Méthodes d'optimisation méta-heuristiques

BUT Info 2e année

Florent Foucaud
Dipayan Chakraborty, Malika More, Adrien Wohrer

IUT CLERMONT AUVERGNE
Aurillac - Clermont-Ferrand - Le Puy-en-Velay
Montluçon - Moulins - Vichy

2023-2024

Heuristiques et méta-heuristiques

Définition (Algorithme heuristique)

Algorithme qui calcule une solution d'un problème sans garantie de qualité.

Exemple : algorithme glouton pour la couverture par sommets (voir cours précédent)

Heuristiques et méta-heuristiques

Définition (Algorithme heuristique)

Algorithme qui calcule une solution d'un problème sans garantie de qualité.

Exemple : algorithme glouton pour la couverture par sommets (voir cours précédent)

Définition (Méta-heuristique)

Algorithme heuristique générique, applicable à un grand nombre de problèmes.

Heuristiques et méta-heuristiques

Définition (Algorithme heuristique)

Algorithme qui calcule une solution d'un problème sans garantie de qualité.

Exemple : algorithme glouton pour la couverture par sommets (voir cours précédent)

Définition (Méta-heuristique)

Algorithme heuristique générique, applicable à un grand nombre de problèmes.

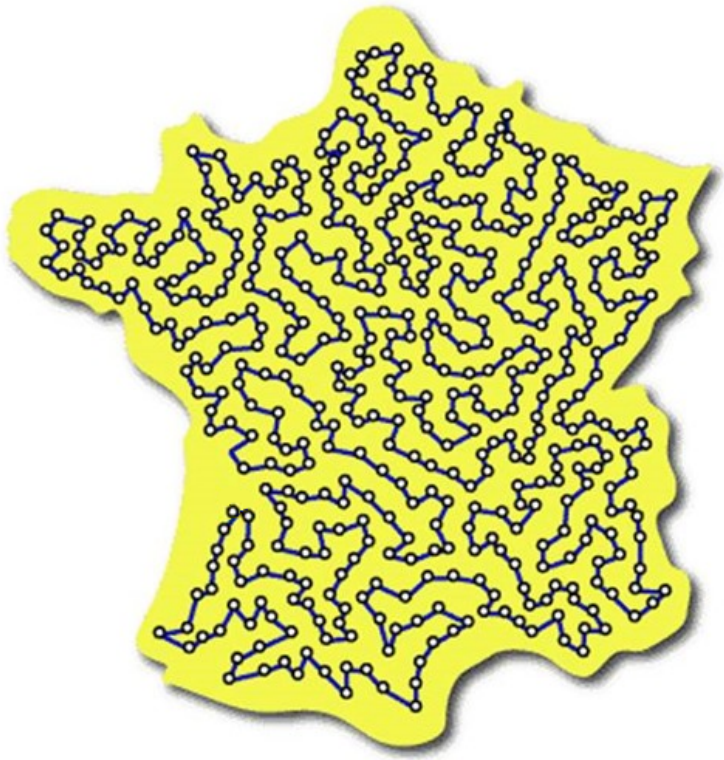
Deux grandes familles de méta-heuristiques sont utilisées en Optimisation :

- Méta-heuristiques de recherche locale : descente de gradient, recuit simulé, méthode tabou...
- Méta-heuristiques à population : algorithmes génétiques, colonies de fourmis, essaims de particules...

Exemple de problème : le voyageur de commerce

Entrée : un ensemble de *n* villes et les distances entre toute paire de villes
Sortie : une tournée (permutation des *n* villes) la plus courte possible

(en anglais : Travelling Salesman Problem, TSP)



Un problème très célèbre depuis le début du 20e siècle, et très difficile à résoudre !
→ NP-complet (probablement pas d'algorithme polynomial)

Voyageur de commerce : combien de solutions ?

Les n villes sont indexées par $0, \ldots, n-1$

Une tournée sur n villes est une permutation de $0, \ldots, n-1$

Voyageur de commerce : combien de solutions ?

Les n villes sont indexées par $0, \ldots, n-1$

Une tournée sur n villes est une permutation de $0, \ldots, n-1$

Pour $n = 3$: $[0, 1, 2]$, $[0, 2, 1]$, $[1, 0, 2]$, $[1, 2, 0]$, $[2, 0, 1]$, $[2, 1, 0]$ (6 tournées)

Voyageur de commerce : combien de solutions ?

Les n villes sont indexées par $0, \ldots, n-1$

Une tournée sur n villes est une permutation de $0, \ldots, n-1$

Pour $n = 3$: $[0, 1, 2]$, $[0, 2, 1]$, $[1, 0, 2]$, $[1, 2, 0]$, $[2, 0, 1]$, $[2, 1, 0]$ (6 tournées)

Il y en a $n! = 1 \times 2 \times \ldots \times (n-1) \times n \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \approx n^n$ différentes

Voyageur de commerce : combien de solutions ?

Les n villes sont indexées par $0, \ldots, n-1$

Une tournée sur n villes est une permutation de $0, \ldots, n-1$

Pour $n = 3$: $[0,1,2]$, $[0,2,1]$, $[1,0,2]$, $[1,2,0]$, $[2,0,1]$, $[2,1,0]$ (6 tournées)

Il y en a $n! = 1 \times 2 \times \ldots \times (n-1) \times n \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \approx n^n$ différentes

$3! = 6$, $4! = 24$, $5! = 120$, $6! = 720$, $7! = 5\,040$, $8! = 40\,320$, $9! = 362\,880$,
$10! = 3\,628\,800$, $11! = 39\,916\,800$, $12! = 479\,001\,600$, $13! = 6\,227\,020\,800$

Voyageur de commerce : combien de solutions ?

Les n villes sont indexées par $0, \ldots, n-1$

Une tournée sur n villes est une permutation de $0, \ldots, n-1$

Pour $n = 3$: $[0,1,2]$, $[0,2,1]$, $[1,0,2]$, $[1,2,0]$, $[2,0,1]$, $[2,1,0]$ (6 tournées)

Il y en a $n! = 1 \times 2 \times \ldots \times (n-1) \times n \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \approx n^n$ différentes

$3! = 6$, $4! = 24$, $5! = 120$, $6! = 720$, $7! = 5\,040$, $8! = 40\,320$, $9! = 362\,880$,
$10! = 3\,628\,800$, $11! = 39\,916\,800$, $12! = 479\,001\,600$, $13! = 6\,227\,020\,800$

Algorithme naïf (en temps exponentiel) :

- Mesurer la longueur de chaque tournée possible
- Sélectionner la plus courte

PLNE pour le voyageur de commerce

	Clermont-Ferrand	Bordeaux	Toulouse	Lyon
Clermont-Ferrand	-	376	377	167
Bordeaux	376	-	244	556
Toulouse	377	244	-	538
Lyon	167	556	538	-

PLNE pour le voyageur de commerce

	Clermont-Ferrand	Bordeaux	Toulouse	Lyon
Clermont-Ferrand	-	376	377	167
Bordeaux	376	-	244	556
Toulouse	377	244	-	538
Lyon	167	556	538	-

On prend une variable pour chaque trajet direct possible (il y en a 6) :

$$x_{cb}, x_{cl}, x_{ct}, x_{bt}, x_{bl}, x_{lt}.$$

Chaque trajet est utilisé ou pas (0 ou 1)

Pour chaque ville, il faut s'asurer qu'on rentre et qu'on sort.

minimiser : $376x_{cb} + 167x_{cl} + 377x_{ct} + 244x_{bt} + 556x_{bl} + 538x_{lt}$

tel que :

$$\begin{aligned}
x_{cb} + x_{cl} + x_{ct} &= 2 && \textit{Clermont}\\
x_{cb} + x_{bt} + x_{bl} &= 2 && \textit{Bordeaux}\\
x_{lt} + x_{cl} + x_{bl} &= 2 && \textit{Lyon}\\
x_{lt} + x_{ct} + x_{bt} &= 2 && \textit{Toulouse}\\
x_{cb}, x_{cl}, x_{ct}, x_{bt}, x_{bl}, x_{lt} &\leq 1\\
x_{cb}, x_{cl}, x_{ct}, x_{bt}, x_{bl}, x_{lt} &\geq 0\\
x_{cb}, x_{cl}, x_{ct}, x_{bt}, x_{bl}, x_{lt} &\in \mathbb{N}
\end{aligned}$$

PLNE pour le voyageur de commerce

En général, on note : x_{ij} $(i \neq j)$ la variable qui correspond au trajet depuis la ville i vers la ville j.

(Ici, on distingue x_{ij} de x_{ji}, mais ce n'est pas obligatoire.)

minimiser : $\sum_{i,j(i\neq j)} d(i,j)x_{ij}$

tel que :

$$\sum_{j:j\neq i} x_{ij} = 1 \quad \forall i \in \{1,\ldots,n\} \text{ (un seul trajet sortant pour la ville } i)$$

$$\sum_{j:j\neq i} x_{ji} = 1 \quad \forall i \in \{1,\ldots,n\} \text{ (un seul trajet entrant pour la ville } i)$$

$$\sum_{i,j\in Q(i\neq j)} x_{ij} \leq |Q|-1 \quad \forall Q \subseteq \{1,\ldots,n\},\ |Q| \geq 2 \text{ (connexité)}$$

$$x_{ij} \leq 1 \quad \forall i,j \in \{1,\ldots,n\}\ (i \neq j)$$
$$x_{ij} \geq 0 \quad \forall i,j \in \{1,\ldots,n\}\ (i \neq j)$$
$$x_{ij} \in \mathbb{N} \quad \forall i,j \in \{1,\ldots,n\}\ (i \neq j)$$

Algorithme glouton pour le voyageur de commerce

Les n villes sont indexées par $0, \ldots, n-1$

Tournée : tableau T contenant la suite des indices des villes visitées
ex : $T = [1, 4, 0, 3, 2]$

1. Choisir une première ville i (par exemple $i = 0$, ou bien, au hasard avec `randrange(0,n)` qui renvoie un entier entre 0 et $n-1$)
2. initialiser la tournée : $T[0] = i$
3. Tant qu'il reste des villes non visitées (pas encore placées dans T) :
   - choisir la ville j non encore visitée qui minimise la distance à la dernière ville visitée
   - ajouter j à T
4. ajouter la ville de départ i à la fin de T pour clôre la tournée
5. renvoyer T

Algorithme glouton pour le voyageur de commerce

Les n villes sont indexées par $0, \ldots, n-1$

Tournée : tableau T contenant la suite des indices des villes visitées
ex : $T = [1, 4, 0, 3, 2]$

1. Choisir une première ville i (par exemple $i = 0$, ou bien, au hasard avec `randrange(0,n)` qui renvoie un entier entre 0 et $n-1$)
2. initialiser la tournée : $T[0] = i$
3. Tant qu'il reste des villes non visitées (pas encore placées dans T) :
   - choisir la ville j non encore visitée qui minimise la distance à la dernière ville visitée
   - ajouter j à T
4. ajouter la ville de départ i à la fin de T pour clôre la tournée
5. renvoyer T

Exemple :

	Clermont-Ferrand	Bordeaux	Toulouse	Lyon
Clermont-Ferrand	-	376	377	167
Bordeaux	376	-	244	556
Toulouse	377	244	-	538
Lyon	167	556	538	-

Algorithme glouton pour le voyageur de commerce

Les n villes sont indexées par $0, \ldots, n-1$

Tournée : tableau T contenant la suite des indices des villes visitées
ex : $T = [1, 4, 0, 3, 2]$

1. Choisir une première ville i (par exemple $i = 0$, ou bien, au hasard avec `randrange(0,n)` qui renvoie un entier entre 0 et $n-1$)
2. initialiser la tournée : $T[0] = i$
3. Tant qu'il reste des villes non visitées (pas encore placées dans T) :
   - choisir la ville j non encore visitée qui minimise la distance à la dernière ville visitée
   - ajouter j à T
4. ajouter la ville de départ i à la fin de T pour clôre la tournée
5. renvoyer T

Exemple :

Première ville : Clermont-Ferrand

	Clermont-Ferrand	Bordeaux	Toulouse	Lyon
Clermont-Ferrand	-	376	377	167
Bordeaux	376	-	244	556
Toulouse	377	244	-	538
Lyon	167	556	538	-

Algorithme glouton pour le voyageur de commerce

Les n villes sont indexées par $0, \ldots, n-1$

Tournée : tableau T contenant la suite des indices des villes visitées
ex : $T = [1, 4, 0, 3, 2]$

1. Choisir une première ville i (par exemple $i = 0$, ou bien, au hasard avec `randrange(0,n)` qui renvoie un entier entre 0 et $n-1$)
2. initialiser la tournée : $T[0] = i$
3. Tant qu'il reste des villes non visitées (pas encore placées dans T) :
   - choisir la ville j non encore visitée qui minimise la distance à la dernière ville visitée
   - ajouter j à T
4. ajouter la ville de départ i à la fin de T pour clôre la tournée
5. renvoyer T

Exemple :

Première ville : Clermont-Ferrand
Deuxième ville : Lyon (167km)

	Clermont-Ferrand	Bordeaux	Toulouse	Lyon
Clermont-Ferrand	-	376	377	167
Bordeaux	376	-	244	556
Toulouse	377	244	-	538
Lyon	167	556	538	-

Algorithme glouton pour le voyageur de commerce

Les n villes sont indexées par $0, \ldots, n-1$

Tournée : tableau T contenant la suite des indices des villes visitées
ex : $T = [1, 4, 0, 3, 2]$

1. Choisir une première ville i (par exemple $i = 0$, ou bien, au hasard avec `randrange(0,n)` qui renvoie un entier entre 0 et $n-1$)
2. initialiser la tournée : $T[0] = i$
3. Tant qu'il reste des villes non visitées (pas encore placées dans T) :
   - choisir la ville j non encore visitée qui minimise la distance à la dernière ville visitée
   - ajouter j à T
4. ajouter la ville de départ i à la fin de T pour clôre la tournée
5. renvoyer T

Exemple :

Première ville : Clermont-Ferrand
Deuxième ville : Lyon (167km)
Troisième ville : Toulouse (538km)

	Clermont-Ferrand	Bordeaux	Toulouse	Lyon
Clermont-Ferrand	-	376	377	167
Bordeaux	376	-	244	556
Toulouse	377	244	-	538
Lyon	167	556	538	-

Algorithme glouton pour le voyageur de commerce

Les n villes sont indexées par $0, \ldots, n-1$

Tournée : tableau T contenant la suite des indices des villes visitées
ex : $T = [1, 4, 0, 3, 2]$

1. Choisir une première ville i (par exemple $i = 0$, ou bien, au hasard avec `randrange(0,n)` qui renvoie un entier entre 0 et $n-1$)
2. initialiser la tournée : $T[0] = i$
3. Tant qu'il reste des villes non visitées (pas encore placées dans T) :
   - choisir la ville j non encore visitée qui minimise la distance à la dernière ville visitée
   - ajouter j à T
4. ajouter la ville de départ i à la fin de T pour clôre la tournée
5. renvoyer T

Exemple :

Première ville : Clermont-Ferrand
Deuxième ville : Lyon (167km)
Troisième ville : Toulouse (538km)
Quatrième ville : Bordeaux (244km)

	Clermont-Ferrand	Bordeaux	Toulouse	Lyon
Clermont-Ferrand	-	376	377	167
Bordeaux	376	-	244	556
Toulouse	377	244	-	538
Lyon	167	556	538	-

Algorithme glouton pour le voyageur de commerce

Les n villes sont indexées par $0, \ldots, n-1$

Tournée : tableau T contenant la suite des indices des villes visitées
ex : $T = [1, 4, 0, 3, 2]$

1. Choisir une première ville i (par exemple $i = 0$, ou bien, au hasard avec `randrange(0,n)` qui renvoie un entier entre 0 et $n-1$)
2. initialiser la tournée : $T[0] = i$
3. Tant qu'il reste des villes non visitées (pas encore placées dans T) :
   - choisir la ville j non encore visitée qui minimise la distance à la dernière ville visitée
   - ajouter j à T
4. ajouter la ville de départ i à la fin de T pour clôre la tournée
5. renvoyer T

Exemple :

Première ville : Clermont-Ferrand
Deuxième ville : Lyon (167km)
Troisième ville : Toulouse (538km)
Quatrième ville : Bordeaux (244km)

	Clermont-Ferrand	Bordeaux	Toulouse	Lyon
Clermont-Ferrand	-	376	377	167
Bordeaux	376	-	244	556
Toulouse	377	244	-	538
Lyon	167	556	538	-

Total : $167 + 538 + 244 + 376 = 1325 km$

Méta-heuristiques de recherche locale

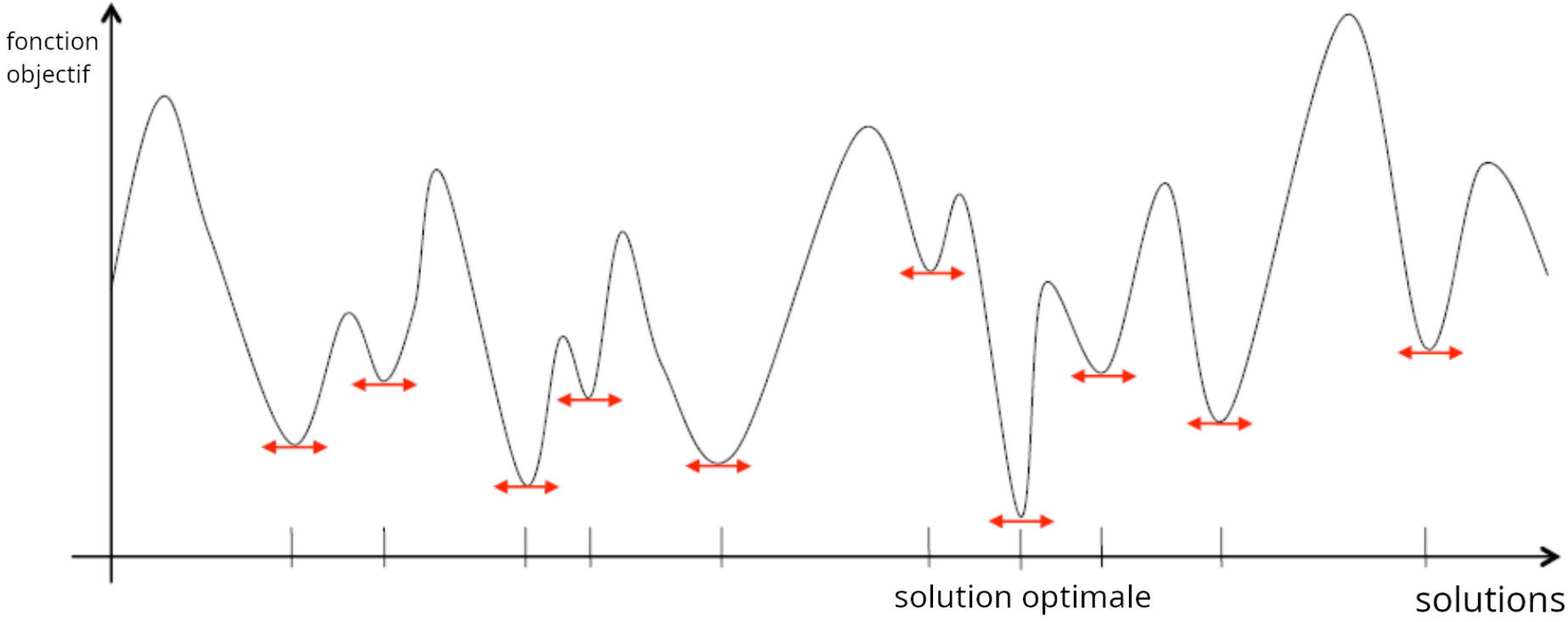
Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

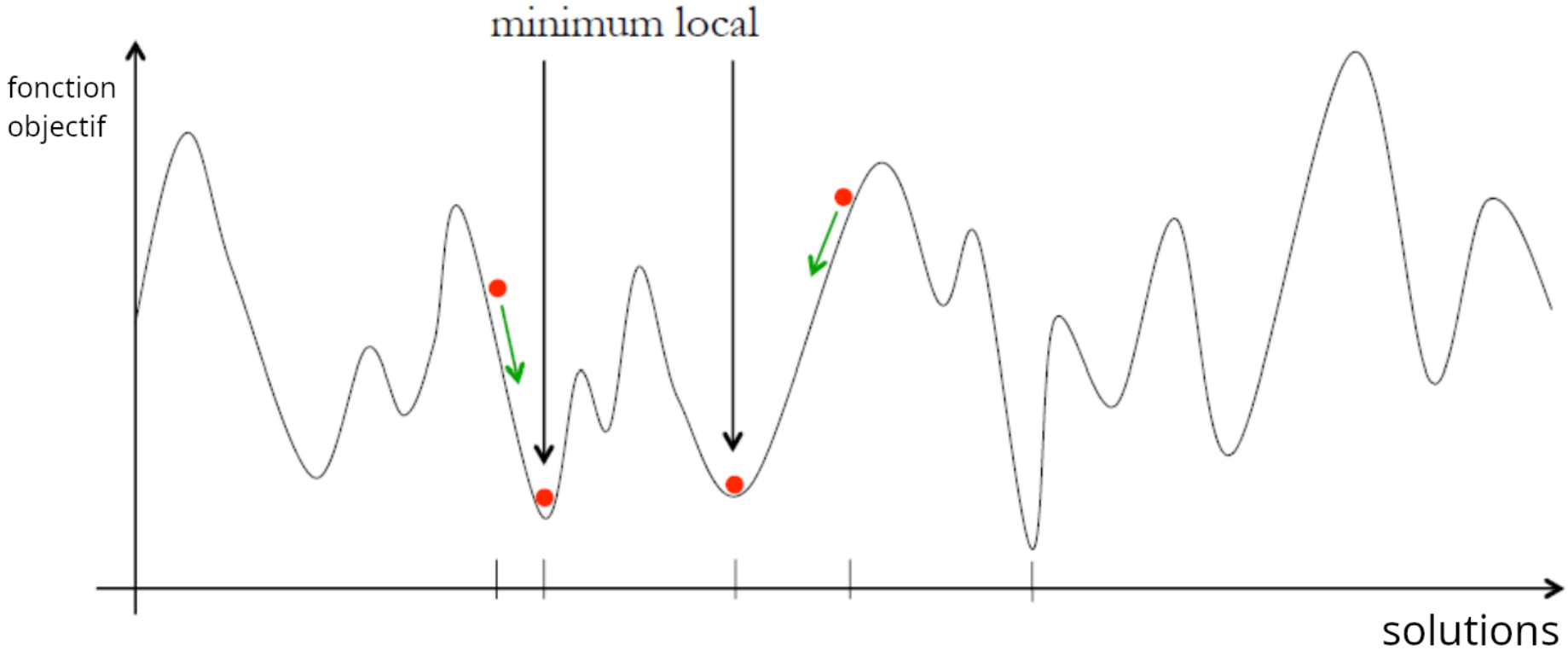


Ici, pour un problème de minimisation

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

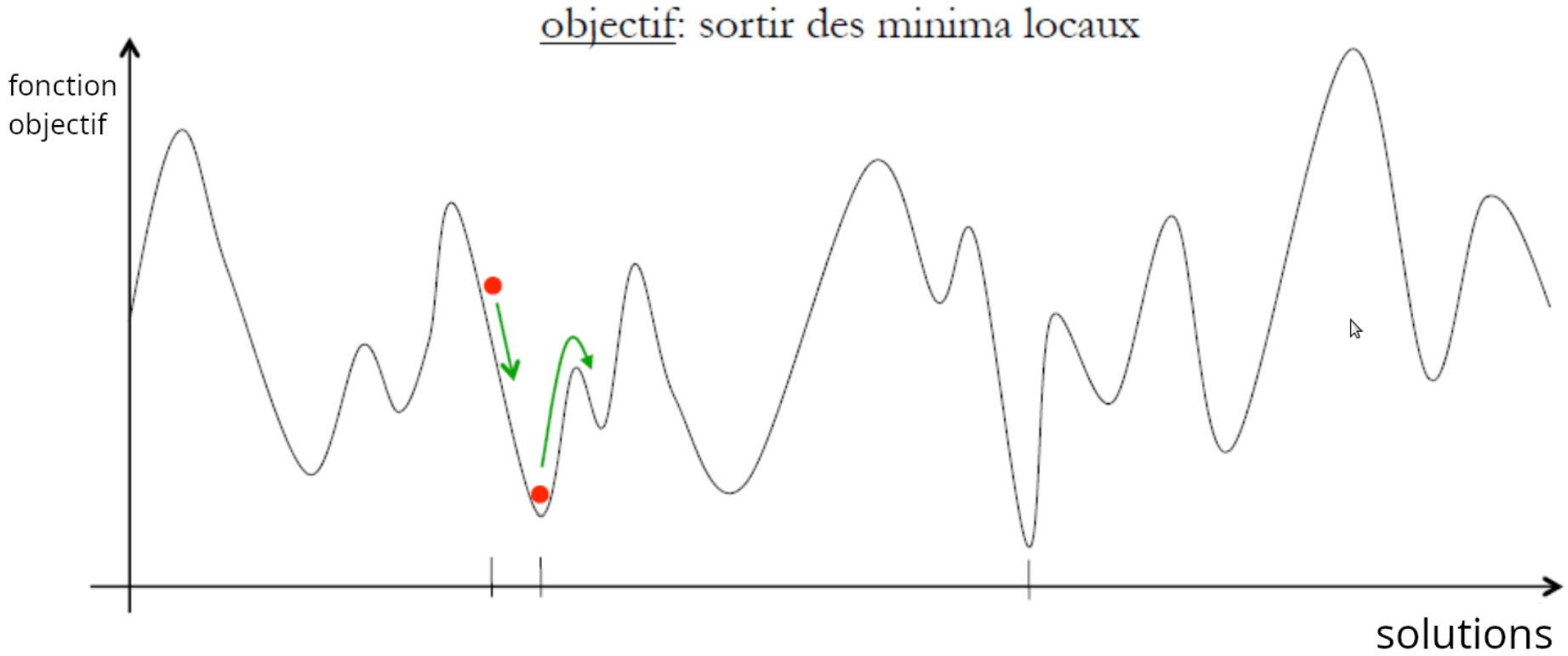


Ici, pour un problème de minimisation

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.



Ici, pour un problème de minimisation

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

On définit une opération de transformation d'une solution en une autre, proche. Pour une solution S, soit $N(S)$ l'ensemble des solutions "voisines".

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

On définit une opération de transformation d'une solution en une autre, proche. Pour une solution S, soit $N(S)$ l'ensemble des solutions "voisines".

Quelle transformation ?

Voyageur de commerce : on échange deux villes de la tournée.

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

On définit une opération de transformation d'une solution en une autre, proche. Pour une solution S, soit $N(S)$ l'ensemble des solutions "voisines".

Quelle transformation ?

Voyageur de commerce : on échange deux villes de la tournée.

T=[0,1,3,2,4]

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

On définit une opération de transformation d'une solution en une autre, proche. Pour une solution S, soit $N(S)$ l'ensemble des solutions "voisines".

Quelle transformation ?

Voyageur de commerce : on échange deux villes de la tournée.

T=[**0**,**1**,3,2,4]

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

On définit une opération de transformation d'une solution en une autre, proche. Pour une solution S, soit $N(S)$ l'ensemble des solutions "voisines".

Quelle transformation ?

Voyageur de commerce : on échange deux villes de la tournée.

T=[**0**,**1**,3,2,4] → T1=[**1**,**0**,3,2,4]

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

On définit une opération de transformation d'une solution en une autre, proche. Pour une solution S, soit $N(S)$ l'ensemble des solutions "voisines".

Quelle transformation ?

Voyageur de commerce : on échange deux villes de la tournée.

T=[**0**,1,**3**,2,4] → T1=[**1**,**0**,3,2,4]

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

On définit une opération de transformation d'une solution en une autre, proche. Pour une solution S, soit $N(S)$ l'ensemble des solutions "voisines".

Quelle transformation ?

Voyageur de commerce : on échange deux villes de la tournée.

T=[**0**,1,**3**,2,4] → T1=[**1**,**0**,3,2,4] T2=[**3**,1,**0**,2,4]

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

On définit une opération de transformation d'une solution en une autre, proche. Pour une solution S, soit $N(S)$ l'ensemble des solutions "voisines".

Quelle transformation ?

Voyageur de commerce : on échange deux villes de la tournée.

T=[0,**1**,**3**,2,4] → T1=[**1**,**0**,3,2,4] T2=[**3**,1,**0**,2,4]

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

On définit une opération de transformation d'une solution en une autre, proche. Pour une solution S, soit $N(S)$ l'ensemble des solutions "voisines".

Quelle transformation ?

Voyageur de commerce : on échange deux villes de la tournée.

T=[0,**1**,**3**,2,4] → T1=[**1**,**0**,3,2,4] T2=[**3**,1,**0**,2,4] T3=[0,**3**,**1**,2,4]

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

On définit une opération de transformation d'une solution en une autre, proche. Pour une solution S, soit $N(S)$ l'ensemble des solutions "voisines".

Quelle transformation ?

Voyageur de commerce : on échange deux villes de la tournée.

T=[0,**1**,3,**2**,4] → T1=[**1**,**0**,3,2,4] T2=[**3**,1,**0**,2,4] T3=[0,**3**,**1**,2,4]

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

On définit une opération de transformation d'une solution en une autre, proche. Pour une solution S, soit $N(S)$ l'ensemble des solutions "voisines".

Quelle transformation ?

Voyageur de commerce : on échange deux villes de la tournée.

T=[0,**1**,3,**2**,4] → T1=[**1**,**0**,3,2,4] T2=[**3**,1,**0**,2,4] T3=[0,**3**,**1**,2,4] T4=[0,**2**,3,**1**,4]

...

Méta-heuristiques de recherche locale

Méthode de la descente de gradient (gradient descent / hill climbing) :

On parcourt l'espace de solutions par des transformations successives jusqu'à trouver un optimum local.

On définit une opération de transformation d'une solution en une autre, proche. Pour une solution S, soit $N(S)$ l'ensemble des solutions "voisines".

Quelle transformation ?

Voyageur de commerce : on échange deux villes de la tournée.

T=[0,**1**,3,**2**,4] → T1=[**1**,**0**,3,2,4] T2=[**3**,1,**0**,2,4] T3=[0,**3**,**1**,2,4] T4=[0,**2**,3,**1**,4]
. . .

Algorithme de recherche locale par descente de gradient :

1. On génère une première solution S_0
   $S \leftarrow S_0$ #solution courante
2. Tant que $N(S)$ contient une solution meilleure que S :
   - choisir une nouvelle solution S' dans $N(S)$
   - $S \leftarrow S'$
3. Retourner S

Variations

Algorithme de recherche locale par descente de gradient :

1. On génère une première solution S_0
   $S \leftarrow S_0$ #solution courante
2. Tant que $N(S)$ contient une solution meilleure que S :
   - choisir une nouvelle solution S' dans $N(S)$
   - $S \leftarrow S'$
3. Retourner S

Variations

Algorithme de recherche locale par descente de gradient :

1. On génère une première solution S_0
   $S \leftarrow S_0$ #solution courante
2. Tant que $N(S)$ contient une solution meilleure que S :
   - choisir une nouvelle solution S' dans $N(S)$
   - $S \leftarrow S'$
3. Retourner S

Variantes :

Variations

Algorithme de recherche locale par descente de gradient :

1. On génère une première solution S_0
   $S \leftarrow S_0$ #solution courante
2. Tant que $N(S)$ contient une solution meilleure que S :
   - choisir une nouvelle solution S' dans $N(S)$
   - $S \leftarrow S'$
3. Retourner S

Variantes :

- Solution initiale S_0 : aléatoire, algo glouton, solution triviale...

Variations

Algorithme de recherche locale par descente de gradient :

1. On génère une première solution S_0
   $S \leftarrow S_0$ #solution courante
2. Tant que $N(S)$ contient une solution meilleure que S :
   - choisir une nouvelle solution S' dans $N(S)$
   - $S \leftarrow S'$
3. Retourner S

Variantes :

• Solution initiale S_0 : aléatoire, algo glouton, solution triviale...

• Sélection de la nouvelle solution : meilleur voisin, premier voisin améliorant, voisin aléatoire...

Variations

Algorithme de recherche locale par descente de gradient :

1. On génère une première solution S_0
   $S \leftarrow S_0$ #solution courante
2. Tant que $N(S)$ contient une solution meilleure que S :
   - choisir une nouvelle solution S' dans $N(S)$
   - $S \leftarrow S'$
3. Retourner S

Variantes :

• Solution initiale S_0 : aléatoire, algo glouton, solution triviale...

• Sélection de la nouvelle solution : meilleur voisin, premier voisin améliorant, voisin aléatoire...

Passer à une solution non-améliorante pour quitter un mauvais optimum local (diversification).

Variations

Algorithme de recherche locale par descente de gradient :

1. On génère une première solution S_0
   $S \leftarrow S_0$ #solution courante
2. Tant que $N(S)$ contient une solution meilleure que S :
   - choisir une nouvelle solution S' dans $N(S)$
   - $S \leftarrow S'$
3. Retourner S

Variantes :

• Solution initiale S_0 : aléatoire, algo glouton, solution triviale...

• Sélection de la nouvelle solution : meilleur voisin, premier voisin améliorant, voisin aléatoire...

Passer à une solution non-améliorante pour quitter un mauvais optimum local (diversification).

→ Variante du recuit simulé (1983) :

On définit une température qui décroit lentement au cours de l'algorithme.
La probabilité de diversification dépend de la température.

Le recuit, un technique métallurgique

Scott Kirkpatrick

C. Daniel Gelatt

Mario P. Vecchi

Méta-heuristiques à populations de solutions

Idée générale : on fait évoluer des ensembles de solutions (populations)

Algorithmes génétiques (J. H. Holland, 1975)

- Population initiale : ensemble de solutions
- On “croise” des paires de solutions pour obtenir la population suivante
- Des “mutations” apparaissent aléatoirement

Méta-heuristiques à populations de solutions

Idée générale : on fait évoluer des ensembles de solutions (populations)

Algorithmes génétiques (J. H. Holland, 1975)

- Population initiale : ensemble de solutions
- On “croise” des paires de solutions pour obtenir la population suivante
- Des “mutations” apparaissent aléatoirement

Algorithmes de colonies de fourmis (M. Dorigo, 1992)

- Des “fourmis” se déplacent plus ou moins aléatoirement dans l’instance et laissent des phéromones
- Les fourmis suivantes vont là où il y a le plus de phéromones
- Les phéromones disparaissent après un certain temps

Méta-heuristiques à populations de solutions

Idée générale : on fait évoluer des ensembles de solutions (populations)

Algorithmes génétiques (J. H. Holland, 1975)

- Population initiale : ensemble de solutions
- On “croise” des paires de solutions pour obtenir la population suivante
- Des “mutations” apparaissent aléatoirement

Algorithmes de colonies de fourmis (M. Dorigo, 1992)

- Des “fourmis” se déplacent plus ou moins aléatoirement dans l’instance et laissent des phéromones
- Les fourmis suivantes vont là où il y a le plus de phéromones
- Les phéromones disparaissent après un certain temps

Algorithmes d’essaims de particules (J. Kennedy, R. Eberhart, 1995)

- Essaim de particules : ensemble de solutions
- Les “particules” se déplacent en fonction de leurs voisines vers la meilleure solution connue

Détails sur les algorithmes génétiques

Population de solutions :

T1=[0,1,2,3,4] T2=[3,4,1,0,2] T3=[4,1,2,0,3] T4=[3,2,4,1,0] T5=[3,0,4,1,2]
T6=[0,4,2,1,3]

Détails sur les algorithmes génétiques

Population de solutions :

T1=[0,1,2,3,4] T2=[3,4,1,0,2] T3=[4,1,2,0,3] T4=[3,2,4,1,0] T5=[3,0,4,1,2] T6=[0,4,2,1,3]

Sélection :

On prend les trois meilleures → T2=[3,4,1,0,2] T5=[3,0,4,1,2] T6=[0,4,2,1,3]

Détails sur les algorithmes génétiques

Population de solutions :

T1=[0,1,2,3,4] T2=[3,4,1,0,2] T3=[4,1,2,0,3] T4=[3,2,4,1,0] T5=[3,0,4,1,2] T6=[0,4,2,1,3]

Sélection :

On prend les trois meilleures → T2=[3,4,1,0,2] T5=[3,0,4,1,2] T6=[0,4,2,1,3]

Reproduction :

Détails sur les algorithmes génétiques

Population de solutions :

T1=[0,1,2,3,4] T2=[3,4,1,0,2] T3=[4,1,2,0,3] T4=[3,2,4,1,0] T5=[3,0,4,1,2] T6=[0,4,2,1,3]

Sélection :

On prend les trois meilleures → T2=[3,4,1,0,2] T5=[3,0,4,1,2] T6=[0,4,2,1,3]

Reproduction :

on croise T2 et T5 → [**3,4**,0,1,2] [**3,4,1**,0,2] [**3,0**,4,1,2] [**3,0,4**,1,2]

Détails sur les algorithmes génétiques

Population de solutions :

T1=[0,1,2,3,4] T2=[3,4,1,0,2] T3=[4,1,2,0,3] T4=[3,2,4,1,0] T5=[3,0,4,1,2]
T6=[0,4,2,1,3]

Sélection :

On prend les trois meilleures → T2=[3,4,1,0,2] T5=[3,0,4,1,2] T6=[0,4,2,1,3]

Reproduction :

on croise T2 et T5 → [**3,4**,0,1,2] [**3,4,1**,0,2] [**3,0**,4,1,2] [**3,0,4**,1,2]
on croise T2 et T6 → [**3,4**,0,2,1] [**3,4,1**,0,2] [**0,4**,3,1,2] [**0,4,2**,3,1]

Détails sur les algorithmes génétiques

Population de solutions :

T1=[0,1,2,3,4] T2=[3,4,1,0,2] T3=[4,1,2,0,3] T4=[3,2,4,1,0] T5=[3,0,4,1,2]
T6=[0,4,2,1,3]

Sélection :

On prend les trois meilleures → T2=[3,4,1,0,2] T5=[3,0,4,1,2] T6=[0,4,2,1,3]

Reproduction :

on croise T2 et T5 → [**3,4**,0,1,2] [**3,4,1**,0,2] [**3,0**,4,1,2] [**3,0,4**,1,2]
on croise T2 et T6 → [**3,4**,0,2,1] [**3,4,1**,0,2] [**0,4**,3,1,2] [**0,4,2**,3,1]
on croise T5 et T6 → [**3,0**,4,2,1] [**3,0,4**,2,1] [**0,4**,3,1,2] [**0,4,2**,3,1]

Détails sur les algorithmes génétiques

Population de solutions :

T1=[0,1,2,3,4] T2=[3,4,1,0,2] T3=[4,1,2,0,3] T4=[3,2,4,1,0] T5=[3,0,4,1,2]
T6=[0,4,2,1,3]

Sélection :

On prend les trois meilleures → T2=[3,4,1,0,2] T5=[3,0,4,1,2] T6=[0,4,2,1,3]

Reproduction :

on croise T2 et T5 → [**3,4**,0,1,2] [**3,4,1**,0,2] [**3,0**,4,1,2] [**3,0,4**,1,2]
on croise T2 et T6 → [**3,4**,0,2,1] [**3,4,1**,0,2] [**0,4**,3,1,2] [**0,4,2**,3,1]
on croise T5 et T6 → [**3,0**,4,2,1] [**3,0,4**,2,1] [**0,4**,3,1,2] [**0,4,2**,3,1]

Mutation :

on échange deux villes au hasard avec une (faible) probabilité, par ex. 1%

Détails sur les algorithmes génétiques

Population de solutions :
T1=[0,1,2,3,4] T2=[3,4,1,0,2] T3=[4,1,2,0,3] T4=[3,2,4,1,0] T5=[3,0,4,1,2]
T6=[0,4,2,1,3]

Sélection :
On prend les trois meilleures → T2=[3,4,1,0,2] T5=[3,0,4,1,2] T6=[0,4,2,1,3]

Reproduction :
on croise T2 et T5 → [**3,4**,0,1,2] [**3,4,1**,0,2] [**3,0**,4,1,2] [**3,0,4**,1,2]
on croise T2 et T6 → [**3,4**,0,2,1] [**3,4,1**,0,2] [**0,4**,3,1,2] [**0,4,2**,3,1]
on croise T5 et T6 → [**3,0**,4,2,1] [**3,0,4**,2,1] [**0,4**,3,1,2] [**0,4,2**,3,1]

Mutation :
on échange deux villes au hasard avec une (faible) probabilité, par ex. 1%

On recommence avec la nouvelle génération !

Détails sur les colonies de fourmis

- Tant qu'on le souhaite, on répète :
  - Une nouvelle fourmi part de la ville 0
  - Tant qu'elle n'a pas visité toutes les villes :

    Elle choisit la suivante au hasard de la manière suivante.
    - Plus la ville est loin, moins elle a de chances d'être choisie
    - Plus le chemin menant à cette ville est couvert de phéromones de fourmis, plus cette ville a des chances d'être choisie
  - Plus la tournée obtenue est longue, moins on y place de phéromones
    (par ex. on dépose une quantité de $1/\textit{longueur}$ phéromones)
  - Les phéromones s'évaporent : on les réduit de x%, par ex. $x = 20\%$
- Pour obtenir la tournée finale, on suit les phéromones de façon déterministe
  (dernière fourmi)

Détails sur les colonies de fourmis

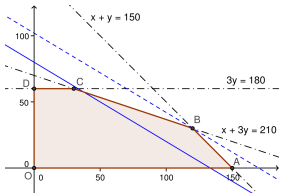
- Tant qu'on le souhaite, on répète :
  - Une nouvelle fourmi part de la ville 0
  - Tant qu'elle n'a pas visité toutes les villes :

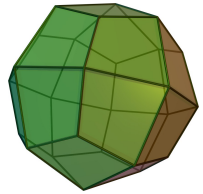
    Elle choisit la suivante au hasard de la manière suivante.
    - Plus la ville est loin, moins elle a de chances d'être choisie
    - Plus le chemin menant à cette ville est couvert de phéromones de fourmis, plus cette ville a des chances d'être choisie
  - Plus la tournée obtenue est longue, moins on y place de phéromones
    (par ex. on dépose une quantité de $1/\textit{longueur}$ phéromones)
  - Les phéromones s'évaporent : on les réduit de x%, par ex. $x = 20\%$
- Pour obtenir la tournée finale, on suit les phéromones de façon déterministe
  (dernière fourmi)

Conclusion générale

Programmes linéaires (PL)

- peuvent exprimer **certains** problèmes importants
- simplexe : efficace en pratique pour trouver la solution optimale (presque tout le temps mais pas toujours...)
- + il existe des méthodes plus complexes, efficaces tout le temps

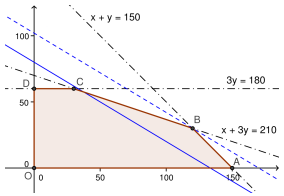


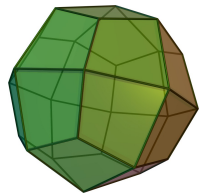


Conclusion générale

Programmes linéaires (PL)

- peuvent exprimer **certains** problèmes importants
- simplexe : efficace en pratique pour trouver la solution optimale (presque tout le temps mais pas toujours...)
- + il existe des méthodes plus complexes, efficaces tout le temps





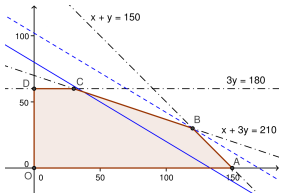
Programmes linéaires en nombres entiers (PLNE)

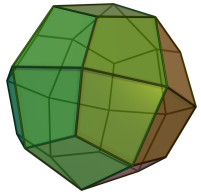
- peuvent exprimer **presque tous** les problèmes importants
- MAIS : pas d'algorithme efficace connu pour trouver la solution optimale !

Conclusion générale

Programmes linéaires (PL)

- peuvent exprimer **certains** problèmes importants
- simplexe : efficace en pratique pour trouver la solution optimale (presque tout le temps mais pas toujours...)
- + il existe des méthodes plus complexes, efficaces tout le temps





Programmes linéaires en nombres entiers (PLNE)

- peuvent exprimer **presque tous** les problèmes importants
- MAIS : pas d'algorithme efficace connu pour trouver la solution optimale !

On peut utiliser :

- des **méthodes de branchement** exactes mais lentes
  → branch and bound, ...
- des **algorithmes d'approximation** ou **algorithmes heuristiques** pour avoir une solution (non-optimale) en temps raisonnable
  → relaxation linéaire, ...
  → algos gloutons, descente de gradient, populations, ...