

Résoudre des problèmes NP-dur ?

Recherche locale – les bases

ACT – Algorithmes et Complexité – M1 Info

Bilel Derbel
Département Informatique - Univ. Lille

Paradigmes de recherche

- **Recherche systématique / complète**

- 'Traverser/parcourir' l'espace de recherche de façon systématique
- Evaluate (implicitement) toute solution jusqu'à solution optimale trouvée (B&B, A*, ...).

- **Recherche Constructive (le retour des gloutons!)**

- Espace de recherche = solutions candidates partielles
- Etape de recherche = extension avec un ou plusieurs nouveaux composants

- **Recherche perturbative**

- Espace de recherche = ensemble complet de solutions candidates
- Une étape de recherche = modification d'une ou plusieurs composants d'une solution

- **Recherche Locale**

- Commencer avec une solution initiale
- 'itérativement' visiter une solution 'voisine'
- Autres ...

Paradigmes de recherche

- **Recherche systématique / complète**

- 'Traverser/parcourir' l'espace de recherche de façon systématique
- Evaluate (implicitement) toute solution jusqu'à solution optimale trouvée (B&B, A*, ...).

- **Recherche Constructive (le retour des gloutons!)**

- Espace de recherche = solutions candidates partielles
- Etape de recherche = extension avec un ou plusieurs nouveaux composants

- **Recherche perturbative**

- Espace de recherche = ensemble complet de solutions candidates
- Une étape de recherche = modification d'une ou plusieurs composants d'une solution

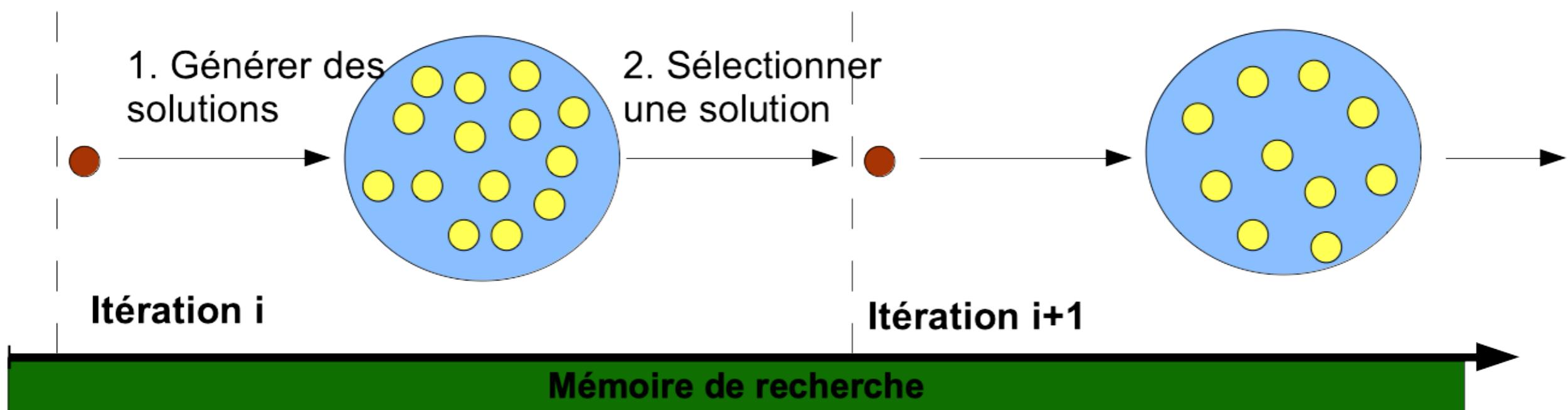
- **Recherche Locale**

- Commencer avec une solution initiale
- 'itérativement' visiter une solution 'voisine'
- Autres ...

Recherche locale – Principe

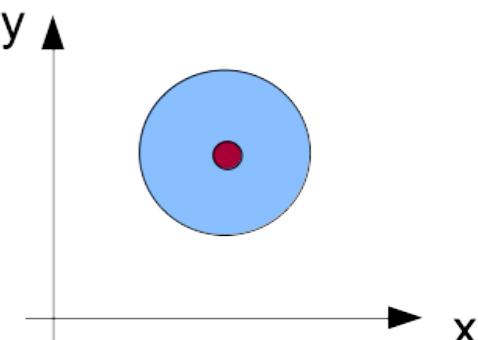
- **Itérativement, parcourir l'espace de recherche de proche en proche à la recherche d'une 'bonne' solution**

- Une solution (candidate) initiale de départ
- 'Améliorer' cette solution en regardant dans son **voisinage proche**



Qu'est ce qu'un voisinage ?

- **Voisinage = Un ensemble de solutions qui diffèrent peu de s**
 - Assez intuitif pour des variables réelles
 - Par exemple, le voisinage d'un point dans le plan est l'ensemble des points dans une boule de rayon (un petit epsilon) ϵ
- Dans un espace (discret) de recherche S , **le voisinage d'une solution est défini par une fonction $N : X \rightarrow 2^X$**
 - N est appelé un opérateur/relation de voisinage
 - Un opérateur/relation de voisinage peut être vue comme définissant une distance dans l'espace de recherche
- **Un opérateur de voisinage est par rapport à une représentation du problème/solution (genotype)**



Exemple #1 : Binaire

- **Problème du sac à dos**

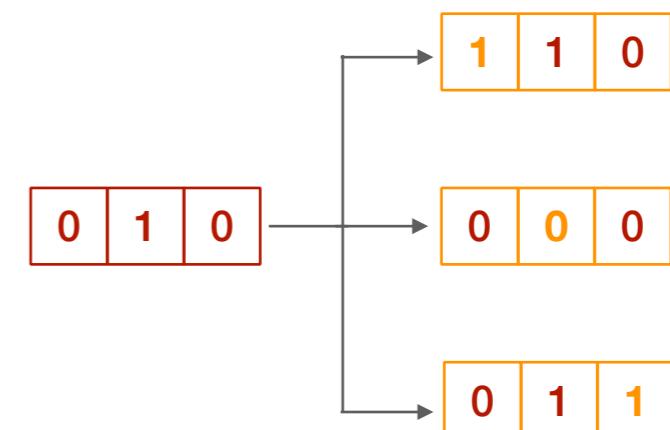
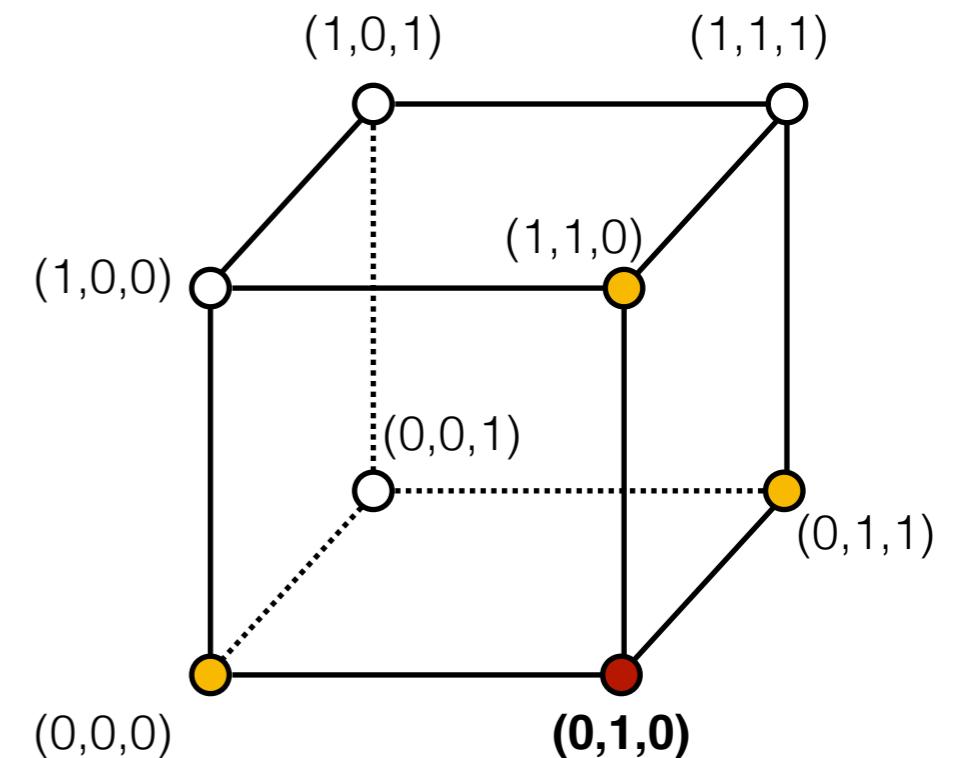
- Donnée : n objets $\{1, \dots, i, \dots, n\}$, chacun de poids p_i et de valeur v_i , et un sac de capacité C
- Problème : remplir le sac en maximisant sa valeur totale (on ne peut pas fractionner les objets)

- **Représentation d'une solution potentielle : une chaîne binaire $x_1, x_2, \dots, x_i, \dots, x_n$**

- $x_i = 0$: l'objet n'est pas dans le sac; $x_i = 1$: l'objet est dans le sac

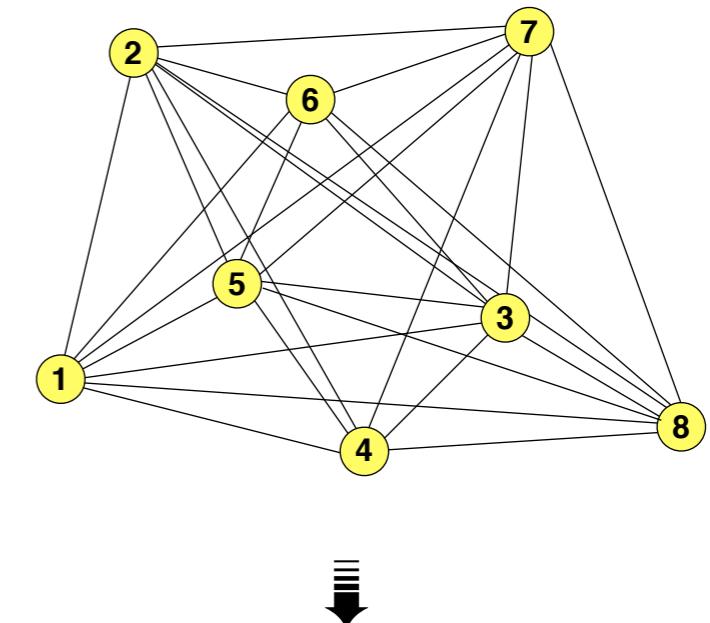
- **Voisinage basée sur la distance de Hamming**

- Hypercube des voisins à une distance de Hamming 1
- Pour un problème de taille n , et une solution donnée, possède n voisins (qui diffèrent de 1 bit)

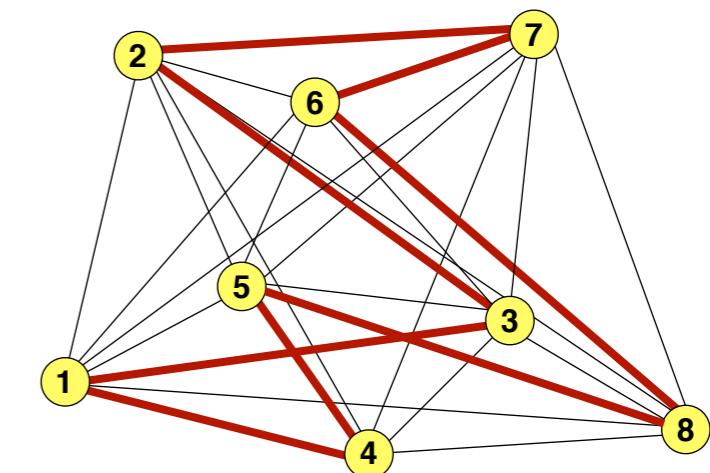


Exemple #2 : Permutation

- **TSP (Voyageur de commerce)**
 - Donnée: Un graphe $G=(V,A,\omega)$ à n sommets
 - Problème: Trouver une tournée (qui passe par chaque sommet une et une seule fois) de poids minimale
- Représentation d'une solution : Une permutation des sommets

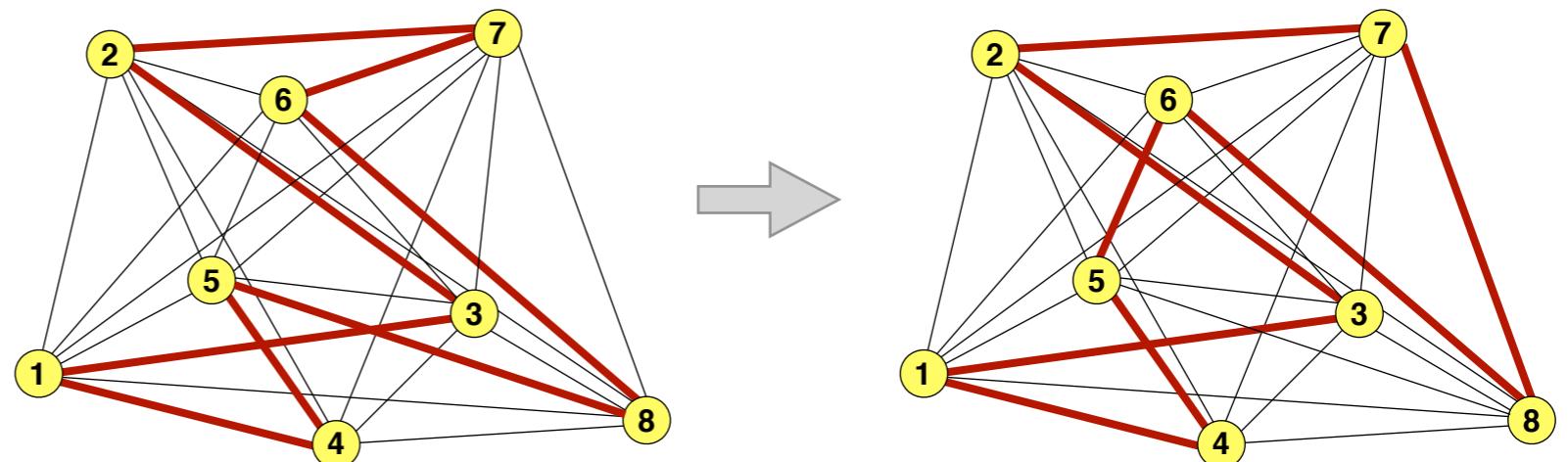


1	3	2	7	6	8	5	4
---	---	---	---	---	---	---	---

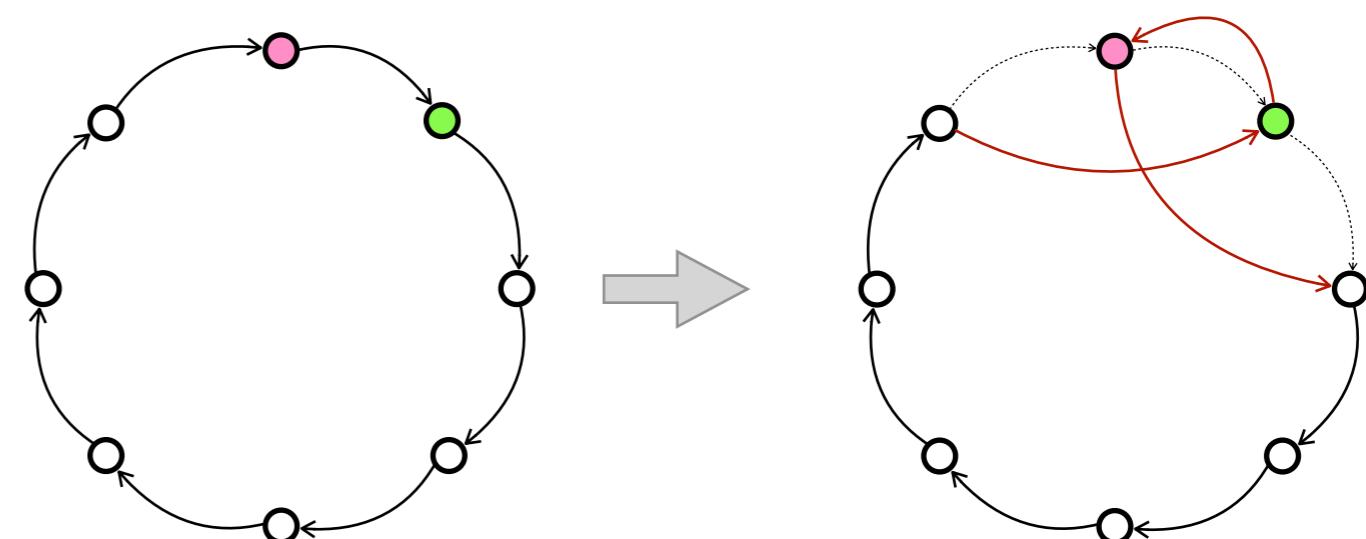
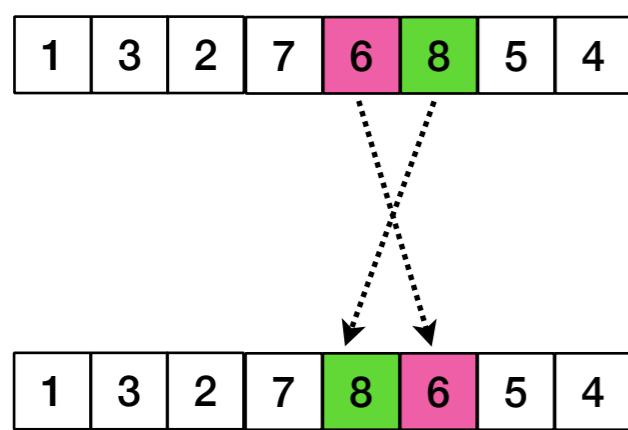


Exemple #2 : Permutation

- **Voisinages de permutation :**
 - **Inversion** : $O(n)$ voisins



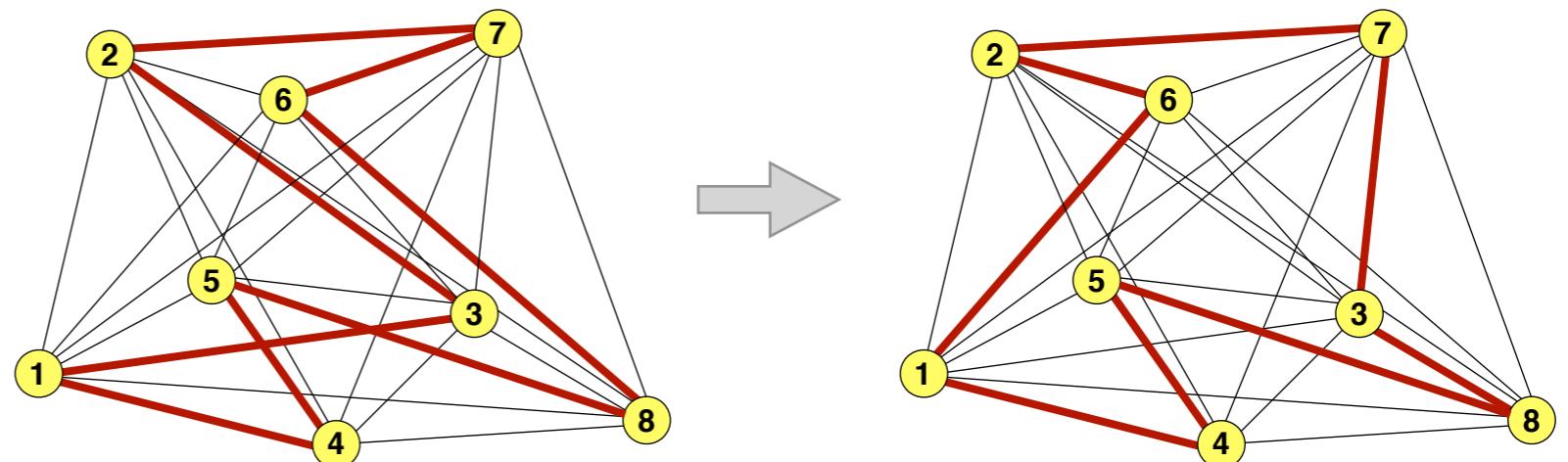
Inversion



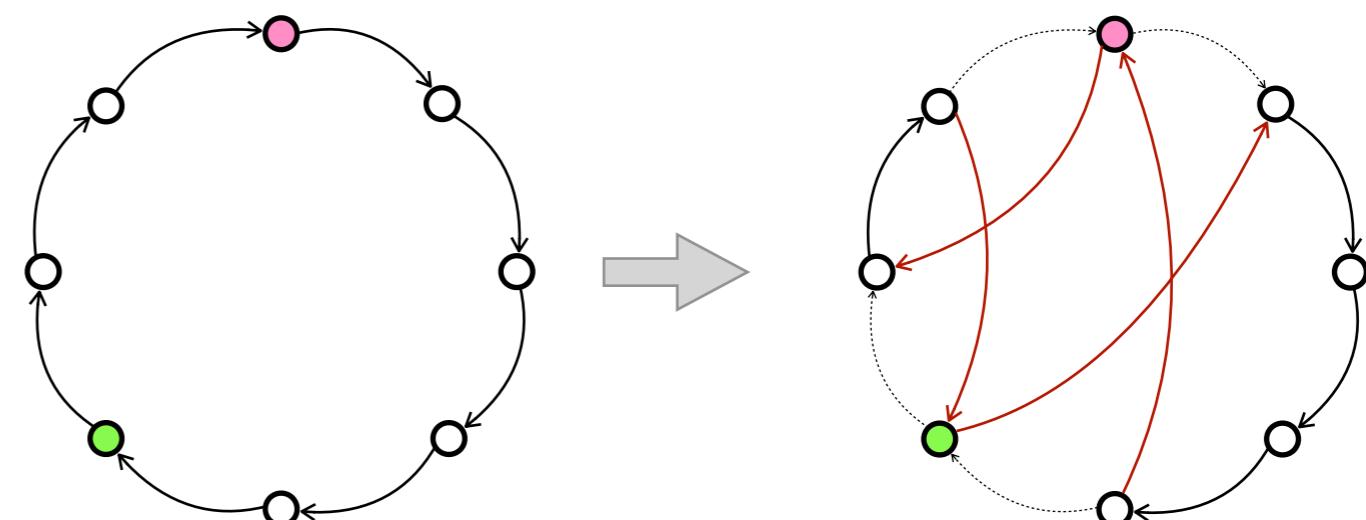
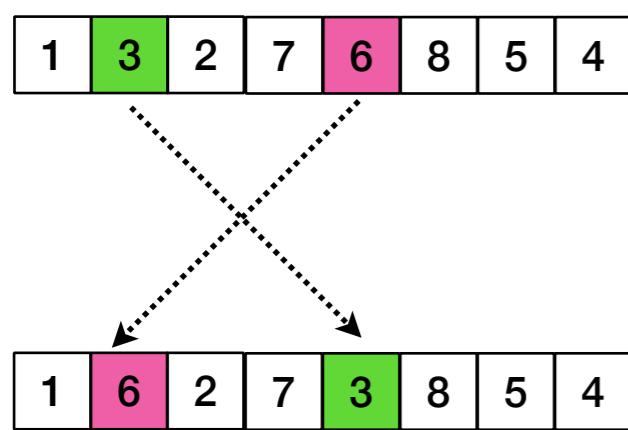
Exemple #2 : Permutation

- **Voisinages de permutation :**

- **Inversion** : $O(n)$ voisins
- **Swap** : $O(n^2)$ voisins



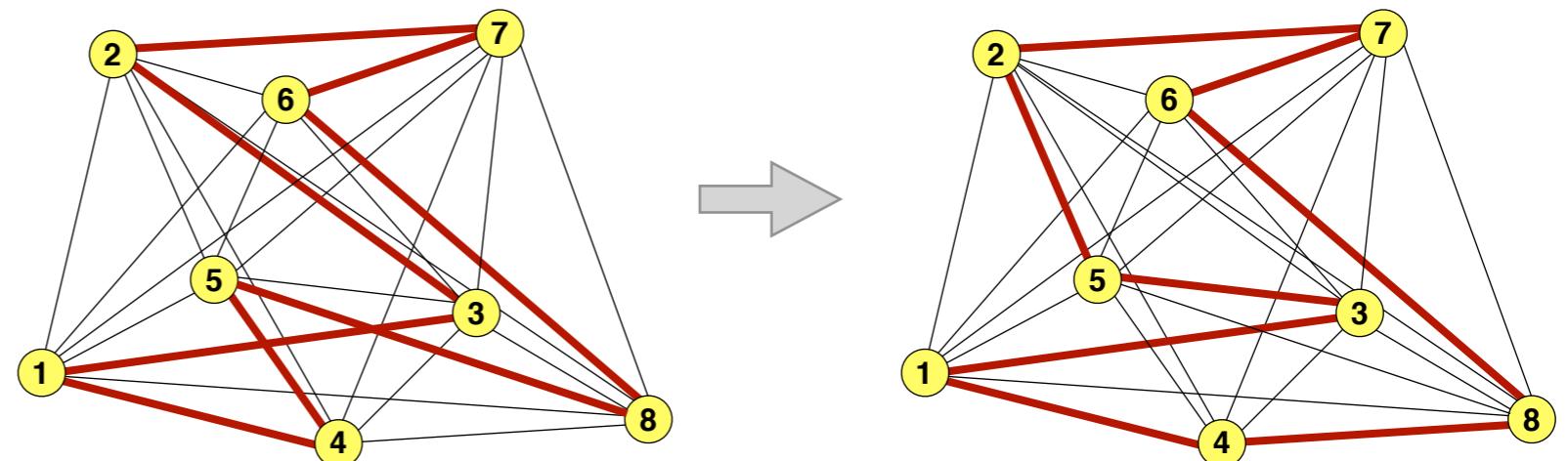
Swap



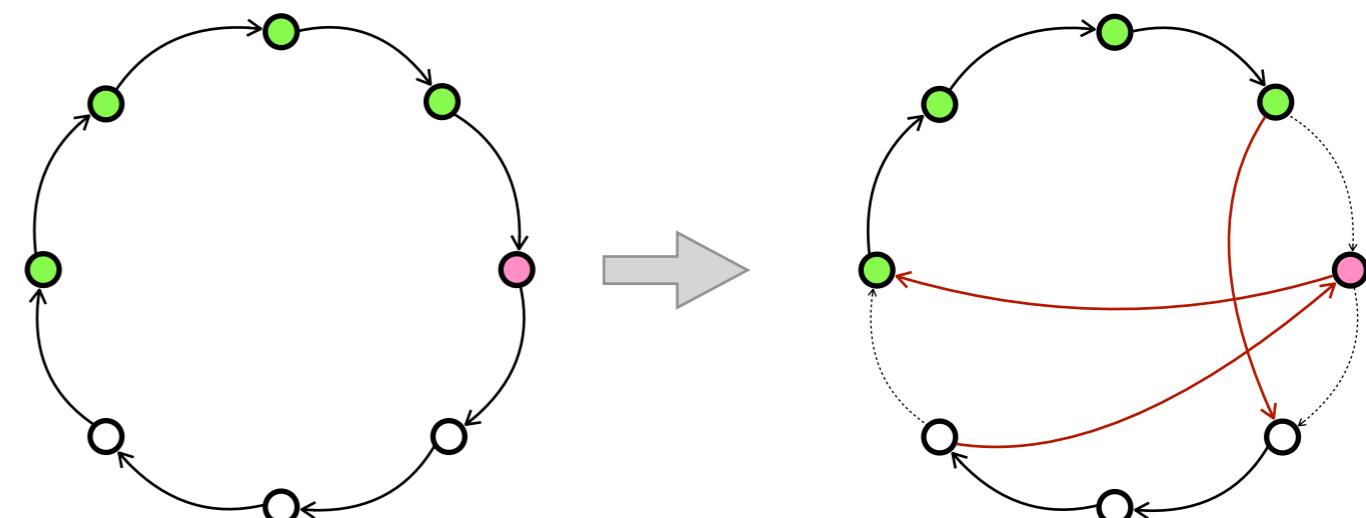
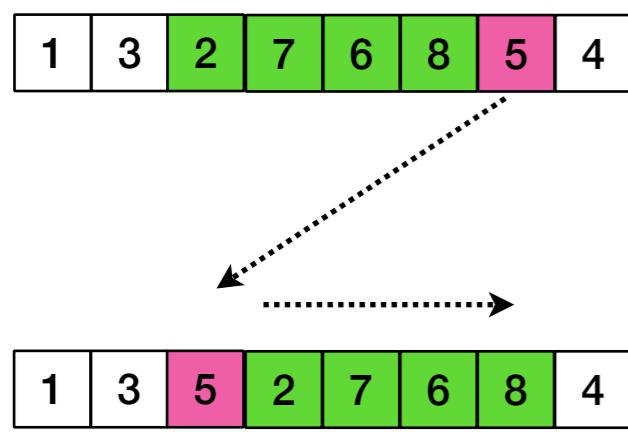
Exemple #2 : Permutation

- **Voisinages de permutation :**

- **Inversion** : $O(n)$ voisins
- **Swap** : $O(n^2)$ voisins
- **Insertion** : $O(n^2)$ voisins



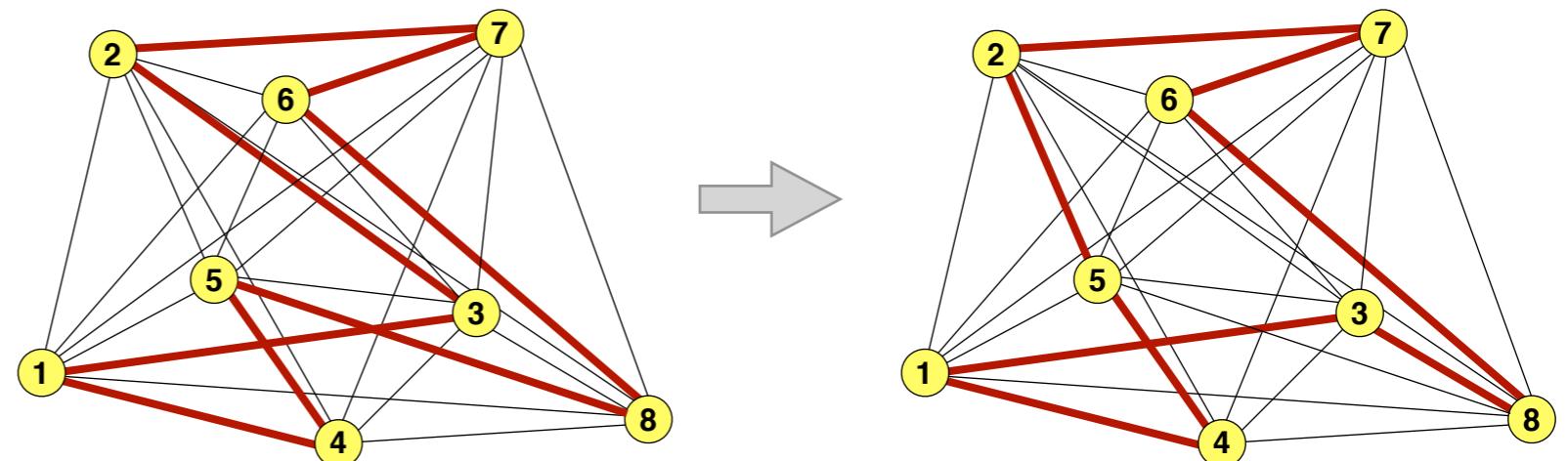
Insertion



Exemple #2 : Permutation

- **Voisinages de permutation :**

- **Inversion** : $O(n)$ voisins
- **Swap** : $O(n^2)$ voisins
- **Insertion** : $O(n^2)$ voisins
- **2-opt** (et ses variantes) : $O(n^2)$ voisins

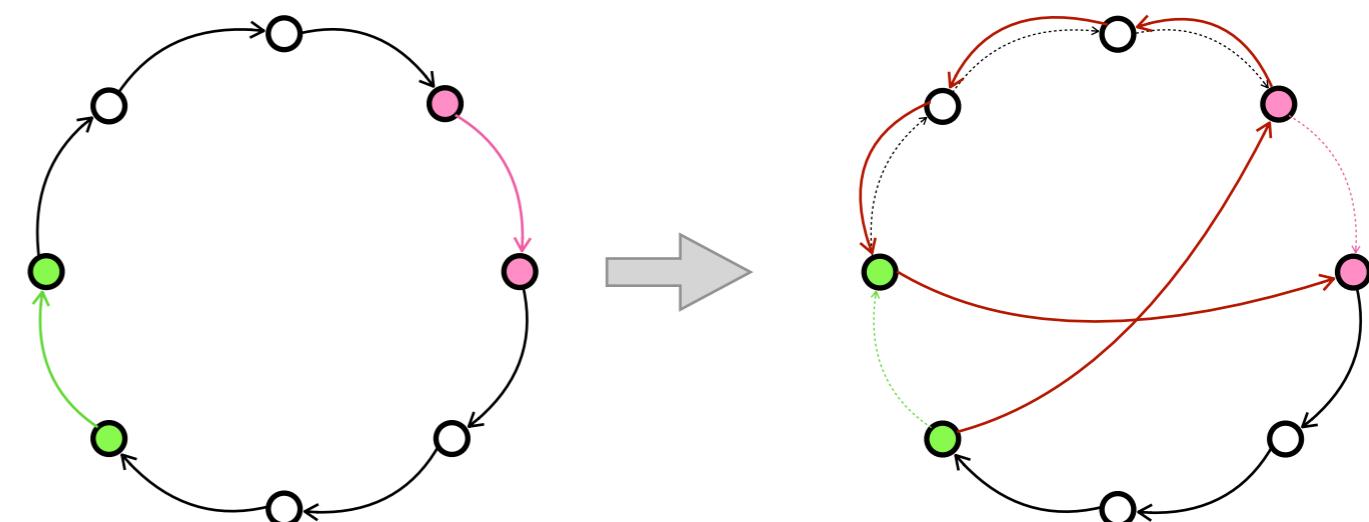


1	3	2	7	6	8	5	4
---	---	---	---	---	---	---	---



1	3	8	6	7	2	5	4
---	---	---	---	---	---	---	---

2-opt



À tout problème un voisinage !

- **Problème ➔ Représentation ➔ Voisinage**
 - Knapsack ➔ binaire ➔ bit-flip, ...
 - SAT ➔ binaire ➔ bit-flip, ...
 - TSP ➔ permutation ➔ Swap, ...
 - Bin Packing ➔ ??? ➔ ???
- **Attention à l'interprétation !**
 - Deux problèmes différents ayant la même representation peuvent en théorie utiliser le même voisinage (problèmes boîte-noires)
 - Un voisinage adapté à un problème ne l'est pas forcément pour un autre problème
- **Un voisinage permet d'avoir une vision locale autour d'une solution**

Recherche locale – schéma de base

s \leftarrow solution initiale ;

Repeat :

N(s) \leftarrow opérateur de voisinage ;

s' \leftarrow **choisir** une solution dans **N(s)** ;

s \leftarrow **s'** ;

Until **Condition d'arrêt**

1. Comment choisir/remplacer ?

2. Comment générer une solution initiale ?

3. Quelle condition d'arrêt ?

Recherche locale – aléatoire

$s \leftarrow$ solution initiale ;

Repeat :

$N(s) \leftarrow$ opérateur de voisinage ;

$s' \leftarrow$ choisir aléatoirement une solution dans $N(s)$;

$s \leftarrow s'$;

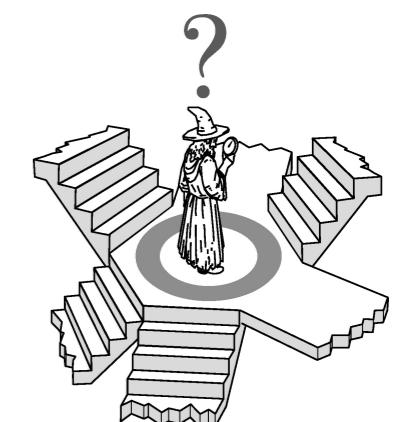
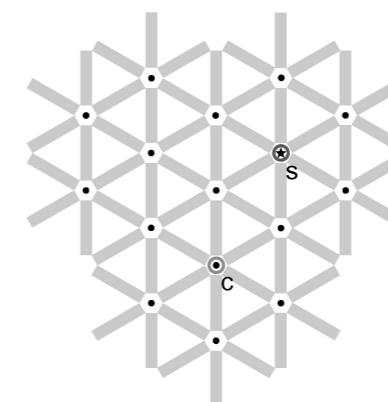
Until Condition d'arrêt



- **Vision globale Vs Vision Locale : Quelle boussole utiliser ?**

- **Paysage de recherche** (structurer l'espace de recherche)

- Sommets : solutions candidates (positions)
- Arêtes : voisins d'une solution selon l'opérateur N



- s^* : solution (optimale) , s : solution candidate courante, s' choisit de façon locale à partir de la position courante s
- Processus Markovien, convergence, ...

Recherche locale – fitness/ évaluation

$s \leftarrow$ solution initiale ;

Repeat :

$N(s) \leftarrow$ opérateur de voisinage ;

$s' \leftarrow$ choisir une solution dans $N(s)$ telle que $f(s') < f(s)$;

$s \leftarrow s'$;

Until Condition d'arrêt

- **Fonction de fitness (évaluation) $f : S \rightarrow \mathbb{IR}$**

- Définit un score/qualité pour chaque solution candidate
- Souvent (mais pas toujours) la même que la fonction objective qui est inhérente au problème !
 - TSP : longueur d'une tournée
 - Knapsack : valeur du sac + pénalité en fonction de combien la capacité du sac est dépassée

Recherche locale – Move

```
s ← solution initiale ;
```

```
Repeat :
```

```
    N(s) ← opérateur de voisinage ;
```

```
    s' ← choisir une solution dans N(s) telle que f(s') < f(s) ;
```

```
    s ← s' ;
```

```
Until Condition d'arrêt
```

- **(Exemples de) Règle pivot / Mouvement :**

- *Best Improvement* : Choisir le meilleur voisin
- *Worst Improvement* : Choisir le moins bon améliorant !
- *Random Improvement* : Choisir un améliorant au hasard
- *First improvement* : Choisir le premier voisin s' suivant un ordre fixe
- **Attention à l'impact sur la complexité ET la qualité !**

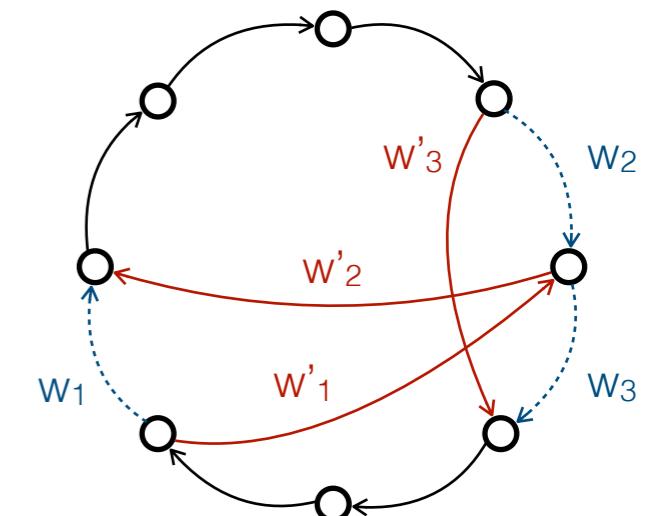
Recherche locale – évaluation incrémentale

- **Le coût de l'évaluation d'une solution peut être important**

- Exemple pour TSP: Un seul move de type 'Best Improvement' coûterait $O(n^3)$
 - Calculer la longueur d'une tournée est en $O(n)$
 - Le nombre de voisins avec 2-opt est en $O(n^2)$

- **Évaluation incrémentale (peut faire la différence en pratique)**

- Un opérateur de voisinage change en général quelques composantes seulement
 - Calculer la différence de $f(s')$ par rapport à $f(s)$ et en déduire $f(s')$
 - Calculer la différence d'une solution voisine s' par rapport à s
 - e.g., décomposer la fonction d'évaluation (exemple des problèmes k-bornées)
 - Exemple pour TSP :
 - $w(p') = w(p) - \text{arêtes dans } p \text{ mais non dans } p' + \text{arêtes dans } p' \text{ mais non dans } p$
 - Complexité $O(n^2)$ au lieu de $O(n^3)$
- **Exploration de très grand voisinage de façon efficace**



$$f(s') = f(s) - w_1 - w_2 - w_3 + w'_1 + w'_2 + w'_3$$

Recherche locale – Init

s \leftarrow solution initiale ;

Repeat :

N(s) \leftarrow opérateur de voisinage ;

s' \leftarrow choisir une solution dans N(s) telle que f(s') < f(s) ;

s \leftarrow s' ; // remplacer s

Until Condition d'arrêt

- Essentiellement deux stratégies
 - Random
 - Heuristique (e.g., constructive)
 - Effectif en pratique, MAIS:
 - Risque de stagnation !
 - La génération d'une solution initiale de bonne qualité peut être difficile (problèmes fortement contraints)

Recherche locale – Hill Climbing, Descent, Iterative improvement, ...

```
s ← solution initiale ;
```

```
Repeat :
```

```
    N(s) ← opérateur de voisinage ;
```

```
    s' ← choisir une solution dans N(s) telle que f(s') < f(s) ;
```

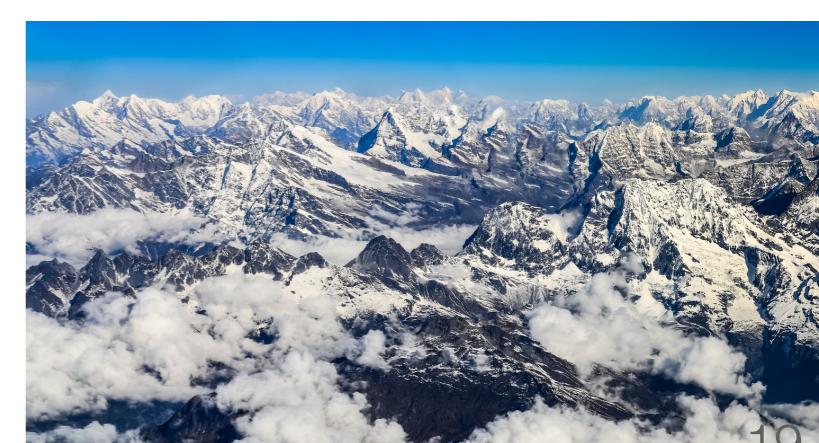
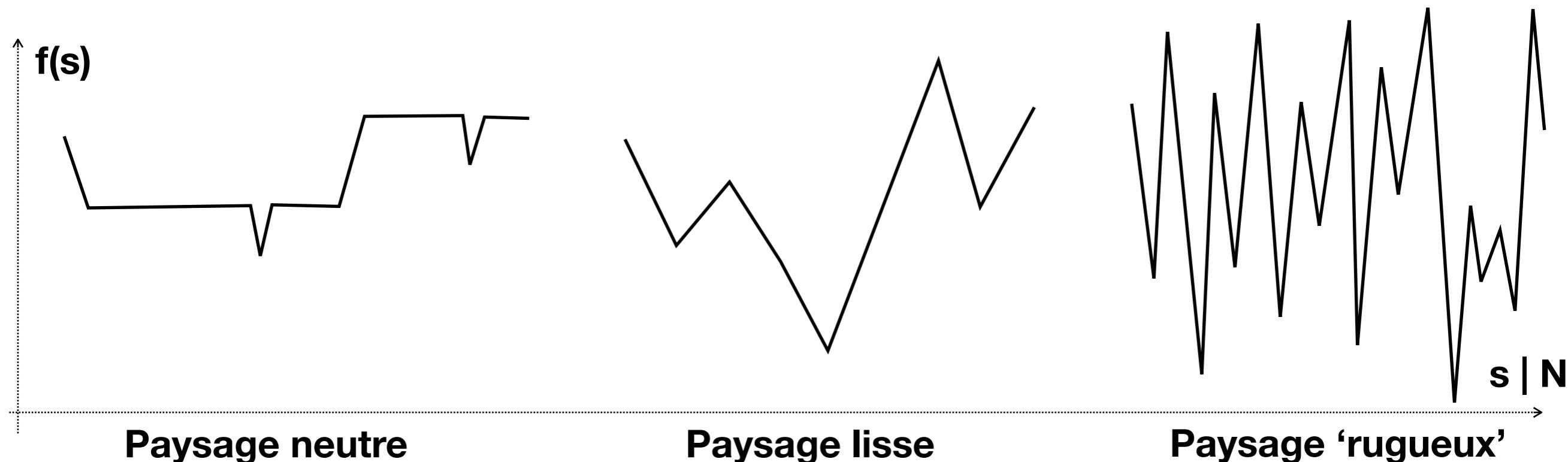
```
    s ← s' ; // remplacer s
```

Until ***Pas d'amélioration possible (Optimum Local)***

- **Optimum local (strict)**
 - Une solution s telle que $f(s) < f(s')$ pour tout s' dans $N(s)$
 - Trajectoire de recherche :
 - $s_0 \text{ (init)} \rightarrow (N, f, move) s_1 \rightarrow (N, f, move) s_2 \rightarrow \dots \rightarrow (N, f, move) s^*$: optimum local

Paysage de recherche – une métaphore ...

- Un voisinage permet de ‘définir’ un **paysage de recherche**
 - Un nombre (typiquement exponentiel) d’optima locaux parmi lesquels on espère trouver un global



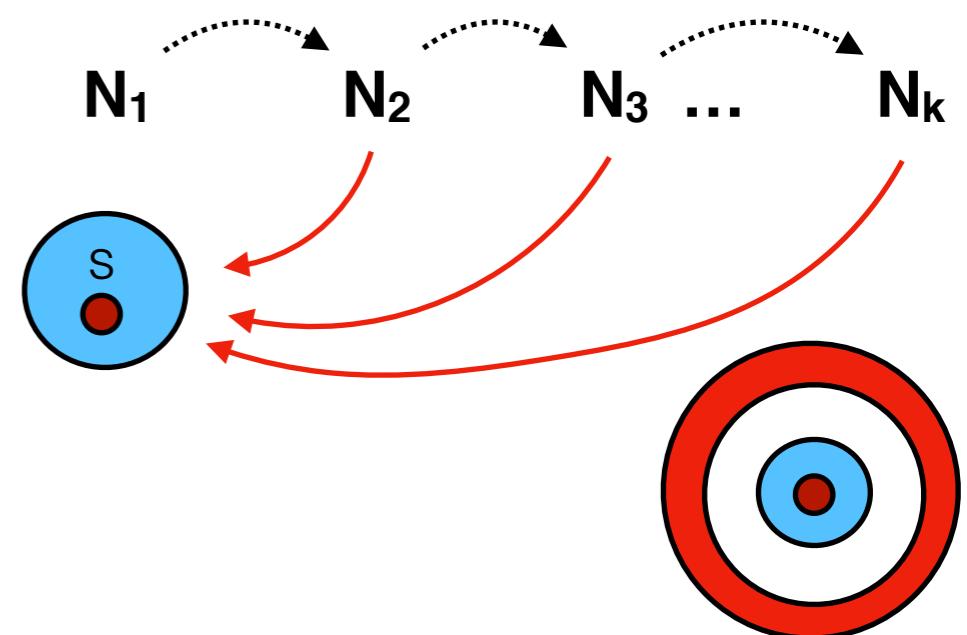
Échapper aux optima locaux

- Comment trouver des ‘bon’ optima locaux tout en évitant d’y rester coincer ?
 - Après tout, peu importe de terminer dans optimum local, l’essentiel est de passer par l’optimum global !
- **Équilibre entre Intensification et Diversification**
 - **Intensification (exploitation)** : améliorer la qualité d’une solution (HillClimbing)
 - **Diversification (exploration)** : éviter de stagner dans une région (Random search)
- **Un optimum local dépend de la donnée de : Init, f, N, et Move/Pivot**
- **Idée générale pour échapper aux optima locaux: Jouer sur ces ‘composants’ de bases**
 - Utiliser un grand voisinage (Large neighborhood search) : pas toujours effectif
 - Recommencer avec une nouvelle solution initiale (Restart) : pas suffisant en pratique car nombre exponentiel d’optima locaux
 - Autres ...

Variable Neighborhood Descent (VND)

- **Principe : Un optimum local pour un voisinage N n'est pas forcément un optimum local pour un voisinage N'**
 - Pour échapper à un optimum local respectivement à un voisinage, changer/alterner de voisinage
- **VND avec k voisinages $N_1, N_2, \dots, N_i, \dots, N_k$**
 - Scanner les voisinages dans un ordre donnée
 - Si on est dans un optimum local passer au voisinage suivant, sinon revenir au premier voisinage
 - À la terminaison, la solution trouvée est un optimum local respectivement aux k voisinages
- Plusieurs variantes :
 - Choix de l'ordre (e.g., du plus petit au plus grand, aléatoire, nested, etc)
 - Choix de la descente (e.g., un move, une descente complète, etc)
 - Choix du backtrack (e.g., premier voisinage, avant dernier, etc)

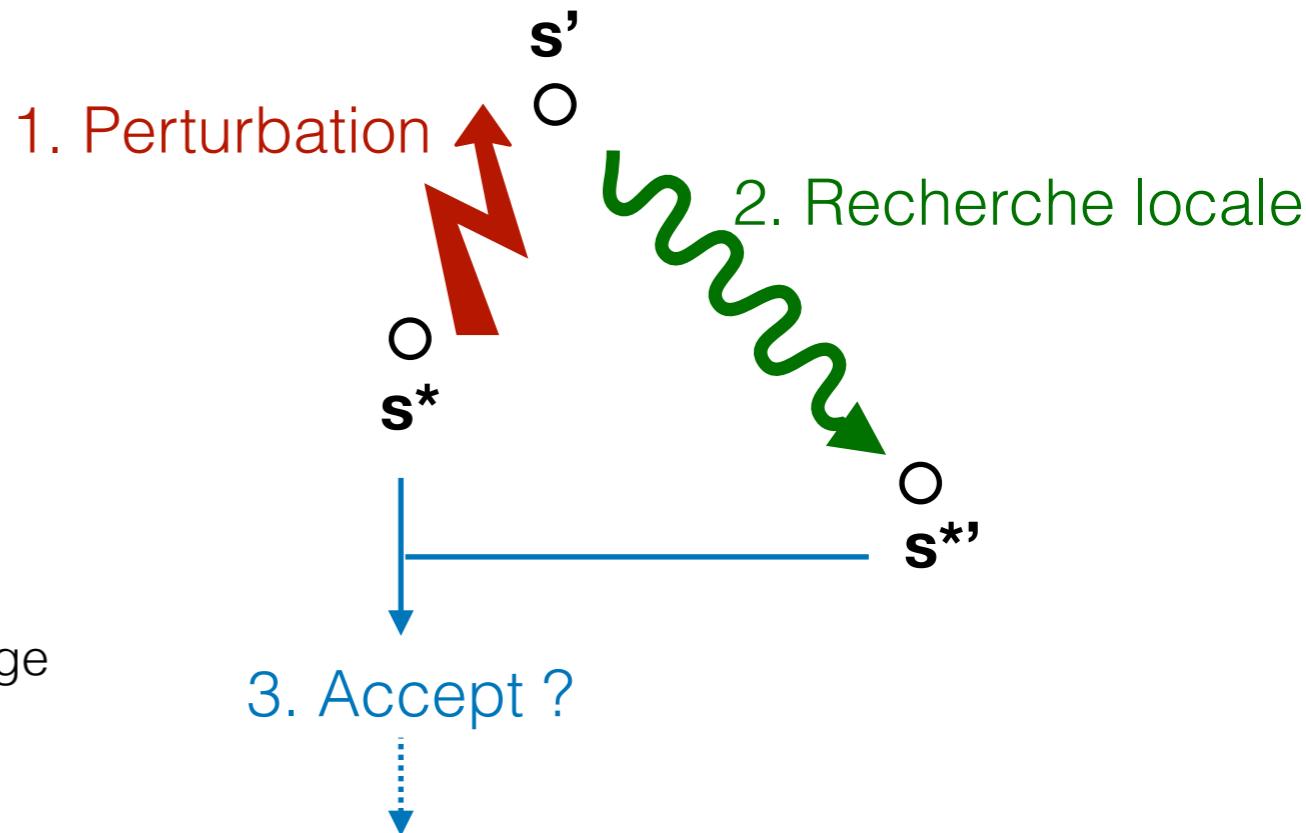
```
s ← une solution initiale ;  
i ← 1 ;  
Repeat  
    s' ← meilleur voisin de s selon  $N_i$ ;  
    If  $f(s') < f(s)$  then  
         $s \leftarrow s'$  ;  
        i ← 1 ;  
    Else  
        i ← i+1 ;  
Until  $i > k$ 
```



Iterated Local Search (ILS)

- **Principe : Un bon optimum local ne doit pas être très loin d'un autre probablement encore meilleur**
- **Recherche Locale itérée (ILS) :**
 - Recherche locale itérative, e.g., HillClimbing, VND
 - atteindre rapidement un optimum local (Intensification)
- **Perturbation**
 - échapper aux optima locaux (Diversification)
- **Critère d'acceptation**
 - équilibre entre diversification et intensification
- ILS effectue une 'marche' dans l'espace des optima locaux
 - La recherche locale définit en quelque sorte un voisinage
 - On parle de bassins d'attraction...

```
s ← une solution initiale ;  
s* ← LocalSearch(s) ;  
Repeat  
    s' ← Perturbation(s*,historique);  
    s*' ← LocalSearch(s')  
    s* ← Accept(s*,s',historique)  
Until Condition d'arrêt
```



Iterated Local Search (ILS)

- La perturbation est complémentaire à la recherche locale

- Quelques exemple simples :

- Effectuer k 'move' aléatoire par rapport à un voisinage donnée
- Plus k est grand plus le degré de la perturbation est élevée

- Degré de la perturbation

- Trop élevé : ~ équivalent à un restart aléatoire
- Trop faible : la recherche locale peut défaire la perturbation et donc échouer à échapper d'un optimum local vers un autre

- Le degré de la perturbation peut varier de façon adaptative

- Une perturbation aléatoire est simple mais pas toujours effective

$s \leftarrow$ une solution initiale ;

$s^* \leftarrow \text{LocalSearch}(s)$;

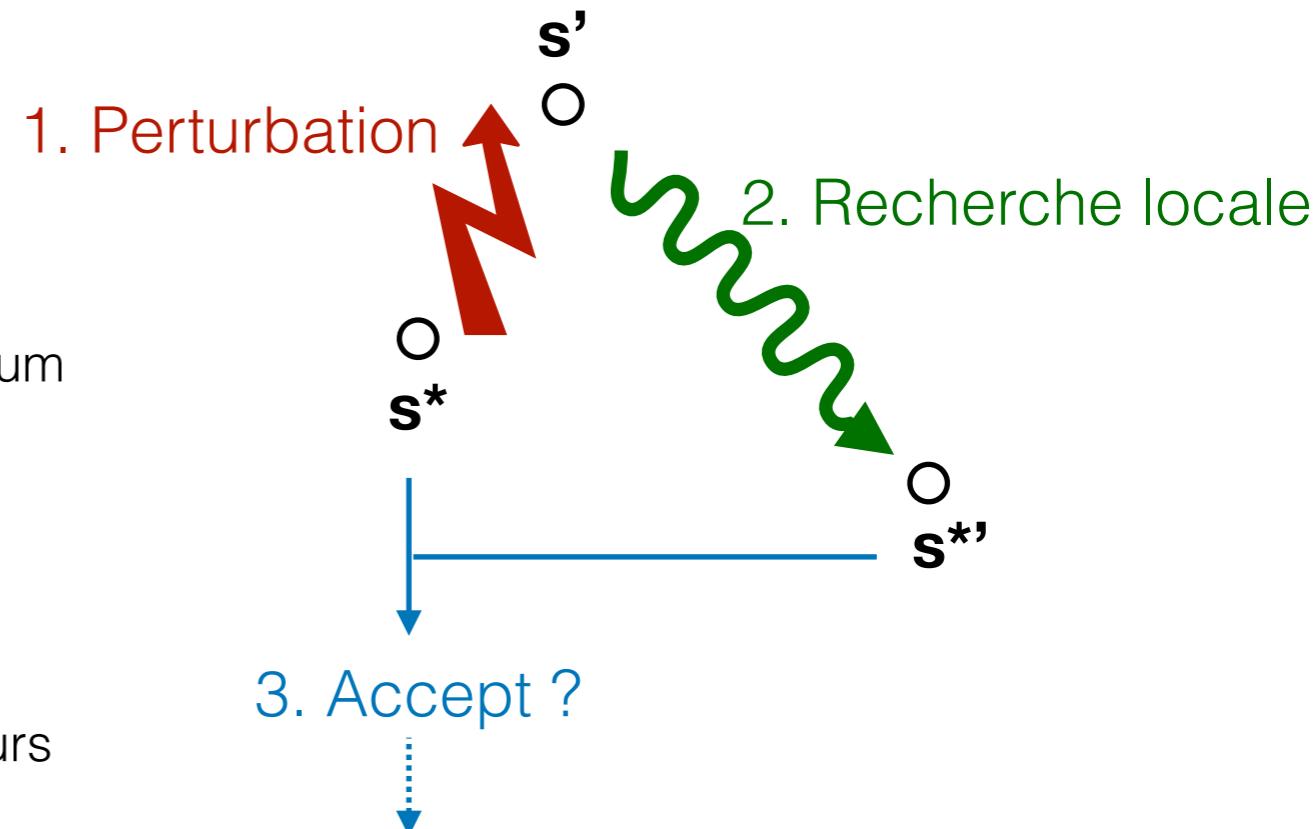
Repeat

$s' \leftarrow \text{Perturbation}(s^*, \text{historique})$;

$s^{*'} \leftarrow \text{LocalSearch}(s')$

$s^* \leftarrow \text{Accept}(s^*, s^{*'}, \text{historique})$

Until Condition d'arrêt



Iterated Local Search (ILS)

- **Critère d'acceptation**

- Intensification vs. Diversification dans l'espace des optima locaux

- **Quelques exemple simples :**

- Toujours accepter s'^*
- N'accepter que les s'^* qui améliorent s^*
- Choix intermédiaires:
 - (Grand déluge) accepter si $f(s'^*) < (1+\varepsilon).f(s^*)$
 - Probabiliste (voir cours prochain)
- Le choix dépend de la façon avec laquelle les optima locaux sont connectés entre eux

```
s ← une solution initiale ;
```

```
s* ← LocalSearch(s) ;
```

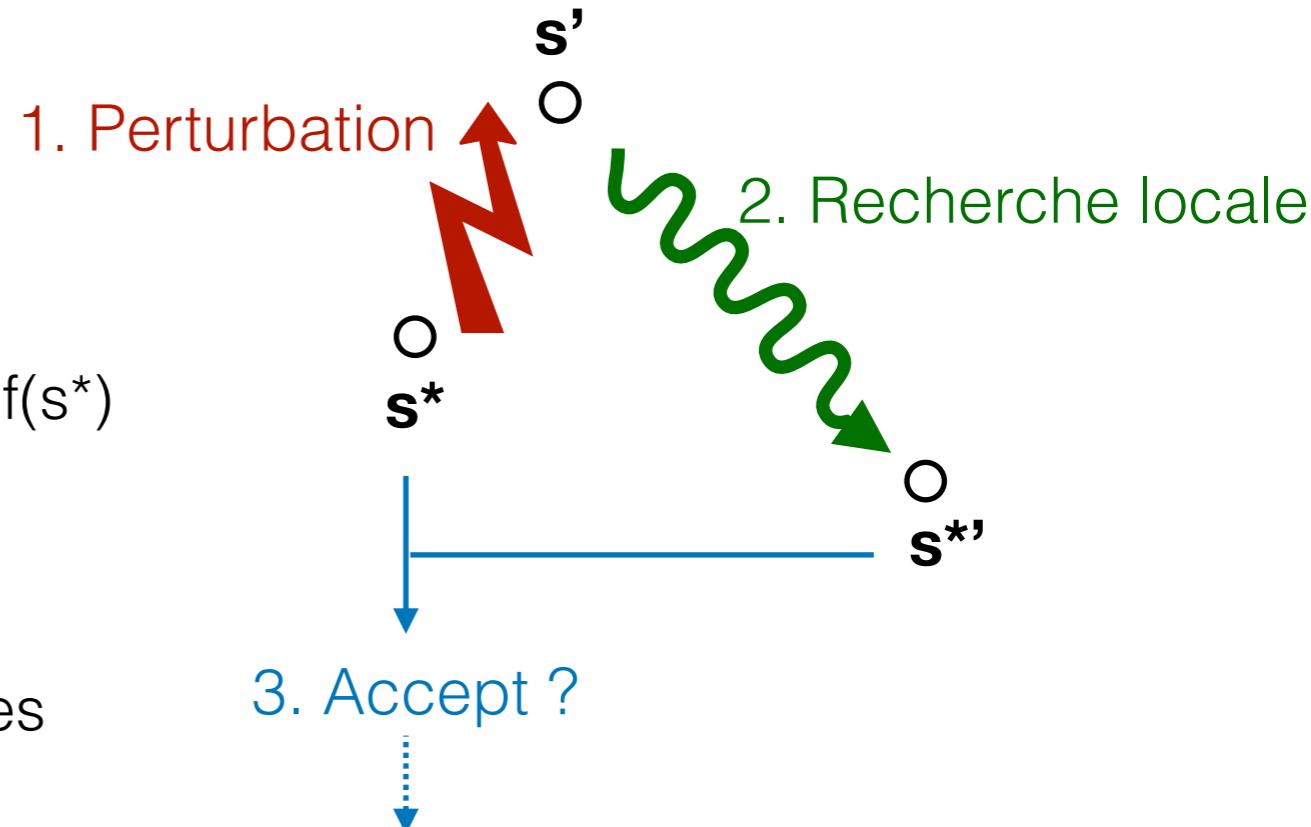
Repeat

```
s' ← Perturbation(s*,historique);
```

```
s*' ← LocalSearch(s')
```

```
s* ← Accept(s*,s*',historique)
```

Until Condition d'arrêt



Iterated Local Search (ILS)

- **Critère d'acceptation**

- Intensification vs. Diversification dans l'espace des optima locaux

- **Quelques exemple simples :**

- Toujours accepter s'^*
- N'accepter que les s'^* qui améliorent s^*
- Choix intermédiaires:
 - (Grand déluge) accepter si $f(s'^*) < (1+\varepsilon).f(s^*)$
 - Probabiliste (voir cours prochain)
 - Le choix dépend de la façon avec laquelle les optima locaux sont connectés entre eux

```
s ← une solution initiale ;
```

```
s* ← LocalSearch(s) ;
```

Repeat

```
    s' ← Perturbation(s*,historique);
```

```
    s*' ← LocalSearch(s')
```

```
    s* ← Accept(s*,s*',historique)
```

Until Condition d'arrêt

- **Historique (des optima locaux, perturbations, etc) !**

- **Condition d'arrêt ?**

- Temps CPU maximum

- Borne sur #itérations/#évaluations

- Borne sur le temps/#itérations/... après la dernière amélioration

Recherche locale (les bases) — Bilan

- **Problème** → **Représentation** → **Voisinage**
- **Trouver un optimum local**
 - **Hill Climbing** : Init, f, N, move/pivot
- **Échapper aux optima locaux**
 - **VND** : Changer/alterner les voisinages
 - **ILS** : sauter d'un optimum local à un autre en combinant perturbation (diversification) et descente (intensification)