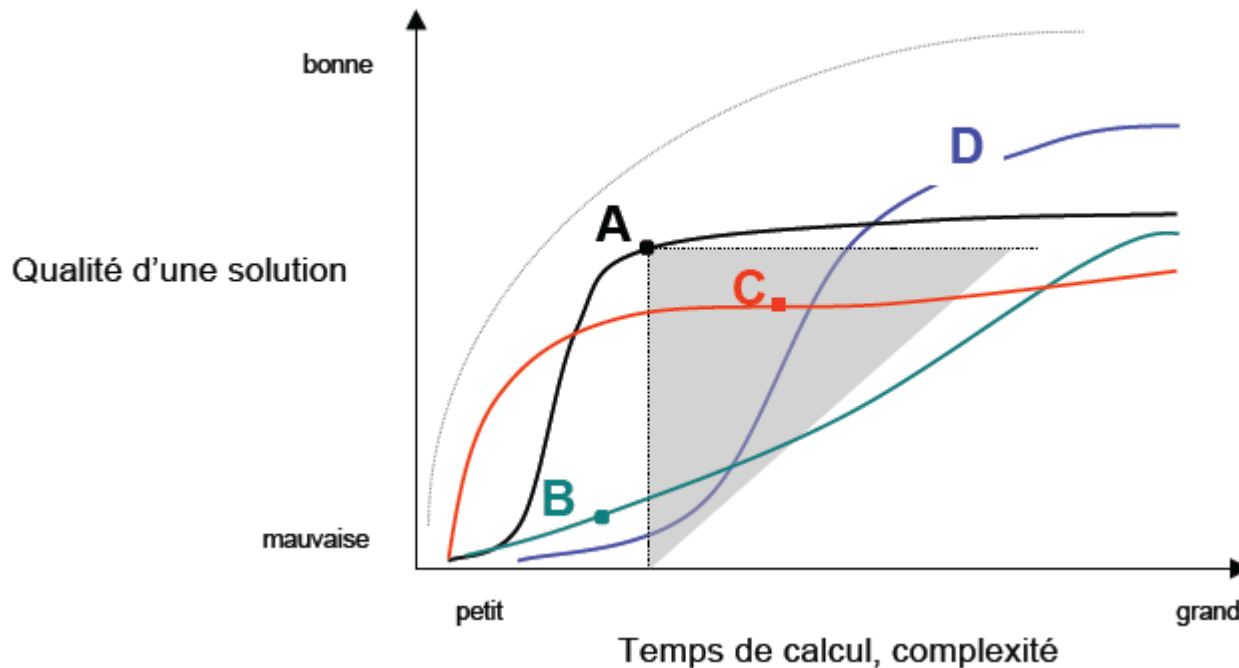
- **Principe des méthodes approchées**

But : contourner l'explosion combinatoire

- Intensification :
  - Accentuer l'exploration dans des zones prometteuses
- Diversification :
  - Découvrir de nouvelles zones
- Compromis intensification / diversification
  - Aléatoire
  - Guidé par la fonction objectif
  - Guidé par d'autres évaluations (solutions explorées, contraintes, paysage, ...)

# Evaluation d'une méthode approchée (1)

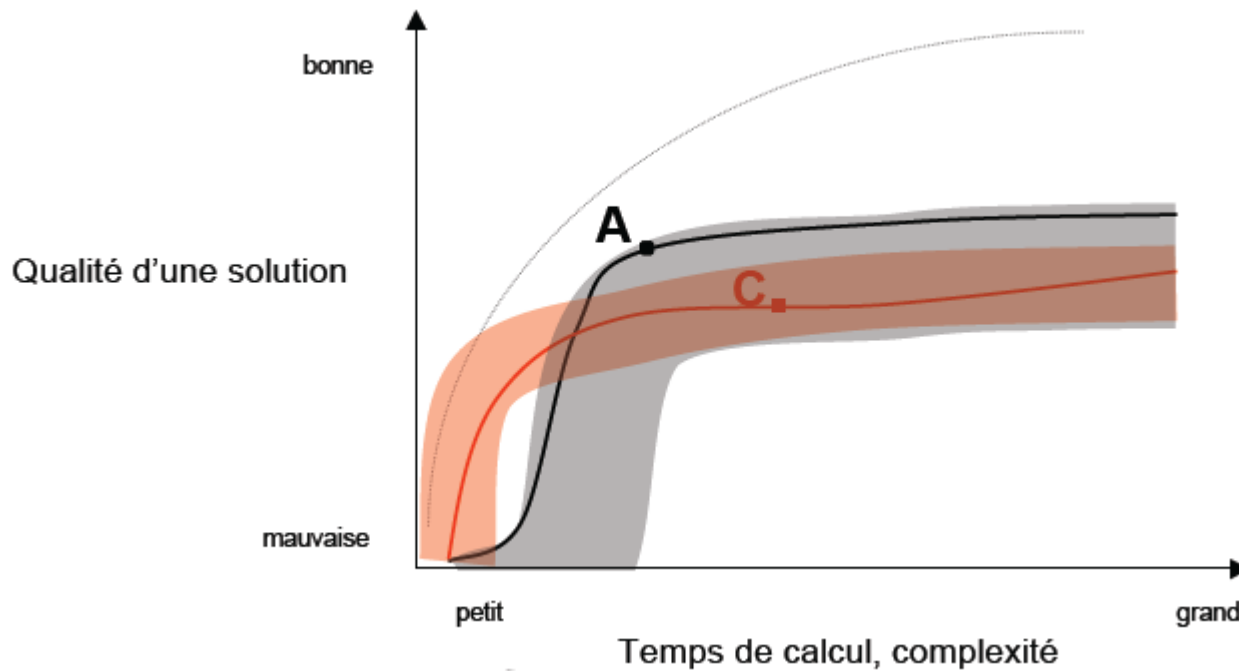
- **Qualité d'une heuristique / métaheuristique**
  - Comparaison : qualité de solution (pour un temps de calcul donné)





# Evaluation d'une méthode approchée (2)

- Qualité / temps de calcul



# Evaluation d'une méthode approchée (3)

- **Comparaison solution optimale / borne**

- Hypothèse : problème de minimisation

- **Solution obtenue (heuristique / métaheuristique)**

- Borne supérieure de l'optimum

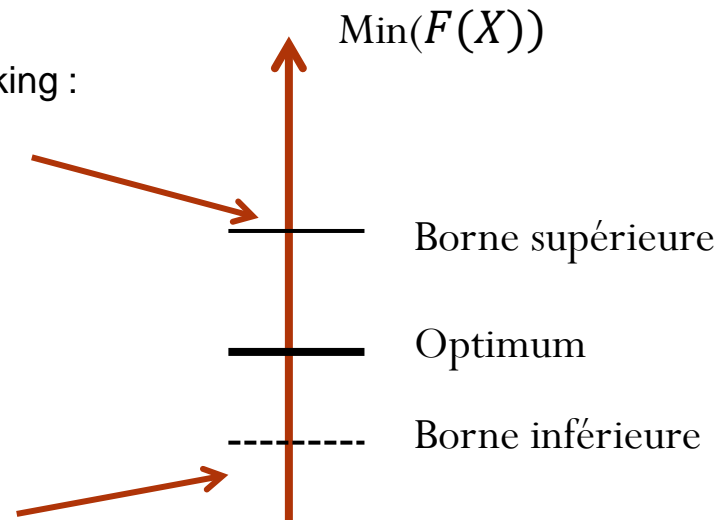
- **Comparaison**

- Optimum si connu
- Borne inférieure de l'optimum
  - Calcul analytique

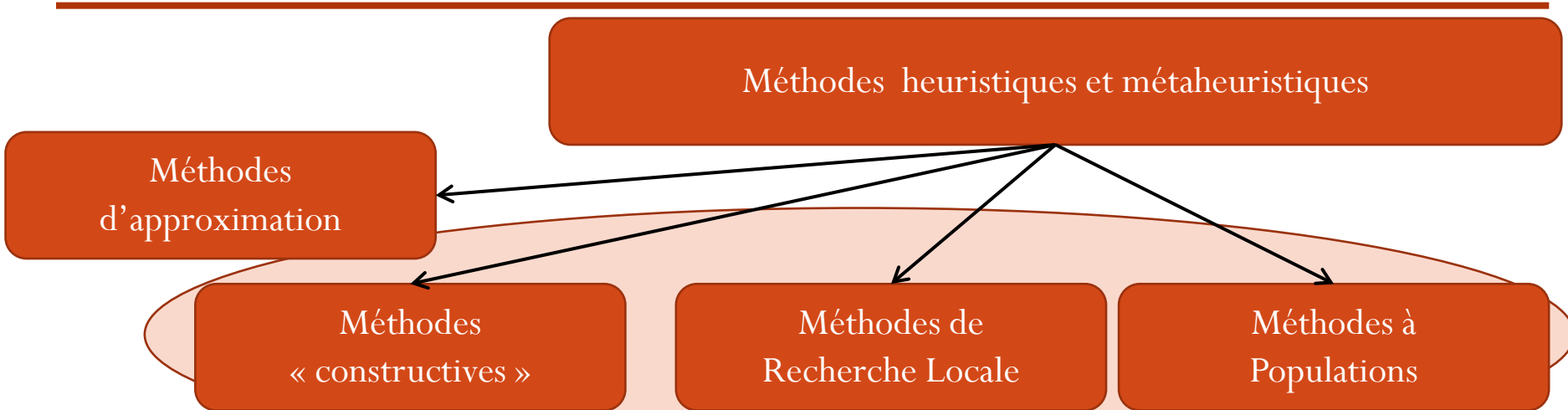
Ex pour le Bin-Packing :  
Algorithme Next-Fit

Ex pour le Bin-Packing :

$$\text{Borne inférieure} = \frac{\sum_{i=1}^n w(b_i)}{c}$$



# Différentes familles de méthodes approchées



- **Heuristiques « gloutonnes » (constructives)**

- Obtenir une solution à partir de « règles » de décision

- **Méta-Heuristiques**

- **Méthode à solution unique : Recherche locale**
  - Exploration du voisinage d'une solution
- **Méthodes à population de solutions**
  - Evolution d'un ensemble de solutions
- **Hybridation de méthodes**

Méthodes Approchées

# Intérêts des méthodes approchées

---

- **Quand utiliser une méthode approchée ?**

- Impossible de passer à l'échelle avec une méthode exacte
  - Contraintes de temps de calcul, de mémoire, de modélisation, ....

- **Une bonne méthode approchée ?**

- Complexité raisonnable
- Solution de bonne qualité, rarement de mauvaise solution
- Simple à mettre en œuvre ....

- **Evaluation d'une méthode approchée**

- Comparaison méthodes exactes sur des jeux de données de taille raisonnable
- Comparaison / bornes / temps de calcul / solutions ...

# Autres qualités pour une métaheuristique

---

- Robustesse
  - Variabilité dans les données (et les contraintes, ....)
- Simplicité de mise en œuvre
  - Paramétrage de la méthode
- Applicable à une grande variété de problèmes
  - Optimisation combinatoire
  - Optimisation continue
- Attractivité de la métaphore
  - Recuit simulé
  - Recherche Tabou
  - Algorithmes génétiques
  - Colonies de Fourmis
  - Systèmes immunitaires artificiels, essaims particulaires, scatter search, .....

# (Bilan) sur les caractéristiques des méthodes

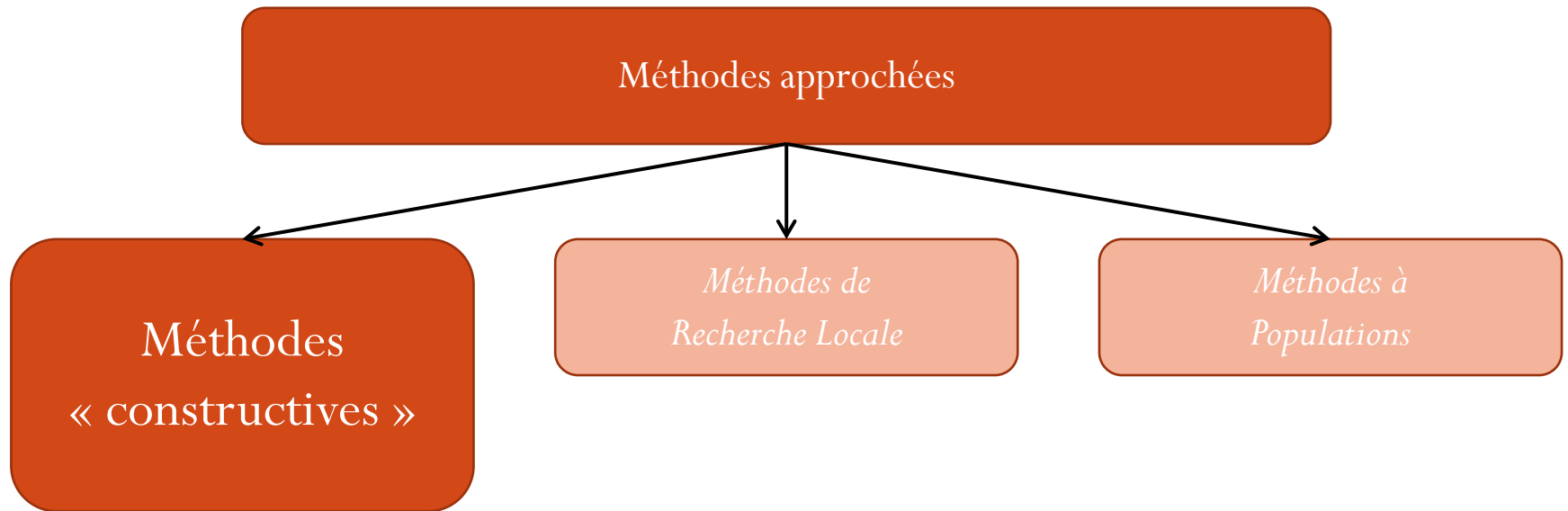
---

- **Résolution de problèmes NP-Complets / NP-Difficiles**
- **Méthodes exactes**
  - Fournissent solution optimale (si existe)
  - Complexité exponentielle : sont limitées à des instances de taille raisonnable
- **Méthodes approchées**
  - Fournissent une solution réalisable mais pas nécessairement optimale
  - Complexité « correcte »
- **En pratique**
  - Certaines instances peuvent être faciles à résoudre
    - Comprendre pourquoi : données ? contraintes ? cas particulier polynomial ?
  - Certains problèmes NP-difficiles admettent des approximations polynomiales
    - Algorithmes polynomiaux permettant le calcul d'une solution avec erreur bornée par rapport à la solution optimale

## Section 3. Méthodes approchées

---

- **Heuristiques gloutonnes**



# Principes d'une Recherche Gloutonne

---

- **Méthode naïve**

**Greedy Search**

- Partir d'une affectation vide des variables
- Tant qu'il y a des variables non affectées
  - Choisir une variable
  - Lui affecter une valeur

**Méthode « Constructive »**

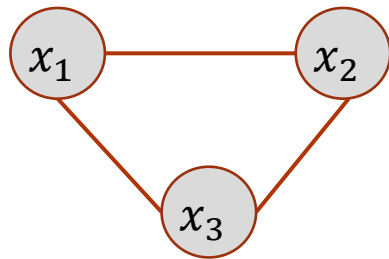
- Les choix successifs ne sont pas remis en cause
- Une branche d'une recherche arborescente (sans backtrack)
- Les choix effectués doivent garantir l'admissibilité de la solution obtenue (ie. respect des contraintes)
- **Peut-on toujours le garantir ?**



# Exemple heuristique gloutonne (1)

- Coloration de graphe

{bleu, rouge}      {bleu, rouge, vert}



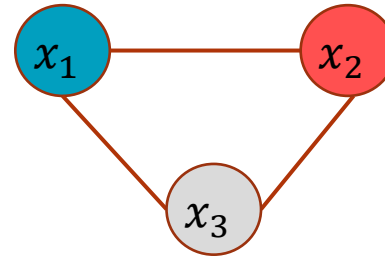
{bleu, rouge}

Minimiser nombre de couleurs

**Solution pas toujours  
admissibles**

Ordre Instanciation :  $x_1, x_2, x_3$

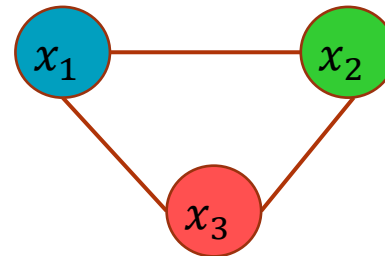
{bleu, rouge}      {bleu, rouge, vert}



{bleu, rouge}

Ordre Instanciation :  $x_1, x_3, x_2$

{bleu, rouge}      {bleu, rouge, vert}



{bleu, rouge}

# Exemple heuristique gloutonne (2)

- Planification

activités	durée	ressource
A1	1	1
A2	1	2
A3	2	1
A4	1	2
A5	2	1

$$A_1 < A_2 < A_5$$

$$A_3 < A_4$$

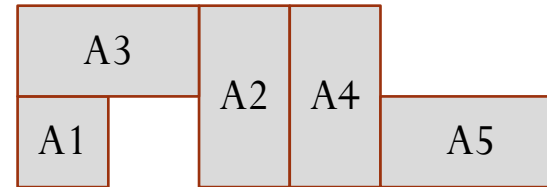
Quantité de ressource disponible : 2

Minimiser durée totale

**Solutions admissibles**

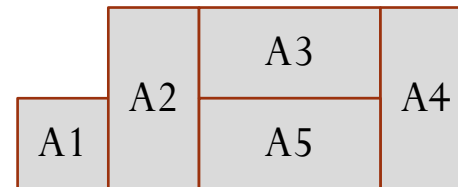
**Valeurs différentes / objectif**

Instanciación :  $A_1; A_3; A_2; A_4; A_5$



Durée Totale = 6

Instanciación :  $A_1; A_2; A_3; A_5; A_4$

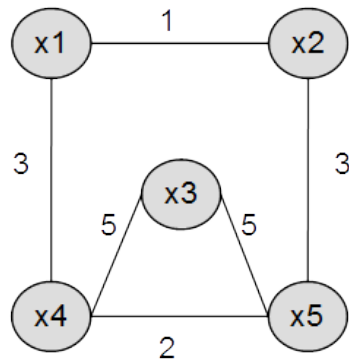


Durée Totale = 5

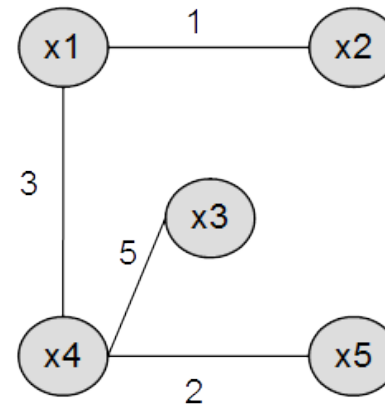
# Exemple heuristique gloutonne (3)

- **Arbres couvrants de cout minimal**

- Soit un graphe non orienté et pondéré
- Trouver un arbre couvrant de poids minimal :
  - Sélectionner les arêtes telles que la somme des poids des arêtes soit minimale



Graphe initial



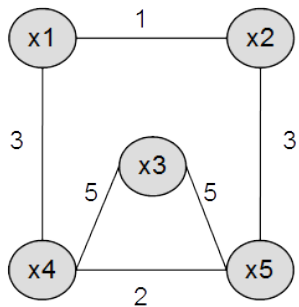
Arbre couvrant de cout minimal (11)

# Exemple heuristique gloutonne (4)

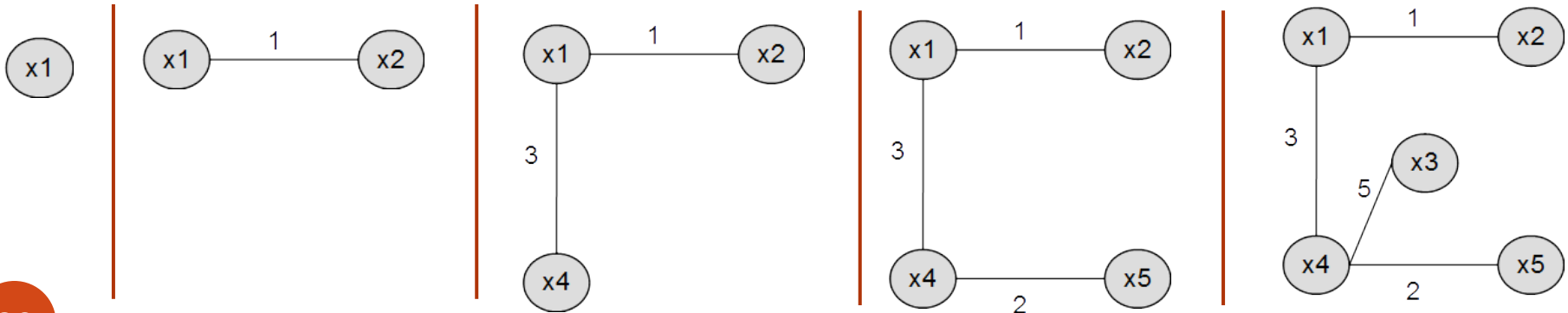
- **Arbres couvrants de cout minimal**

- Algorithme de Prim : ajout progressif d'arêtes

- arbre = graphe connexe avec un nombre minimal d'arêtes
- Maintenir un graphe connexe à chaque itération en ajoutant une arêtes pour connecter la partie connexe aux sommets non encore couverts



Algorithme glouton optimal (propriétés math du problème)



# Caractéristiques d'une Recherche Gloutonne

---

- **Qualité de la solution / fonction objectif**
  - Dépend de l'ordre sur les variables et des choix de valeurs
  - **Pas de garantie d'optimalité**
  - Obtention d'une **borne supérieure** de la valeur optimale (minimisation)
- **Ordre d'instanciation**
  - Quelle variable choisir ?
  - Quelle valeur lui affecter ?
  - **Principe général**
    - Choix de variable : la plus importante d'abord (fail-first)
    - Choix de valeur : celle ayant le plus de chance de conduire à une (bonne) solution (succeed-first)

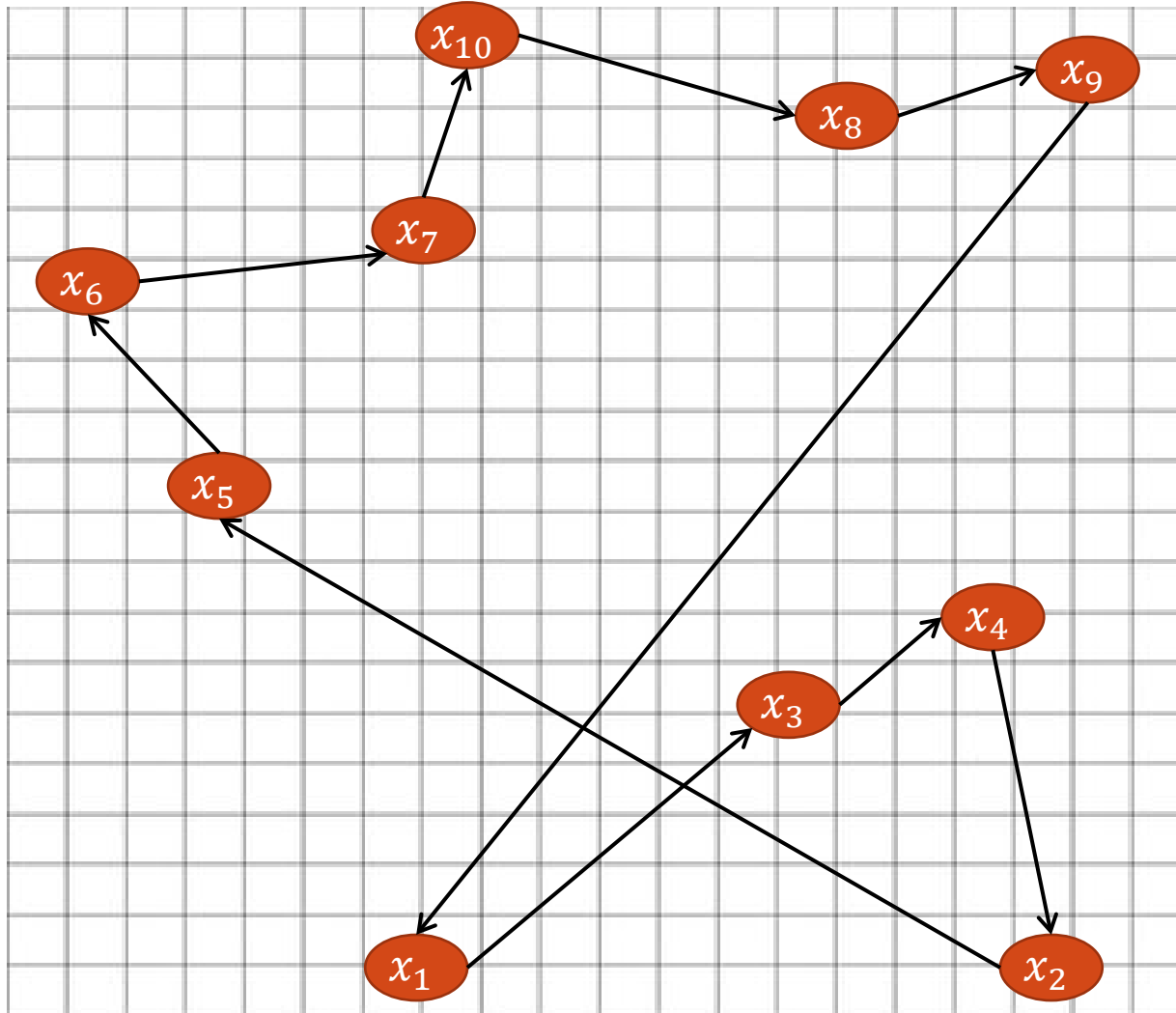
# Application 1 : TSP

---

- **Heuristique : Plus proche Voisin**

- Choisir un sommet de départ
- Tant qu'il reste des sommets non traités
  - Connecter le dernier sommet atteint au sommet libre le plus proche
- Relier le dernier sommet au sommet initial
- Données :
  - Graphe ( $n$  sommets) avec matrice de distance (2 à 2)
- Résultat : cycle (permutation des sommets)
- Nombre de solutions :  $(n - 1)!$
- Complexité heuristique :  $(O(n^2))$ 
  - $n$  itérations
  - À chaque itération : trouver le plus proche ( $O(n)$ )

# Application 1 : TSP



Borne supérieure de la solution optimale

Garantie de qualité (si symétrique et inégalités triangulaires)

# Application 1 : TSP

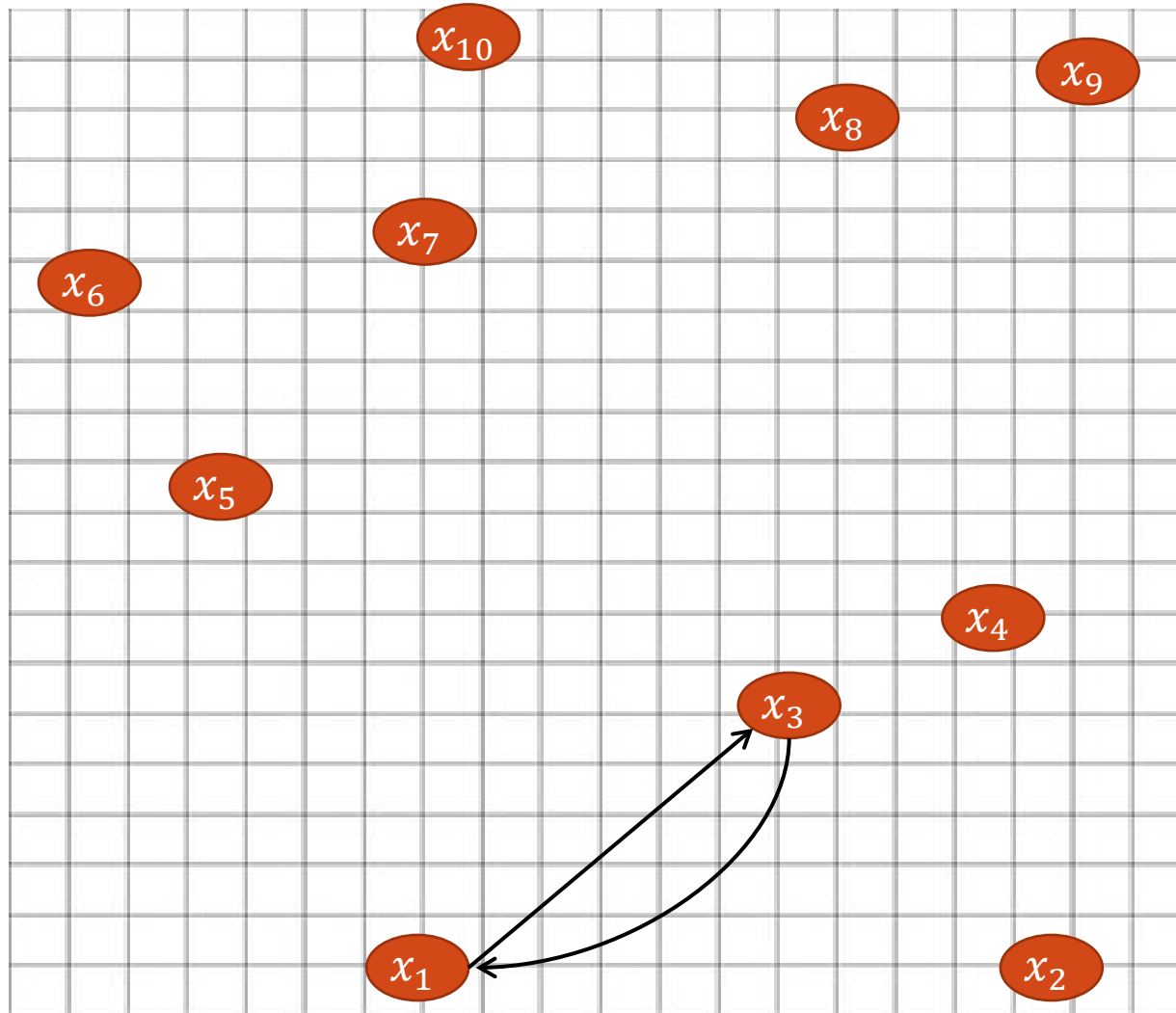
---

- **Autre Heuristique : Insertion dans un cycle**
  - Choisir un sommet de départ
  - A chaque étape : il existe un cycle
    - Insérer dans ce cycle le sommet minimisant un critère
      - Insertion du plus proche voisin : insérer dans le cycle le sommet le plus proche de ceux déjà présents (nearest insertion) – après le plus proche
      - Insertion du voisin à moindre coût : insérer le sommet engendrant la plus petite augmentation de la longueur du cycle (cheapest insertion)
- Complexité heuristique d'insertion :
  - $n$  itérations
  - À chaque itération :
    - Nearest insertion :  $O(n)$
    - Cheapest insertion :  $O(n^2)$

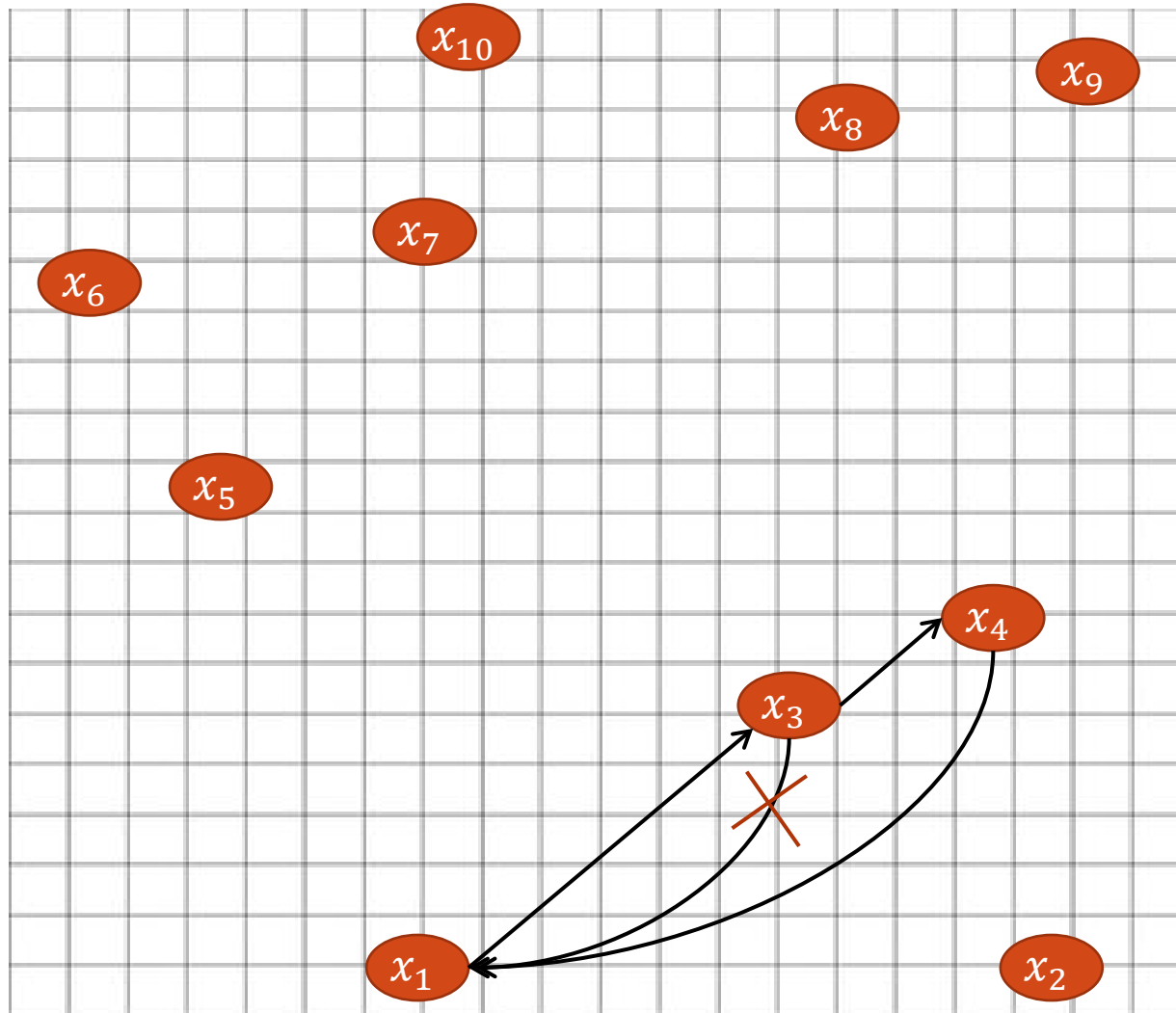


# Application 1 : TSP

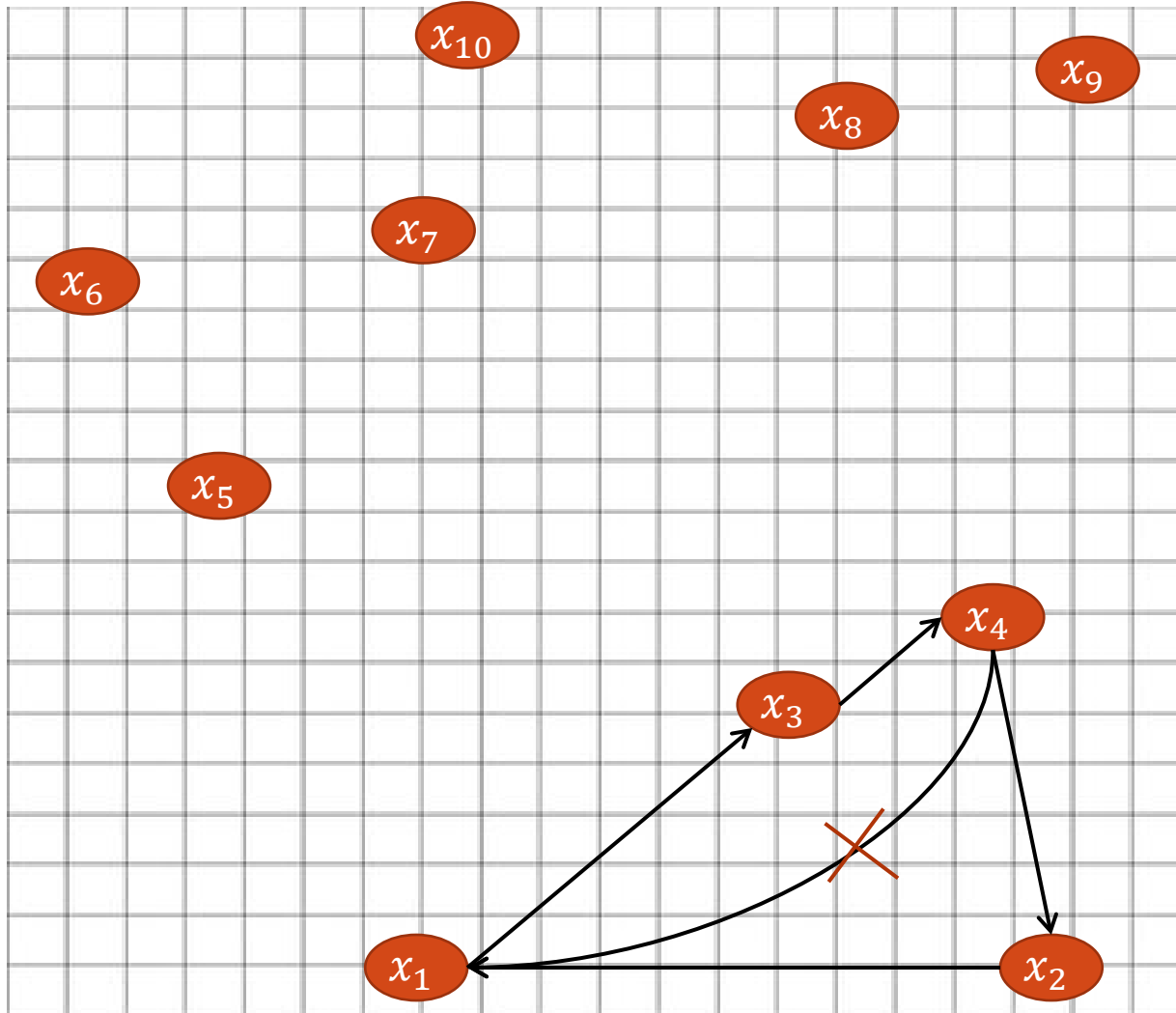
---



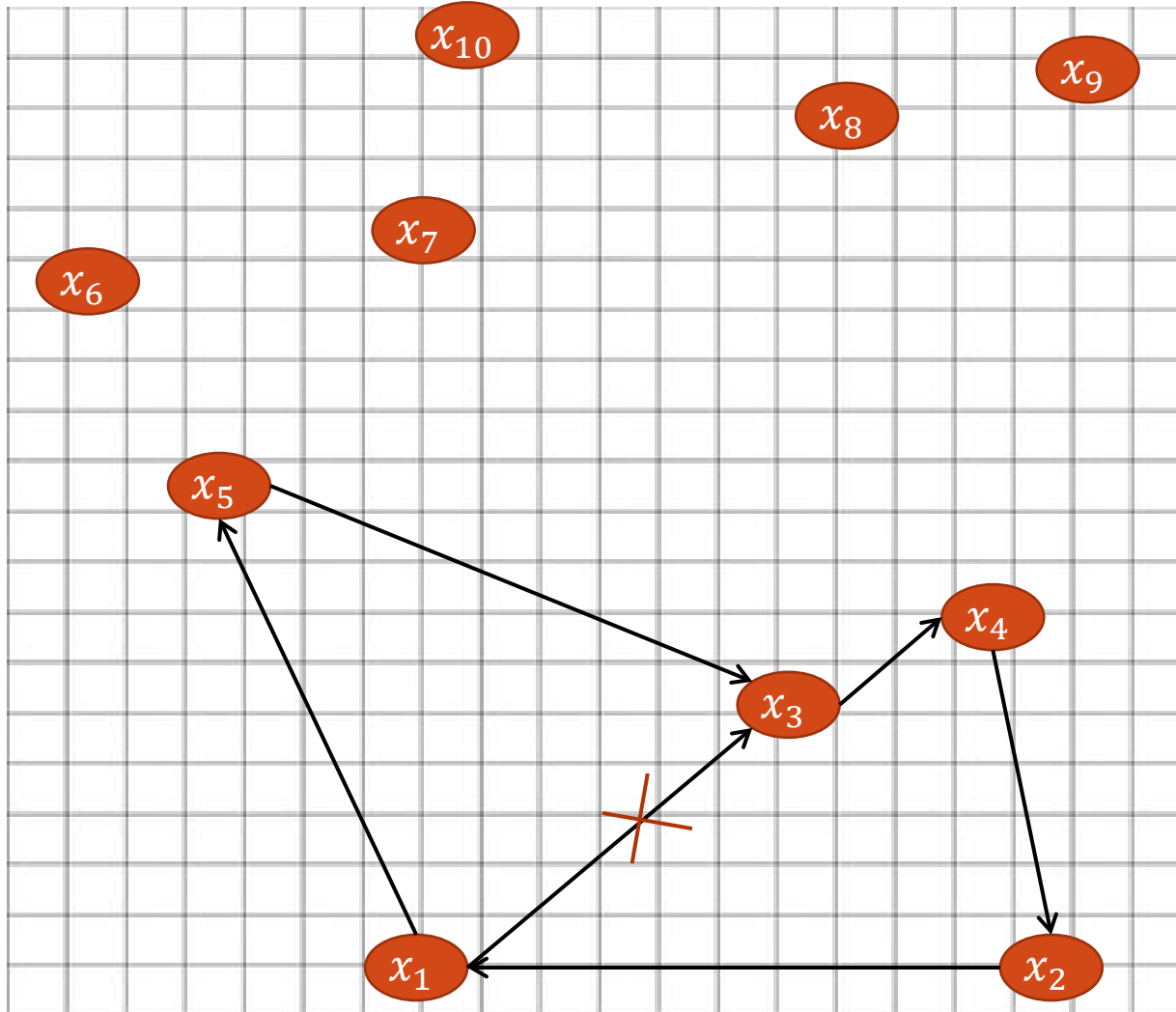
# Application 1 : TSP



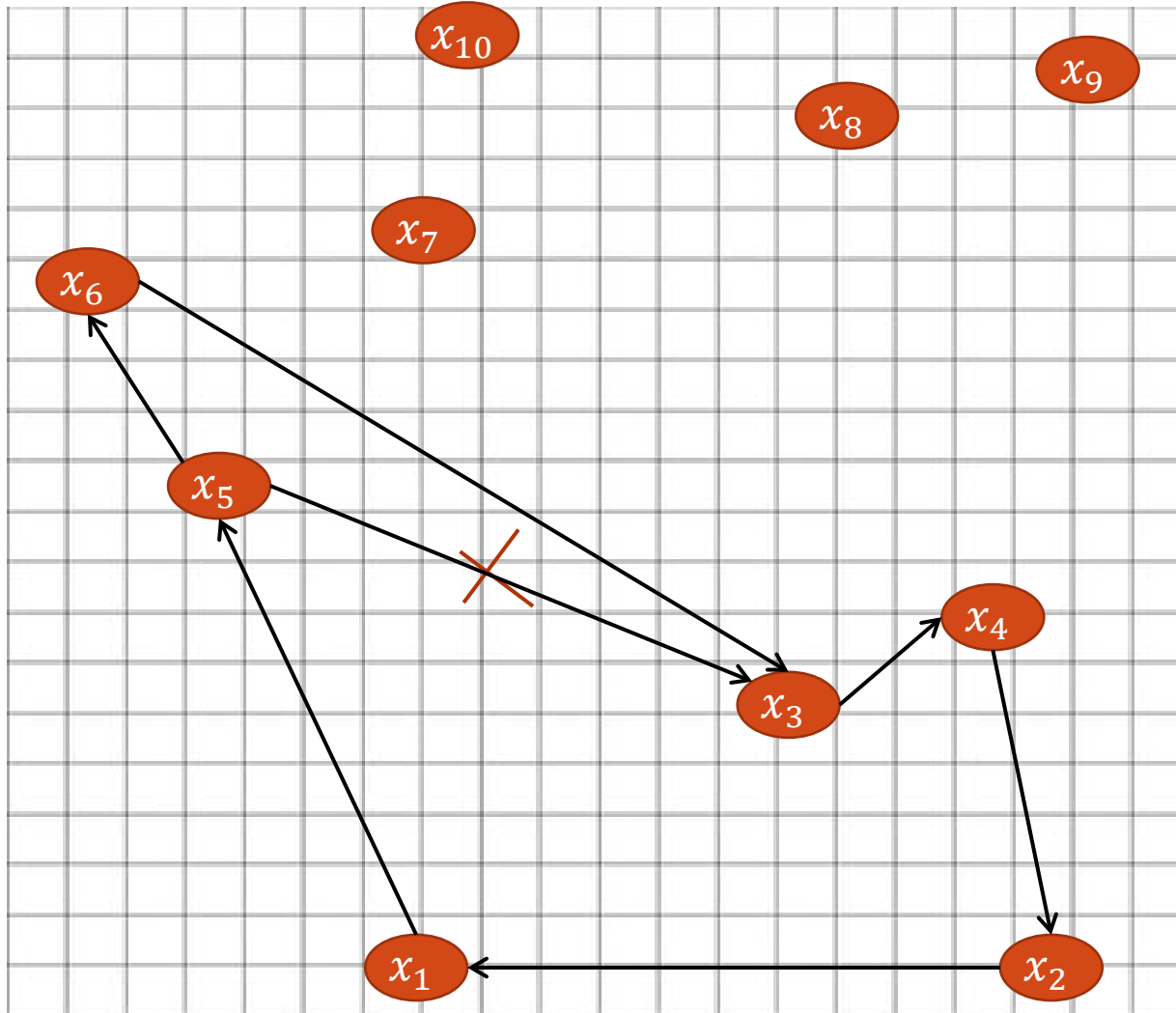
# Application 1 : TSP



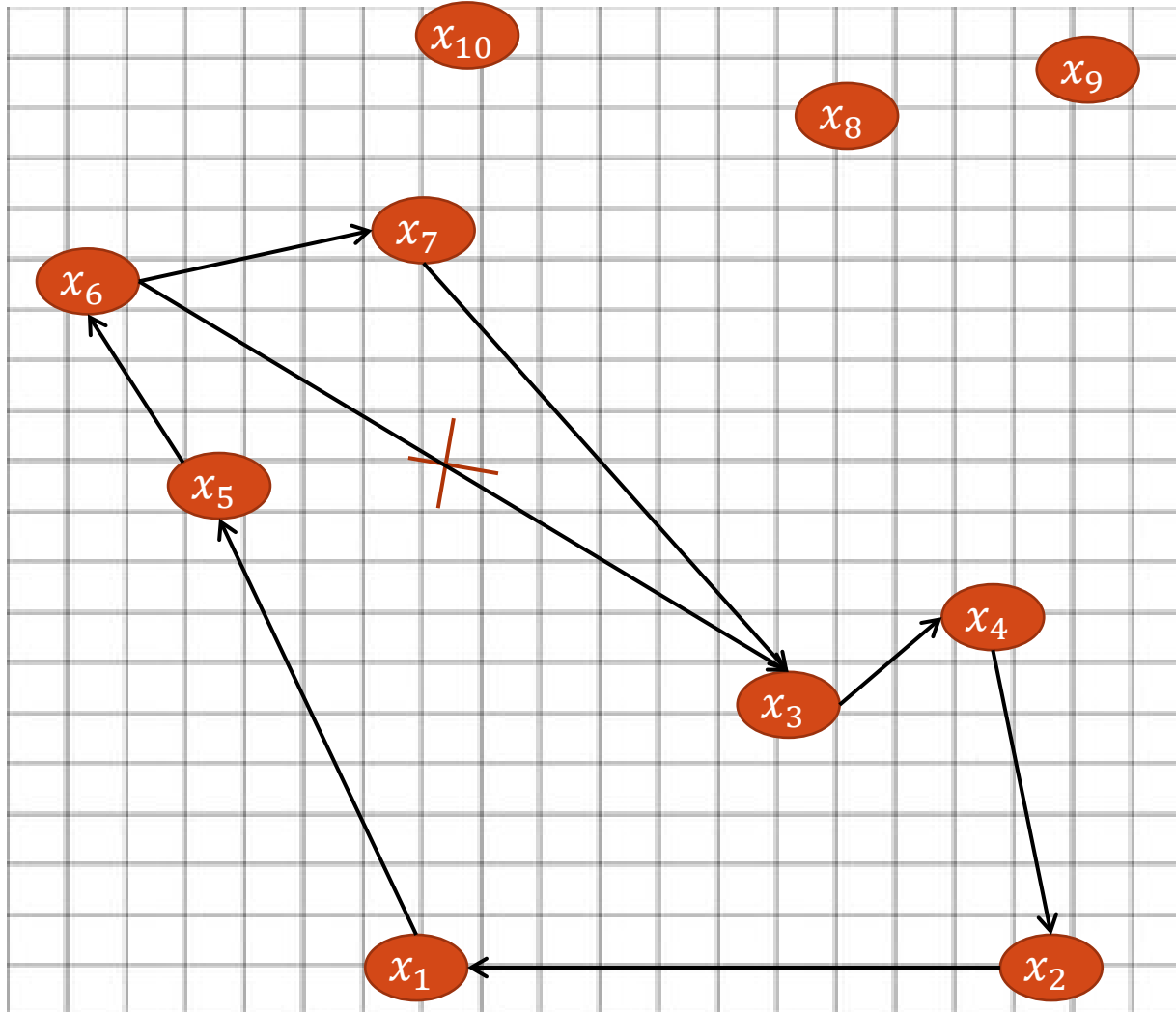
# Application 1 : TSP



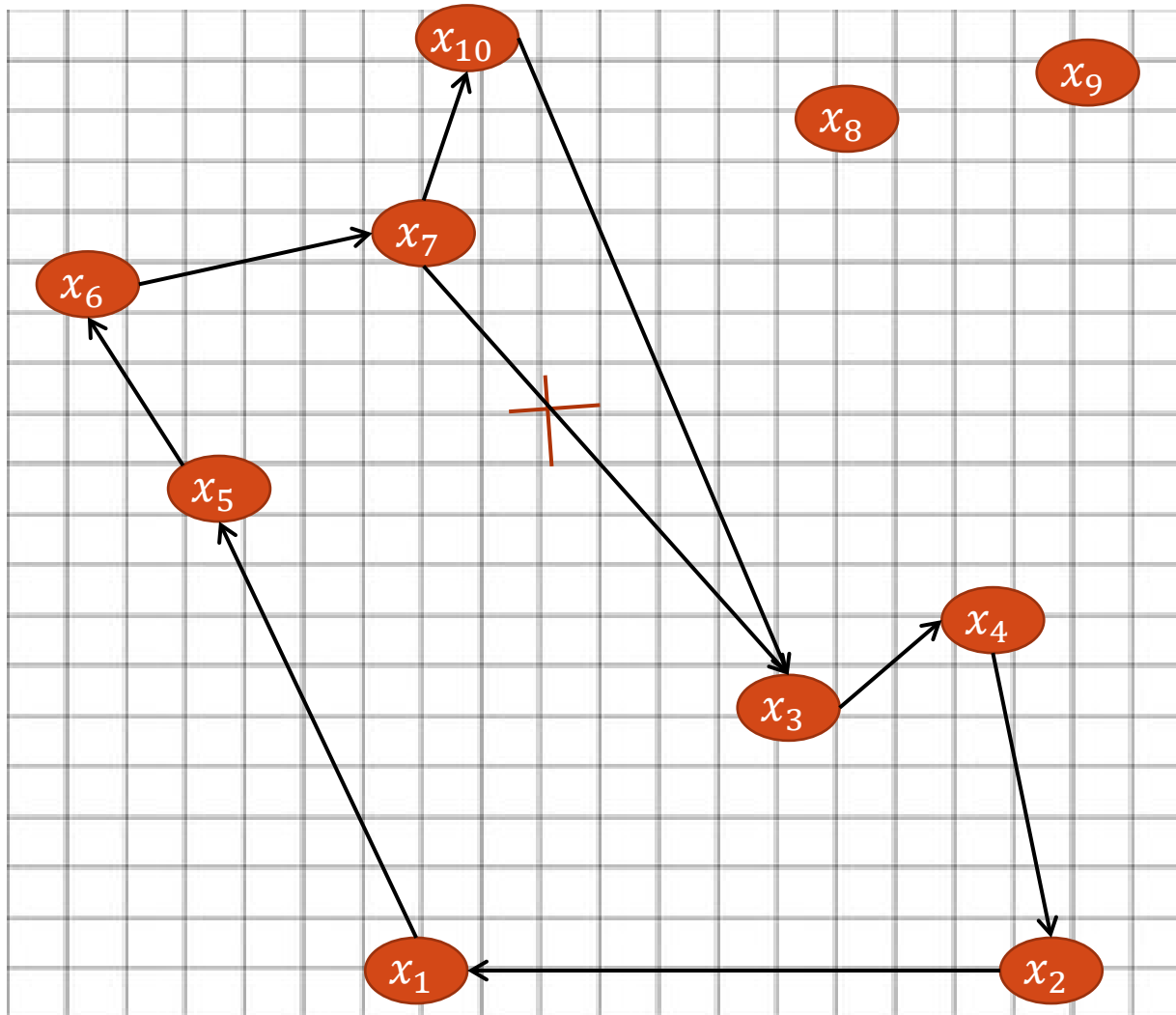
# Application 1 : TSP



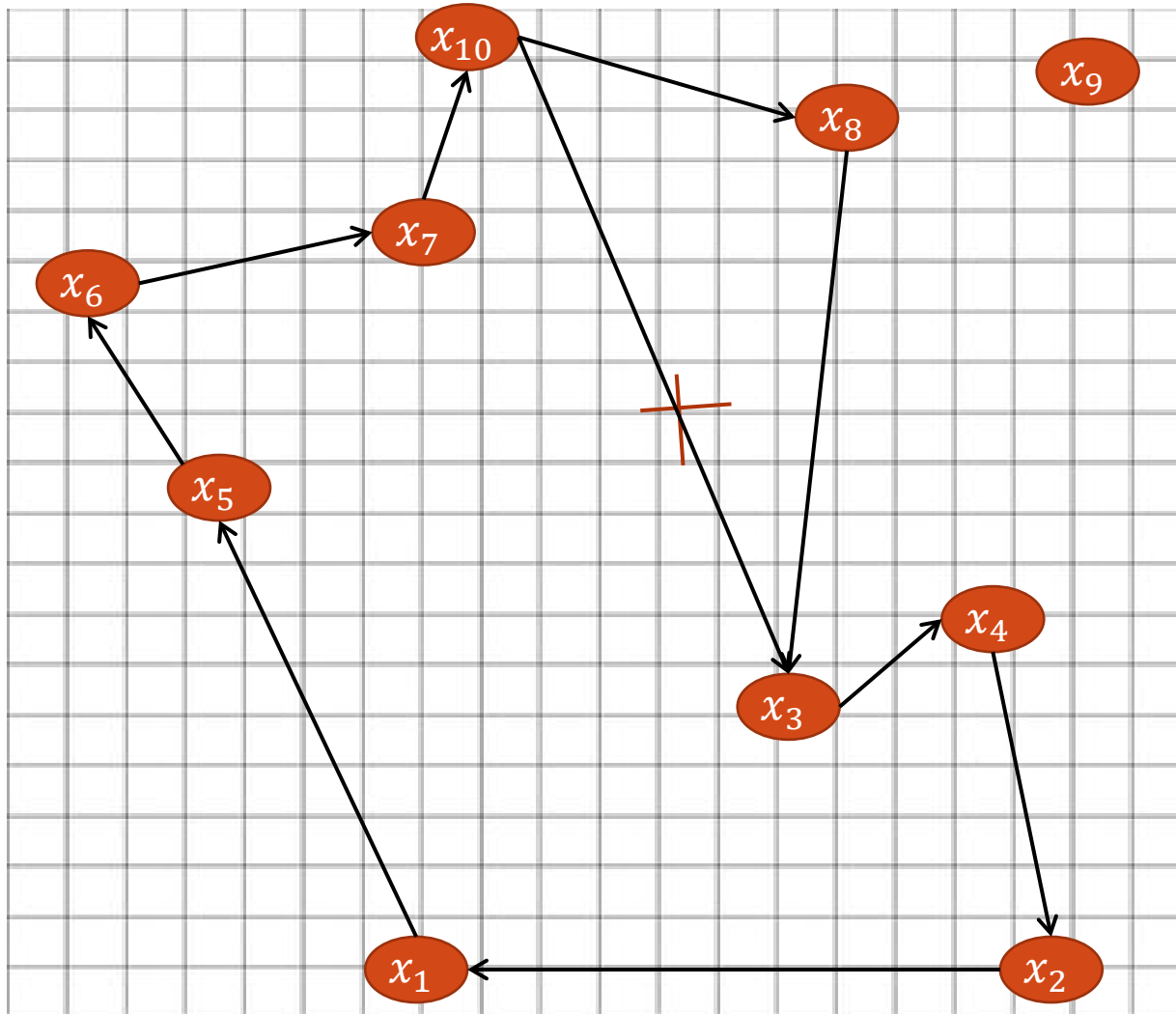
# Application 1 : TSP



# Application 1 : TSP

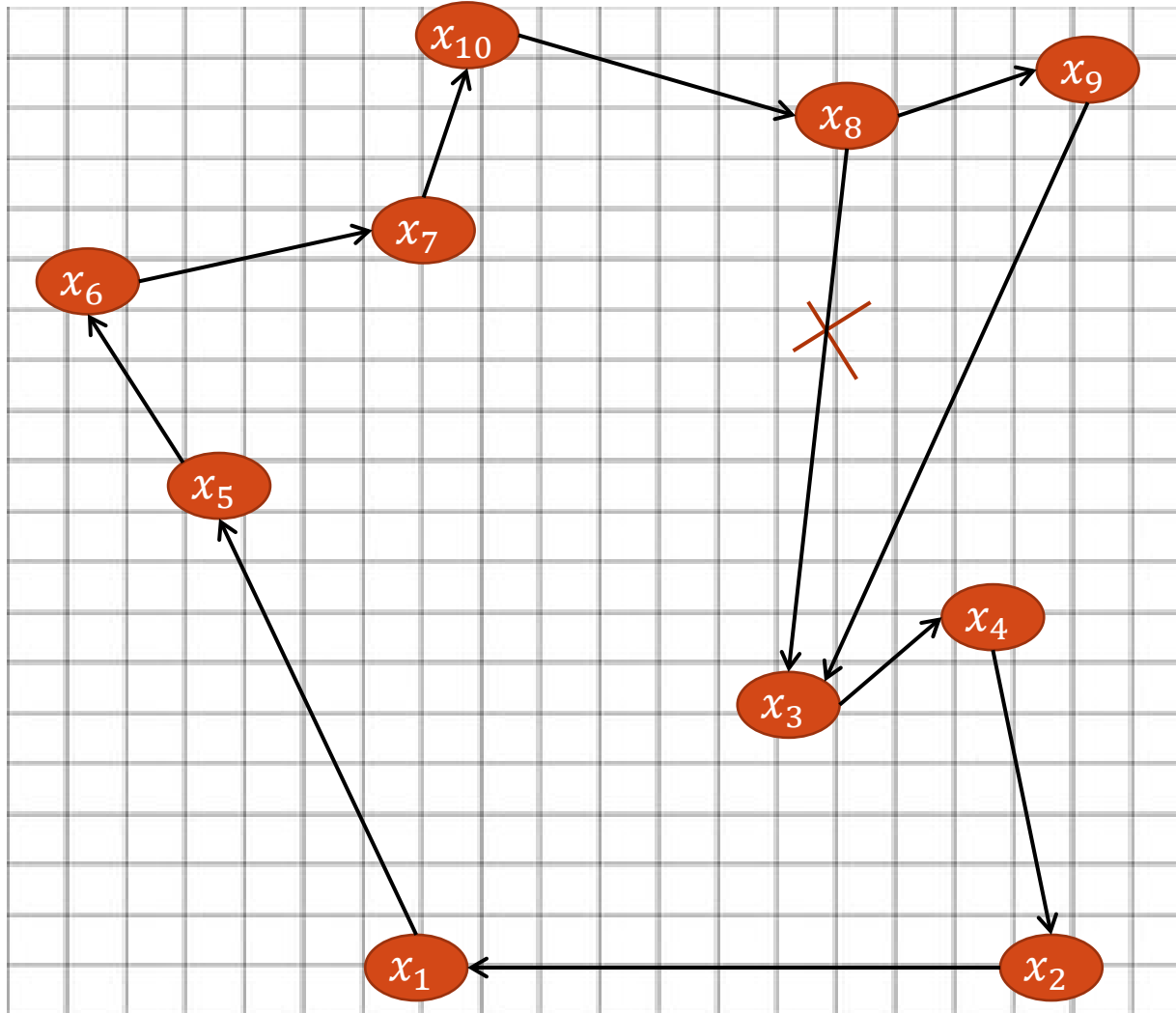


# Application 1 : TSP

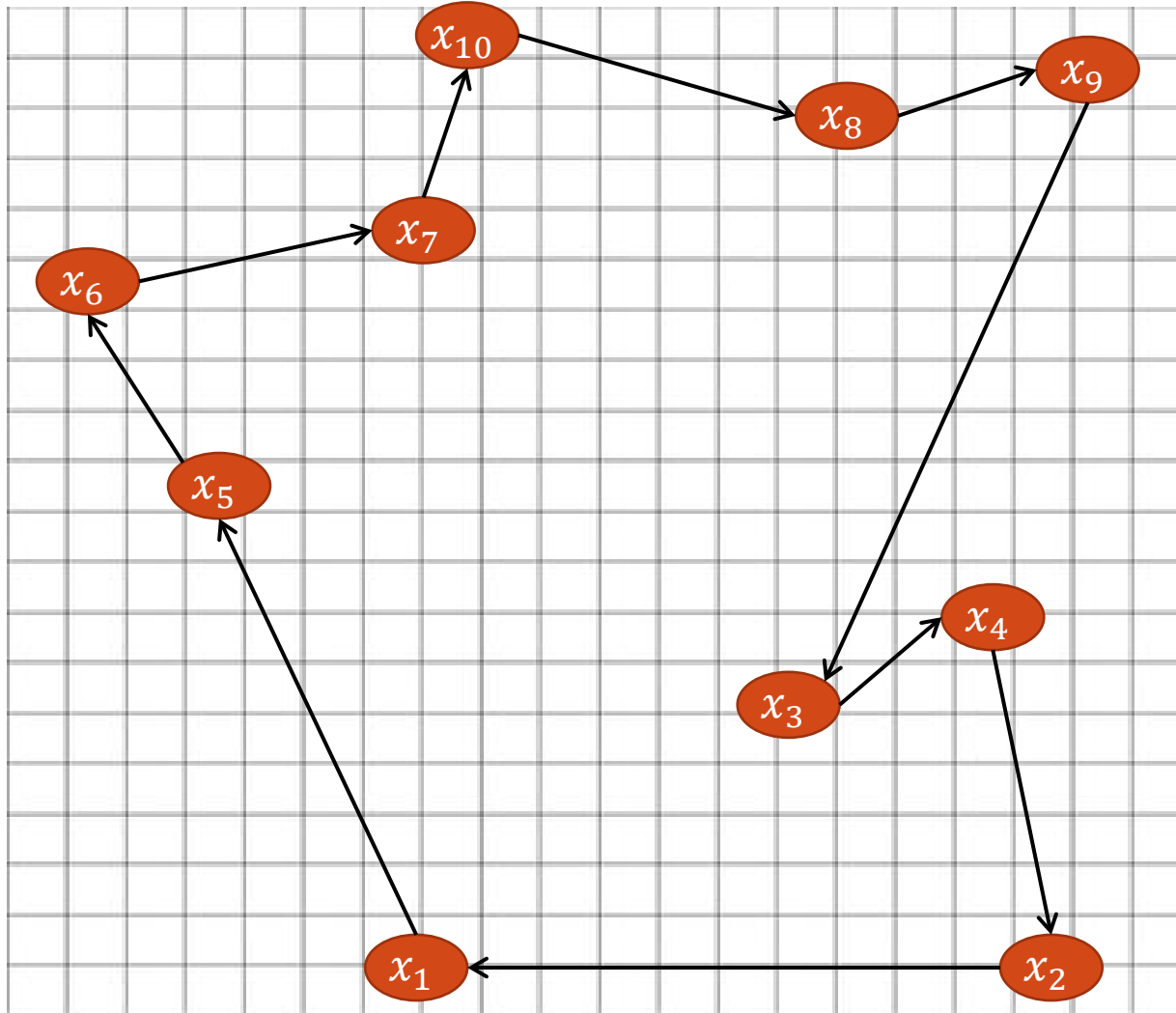




# Application 1 : TSP



# Application 1 : TSP



Borne supérieure de la solution optimale

Garantie de qualité (si symétrique et inégalités triangulaires)

# Application 2 : Problème du Sac à dos (1)

## • Problème :

## Knapsack Problem

- Un ensemble de  $n$  objets  $N = \{a_1, \dots, a_n\}$
- A chaque objet :
  - Un poids  $w_i$  et une utilité  $u_i$
- Un sac à dos dont le poids total ne doit pas dépasser la capacité  $W$
- Sélectionner les objets pour maximiser l'utilité
- Exemple sac de randonnée de capacité maximale 3 kg
  - Remplir le sac avec les objets les plus utiles

	Utilité	Poids (kg)
carte	10	0,2
gourde	8	1,5
2e gourde	3	1,5
pull	6	1,2
Kway	2	0,5
fromage	2	0,6
fruits secs	4	0,5

# Exemple 2 : Problème du Sac à dos (2)

---

- **Heuristique par intérêt décroissant**

- Calculer pour chaque objet le ratio Utilité / Poids :  $\frac{u_i}{w_i}$
- Trier les objets par ordre décroissant sur le ratio  $\frac{u_i}{w_i}$
- Variable booléenne : prendre ou pas un objet
- Objets sélectionnés :  $S = \emptyset$ 
  - Parcourir les objets dans l'ordre
    - Soit  $i$  l'objet courant
    - Si  $w(S) + w_i \leq W$  alors prendre l'objet  $i$  :  $S = S \cup \{i\}$
- Complexité :  $O(n)$ 
  - Calcul du ratio pour tous les objets ( $O(n)$ ) et tri ( $O(\log(n))$ )
  - Parcours de chaque objet et vérification du poids total ( $O(n)$ )

# Application 2 : Problème du Sac à dos (3)

- Application

	Utilité	Poids	Ratio	Ordre
carte	10	0,2	50,00	1
gourde	8	1,5	5,33	3
2e gourde	3	1,5	2,00	7
pull	6	1,2	5,00	4
Kway	2	0,5	4,00	5
fromage	2	0,6	3,33	6
fruits secs	4	0,5	8,00	2

- $W(S) = 0$
- $i = \text{Carte} : W(S) + w(i) = 0,2 \leq 3 \rightarrow \text{OK}$
- $i = \text{Fruits secs} : W(S) + w(i) = 0,2 + 0,5 = 0,7 \leq 3 \rightarrow \text{OK}$
- $i = \text{Gourde} : W(S) + w(i) = 0,7 + 1,5 = 2,2 \leq 3 \rightarrow \text{OK}$
- $i = \text{Pull} : W(S) + w(i) = 2,2 + 1,5 = 3,7 > 3 \rightarrow \text{NON}$
- $i = \text{Kway} : W(S) + w(i) = 2,2 + 0,5 = 2,7 \leq 3 \rightarrow \text{OK}$
- $i = \text{Fromage} : W(S) + w(i) = 2,7 + 0,6 = 3,3 > 3 \rightarrow \text{NON}$
- $i = \text{2e gourde} : W(S) + w(i) = 2,7 + 1,5 = 4,2 > 3 \rightarrow \text{NON}$
- **Utilité** =  $10 + 8 + 2 + 4 = 24$

**Borne inférieure / utilité max**

# Exercice : Coloration de sommets

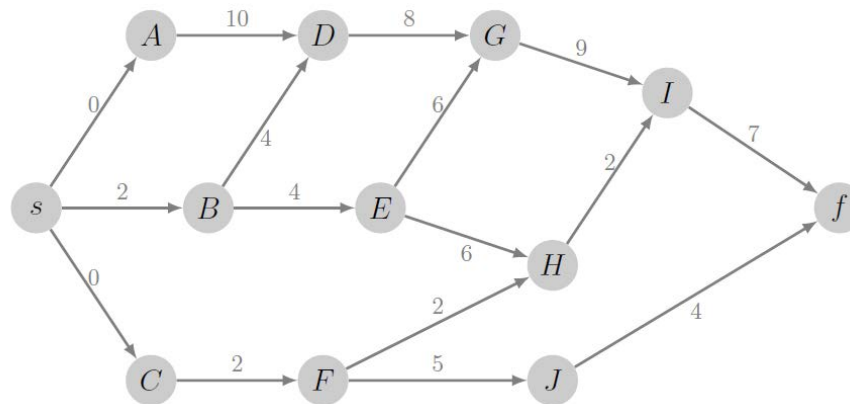
- Planifier sur la semaine les rattrapages de 6 étudiants et de 6 UF avec 2 créneaux par jour tout en minimisant le nombre de créneaux

Etudiants	Epreuves
E1	UF1, UF2, UF5
E2	UF3, UF4
E3	UF2, UF6
E4	UF3, UF4, UF5
E5	UF3, UF6
E6	UF1, UF2, UF3

- Pistes :
  - Modéliser le problème par un graphe d'incompatibilité entre UF et se ramener à un problème de coloration de sommets
  - Calculer une borne inférieure du nombre de créneaux
  - Proposer une heuristique gloutonne

# Exercice : Ordonnancement d'activités (1)

- On souhaite planifier la réalisation d'un ensemble d'activités (notées de  $A$  à  $J$ ) utilisant deux ressources ( $R_1$  et  $R_2$ ) pour minimiser la durée totale de réalisation (appelée Makespan)
- Le séquençage temporel entre activités est représenté par le graphe ci-dessous

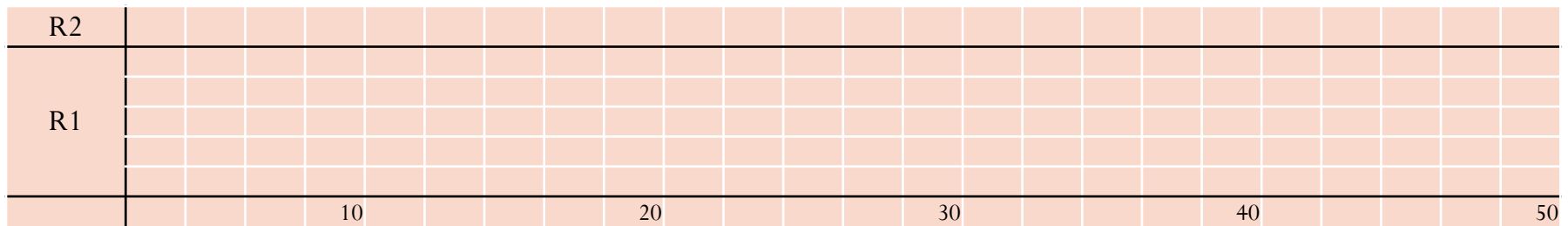


- Les utilisations de ressources sont les suivantes

	$A$	$B$	$C$	$D$	$E$	$F$	$G$	$H$	$I$	$J$
$R_1$ (5)	3	3	1	1	1	2	3	2	1	2
$R_2$ (1)	0	0	0	1	1	1	0	1	0	0

# Exercice : Ordonnancement d'activités (2)

- **Question 1.** Calculer la durée minimale de l'ordonnancement (sans prendre en compte les contraintes de ressource)
- **Question 2.** Donner les dates de début au plus tôt et au plus tard
- **Question 3.** Appliquer une heuristique gloutonne en classant les tâches par dates de début au plus tard croissantes
- **Question 4.** Donner le diagramme de Gantt associé et donner la valeur du Makespan de la solution obtenue.



- **Question 5.** A-t-on l'optimum ? Une borne supérieure ? Une borne inférieure ?



# Récapitulatif – Heuristiques Gloutonnes

---

- **Caractéristiques d'une méthode gloutonne**
  - Exploiter des connaissances pour faire les meilleurs choix à chaque étape
  - Simple à mettre en œuvre
  - Temps de calcul limité :
    - Ex : complexité linéaire en fonction du nombre de variables
  - Aucune garantie d'optimalité :
    - Borne supérieure/ objectif en minimisation
- **Attention**
  - Peut ne pas aboutir à une solution réalisable
  - Sauf si problème peu contraint

# Variantes – Heuristiques Gloutonnes

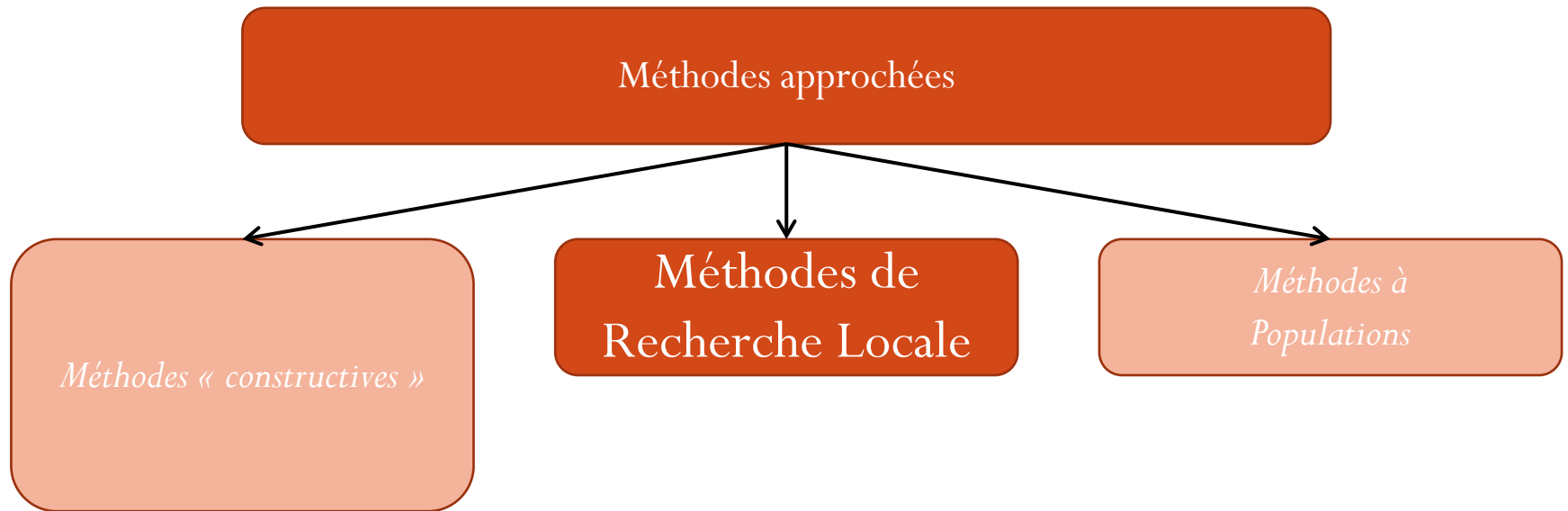
---

- **Heuristiques statiques**
  - Ordre d'exploration est défini a priori
- **Heuristiques dynamiques**
  - Ordre d'exploration est recalculé lors de chaque étape
- **Heuristiques avec de l'aléatoire (« randomisées »)**
  - Introduire de l'aléatoire
    - Sur les choix de variables et de valeurs
  - Lancer plusieurs exécutions de la recherche gloutonne
  - Arrêt de la méthode
    - Toutes les variables sont instanciées
  - Récupérer la meilleure solution

## Section 3. Méthodes approchées

---

- **Méthodes de Recherche Locale**



# Principes d'une recherche locale (1)

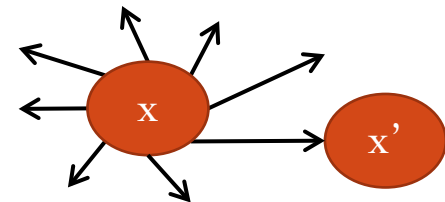
---

- **Idée :**

- Les bonnes solutions ont des caractéristiques communes

- **Principe :**

- Partir d'une solution initiale
- Modification de cette solution
  - Mouvement
    - Ex : Echange de 2 sommets dans un cycle



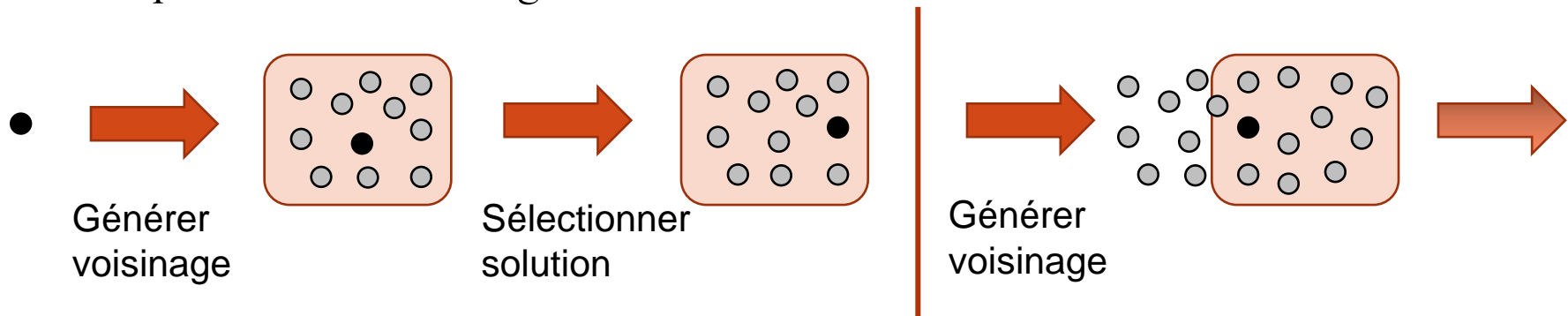
- Voisinage d'une solution  $x$

- $N(x)$  : Ensemble des solutions atteignables par un mouvement donné
- $x' \in N(x)$  : voisin de la solution  $x$

# Principes d'une recherche locale (2)

- **Principe d'exploration :**

- Exploration de voisinages successifs



- But : améliorer la valeur de la fonction objectif (et assurer l'obtention de solution réalisable)
- Solution obtenue :
  - Issue de mouvements dans une structure de voisinage
- Conditions d'arrêt ?
- Cycles entre différentes solutions ?

# Principes d'une recherche locale (3)

---

- **Composants**

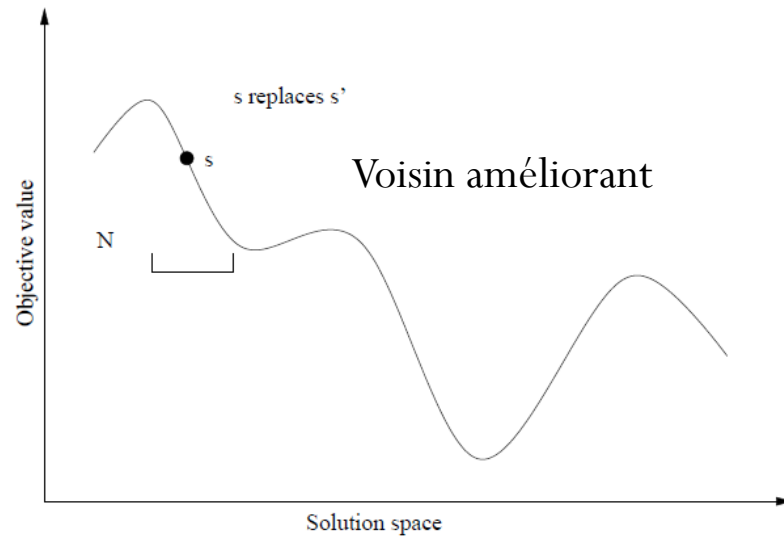
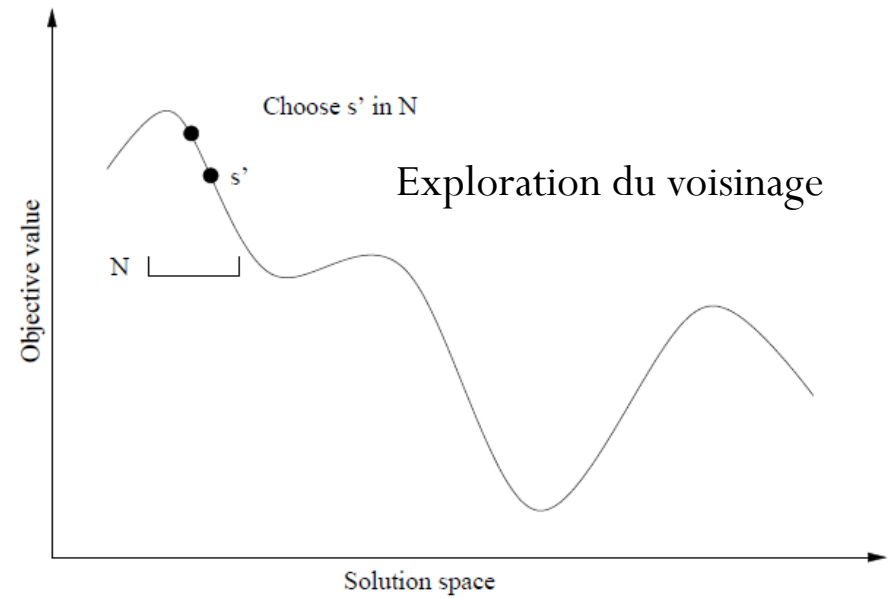
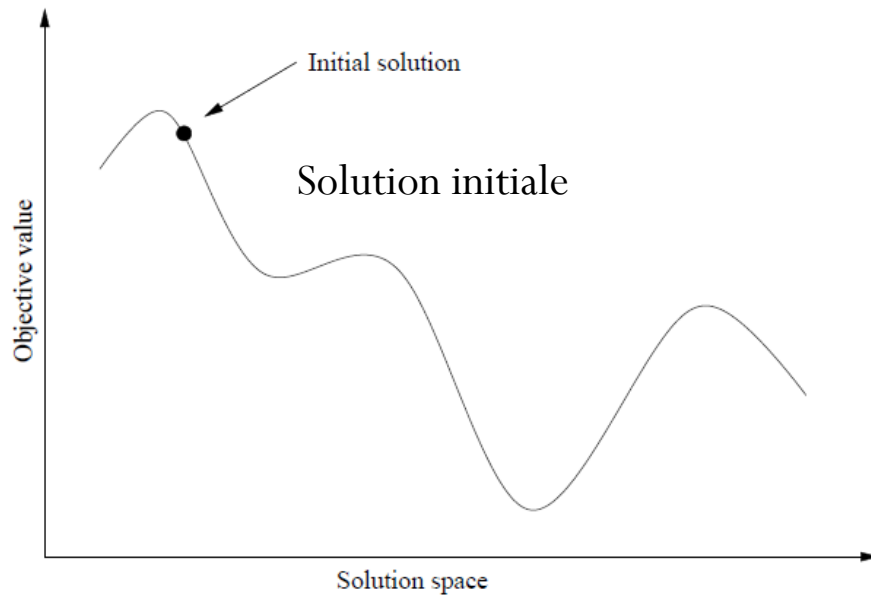
- Espace de recherche des solutions :  $E$
- Fonction objectif :  $f$  (cout à minimiser)
- Voisinage : mouvement qui pour tout  $x \in E \rightarrow N(x) \in E$

- **Algorithme de Recherche locale**

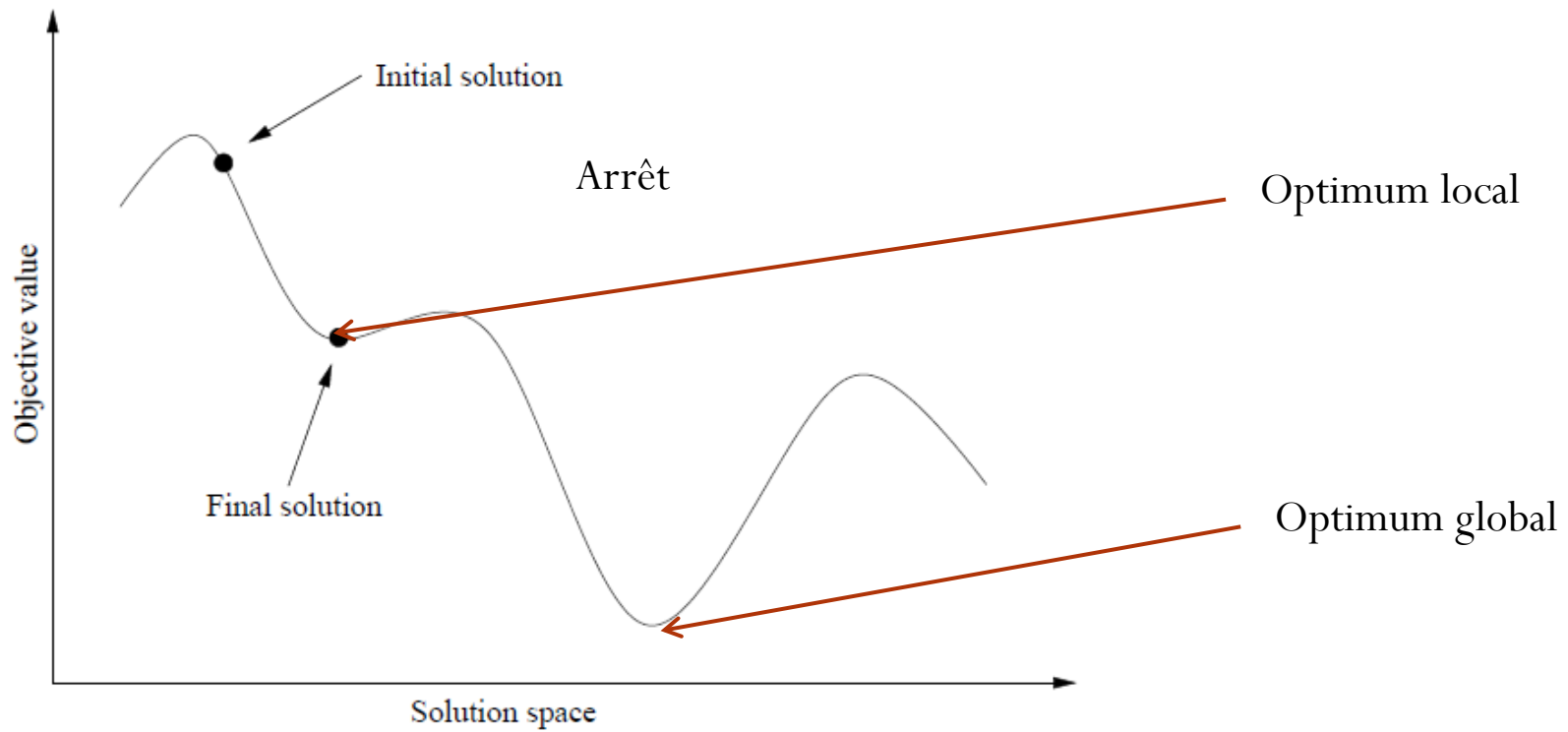
- Solution initiale :  $s \in E$  ;  $z \leftarrow f(s)$
- Meilleure solution :  $s^* \leftarrow s$  ;  $z^* \leftarrow z$
- Répéter
  - Choisir  $s' \in N(s)$
  - $s \leftarrow s'$
  - Si  $f(s) < f(s^*)$  alors  $s^* \leftarrow s$ ,  $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt »

**Local Search**

# Exemple



# Exemple





# Algorithme de recherche locale (1)

---

- **Principe :**

- Solution initiale :  $s \in E$  ;  $z \leftarrow f(s)$  (1)
- Meilleure solution :  $s^* \leftarrow s$  ;  $z^* \leftarrow z$
- Répéter
  - Choisir  $s' \in N(s)$  (2) (3)
  - $s \leftarrow s'$
  - Si  $f(s) < f(s^*)$  alors  $s^* \leftarrow s$ ,  $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt » (4)

- **Points clés**

1. Comment obtenir une solution initiale ?
2. Comment générer un voisinage ?
3. Comment sélectionner la prochaine solution ?
4. Quand s'arrêter ?

# Algorithme de recherche locale (2)

---

- **Algorithme de Recherche locale**

- Solution initiale :  $s \in E$  ;  $z \leftarrow f(s)$  (1)
- Meilleure solution :  $s^* \leftarrow s$  ;  $z^* \leftarrow z$
- Répéter
  - Choisir  $s' \in N(s)$  (2) (3)
  - $s \leftarrow s'$
  - Si  $f(s) < f(s^*)$  alors  $s^* \leftarrow s$ ,  $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt » (4)

- **Points clés**

1. **Comment obtenir une solution initiale ?**

- Solution aléatoire
- Solution connue
- Heuristique gloutonne

# Algorithme de recherche locale (3)

---

- **Algorithme de Recherche locale**

- Solution initiale :  $s \in E$  ;  $z \leftarrow f(s)$  (1)
- Meilleure solution :  $s^* \leftarrow s$  ;  $z^* \leftarrow z$
- Répéter
  - Choisir  $s' \in N(s)$  (2) (3)
  - $s \leftarrow s'$
  - Si  $f(s) < f(s^*)$  alors  $s^* \leftarrow s$ ,  $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt » (4)

- **Points clés**

- 4. **Quand s'arrêter ?**

- Obtention d'une solution de qualité voulue
    - Nombre d'itérations / Temps d'exécution maximum atteint
    - Nombre d'itérations / Temps d'exécution maximum sans amélioration atteint

# Algorithme de recherche locale (4)

---

- **Algorithme de Recherche locale**

- Solution initiale :  $s \in E$  ;  $z \leftarrow f(s)$  (1)
- Meilleure solution :  $s^* \leftarrow s$  ;  $z^* \leftarrow z$
- Répéter
  - Choisir  $s' \in N(s)$  (2) (3)
  - $s \leftarrow s'$
  - Si  $f(s) < f(s^*)$  alors  $s^* \leftarrow s$ ,  $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt » (4)

- **Points clés**

2. **Comment générer un voisinage ?**

# Voisinages (1)

---

- **Voisinage d'une solution**

- Opération de transformation :  $x \in E \rightarrow N(x) \in E$
- Transformation locale sur une solution :
  - La structure globale de la solution n'est pas modifiée
- Doit pouvoir être généré et évalué rapidement
- Lié à la représentation d'une solution

- **Graphe de voisinage :**

- Sommets : espace de recherche  $E$
- Si  $y \in N(x) \rightarrow$  relation  $(x, y)$  dans le graphe

# Voisinages (2)

---

- **Propriétés :**

- Accessibilité : tout élément de l'espace de recherche peut être atteint (toute solution intéressante) est accessible par transformations successives
  - Graphe de voisinage Connexe
- Réversibilité : on peut revenir à la solution initiale
  - Graphe symétrique

- **Choix d'un Voisinage :**

- Lié au codage de la solution
  - Ex : Voyageur de Commerce : ensemble des sommets ou des arêtes
- Lié à sa taille (cout de génération et d'évaluation)
  - Ex :  $O(n)$ ,  $O(n^2)$ , ... / nombre de variables

# Exemple de voisinage (1)

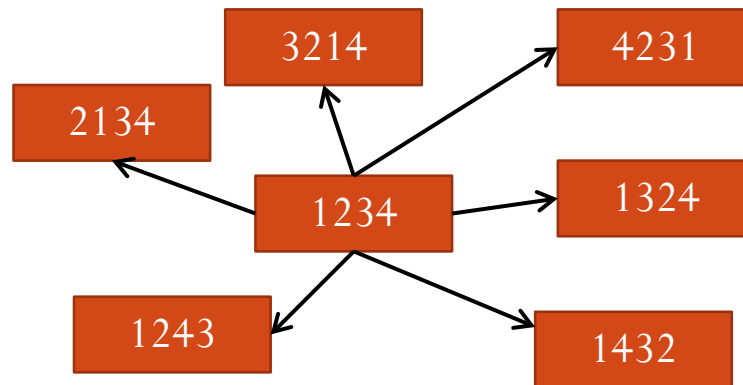
---

- Ensemble de  $N$  variables binaires :
- Voisinage :
  - **Distance** :
    - Ensemble des chaînes binaires à une distance de Hamming de 1
    - Pour 5 variables binaires :  $x = 01101$ 
      - $N(x) = \{11101; 00101; 01001; 01111; 01100\}$
      - $|N(x)| = n = 5$
  - Si distance de Hamming de 2,  $|N(x)| = \frac{n(n-1)}{2}$
  - **Complémentation**
    - Solution = chaîne de variables binaires :
      - remplacer une ou plusieurs valeurs 0/1 par son complémentaire

# Exemple de voisinage (2)

- **Echange / Swap**

- Solution = chaîne de caractères
  - Choisir 2 positions  $i$  et  $j$ . Intervertir les caractères situés à ces deux positions



Taille du voisinage d'une solution de taille  $n$  :  $\frac{n(n-1)}{2}$



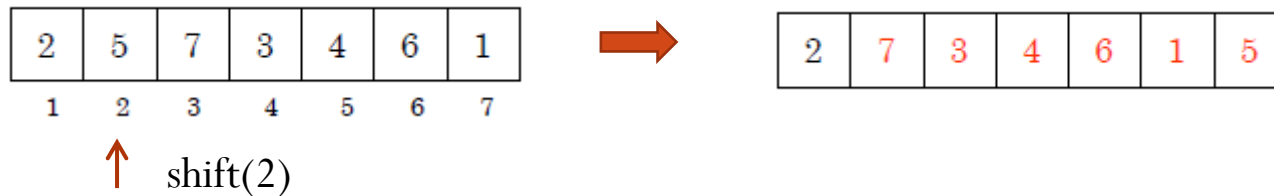
# Exemple de voisinage (3)

- Quelques transformations classiques :

- **Décalage**

- Solution = chaîne de caractères :

- Choisir une position  $i$ . Insérer élément de position  $i$  à la fin et décaler les caractères



- **Inversion**

- Solution = chaîne de caractères

- Choisir 2 positions  $i$  et  $j$ . Inverser l'ordre d'écriture entre  $i$  et  $j$ .



# Voisinages pour le problème du Voyageur de Commerce (1)

---

- **Problème :**

- Un ensemble de villes (sommets d'un graphe) :  $G(X, A)$  et distance entre villes
- Déterminer une tournée minimisant la somme des distances parcourues

- **Espace de recherche  $E$**  : ensemble des permutations de  $X$

- Soit une solution contenant les arêtes  $(u, x)$  et  $(v, y)$

- **Insertion**

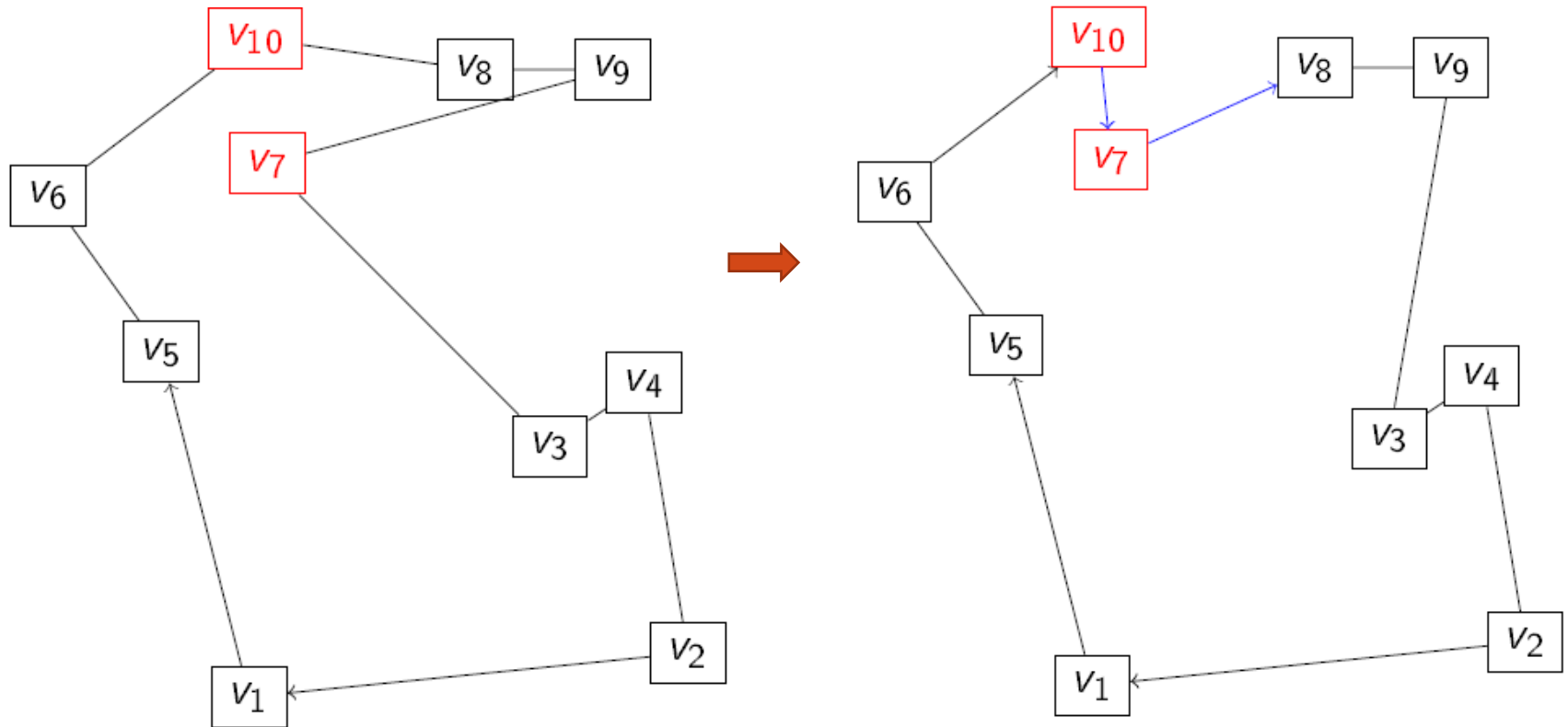
- Supprimer  $u$  du cycle pour l'insérer après  $v$
- Déplacer  $(u, x)$  après  $v$

- **Swap**

- Echanger  $u$  et  $v$

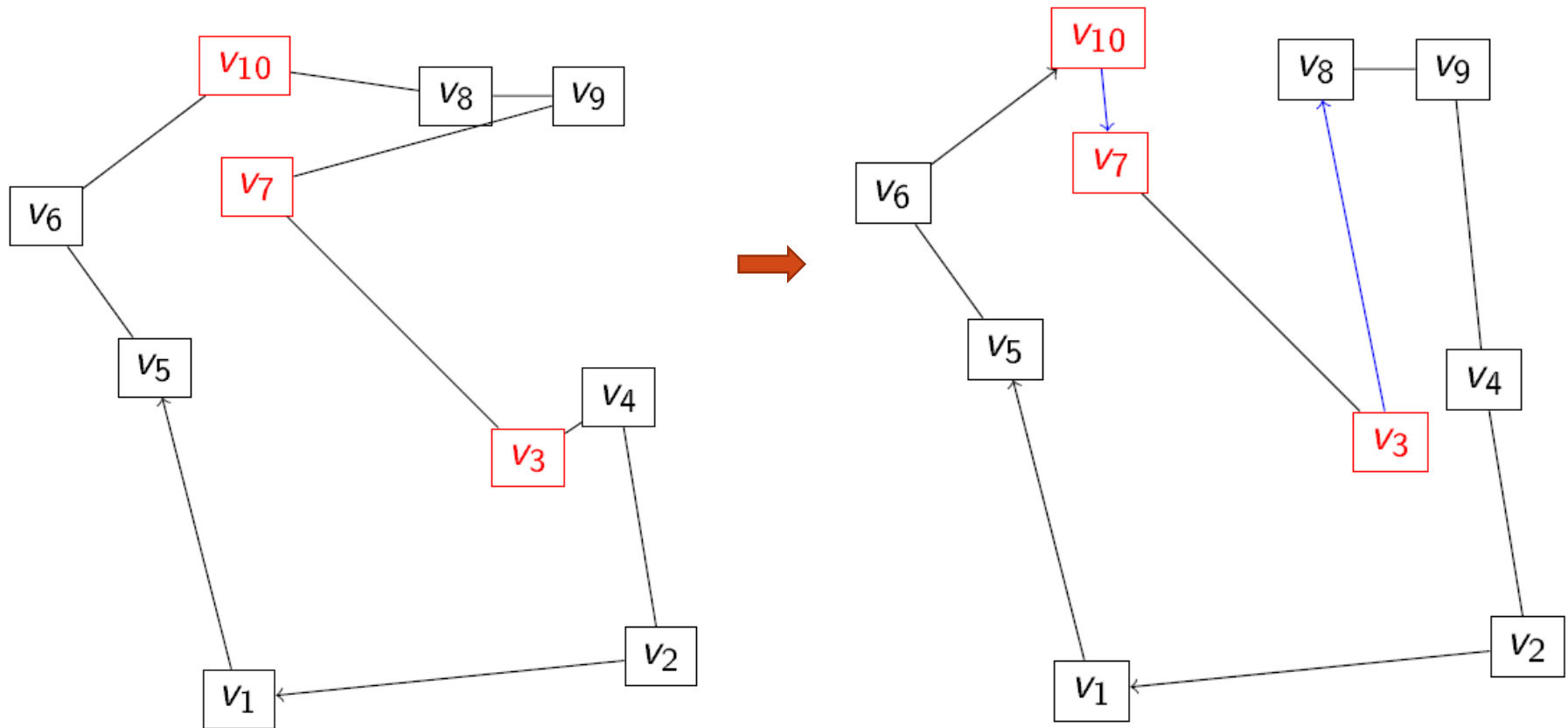
# Voisinages pour le problème du Voyageur de Commerce (2) : Insertion

- Supprimer  $v_7$  pour l'insérer après  $v_{10}$



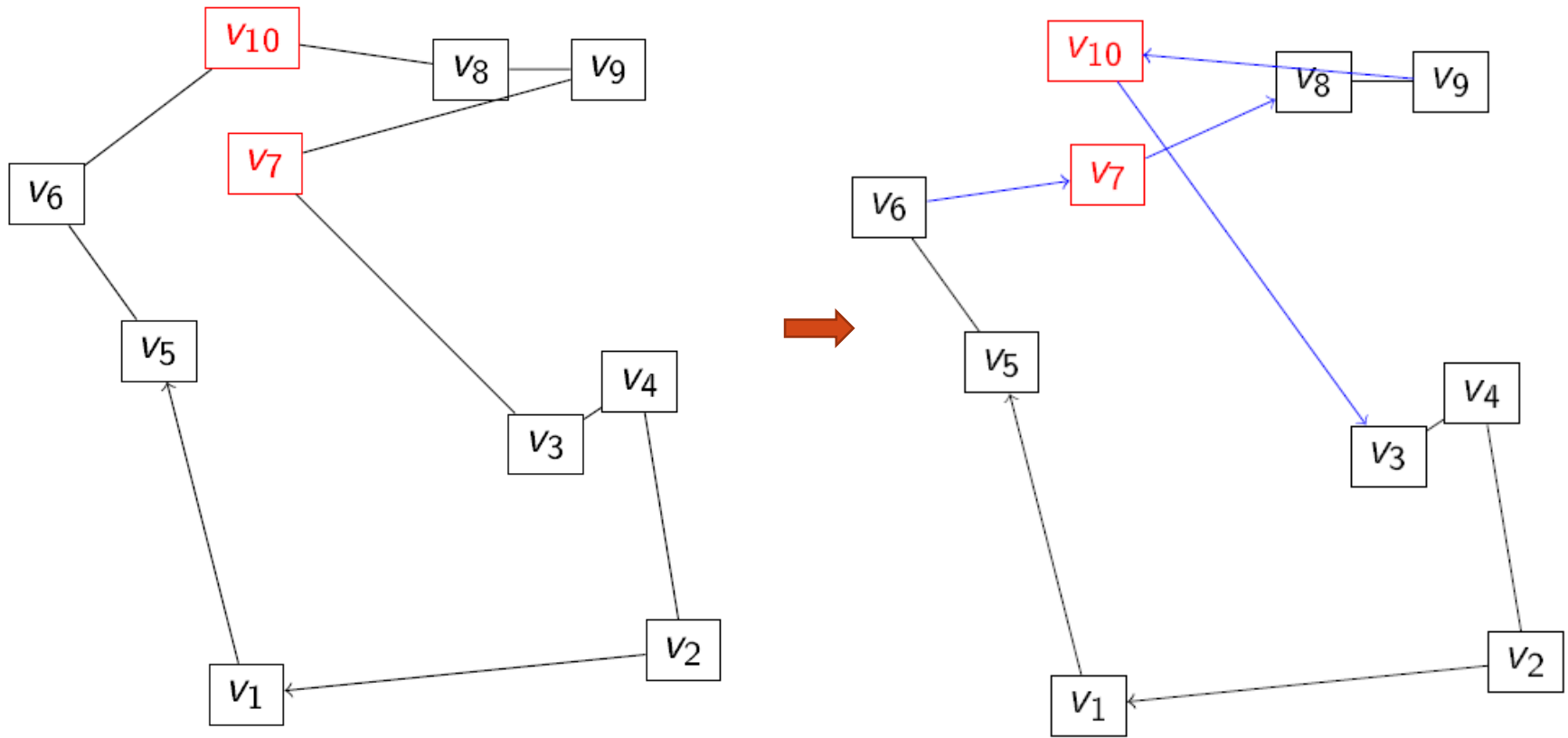
# Voisinages pour le problème du Voyageur de Commerce (2) : Insertion

- Déplacer ( $v_7, v_3$ ) pour le placer après  $v_{10}$



# Voisinages pour le problème du Voyageur de Commerce (3) : Swap

- Echanger  $v_7$  et  $v_{10}$

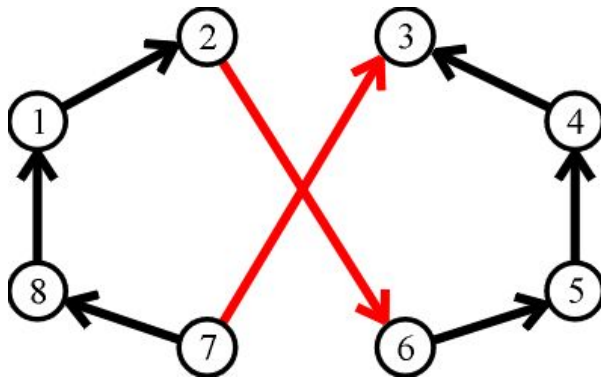


# Voisinages pour le problème du Voyageur de Commerce (3) :

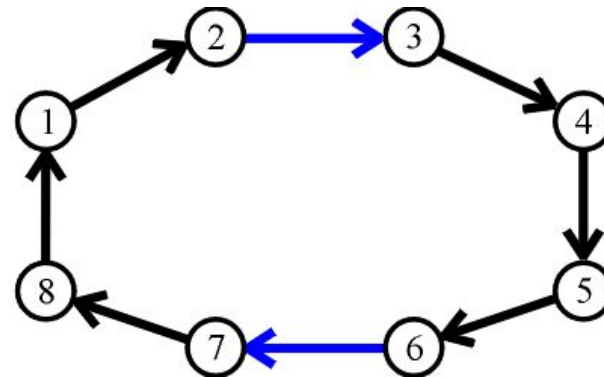
- **Voisinage 2-opt**

- Trouver 2 arêtes non consécutives
  - Les supprimer et reformer le cycle

- Supprimer :  $(x_2, x_6)$  et  $(x_7, x_3)$



- Reformuler :  $(x_2, x_3)$  et  $(x_6, x_7)$



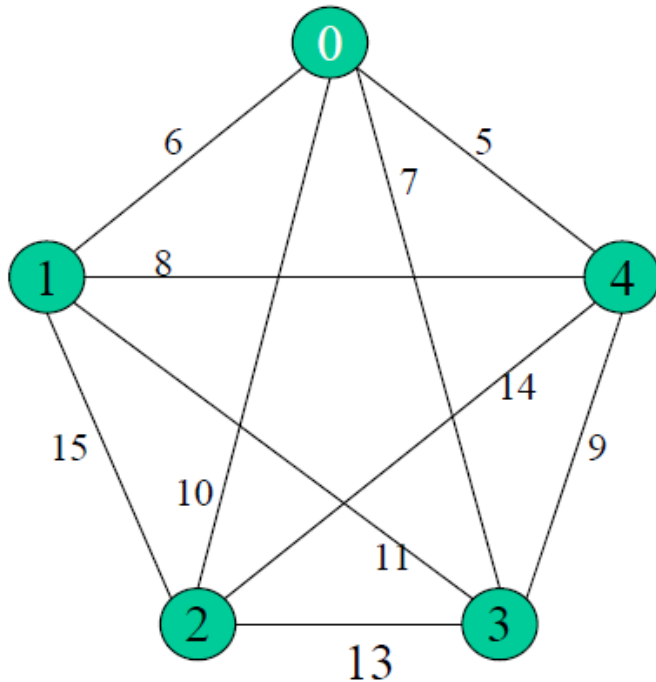
- Une seule façon de reformer le cycle

- Variante : 3-opt, k-opt,

# Graphe de voisinage (1)

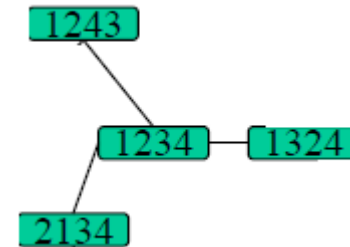
- Exemple sur le TSP

Départ et Retour en 0



Nombre solutions :

Voisinage = échanger 2 sommets consécutifs

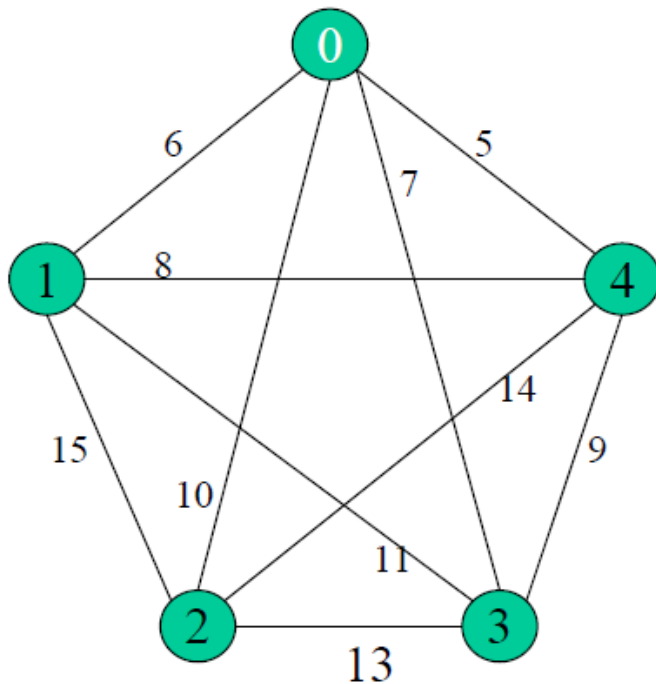


Taille du voisinage :

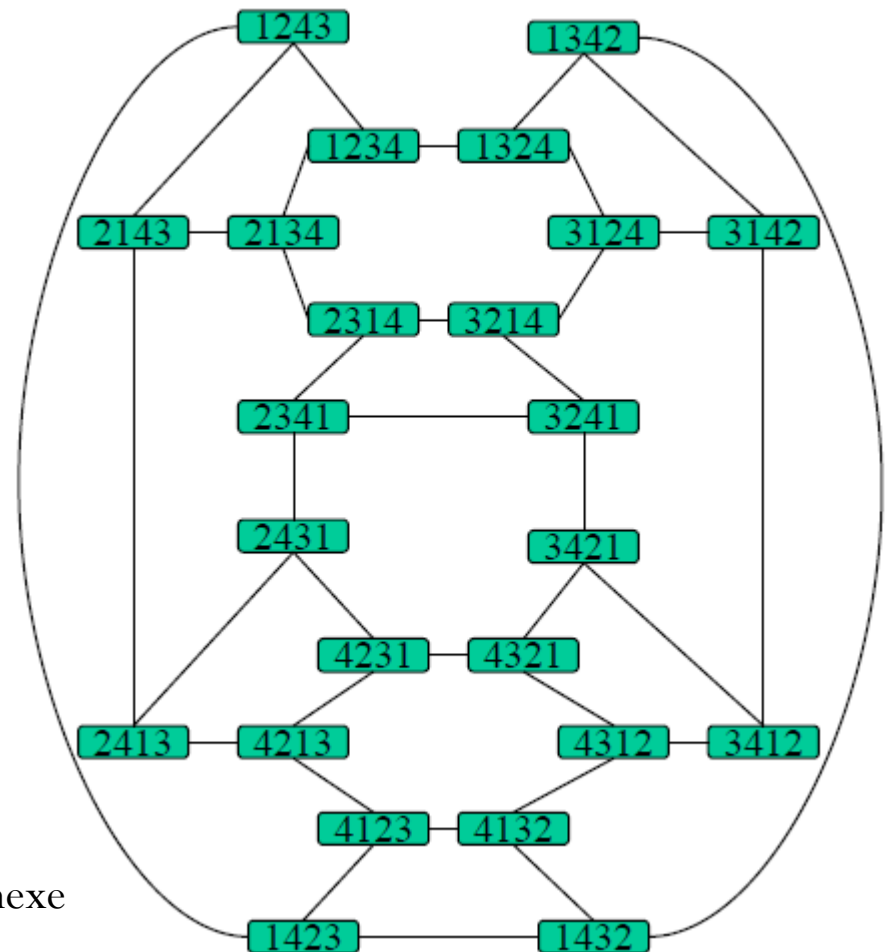
# Graphe de voisinage (2)

- Exemple sur le TSP

Départ et Retour en 0



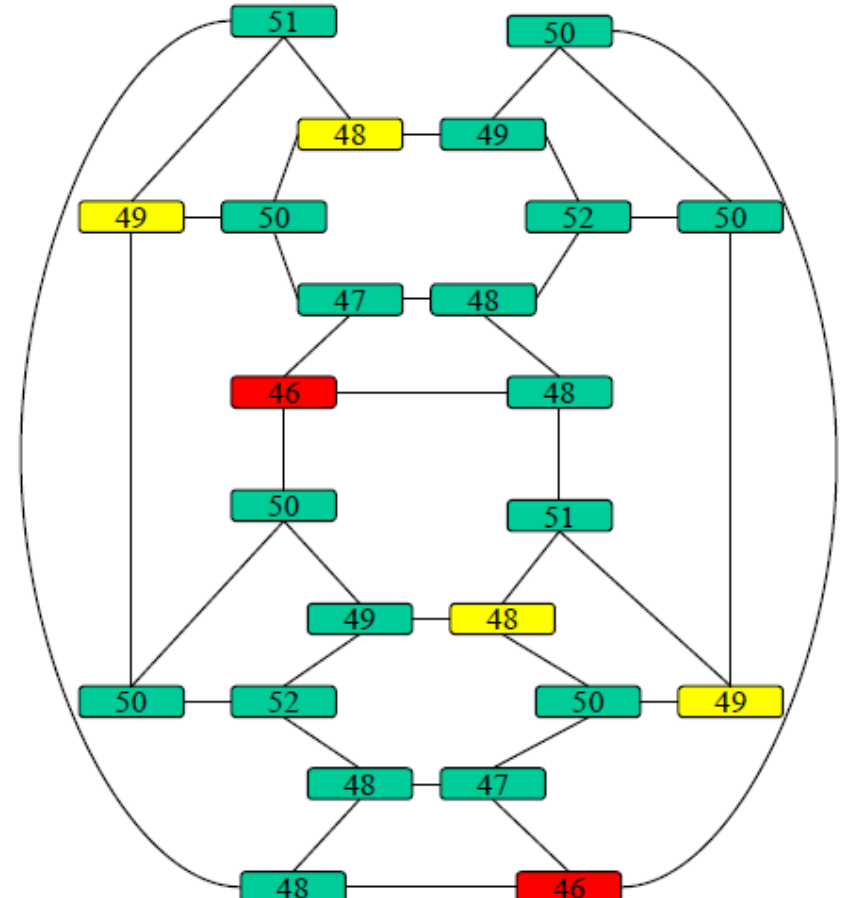
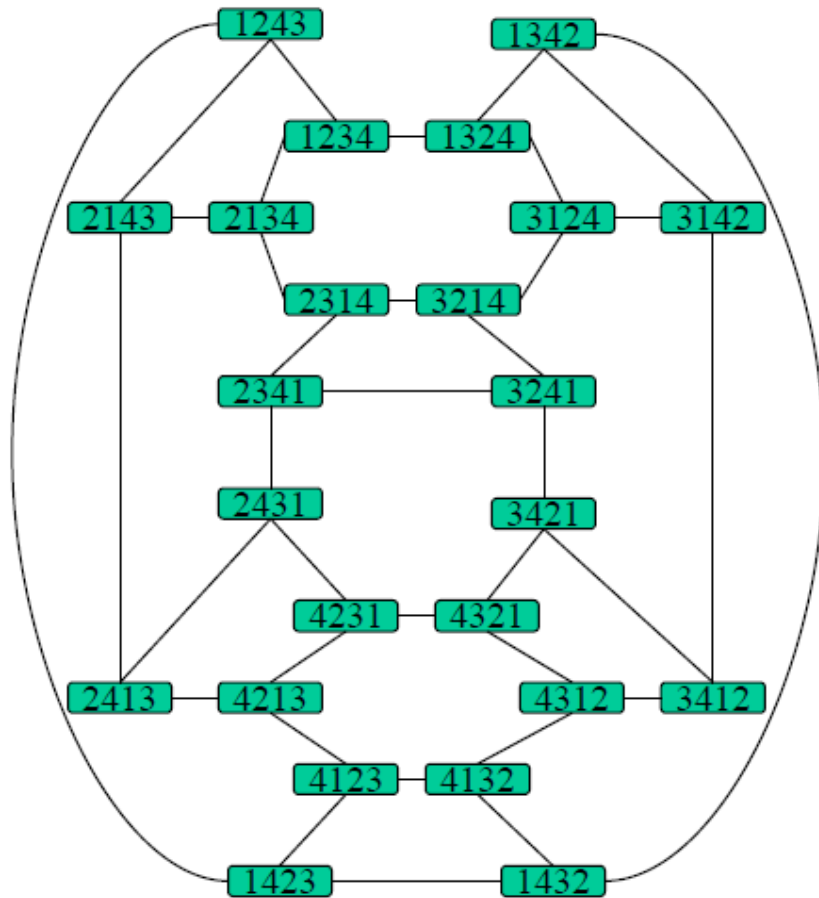
Voisinage = échanger 2 sommets consécutifs



Voisinage → graphe connexe



# Graphe de voisinage (3)



Optimums locaux et globaux

# Paysage de Recherche

---

- Une opération de voisinage → graphe de voisinage
  - Paysage de recherche
- **Optimum local** : tous les voisins sont moins bons / objectif
- **Plateau** : ensemble de points connexes dans le graphe de voisinage et ayant même valeur de fonction objectif
- **Bassin d'attraction** : voisins que l'on peut atteindre sans dégrader la fonction objectif

# Choix d'un voisinage

---

- Dépend du problème à résoudre
- Dépend de la représentation des solutions
- Impact du voisinage sur le paysage des solutions
- Littérature sur le problème considéré
- **Attention à la taille du voisinage**
  - Si trop petit : risque de ne pas avoir de meilleure solution
  - Si trop grand : l'exploration est coûteuse
- **Impact entre voisinage et sélection d'une solution :**
  - 1 seul optimum et pas de plateau : Intensifier
  - Paysages rugueux : Diversifier



# Algorithme de recherche locale (5)

---

- **Algorithme de Recherche locale**

- Solution initiale :  $s \in E$  ;  $z \leftarrow f(s)$  (1)
- Meilleure solution :  $s^* \leftarrow s$  ;  $z^* \leftarrow z$
- Répéter
  - Choisir  $s' \in N(s)$  (2) (3)
  - $s \leftarrow s'$
  - Si  $f(s) < f(s^*)$  alors  $s^* \leftarrow s$ ,  $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt » (4)

- **Points clés**

2. **Comment sélectionner la prochaine solution ?**

# Exploration d'un voisinage (1)

---

- **Algorithme de Recherche locale**

- Solution initiale :  $s \in E$  ;  $z \leftarrow f(s)$  (1)
- Meilleure solution :  $s^* \leftarrow s$  ;  $z^* \leftarrow z$
- Répéter
  - Choisir  $s' \in N(s)$  (2) (3)
  - $s \leftarrow s'$
  - Si  $f(s) < f(s^*)$  alors  $s^* \leftarrow s$ ,  $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt » (4)

- **Points clés**

- 3. **Comment sélectionner la prochaine solution ?**

- Premier voisin améliorant
    - Meilleur voisin améliorant
    - Aléatoire
    - .....

# Exploration d'un voisinage (2)

---

- **Comment sélectionner la prochaine solution ?**
  - Exploration de l'espace de recherche pour déterminer la nouvelle solution
  - Sélectionner une solution de meilleure qualité :
    - **Premier voisin améliorant (First Improvement)**
      - Savoir générer des voisins prometteurs
    - **Meilleur voisin (Best improvement)**
      - Exploration de tout le voisinage
  - Méthode **Hill-Climbing** ou **Plus grande pente** ou **Descente**
    - Méthode du gradient en optimisation continue
    - Permet une **intensification** de l'exploration autour d'une solution initiale
    - Aboutit à un optimum local (ou reste bloqué sur un plateau)

# Algorithmes Hill-Climbing (1)

---

## Algorithme First Improvement

Solution initiale :  $s \in E$  ;  $z \leftarrow f(s)$  (1)

Meilleure solution :  $s^* \leftarrow s$ ;  $z^* \leftarrow z$

Répéter

Choisir  $s' \in N(s)$  tq  $f(s') < f(s)$  (2) (3)

$s \leftarrow s'$

Si  $f(s) < f(s^*)$  alors  $s^* \leftarrow s$ ,  $z^* \leftarrow z$

Jusqu'à « Critères d'arrêt » (4)

## Algorithme Best Improvement

Solution initiale :  $s \in E$  ;  $z \leftarrow f(s)$  (1)

Meilleure solution :  $s^* \leftarrow s$ ;  $z^* \leftarrow z$

Répéter

Choisir  $s' \in N(s)$  tq  $\forall s'' \in N(s) \setminus \{s'\} f(s') < f(s'')$  (2) (3)

$s \leftarrow s'$

Si  $f(s) < f(s^*)$  alors  $s^* \leftarrow s$ ,  $z^* \leftarrow z$

Jusqu'à « Critères d'arrêt » (4)

# Algorithmes Hill-Climbing (2)

---

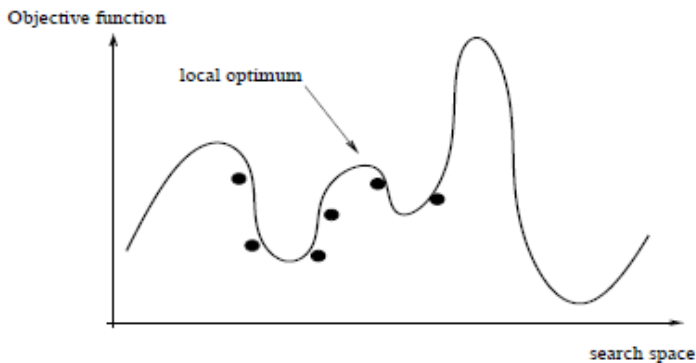
- **Terminaison de la méthode Hill-Climbing**
  - La taille du voisinage ne permet pas de trouver de meilleure solution
  - Solution localement optimale
- Pas de bouclage de la méthode
  - Pas de retour sur une solution déjà trouvée
    - Quand on sélectionne  $s'$  comme solution améliorante,  $s'$  n'a jamais été sélectionnée auparavant comme solution améliorante



# Exploration aléatoire d'un voisinage

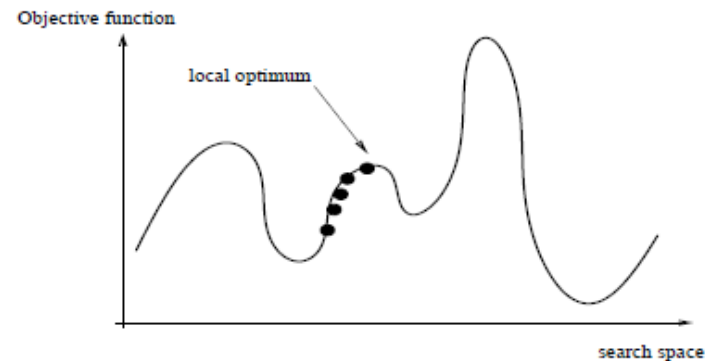
- Sélectionner une solution aléatoire: **Random Walk**
  - Accepte des solutions même non améliorante
  - Permet une **diversification** de l'exploration (sortir d'un optimum local)
- Différence « Random Walk » et « Hill Climbing » en maximisation

Random walk



**Diversification**

Hill-climbing



**Intensification**

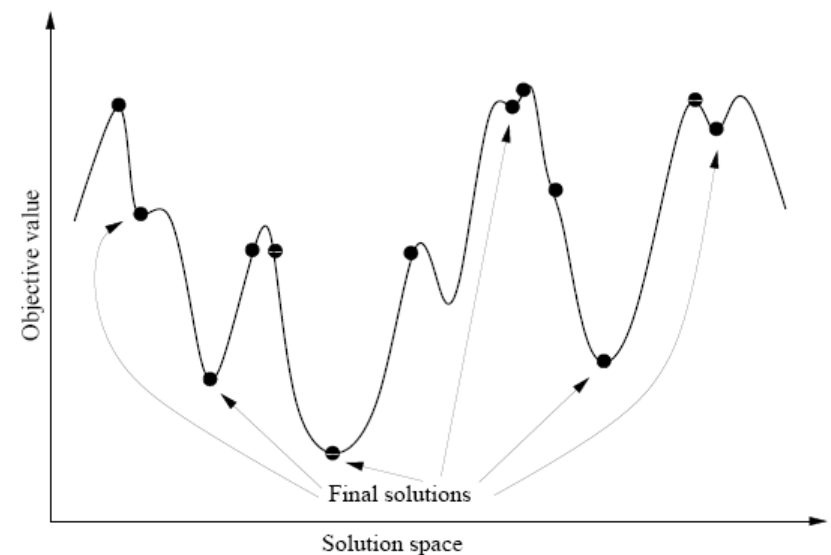
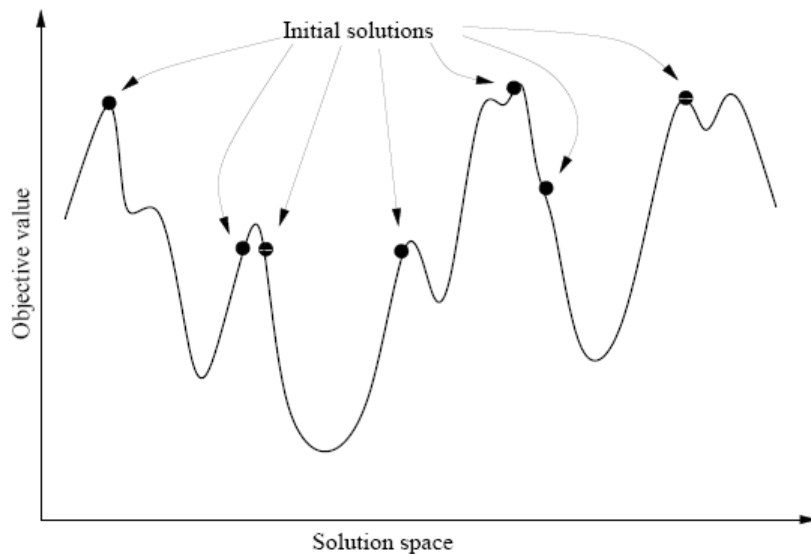
# Intensification / Diversification

---

- **Combinaison intensification / diversification**
  - Fixer une probabilité de choisir un mouvement « random » :  $\rho$
  - Choisir aléatoirement une valeur  $m$ 
    - Si  $m < \rho$  alors
      - Choisir une solution  $s'$  aléatoirement dans  $N(s)$  -- **diversification**
    - Sinon
      - Choisir la meilleure solution  $s' \in N(s)$  -- **intensification**
  - Selon la probabilité  $\rho$  : compromis diversification / intensification
    - $\rho = 1$  : random walk
    - $\rho = 0$  : hill-climbing
    - En pratique :  $\rho$  faible

# Plusieurs solutions initiales

- Autre amélioration de la méthode Hill-Climbing
  - Relancer la méthode de descente à partir de plusieurs solutions initiales
  - Conserver la meilleure solution trouvée
- Méthode de **Descente Multi-Start**



# Algorithme de recherche locale (6)

---

- **Algorithme de Recherche locale**

- Solution initiale :  $s \in E$  ;  $z \leftarrow f(s)$  (1)
- Meilleure solution :  $s^* \leftarrow s$  ;  $z^* \leftarrow z$
- Répéter
  - Choisir  $s' \in N(s)$  (2) (3)
  - $s \leftarrow s'$
  - Si  $f(s) < f(s^*)$  alors  $s^* \leftarrow s$ ,  $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt » (4)

- **Points clés**

1. Comment obtenir une solution initiale ?
2. Comment générer un voisinage ? ➔ **et évaluer**
3. Comment sélectionner la prochaine solution ?
4. Quand s'arrêter ?

# Evaluation d'une solution voisine

---

- **Evaluation d'une solution / d'un voisin**

- Estimation **a priori** de la qualité d'un voisin
- Permet de se guider dans l'espace de recherche pour trouver un chemin vers de bonnes solutions

- **Comment évaluer ?**

- Fonction Objectif
- Mais pas toujours !
  - autre fonction d'évaluation (ex SAT : nombre de clauses non vérifiées)

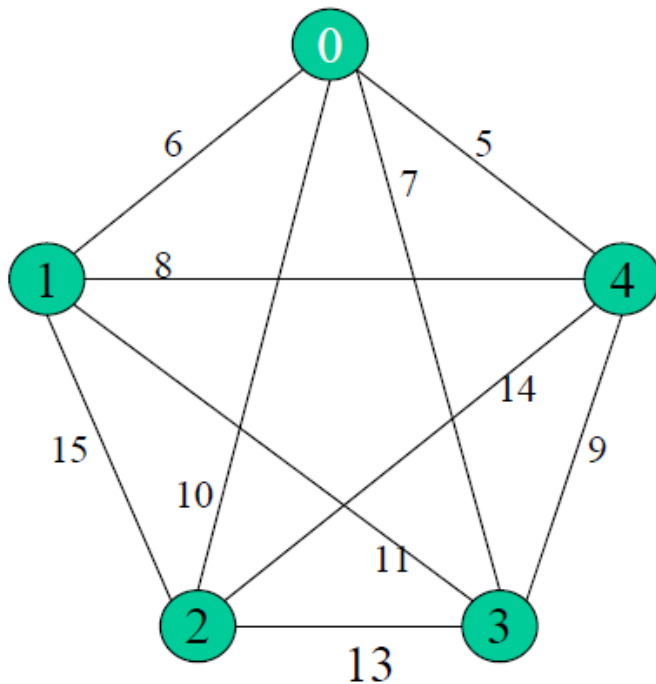
- **L'évaluation : opération de calcul très fréquente**

- Doit être la moins coûteuse en temps de calcul
- Si possible incrémentale :
  - Calculer l'apport / la perte généré par un voisin
  - Variation de  $f(s')$  par rapport à  $f(s)$

# Exemple évaluation

## • Exemple

Départ et Retour en 0



Solution  $s = \{0,1,2,3,4,0\}$ ;  $f(s) = 48$

Voisin  $s'$

- Permutation 1 et 2 :  $\{0,2,1,3,4,0\}$

Evaluation de  $s'$  :

- Retirer (0,1) et (2,3) :  $-6-13 = -19$
- Ajouter (0,2) et (1,3) :  $+10+11=+21$
- $f(s') = f(s) - 19 + 21 = 50$

# S'échapper des optima locaux (1)

---

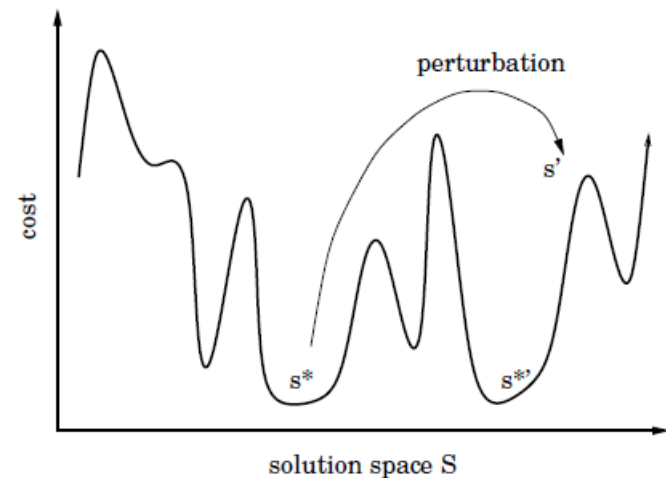
- **Dans une recherche locale :**
  - Impact du choix de la structure de voisinage
  - Impact du choix de la fonction d'évaluation (ie. du paysage parcouru)
- **Pour diversifier**
  - Introduire de l'aléatoire
  - Autoriser des solutions non améliorantes
    - Mais comment éviter de cycliser sur des solutions ?
  - Faire varier les voisinages
  - Re-démarrage
  - Mémorisation
- **Nombreuses variantes en Recherche Locale**

# Variantes (1) : Iterated Local Search

## Perturbation aléatoire de la solution

### Iterated Local Search

1.  $s_0 \leftarrow$  Solution initiale;  $Best \leftarrow s_0$
2.  $s \leftarrow \text{Descent}(s_0)$       // *Exploration*
3. repeat
4.      $s' \leftarrow \text{Perturbation}(s)$
5.      $s'' \leftarrow \text{Descent}(s')$
6.     *Acceptation* :  
      if  $f(s'') < f(s)$  then  
           $s \leftarrow s''$  ; *Mise-à-jour*( $Best, s$ );
7. until : conditions à définir





# Variantes (2) : Méthode à seuil

## Threshold Accepting

- **Idée :**
  - Introduire un seuil d'acceptation  $\tau$  des solutions non améliorantes
  - Choisir le premier voisin  $s'$  tel que  $f(s') - f(s) < \tau$
- **Réglage du seuil**
  - Détermine le compromis diversification / intensification
    - Si  $\tau = \infty$  : aléatoire
    - Si  $\tau \leq 0$  : les mouvements dégradant la solution sont interdits
  - Le seuil peut varier au cours des itérations

# Variantes (3) : Voisinages variables

---

- **Variable Neighborhood Descent/ Search**
- **Définir plusieurs voisinages (diversification)**
  - But : pouvoir sortir des optima locaux et améliorer la qualité de la solution
- **Principe**
  - Effectuer une succession de méthodes de descente
  - Quand un optimum local est atteint par une méthode de descente:
    - changer de voisinage
  - Ordonner a priori l'ensemble des voisinages
    - $N_1, N_2, \dots, N_k$
    - Complexité croissante

## Variantes (3) : Voisinages variables

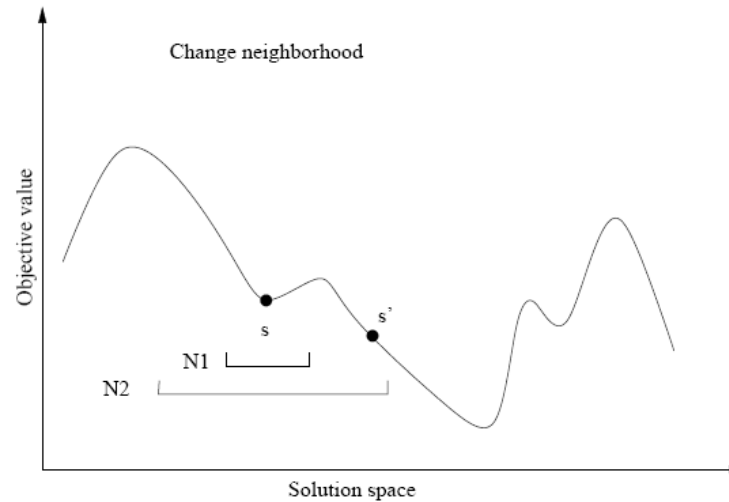
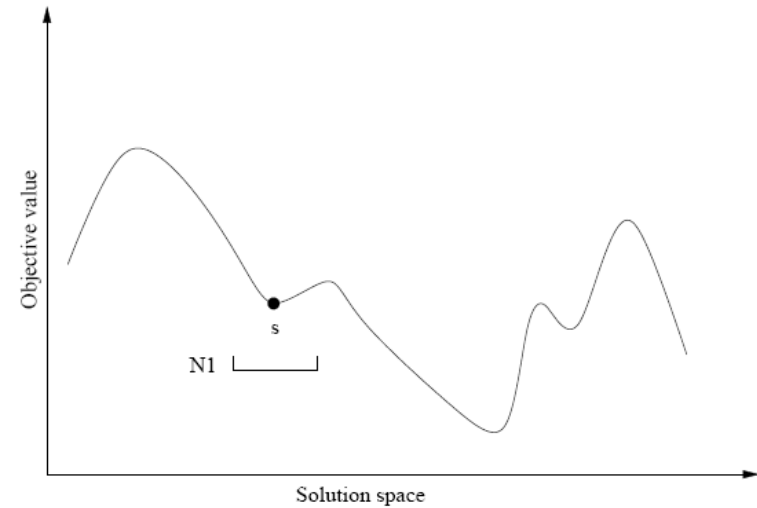
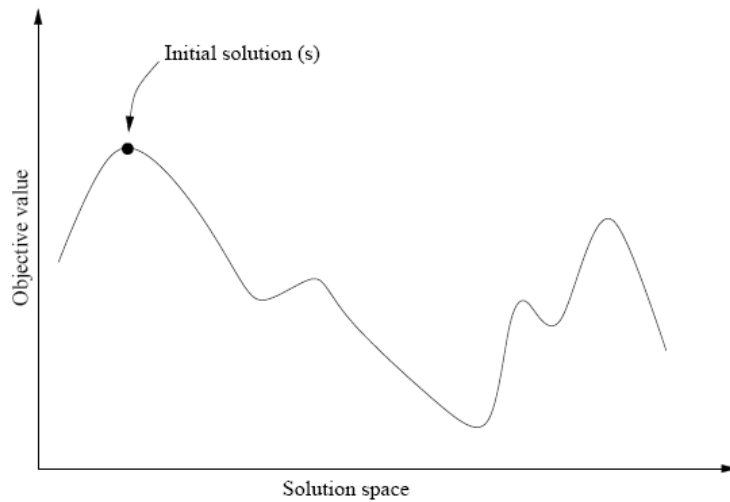
- Principe

```

1.  s ← Solution initiale
2.  i ← 1 : numéro du voisinage
3.  repeat
4.      s' ← Meilleur Voisin dans Ni(s) //Variante : Random(Ni)
                                         // et Descent(s')
5.      if f(s') < f(s) then              //Variante : Rester sur Ni
          i ← 1;      s ← s'            // et appliquer Descent(s')
6.      else
          i ← i+1
      end if
7.  until i > k (nombre de voisinages)

```

# Variantes (3) : Voisinages variables



# Variantes (4) : Recuit Simulé

## Simulated Annealing

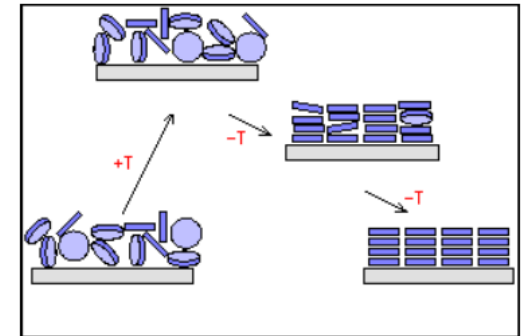
- **Idée**

- Analogie métallurgie : en chauffant un métal puis en le refroidissant doucement on peut obtenir des structures cristallines résistantes

### Principe

Repose sur la capacité d'un système physique d'évoluer vers un état énergétique minimal.

- D'abord, une agitation thermique permet de sortir des minima locaux de l'énergie.
- Ensuite, quand on refroidit le système physique assez lentement, il tend à évoluer vers une structure d'énergie minimale.



- Métaheuristique pour l'optimisation
  - S. Kirkpatrick 1983 / V. Cerny 1985

# Variantes (4) : Recuit Simulé

## Simulated Annealing

- **Pour l'optimisation :**

- Diversifier la recherche en acceptant des voisins qui dégradent la fonction objectif en fonction d'une probabilité d'acceptation qui décroît dans le temps

- **Stratégie d'exploration :**

- **Paramètre  $T$  (température)** qui décroît au cours des itérations
- Choix d'un voisin  $s'$  tel que :
  - Si  $\Delta = f(s') - f(s) < 0$  alors  $s \leftarrow s'$  -- **intensification**
  - Si  $x < e^{\frac{-\Delta}{T}}$  ( $x$  choisi aléatoirement dans  $[0,1]$ ) alors  $s \leftarrow s'$  -- **diversification**
  - Sinon Rester sur la solution  $s$

# Variantes (4) : Recuit Simulé

- **Température** : probabilité d'accepter une solution non améliorante
- **Condition de Métropolis** :

- accepter la nouvelle solution avec une probabilité :  $e^{\frac{-\Delta}{T}}$

- Si  $\Delta \nearrow$  alors  $e^{\frac{-\Delta}{T}} \searrow$  ; Si  $T \searrow$  alors  $e^{\frac{-\Delta}{T}} \searrow$

- **Algorithme**

```
1. s0 ← solution initiale
2. T0 ← Température initiale
3. s ← s0; T ← T0
4. while (Conditions) loop
5.     s' ← Random(N(s))
6.     Deltaf ← f(s') - f(s)
7.     if Deltaf < 0 ou random < exp(-Delta/T) then
8.         s ← s'
9.     end if
10.    T ← k.T
11. end while
12. Return s
```

# Variantes (4) : Recuit Simulé

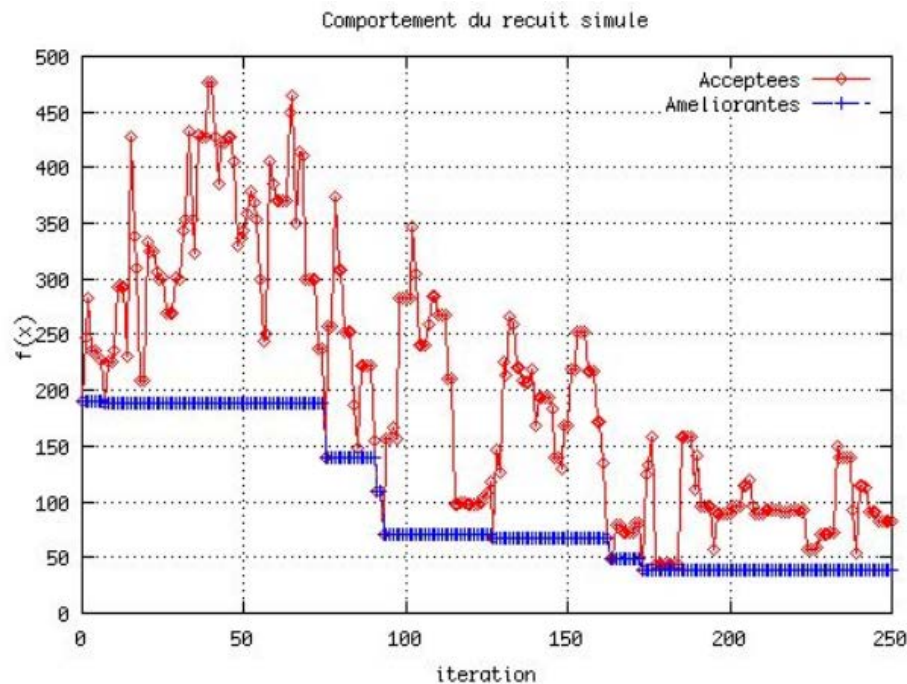
---

- **Au début des itérations :**
  - T élevé : Acceptation fréquente de solutions non améliorantes
- **En fin des itérations :**
  - T faible : acceptation rare de solution non améliorante
- **Réglage des paramètres**
  - Température initiale
  - Variation de température : à chaque étape / par palier / adaptative
  - Conditions d'arrêt (température, fonction objectif, ...)
  - Trouver le bon paramétrage ....



# Variantes (4) : Recuit Simulé

- **Etat initial**
  - Solution initiale
  - Température : doit permettre d'accepter « suffisamment » de solutions voisines
- **Graphique des solutions visitées / améliorantes**

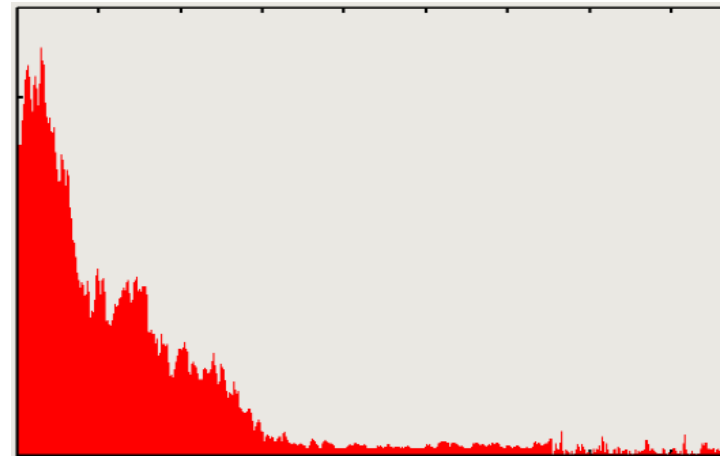
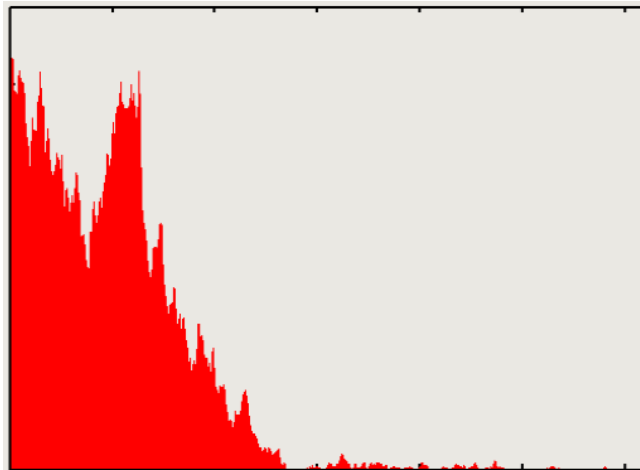


# Variantes (4) : Recuit Simulé

---

- **Schéma de refroidissement**

- Si trop rapide : convergence prématurée : on reste dans un optimum local
- Si trop lente : exploration trop importante



# Variantes (5) : Recherche Tabou

## Tabu Search

- **Constat :**

- Quand on est sur un optimum local : les solutions voisines sont toutes de moins bonne qualité ➔ bassin d'attraction
- Glover 1986 / Hansen 1986

- **Idée :**

- Sortir du bassin d'attraction en acceptant des solutions de moins bonne qualité
    - Choisir le meilleur voisin même si non améliorant
  - Mais interdire de revisiter des solutions déjà explorées
- Structure pour mémoriser des informations sur les solutions visitées, appelée **Liste Tabou** pendant un certain nombre d'itérations

# Variantes (5) : Recherche Tabou

- **Stratégie d'exploration :**

- Introduire une liste  $L$  (initialement vide)
- A chaque itération : ajouter le dernier mouvement effectué dans  $L$
- Choisir une solution voisine  $s'$  telle que :
  - Le mouvement  $s \rightarrow s' \notin L$  -- **diversification**
  - Le cout  $f(s')$  soit minimal -- **intensification**

- **Algorithme**

```
1.  $s \leftarrow$  solution initiale
2.  $best \leftarrow s$ 
3.  $L \leftarrow \emptyset$  // Tabu
4. while (Conditions) loop
5.      $s' \leftarrow$  Meilleur-Voisin( $N(s)$ ,  $L$ ) // Voisin non tabou
6.     if  $f(s') < f(best)$  then
7.          $best \leftarrow s'$ 
8.     end if
9.     Actu_Tabu( $s'$ ,  $L$ )
10. end while
11. Return best
```

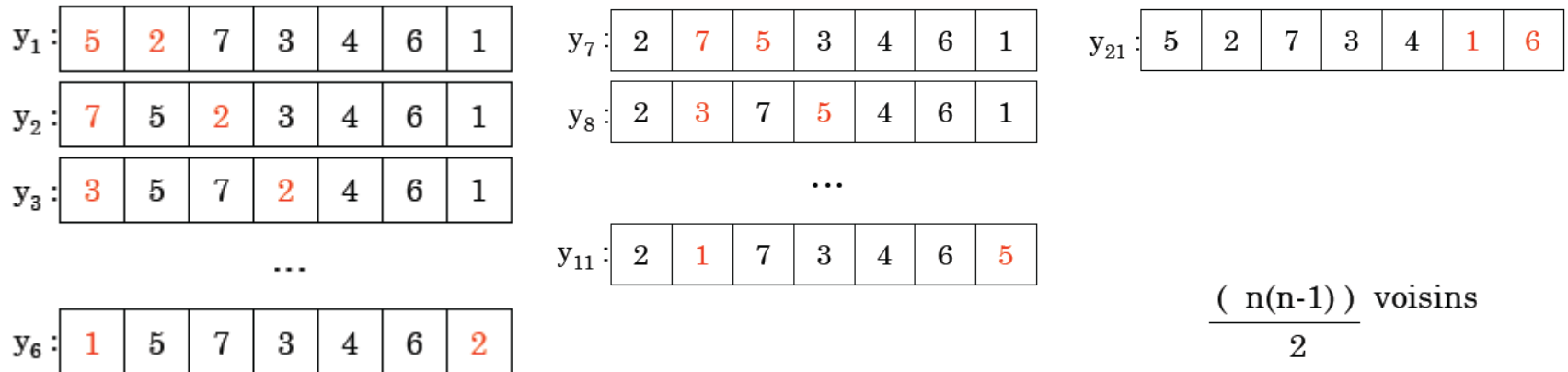
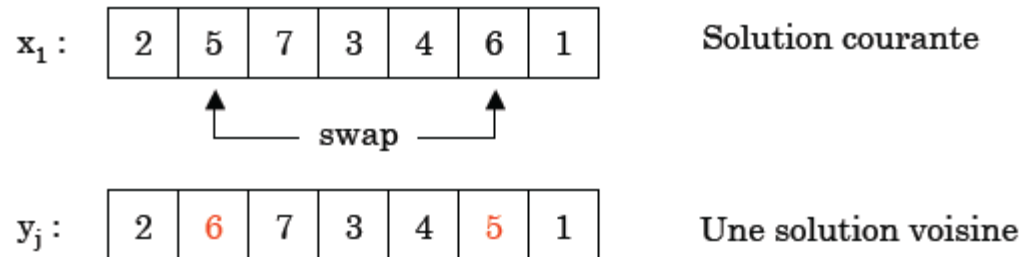
# Variantes (5) : Recherche Tabou

---

- « Liste » Tabou
  - **Conserver les mouvements effectués** et non pas les solutions visités
    - Exemple : variables échangées (swap)
    - Plus rapide à vérifier et moins coûteux à mémoriser ...
  - **Est parcourue fréquemment** dans la recherche de solutions
    - Accès efficace pour vérifier si une solution est tabou
      - Table de hachage (sur les mouvements, sur la fonction objectif)
        - Si collision : Taille de la liste trop petite
  - **Ne pas déconnecter** la solution optimale de la solution courante
    - Les informations restent dans la liste pendant une durée limitée (ie un nombre d'itérations)

# Exemple

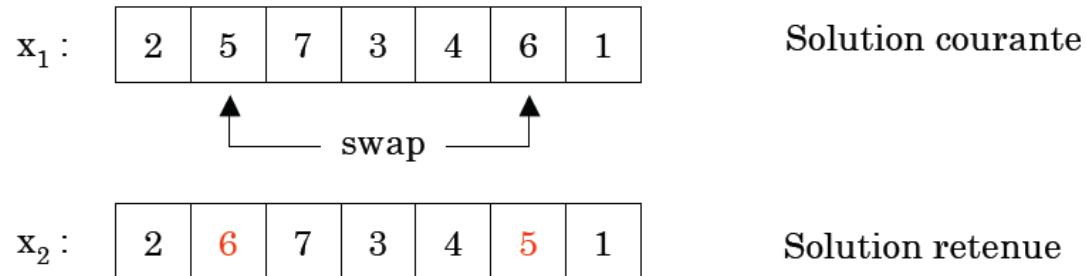
- Exploration d'un voisinage



$$\frac{(n(n-1))}{2} \text{ voisins}$$

# Exemple

- Sélection d'un voisin

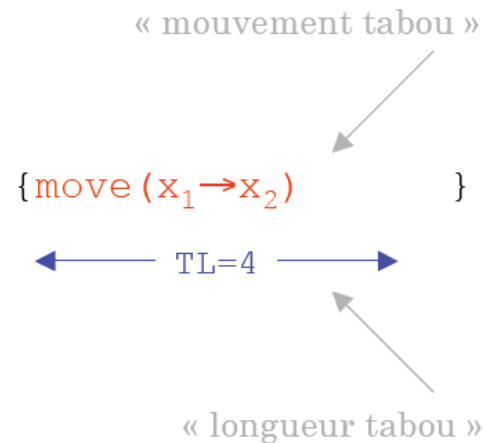


- Mouvement « Tabou »

Déclarer  
le  
mouvement

5  $\leftrightarrow$  6

tabou durant  
#n  
itérations



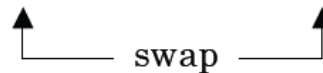
# Exemple

- Sélection d'un voisin

$x_1$  :

2	5	7	3	4	6	1
---	---	---	---	---	---	---

Solution courante



$x_2$  :

2	6	7	3	4	5	1
---	---	---	---	---	---	---

Solution retenue

- Structure pour les mouvements « Tabou »

	2	3	4	5	6	7
1	0	0	0	0	0	0
2		0	0	0	0	0
3			0	0	0	0
4				0	0	0
5					0	0
6						0

Déclarer  
le  
mouvement

5 ↔ 6

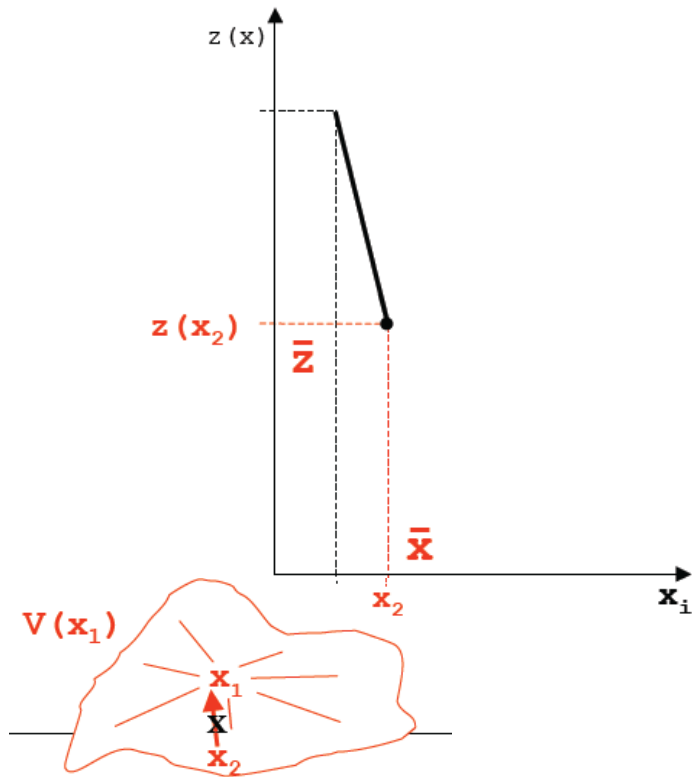
tabou durant  
#n  
itérations

	2	3	4	5	6	7
1	0	0	0	0	0	0
2		0	0	0	0	0
3			0	0	0	0
4				0	0	0
5					4	0
6						0



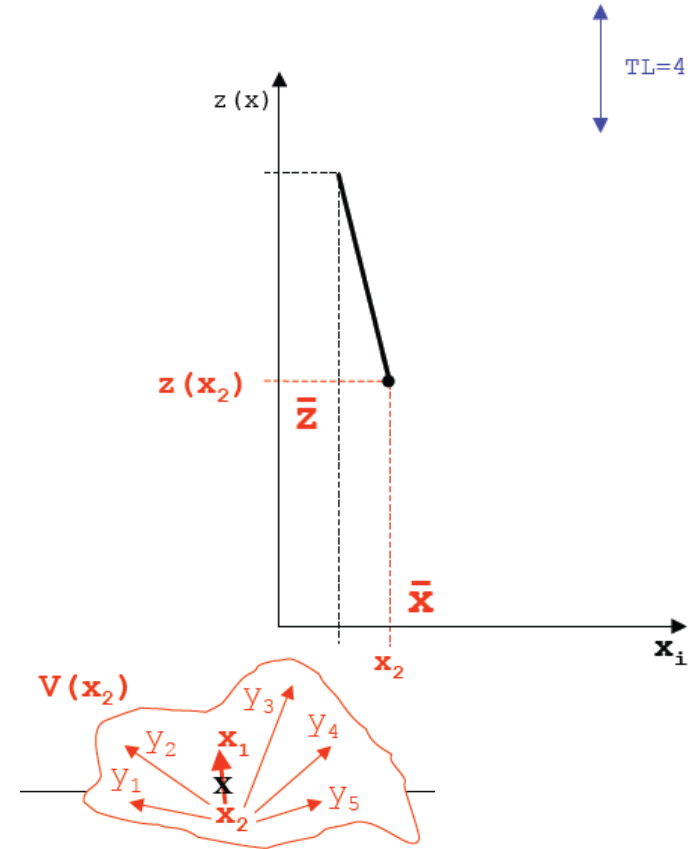
Choix d'un voisin et  
ajout d'un mouvement tabou

$$TM = \{\text{move}(x_1 \rightarrow x_2)\}$$

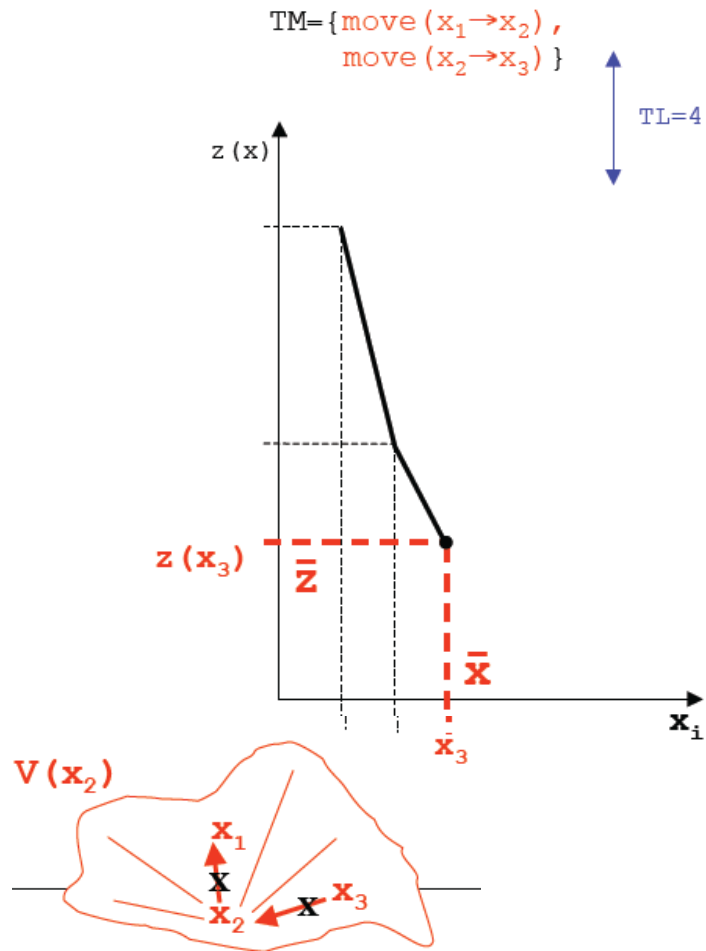


Exploration voisinage  
Sauf le mouvement tabou

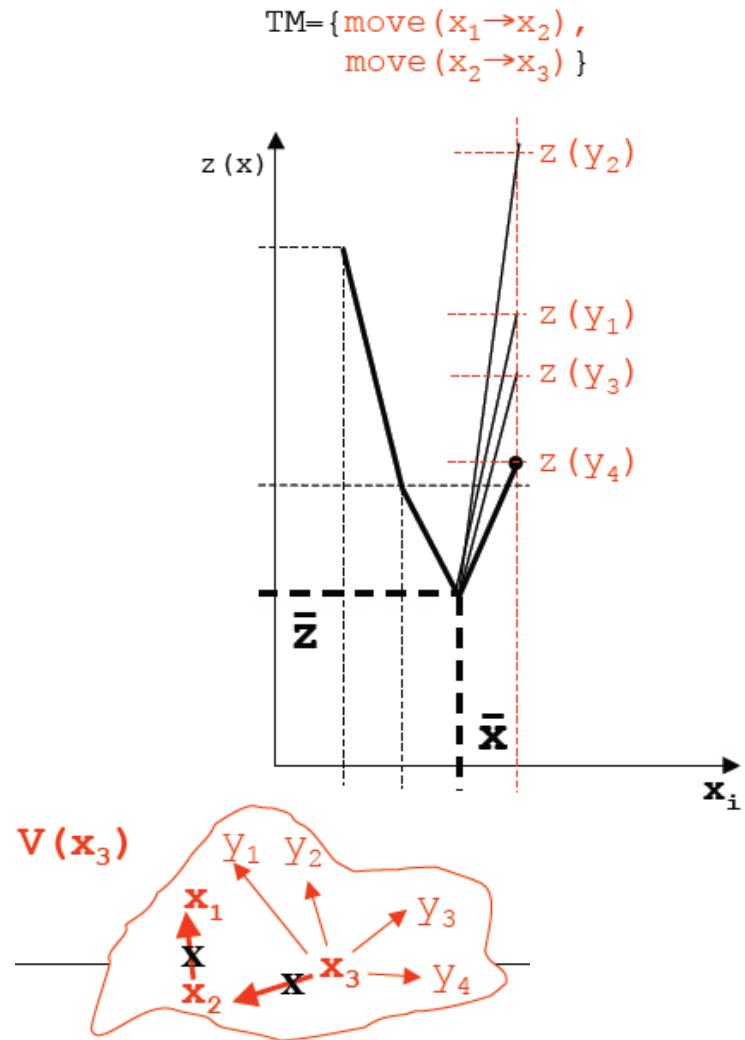
$$TM = \{\text{move}(x_1 \rightarrow x_2)\}$$



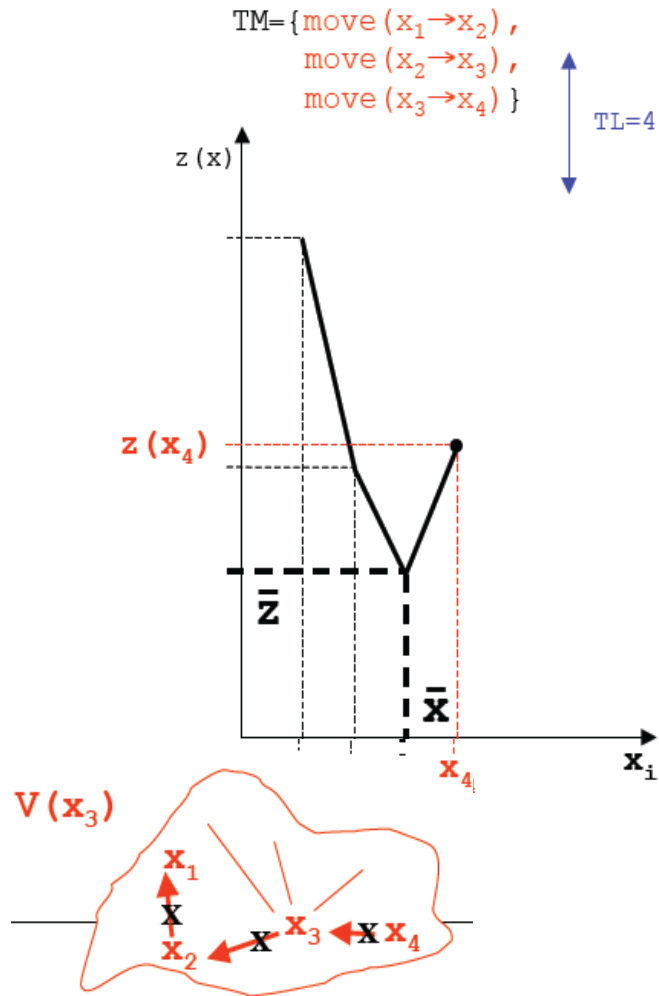
Choix d'un voisin améliorant et  
ajout d'un mouvement tabou



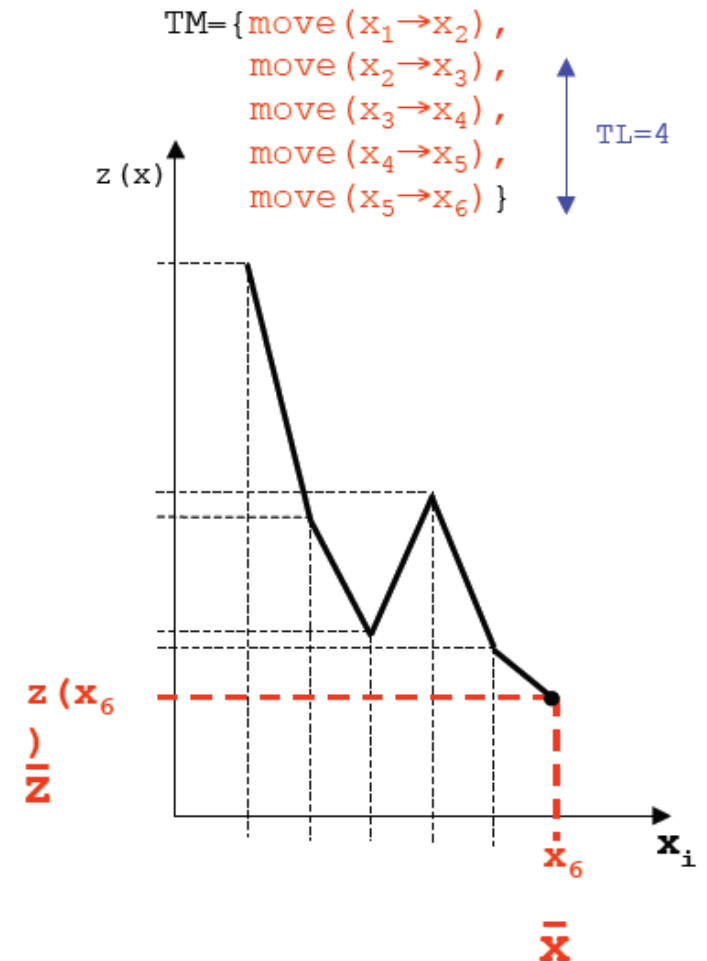
Exploration voisinage  
Sauf le mouvement tabou  
Aucun voisin améliorant



Choix d'un voisin et  
ajout d'un mouvement tabou



Prise en compte de la durée des interdictions



# Variantes (5) : Recherche Tabou

---

- **Exemple d'une liste Tabou**

- **Mouvement effectué sur les solutions :**

- Interdire le mouvement inverse pendant  $k$  itérations
      - Itération  $p$  : solution obtenue après  $\text{swap}(i, j)$
      - Interdire  $\text{swap}(j, i)$  jusqu'à itération  $p + k$
      - Matrice pour mémoriser toutes les paires de swap possibles
    - Mouvement inverse peut être complexe

- **Le contenu de la liste Tabou**

- Peut interdire plus de solutions que celles réellement explorées
  - Ne prévient pas totalement des risques de cycle

# Variantes (5) : Recherche Tabou

---

- **Durée des interdiction**

- **Ne conserver que les  $k$  derniers mouvements effectués**

- Valeur de  $k$  : longueur de la liste → compromis diversification / intensification
      - $k$  faible :
        - peu de voisins interdits risque de rester bloqué sur un optimum local
      - $k$  élevé :
        - beaucoup de voisins interdits / parcours potentiellement plus long
        - diversification importante mais on risque de louper l'optimum global
    - Réglage adaptatif en fonction du problème / d'une instance

- **Annuler une interdiction**

- Autoriser mouvement tabou si amélioration de la fonction objectif
  - Critère d'aspiration

# Variantes (5) : Recherche Tabou

---

- **Conception d'une méthode Tabou**
  - Définir un voisinage et une fonction d'évaluation
  - Définir une structure pour la liste Tabou
    - Quoi mémoriser
    - Pendant combien de temps
  - Définir conditions d'arrêt
  - Mémoriser meilleure solution visitée

# Variantes (5) : Recherche Tabou

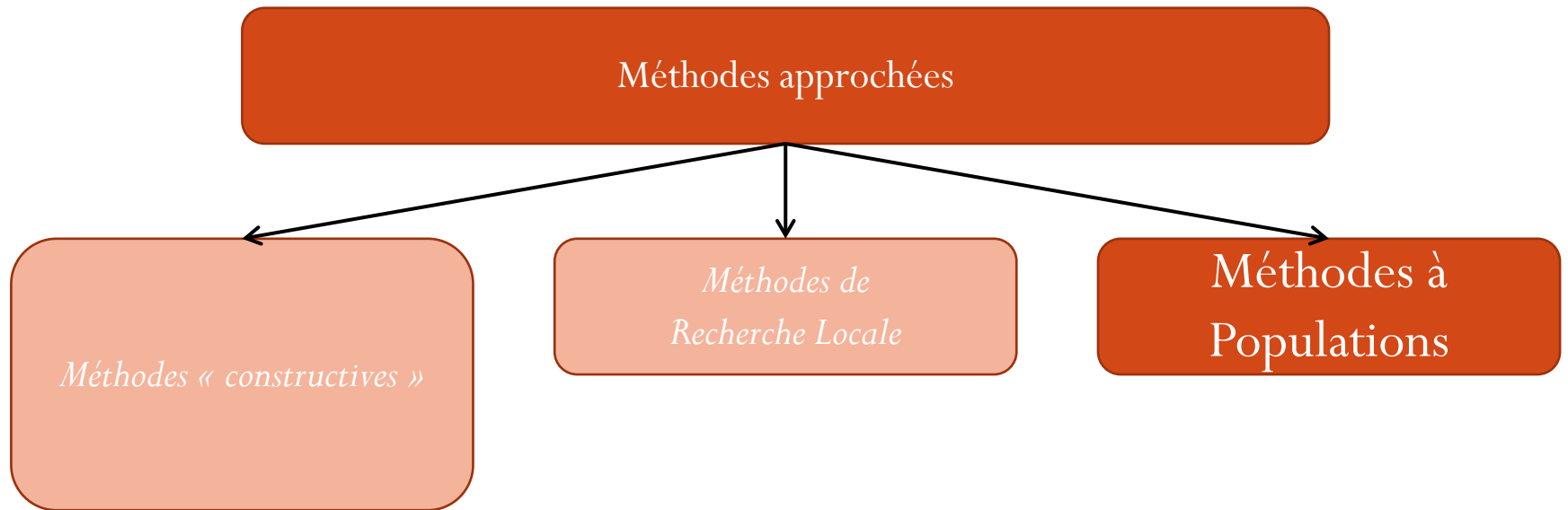
---

- **Attention à l'exploration du voisinage**
  - Taille : se limiter si besoin à une liste de voisins candidats
    - Aléatoire
    - Les plus pertinents a priori
  - Evaluation :
    - doit être efficace (incrémentale, approchée)
- **Variante**
  - Mémoire dite à long terme pour guider la recherche
    - Mémoriser les mouvements effectués et leur qualités respectives
    - Diversification : Guider vers des parties non explorées
    - Intensification : Repartir de caractéristiques de bonnes solutions

## Section 3. Méthodes approchées

---

- **Méthodes à population**





# Méthodes à population

---

- **Idées :**

- Considérer un ensemble de solutions
- Faire évoluer cet ensemble de solutions

- **Méthodes :**

- Algorithmes évolutionnaires (génétiques)
- Colonie de Fourmis (Ant Colony Optimization)
- Essaim (Particle Swarm Optimization)
- .....

# Algorithmes évolutionnaires (1)

---

- **Familles de méthodes inspirées des systèmes vivants**
  - Algorithmes génétiques : année 1970 / 1980 -- J. Holland 1975
  - Pour la résolution de problèmes d'optimisation (continue / discrète)
- **Principe :**
  - Faire évoluer progressivement un ensemble de solutions via des opérateurs « stochastiques » inspirés des processus de sélection naturelle et d'évolution génétique (Darwin)
    - Un enfant « hérite » du patrimoine génétique de ses deux parents
    - Des mutations génétiques (positives ou négatives) peuvent modifier certains gènes
    - Parmi les descendants : seuls les plus « adaptés » à l'environnement survivent
    - Le hasard joue un rôle important pour produire des enfants différents de leurs parents
    - La sélection naturelle effectue le tri entre les évolutions favorables et celles qui ne le sont pas

# Algorithmes évolutionnaires (2)

---

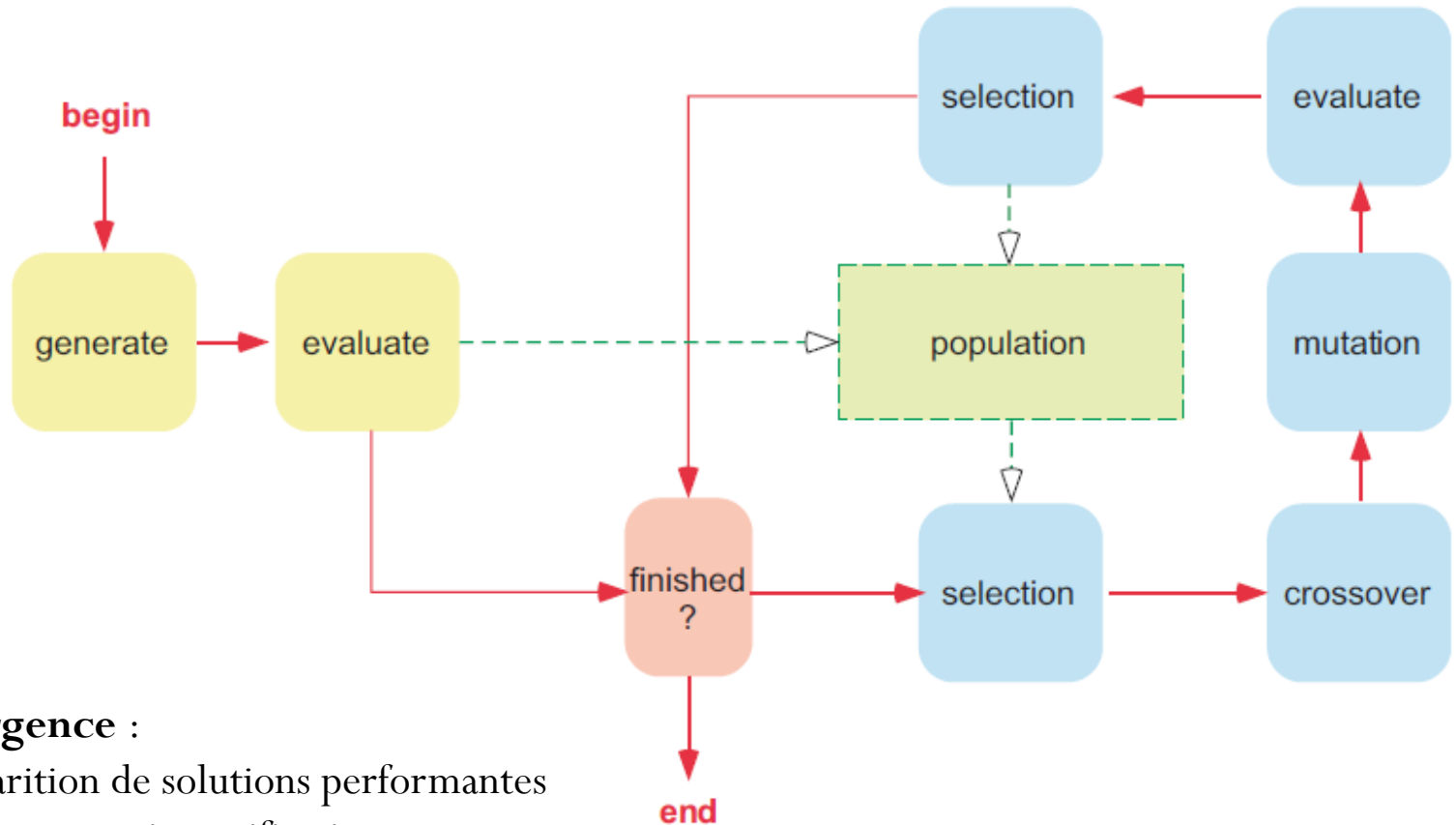
- **Vocabulaire :**

- Solution = un individu ayant une évaluation (fitness)
- Ensemble de solutions = une population
- Evolution de la population de solutions
  - Croisement : combiner solutions (parents) pour obtenir une nouvelle solution (enfant)
  - Mutation : modifier solutions
- Génération = une itération d'évolution

- **Mécanismes :**

- Evaluation des solutions
- Sélection des meilleures solutions
- Croisements
- Mutations

# Algorithmes évolutionnaires (3)



## Convergence :

- Apparition de solutions performantes
- Croisement : intensification
  - Les meilleures solutions « parents » donnent les meilleures solutions « enfants »
- Mutation : diversification

# Algorithmes évolutionnaires (4)

---

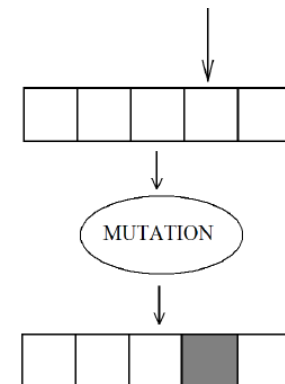
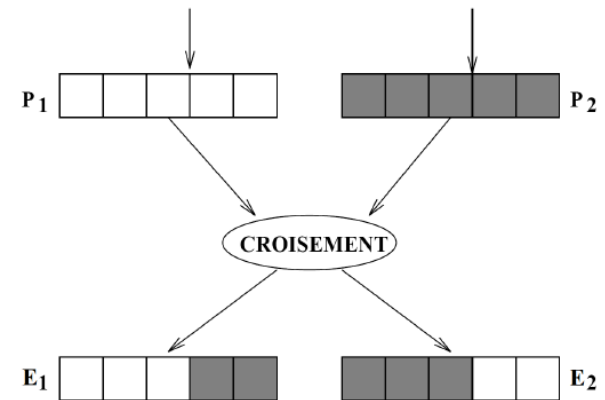
- **Algorithme général**

1. Initialiser une population de solutions
2. Evaluer les individus
3. while (conditions) loop
4.     Sélectionner parents
5.     Combiner parents pour produire enfants
6.     Modifier enfants
7.     Evaluer les nouveaux individus
8.     Sélectionner la nouvelle population
9. end while
10. return Meilleur individu

# Algorithmes évolutionnaires (4)

- **Points clés de la méthode**

- Codage des solutions
- Génération d'une population initiale
- Processus de sélection :
  - parents
  - nouvelle population
- Opérateurs de mutation et de croisement
- Paramètres :
  - Taille de la population
  - Critères d'arrêt
  - Probabilité de mutation



# Représentation des solutions / Evaluation

---

- **Représentation des solutions**

- Variables discrètes :

- codage binaire

0	0	1	1	0	1	0	1
---	---	---	---	---	---	---	---

- permutation

D	A	F	G	H	C	E	B
---	---	---	---	---	---	---	---

- Fonction d'évaluation :

- Mesure du score de chaque solution
      - Fonction objectif ou autres mesures
    - est utilisée pour le processus de sélection (solutions parents; nouvelle génération)
    - Attention au cout de calcul
      - Impact du codage sur la fonction d'évaluation

# Population de solutions

---

- **Population de solutions :**

- Ensemble de solutions réparties dans l'espace de recherche
  - Génération aléatoire
  - Heuristique gloutonne
  - Solution existante
- Introduire de bonnes solutions

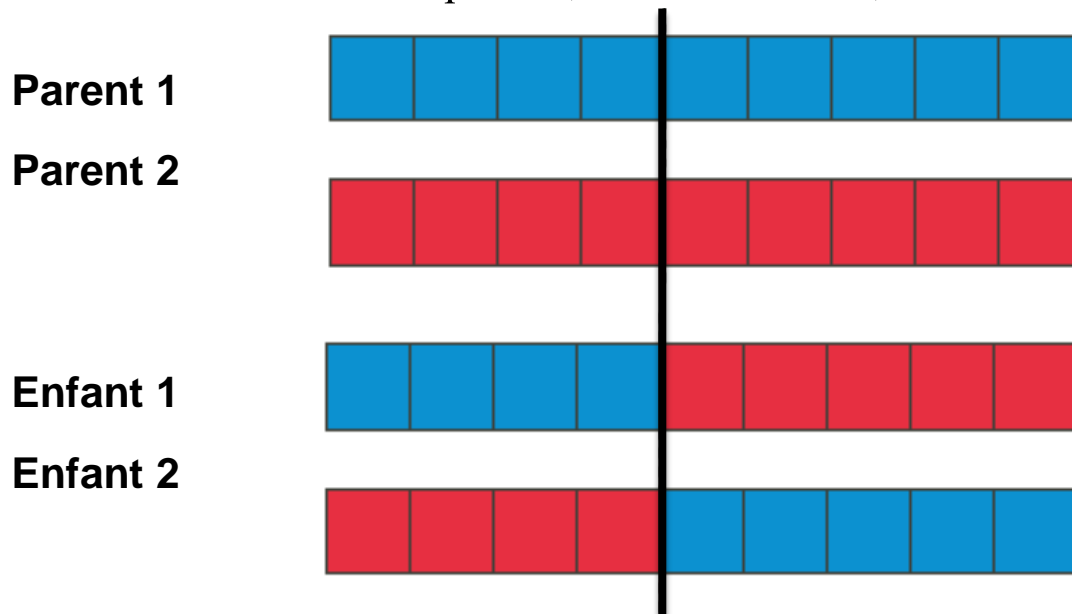
- **Taille population**

- Si trop petite : perte de diversité
- Si trop grande : temps de calcul important
- Paramètre à régler expérimentalement



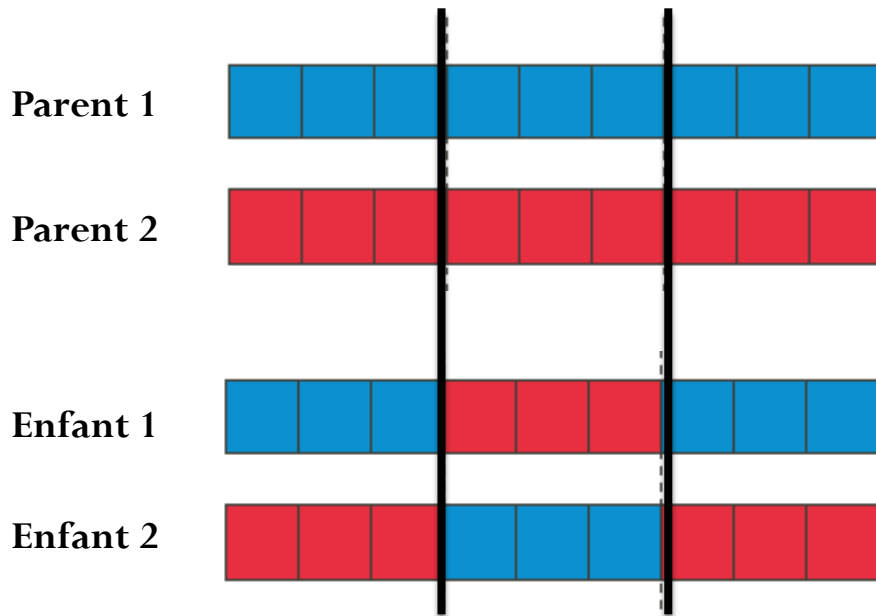
# Evolution de solutions : croisement (1)

- **Evolution de solutions : combiner 2 solutions**
  - Croisement :
    - Découper le vecteur associé à chaque solution en k morceaux
    - Recombiner les morceaux pour obtenir de nouvelles solutions
  - Croisement à 1 point (choisi au hasard)

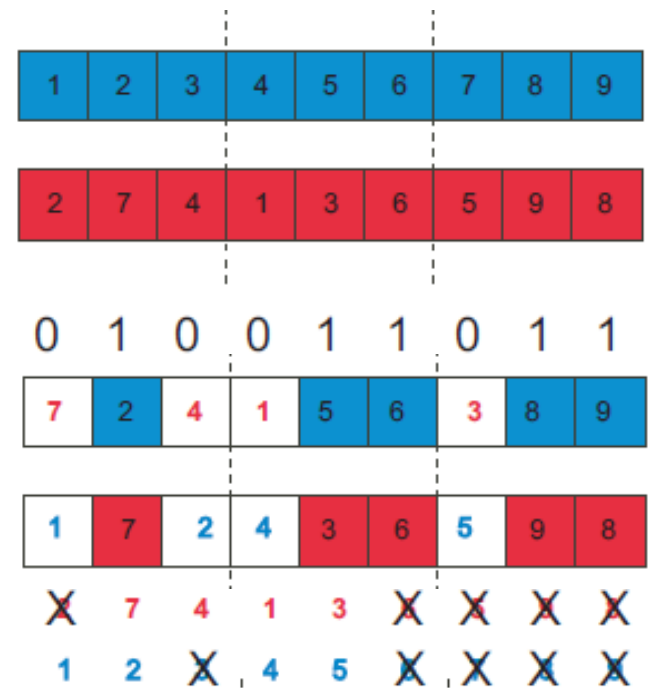


# Evolution de solutions : croisement (2)

## Croisement multi-point



## Masque

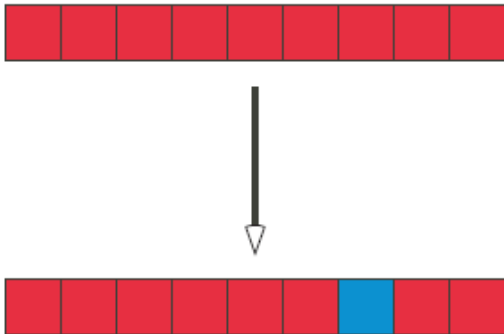


# Evolution de solutions : mutation

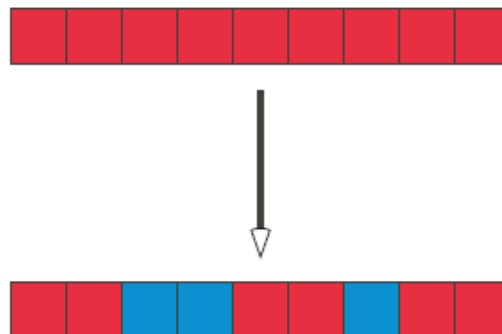
- **Mutation**

- Perturber une solution de manière aléatoire
  - Probabilité assez faible

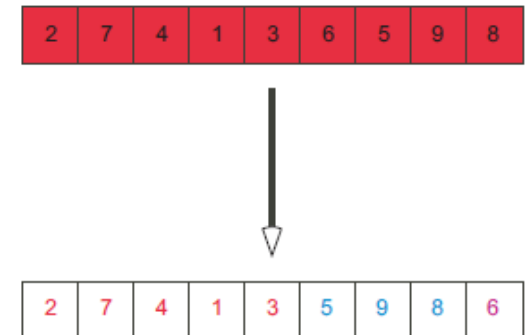
## Mutation 1 élément



## Mutation plusieurs éléments



## Décalage



# Evolution de solutions

- Croisement et Mutation peuvent produire des solutions non réalisable

- Exemple : vecteur = une permutation (TSP)

Parent 1	A	B	C	D	E	F	G	H
Parent 2	D	A	F	G	H	C	E	B
Enfant 1	A	B	C	D	H	C	E	B
Enfant 2	D	A	F	G	E	F	G	H

- **Attention** : les enfants ne sont pas des solutions réalisables

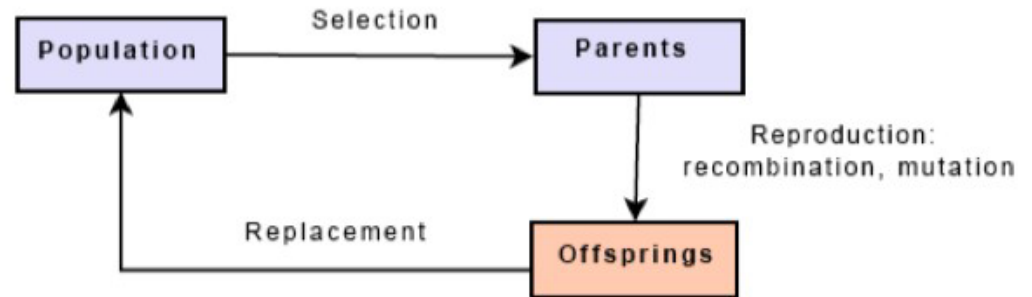
A	B	C	D	H	F	E	G
D	A	F	G	E	B	C	H

- Les réparer
- Recherche locale

# Sélection (1)

- **Deux étapes de sélection**

- Sélection de solutions parents
- Sélection pour nouvelle population



- **Sélection de parents :**

- Fixer un nombre de solutions-enfants à générer

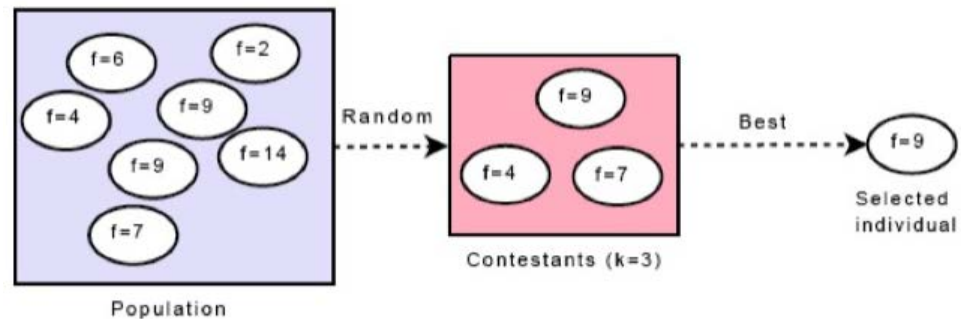
- **Elitisme :**

- Sélectionner uniquement les solutions les plus performantes / fitness
- Risque de convergence prématurée de la méthode

# Sélection (2)

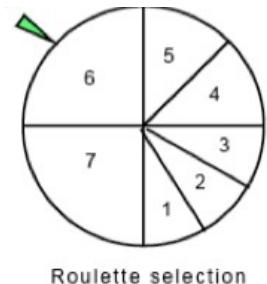
- **Sélection de parents :**

- Par tournoi (2 à 2) :
  - choisir une paire de solutions au hasard et conserver la meilleure. Itérer jusqu'à avoir suffisamment de solutions sélectionnées



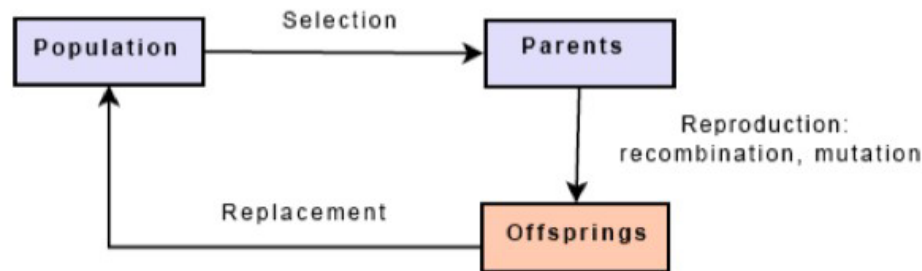
- Par roulette/par rang :
  - associer une valeur / un rang à chaque solution. Tirer au hasard de telle sorte qu'un individu important ait une probabilité de sélection plus forte

Individuals:	1	2	3	4	5	6	7
Fitness:	1	1	1	1,5	1,5	3	3



# Sélection (3)

- Sélection (remplacement) d'une nouvelle population :



- A la fin d'une itération :
  - Solutions de la populations initiales
  - Solutions obtenues par combinaison / mutation
- Nouvelle population :
  - Choisir les  $\mu$  meilleures solutions parmi les  $l$  enfants générés
  - Choisir les  $\mu + l$  meilleures solutions parmi les parents + les enfants
  - Taille constante

# Variantes

---

- **Variantes :**
  - Très nombreuses
    - Liées aux différents paramétrages / options
      - Plein de noms de méthodes ....
  - Algorithmes génétiques : méthode « de base »
    - Codage binaire principalement
  - Programmation génétique :
    - Espace de recherche de très grande taille
    - Parallélisation calculs










# Variantes

---

- **Algorithme général**

1. Générer une population  $P$  de solutions
2. Réparer  $P$  (recherche locale)
3. répéter
4.     // Croisement
5.     Générer un **ensemble**  $P_c$  par **croisement**
6.     Réparer  $P_c$
7.     // Mutation
8.     Générer un **ensemble**  $P_m$  par **mutation**
9.     Réparer  $P_m$
10.    // Nouvelle population
11.    **Sélectionner** la nouvelle population à partir de  $P, P_c, P_m$
12. Jusqu'à (conditions d'arrêt)
13. return Meilleur individu

# Exemple de variante

1. Générer une population  $P$  de solutions de taille  $N$
2.  $G \leftarrow$  Nb max générations  Paramètres
3. Pour  $g$  de 1 à  $G$
4.  $P_n \leftarrow \emptyset$   Taux de croisement
5. pour  $i = 1$  à  $c_{cross} \cdot N$  faire  Croisement 1 point
6.  $(p_1, p_2) \leftarrow$  sélectionner 2 parents
7.  $e \leftarrow$  Croisement( $p_1, p_2$ )  Probabilité mutation
8. si Random( $x \in [0,1]$ )  $< \rho_{mut}$  alors
9.  $e \leftarrow$  Mutation( $e$ )  Swap
10.  $e \leftarrow$  RechercheLocale( $e$ )  Descente (avec swap)
11.  $P_n \leftarrow P_n \cup \{e\}$
1.  $P \leftarrow$  Remplacement( $P, P_n$ )  Elitisme
2. Fin pour
3. return Meilleure solution

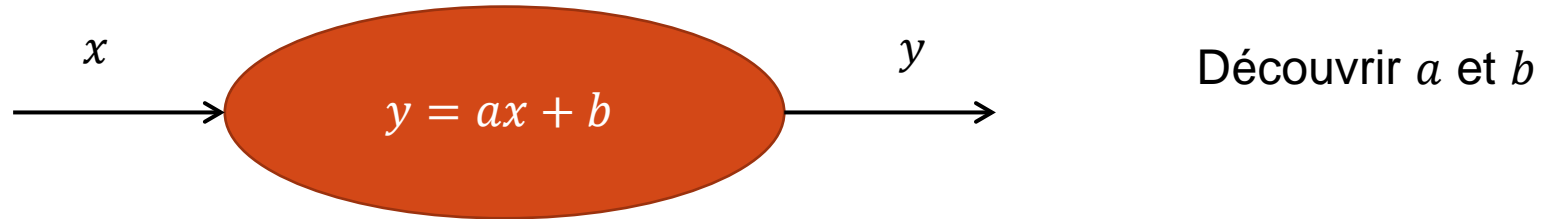
# Autres variantes

---

- Processus « d'éducation » des solutions enfants
  - Ajouter un mécanisme de recherche locale pour la mutation
  - Algorithmes mémétiques
- Fonction d'évaluation :
  - fonction objectif + **diversité des solutions**
- Plateformes :
  - Exemple : DEAP (Distributed Evolutionary Algorithms in Python)

# Application en apprentissage

- **Processus d'apprentissage supervisé**



- Données d'apprentissage
  - $x = 3 \rightarrow y = 5$  ; hypothèse  $a = 0, b = 5; a = 2, b = -1; \dots$
  - ....
- Solutions approchées (ne pas apprendre « par cœur » la base d'apprentissage)
  - Évaluation des meilleures solutions
    - Distance entre résultat fourni et résultat attendu pour un ensemble de données
    - Minimiser cette distance ➔ processus d'apprentissage

# Colonies de Fourmis (1)

## Ant Colony Optimization

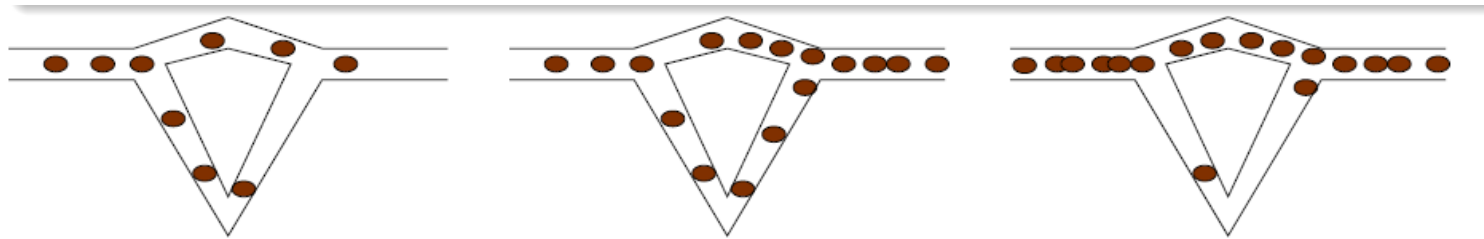
- **Idée :**
  - Auto-organisation des insectes sociaux
  - **Emergence** d'un comportement global à partir **d'interactions** locales
    - Emergence : comportement global non programmé
    - Comportement : structure
    - Interactions : communications directes ou indirectes
  - Systèmes dynamiques
  - Robustesse et flexibilité



# Colonies de Fourmis (2)

- **Recherche de nourriture par une colonie de fourmis**

- Emergence de « plus courts chemins »



- Initialement : tous les chemins sont équiprobables
- Les fourmis prenant le plus court chemin reviennent le plus vite
  - Chemin le plus court a plus grande fréquence de passage
    - Accroissement de la concentration en phéromones sur ce chemin
    - Evaporation sur les autres chemins
- Méthode ACO Dorigo 1992

# ACO (1)

---

- **Principe d'auto-organisation**

- Ensemble d'agents (fourmis) communiquant indirectement par des dépôts de phéromones
  - Remarque : ici les communications directes entre agents ne sont pas considérées
- Les phéromones sont déposées par les agents au cours de leurs déplacements
- La quantité déposée est contrôlée par chaque agent
- Les phéromones déposées attirent les autres agents
- Evaporation au cours du temps

# ACO (1)

---

- **Algorithme général**

Initialisation des traces de phéromones

Répéter

    Chaque fourmi calcule un chemin : aléatoire / phéromones, ...

    Mise à jour des traces de phéromones

Jusqu'à : Condition d'arrêt

- Ajout de recherche locale pour améliorer les solutions
- Définir un graphe pour représenter la construction de solutions
  - Sommets : composants de solutions
  - Arcs : succession de composants
  - Solution : meilleur chemin dans le graphe



# Application ACO pour le TSP

---

- Un ensemble d'agents situés initialement sur le même sommet
- **Choix d'un nouvel arc**
  - Pour chaque agent (placé au sommet  $i$ ) la probabilité de choisir le prochain arc  $(i, j)$  dépend de :
    - Concentration en phéromones sur l'arc  $(i, j)$  par rapport aux autres arcs
    - Mesure de la qualité de l'arc (inverse de sa longueur par exemple)
    - Pondérations entre ces 2 éléments
- **Mise à jour des traces**
  - Renforcement : ajouter des phéromones sur tous les arcs de la meilleure solution
  - Evaporation : diminuer la quantité de phéromones sur chaque arc (de manière proportionnelle à sa qualité)
    - Borner la quantité minimale et maximale de phéromones sur chaque arc

# Composants ACO

---

- **Recherche Gloutonne**

- Construction de solutions par une méthode gloutonne

- **Recherche Aléatoire**

- 

- **Recherche adaptative**

- Adapter l'exploration en fonction d'un historique
  - Capitalisation de l'expérience par les dépôts de phéromones
  - Exploitation en biaisant la recherche gloutonne par rapport aux phéromones

- **Intensification / Diversification**

- Intensifier : dépôt de phéromones sur les zones prometteuses – Sélection en fonction des quantités de phéromones
- Diversifier : évaporation des phéromones

# Essaims particuliers (1)

## Particle Swarm Optimization

- **Idée :**

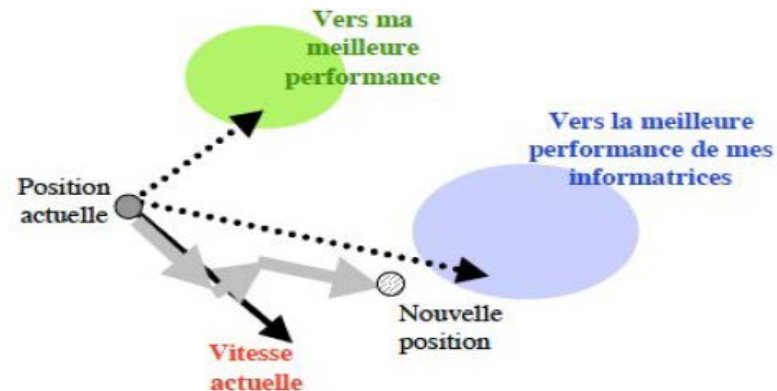
- Population de solutions : essaim
- Compromis entre trajectoire individuel et trajectoire du groupe

- Un individu :
  - Suit sa propre trajectoire
  - Subit l'influence des autres
  - Mémorise sa meilleure performance
- Exploration de l'espace des solutions
- Origine : optimisation continue



# Essaims particuliers (2)

- **Chaque individu est un élément de l'espace de recherche**
  - Position dans l'espace de recherche
  - Vitesse : dépend des voisins, des bonnes solutions visitées
  - Voisinage : ensemble d'individus auxquels il est relié
- **Déplacer chaque individu en fonction :**
  - Comportement « individuel » : suivre sa trajectoire
  - Comportement « conservateur » : revenir vers la meilleure position déjà visitée
  - Comportement « collectif » : suivre le meilleur voisin



## Section 4. Pour aller plus loin ....

---

- Méthodes hybrides
- Méthodes multi-objectif

# Contraintes et Objectifs

---

- **Contraintes vs Objectif**

- Introduire des « contraintes » dans la fonction Objectif
- Objectif :
  - nb de contraintes non satisfaites (exemple capacité à respecter)
  - on cherche à minimiser cet objectif, lorsqu'il vaut 0 :
    - obtention d'une solution admissible

- **Intérêt :**

- Explorer des affectations à la limite entre cohérentes et incohérentes
- Traiter des pbs de satisfaction sous forme de pbs d'optimisation

- **Difficulté :**

- Combiner cet objectif avec celui (ceux) déjà existant(s)
- Combinaison linéaire ➔ 1 seul objectif

# Hybridation de méthodes

---

- **Hybridation de méthodes**

- Algo. Mémétique
  - Combiner algorithmes évolutionnaires et recherche locale
    - Population initiale
    - Mutation → recherche locale
- Explorer plusieurs solutions :
  - Recherche locale + Mutation
- PLNE / PPC + méthodes approchées :
  - Beam Search : Branch and Bound & heuristique
  - Exploration de grands voisinages avec PPC / PLNE

# Optimisation multi-objectif (1)

---

- **Pourquoi ?**

- Prendre en compte simultanément plusieurs objectifs contradictoires
  - Exemple : temps / argent

- **Formulation**

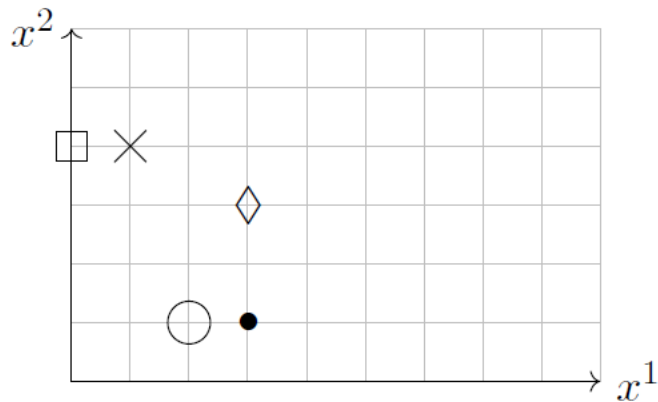
- $Min Z = (c_1(X), c_2(X), \dots, c_p(X))$
- Tel que :  $X \in \Omega$ 
  - $X$  : ensemble des variables,  $\Omega$  : ensemble des contraintes
  - $c_i(X)$  : une fonction objectif;  $p$  : nombre de fonctions objectif



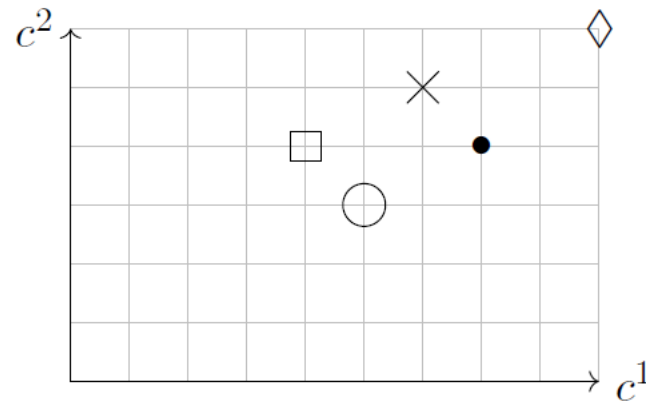
# Optimisation multi-objectif (2)

- Espace des solutions et espaces des objectifs

L'image de  $\mathcal{X}$  par  $c$  forme l'espace des objectifs  $\mathcal{Y} \subseteq \mathbb{N}^p$



(a) Espace des solutions.



(b) Espace des objectifs.

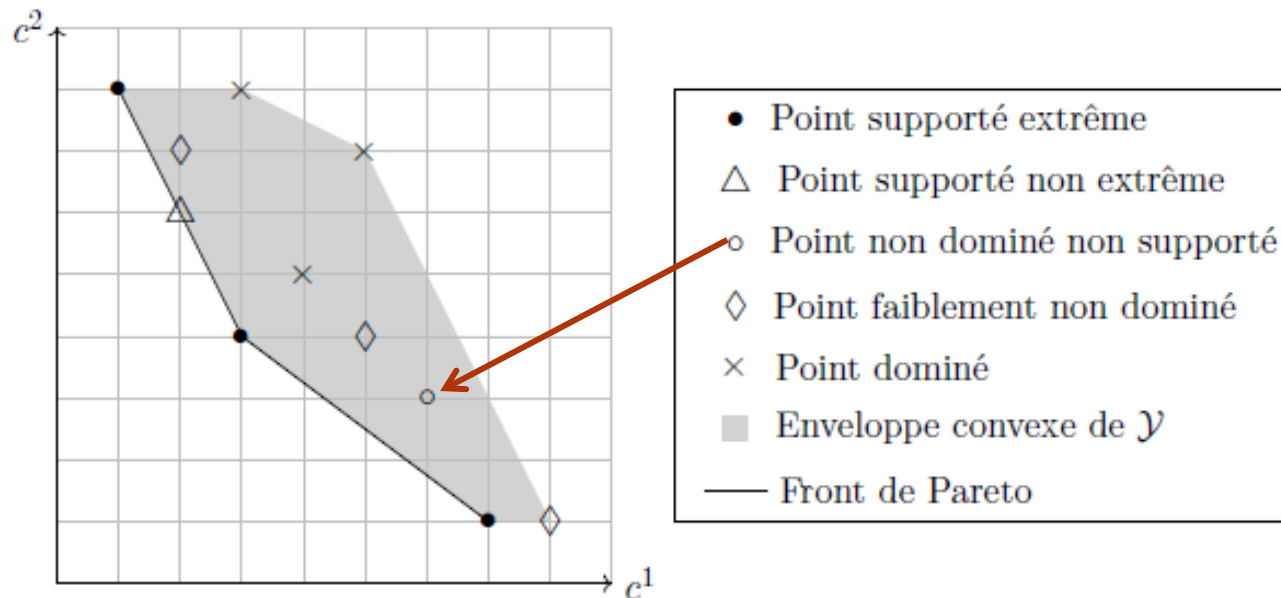
$$c^1(x) = 2x^1 + x^2 \text{ et } c^2(x) = x^1 + x^2$$

# Optimisation multi-objectif (3)

- **Ensemble des solutions réalisable un multi-objectif**
  - Un ensemble (possiblement non convexe) de points dans l'espace des objectifs
  - L'optimum : un ensemble de points : les **solutions de compromis**
    - Rechercher un équilibre tel que :
      - on ne peut pas améliorer un objectif sans détériorer au moins un des autres objectifs
- **Dominance de Pareto**
  - $x$  domine faiblement  $x'$ , noté  $x \leq x' \Leftrightarrow c^i(x) \leq c^i(x') \forall i \in \llbracket 1; p \rrbracket$
  - $x$  domine  $x'$ , noté  $x \leq x' \Leftrightarrow \begin{cases} c^i(x) \leq c^i(x') \forall i \in \llbracket 1; p \rrbracket \\ c^i(x) < c^i(x') \exists i \in \llbracket 1; p \rrbracket \end{cases}$
  - $x$  domine strictement  $x'$ , noté  $x < x' \Leftrightarrow c^i(x) < c^i(x') \forall i \in \llbracket 1; p \rrbracket$

# Optimisation multi-objectif (4)

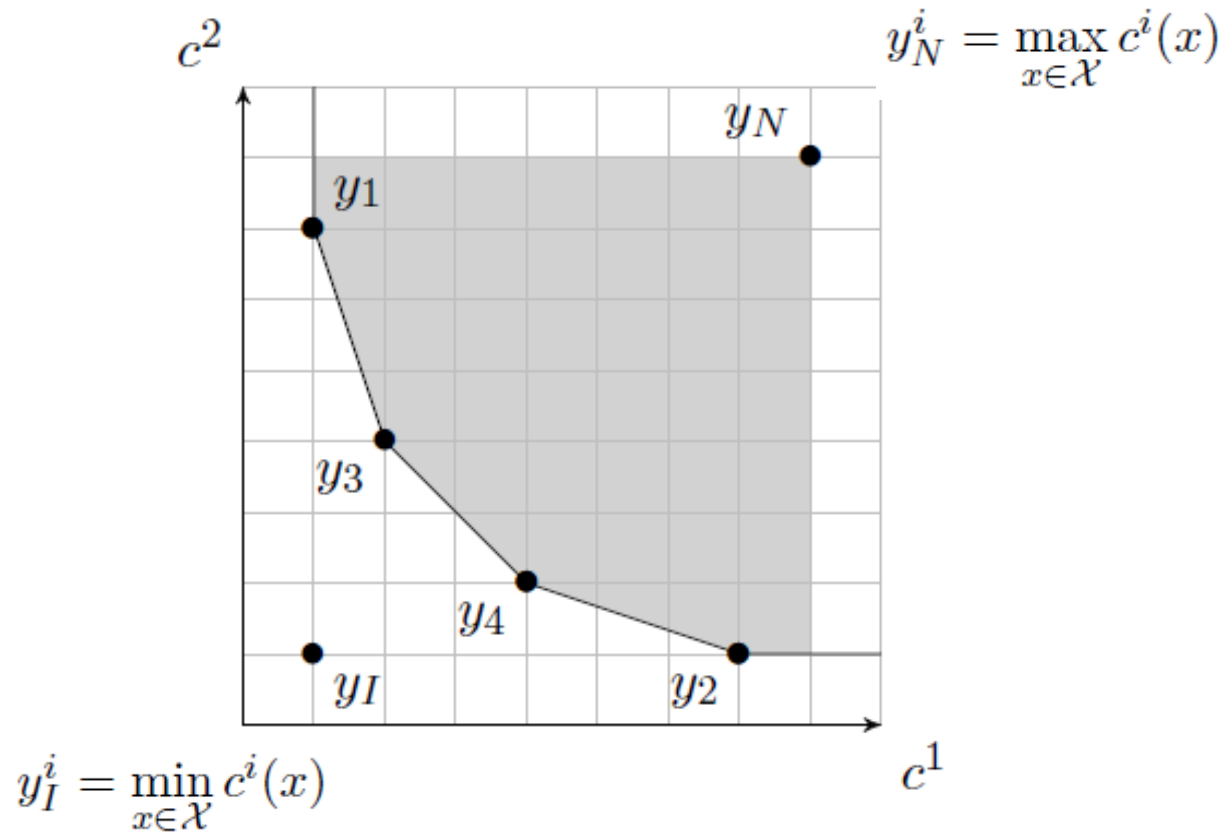
- **Front de Pareto**



- Points supportés : sur l'enveloppe convexe
- Points non supportés : à l'intérieur de l'enveloppe convexe
- Point non dominé supporté extrême = point extrême de l'enveloppe convexe

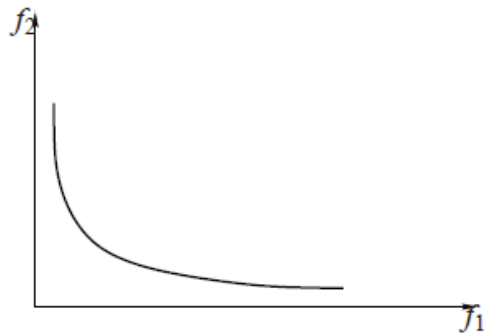
# Optimisation multi-objectif (5)

- Point Idéal  $y_I$  et point Nadir  $y_N$

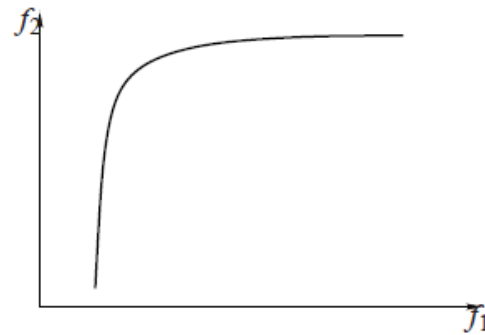


# Formes de Front de Pareto

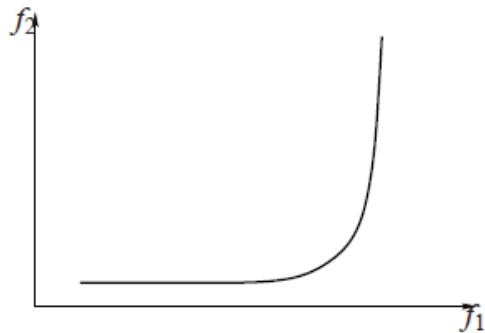
- **Forme des Fronts de Pareto à 2 objectifs**



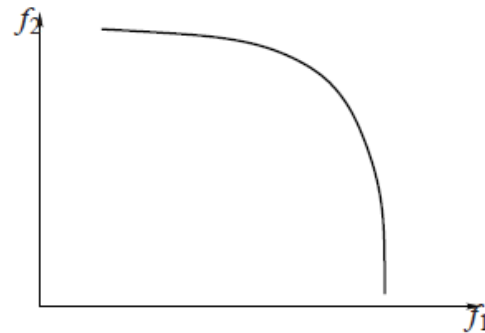
(a) Minimiser  $f_1$  minimiser  $f_2$



(b) Minimiser  $f_1$  maximiser  $f_2$



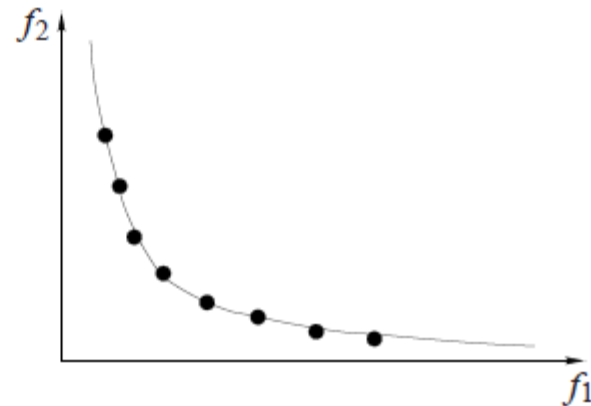
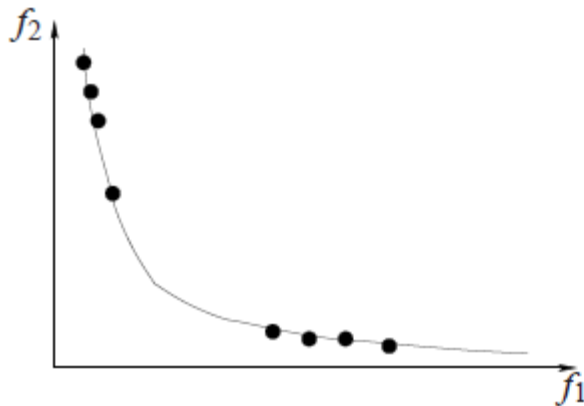
(c) Maximiser  $f_1$  minimiser  $f_2$



(d) Maximiser  $f_1$  maximiser  $f_2$

# Calcul d'un Front de Pareto (1)

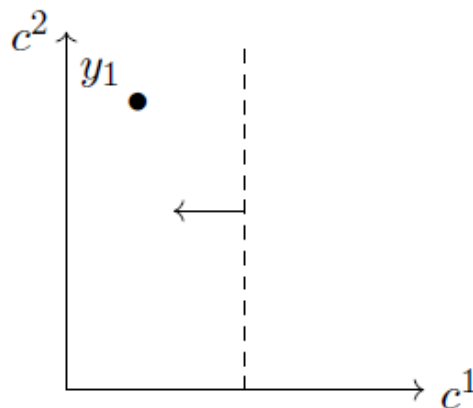
- **Comment obtenir un Front de Pareto**
  - Points extrêmes :
    - Optimiser 1 seul objectif
  - Bonne représentation du Front : solutions diversifiées



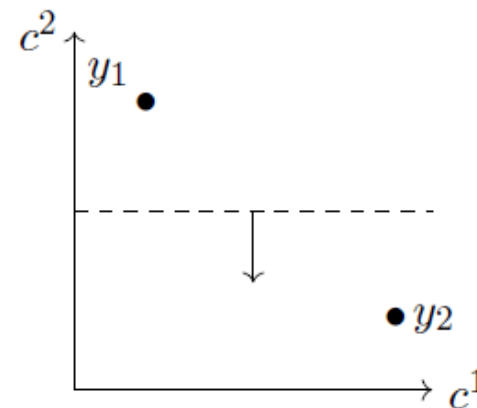
# Calcul d'un Front de Pareto (2)

- **Réduction à un seul objectif**

- Somme pondérée des objectifs
  - Exemple :  $Z(x) = w_1 \times c_1(x) + w_2 \times c_2(x)$
- Méthode dichotomique :
  - Initialisation : optimisation de chaque objectif



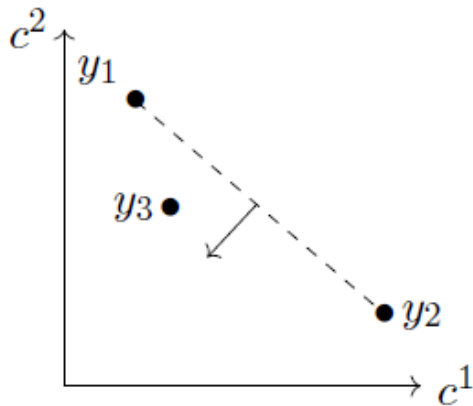
(a) Initialisation : recherche du point  $c^1$ -extrême  $y_1$ .



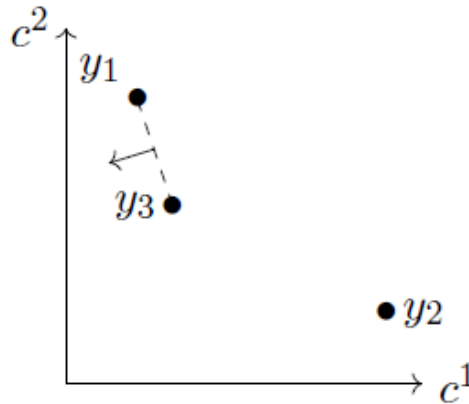
(b) Initialisation : recherche du point  $c^2$ -extrême  $y_2$ .

# Calcul d'un Front de Pareto (3)

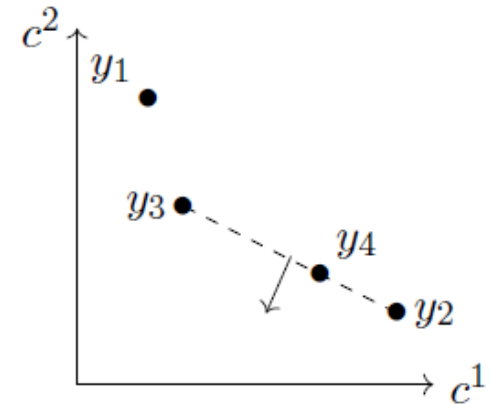
- Méthode dichotomique :



(c) Itération 1 : recherche dans la direction définie par  $y_1$  et  $y_2$  pour obtenir  $y_3$ .



(d) Itération 2 : pas de nouveau point trouvé dans la direction définie par  $y_1$  et  $y_3$ .



(e) Itération 3 : recherche dans la direction définie par  $y_3$  et  $y_2$  pour obtenir  $y_4$ , un point supporté non extrême.



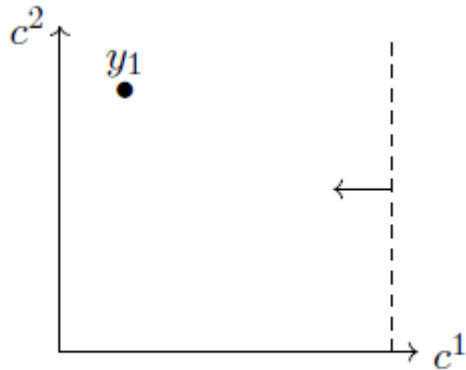
# Calcul d'un Front de Pareto (4)

- **Méthode epsilon contrainte**

- Ne conserver qu'un seul objectif
- Contraindre les autres par  $\epsilon_i$ 
  - Varier ces valeurs pour obtenir un front de Pareto

$$\min c_\varepsilon(x) = c^k(x)$$

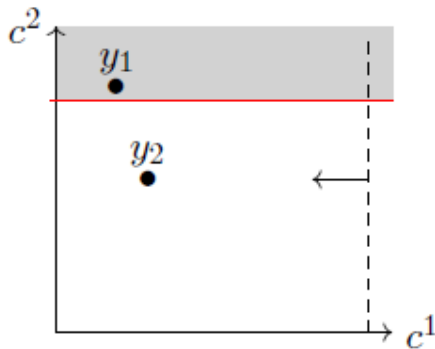
$$t.q. c^i(x) \leq \varepsilon_i, \quad i \in \llbracket 1; p \rrbracket \text{ et } i \neq k$$



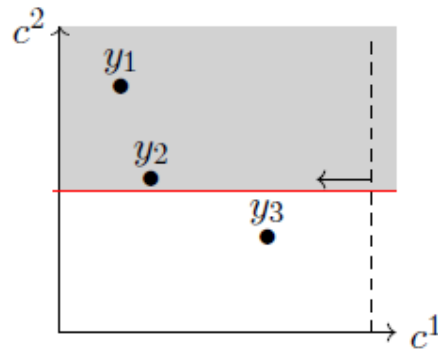
(a) Initialisation : recherche du point  $c^1$ -extrême  $y_1$ .

# Calcul d'un Front de Pareto (5)

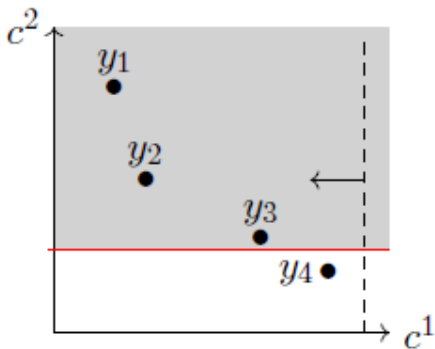
- Itérations



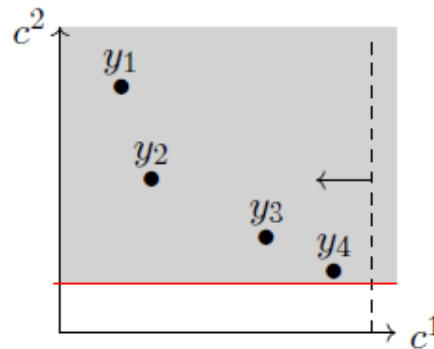
(b) Itération 1 : ajout de la contrainte  $\varepsilon$  et obtention de  $y_2$ .



(c) Itération 2 : modification de la contrainte  $\varepsilon$  et obtention de  $y_3$ .



(d) Itération 3 : modification de la contrainte  $\varepsilon$  et obtention de  $y_4$ .



(e) Itération 4 : Pas de nouveau point obtenu avec modification de la contrainte  $\varepsilon$ .

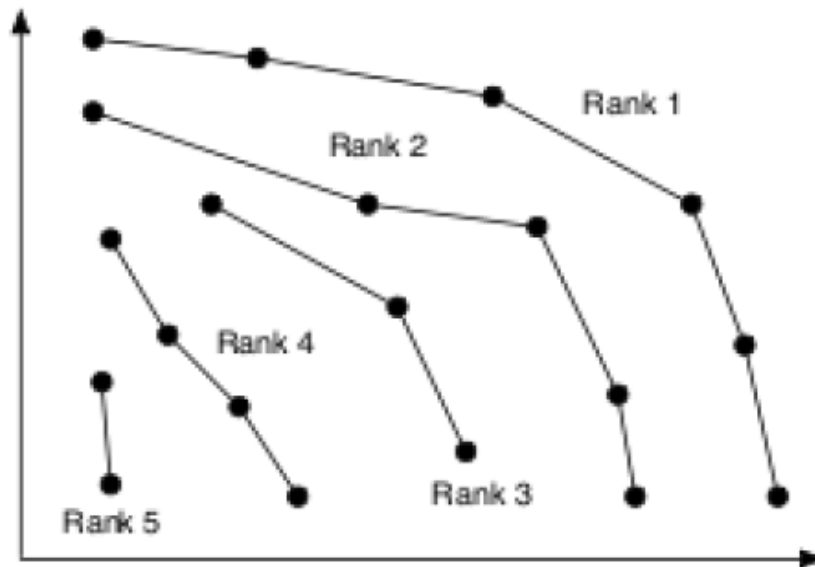
# Calcul d'un Front de Pareto (6)

---

- **De très nombreuses méthodes**
  - Littérature spécialisée sur le sujet
- **Méthode basée sur algorithmes évolutionnaires**
  - Modifier le processus de sélection pour prendre en compte les différents objectifs
  - Utiliser la sélection par tournoi
    - avec ordre lexicographique sur les objectif
      - Exemple :  $f_1 > f_2 > f_3$
    - avec objectif choisi aléatoirement
    - avec comparaison majoritaire sur les fonctions objectifs

# Calcul d'un Front de Pareto (7)

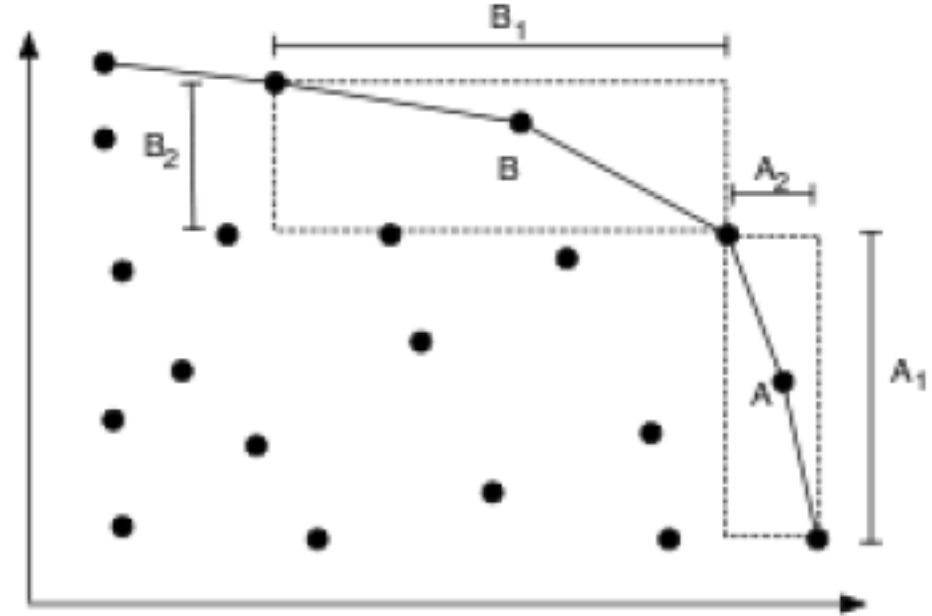
- **Considérer explicitement les objectifs**
  - Utiliser un rang de Pareto
    - Rang 1 : solutions non dominées
    - Rang 2 : solutions dominées uniquement par solutions de rang 1
    - Etc
  - Ignorer les solutions dominées pour la génération suivante



# Calcul d'un Front de Pareto (8)

- **Diversité des solutions**

- Bon échantillonnage du Front de Pareto
- Introduire une mesure d'espacement entre solution (en complément du rang)



- **Algorithme NSGAII**

- Algorithme évolutionnaire
- Rang + espacement
  - Pour 2 solutions de même rang, privilégier celle ayant le plus grand espacement
- Stratégie de remplacement + archivage meilleures solutions

## Section 3. Conclusion

---

# Conclusion (1)

---

- **Analyse du problème**

- Quelles sont les décisions à prendre ?
- Quelles sont les contraintes ?
- Quelles sont les objectifs ?
  - Unique ?
  - Multiple ?

- **Modélisation**

- Caractéristiques du modèle obtenu (taille, ....)
- Lien avec des problèmes classiques
- Complexité

# Conclusion (2)

---

- **Résolution**

- Méthode exacte ou approchée
- Concevoir une méthode spécifique ?
- Utiliser des outils de résolution ?
  - Diversités des outils / plateformes / langages

- **Décomposer**

- Parties essentielles du problème
- Intégrer au fur et à mesure

- **Evaluation**

- Instances « jouet » / Instances taille réelle
- Analyse critique des résultats



# Diversité / Similarités des méthodes (1)

---

- **Deux grandes familles d'approche :**
  - Méthode de Recherche locale (à solution unique) :
    - Evolutions successives d'une solution
  - Méthode à population de solutions :
    - Evolution de la population : combinaison/mutation
    - Processus de sélection
- **Des concepts similaires :**
  - Résultat : solution approchée
  - Sortir des optima locaux
    - Compromis entre Diversification et Intensification
  - Utilisation de l'aléatoire : méthodes stochastiques

# Diversité / Similarités des méthodes (2)

---

- **Intensification**
  - Pousser la recherche autour de solutions de bonne qualité
- **Diversification**
  - Déplacer la recherche vers de nouvelles parties non explorées de l'espace des solutions
- **Compromis entre les deux :**
  - Spécifique de chaque méthode approchée
- **De très (trop ?) nombreuses méthodes**

# Comment choisir ?

---

- **Quelle méthode choisir pour résoudre un problème**
  - Produire une bonne solution
  - Temps de calcul raisonnable
  - Mais pas de recette miracle
    - Tester plusieurs méthodes, bibliographie, expertise, connaissance métier, ...
  - No Free Lunch Theorem (for Search / Optimization / Learning)
    - Il n'y a pas de méthodes meilleure que les autres sur l'ensemble des problèmes
    - "any two optimization algorithms are equivalent when their performance is averaged across all possible problems"

---

---