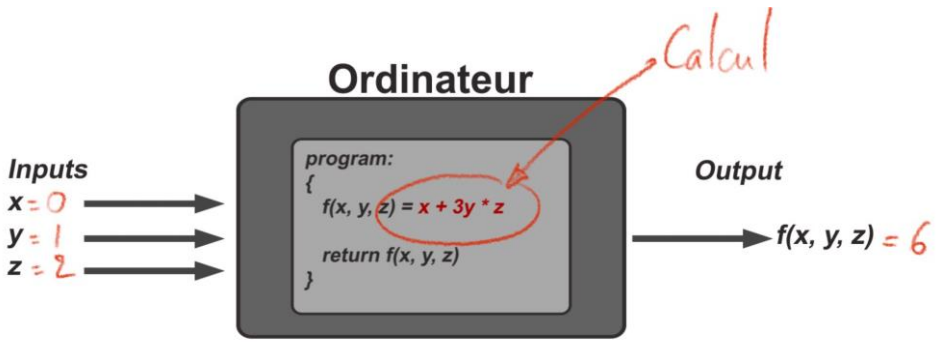
Intelligence Artificielle
IA

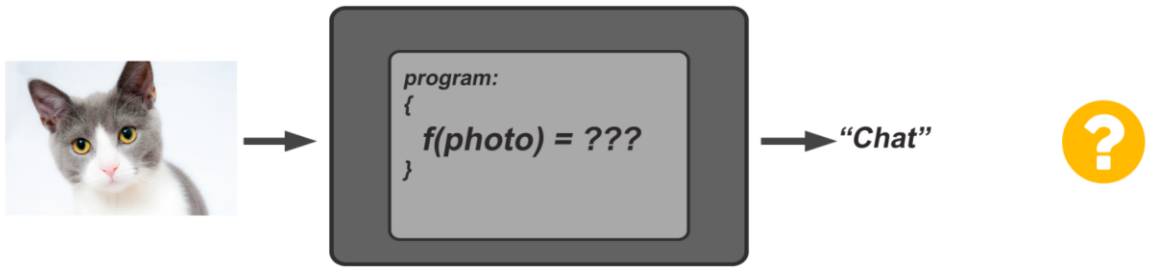
Par Dr. CHAIBI Hasna

A.U 2023-2024

l'Intelligence Artificielle

- Les ordinateurs ne sont pas intelligents, il ne savent faire que des calculs.
- Ils peuvent ainsi effectuer en quelques secondes des calculs qui nous prendraient des milliers d'années à résoudre, par contre, ils sont incapables de reconnaître un chat sur une photo. C'est parce que dans le premier cas, il suffit d'entrer des équations mathématiques bien connues dans l'ordinateur en écrivant un programme, c'est ce qu'on appelle la **programmation**, mais dans le second cas, il n'existe pas de calcul exact pour reconnaître un chat à partir de pixels…





l’Intelligence Artificielle

- D’une manière générale, peut-on décrire le fonctionnement de notre cerveau (raisonnement, mémoire, etc.) par un système d’équations ?

➢ **Non**. Mais il existe une **solution**…

- Face à ce problème, **Alan Turing**, un des pères fondateurs de **l’intelligence artificielle**, s’est posé une question, en **1950**, dans son article scientifique Computing Machinery and Intelligence. Voici la question : « **Can machines do what we do** ? » autrement dit, **est-ce qu’une machine peut faire ce que nous, les êtres humains, faisons ?** Parce que nous, en plus de faire des calculs, et bien nous sommes capables de résoudre des problèmes, jouer aux échecs, conduire une voiture, ou bien reconnaître les objets que nous voyons tout autour de nous…
- Cette question pose en réalité la base de ce qu’on appelle **l’Intelligence Artificielle**

l’Intelligence

- **Être intelligent** c’est savoir répondre rapidement à une situation complexe.
- **Être intelligent** c’est de trouver une réponse satisfaisante plutôt que la réponse optimale.
- **Être intelligent** c’est savoir trouver ce qui important dans une information.
- **Être intelligent** c’est savoir s’adapter si les conditions change ou savoir réagir à une situation.

l’Intelligence Artificielle

- L'**intelligence artificielle** (**IA**) est « l'ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence humaine »

- **L’intelligence artificielle**, c’est toute technologie informatique **algorithmique** qui permet de résoudre des **problèmes complexes** qu'on aurait cru réservés à l'intelligence humaine, **en simulant des capacités humaines comme la perception et le raisonnement. -** Cédric Villani **-**
- le **Machine Learning,** c’est une technique qui permet aux ordinateurs de réussir à faire tout ce genre de choses.

l’Intelligence Artificielle

❑ Machine Learning, une technique au sein de L’IA

- Grâce au Machine Learning, nos ordinateurs sont capables de conduire des voitures et des avions de façon plus sûre que nous, ils peuvent diagnostiquer un patient (cancer, fracture) de façon plus fiable qu’un médecin.

- Le Machine Learning est aujourd’hui au cœur de nos systèmes de prise de décision : Banque, justice, sécurité, business, marketing. Les algorithmes de Google, Youtube, Facebook, Amazon, ou Netflix, fonctionnent tous grâce au Machine Learning.

l’Intelligence Artificielle

- Les **véhicules autonomes** sont une prochaine révolution permise par l’IA qui promet de transformer le monde des transports.

- Dans un futur pas si lointain, nous pourrons nous faire conduire par une voiture autonome équipés de capteurs et d'un ordinateur très puissant. Plus besoin de conduire, nous pourrons consacrer ce temps à lire ou à consulter nos emails. Au-delà d’un confort individuel, cela pourrait permettre de fluidifier la circulation et de réduire les accidents et ça grâce au **Machine learning**.

l’Intelligence Artificielle

- **Le Machine Learning**, ça consiste à écrire un programme qui au début ne sait rien faire, mais qui va **apprendre** à faire quelque chose avec le **temps** et **l’expérience**… Un peu comme un être humain apprendrait à faire du vélo : au début on y arrive pas du tout, mais à force d’en faire et bien on y arrive de mieux en mieux, jusqu’au moment où on en fait super bien.
- Cet exemple colle hyper bien avec la définition du Machine Learning donnée par l’américain **Tom Mitchell** en **1998**. Selon lui, une machine apprend, lorsque sa **Performance P** à faire une **Tâche T** s’améliore grâce à une nouvelle **Expérience E**.
- **Tâche T** : Faire du vélo
- **Performance P** : Rouler droit, ne pas tomber
- **Expérience E** : chaque fois que l’on fait du vélo, y compris les fois où l’on tombe.

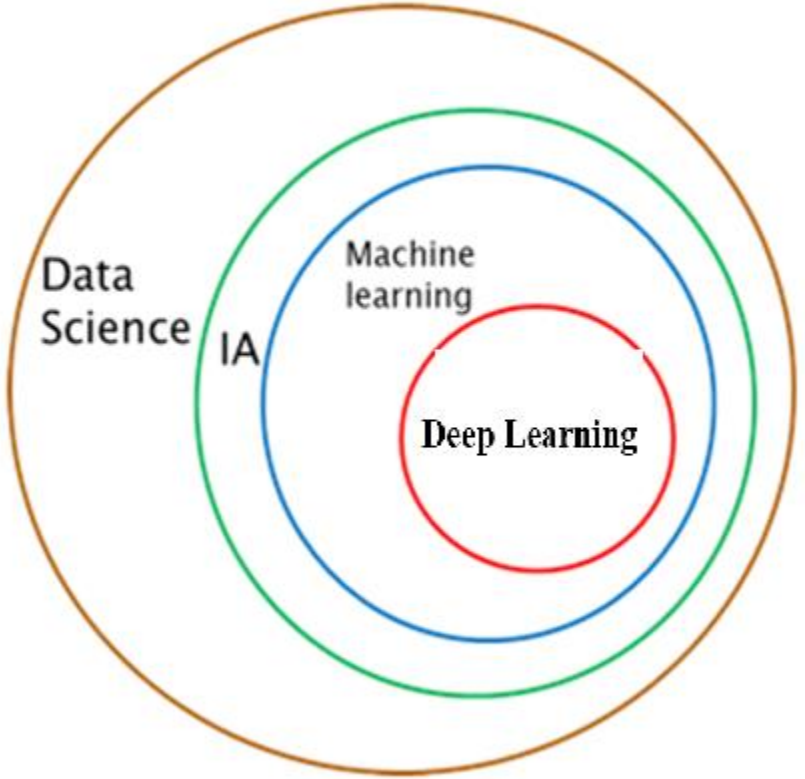
l’Intelligence Artificielle

- **L’apprentissage automatique,** ou ***Machine Learning***, est un sous-ensemble de **l’intelligence artificielle** qui permet à un programme informatique d'effectuer une tâche pour laquelle il n'est pas programmé explicitement : il est programmé pour apprendre à la faire. **On donne au programme de nombreuses données et il apprend à partir de ces données.**
- Cette discipline est notamment utilisée dans votre boîte email pour classer automatiquement un email en **spam**.

l'Intelligence Artificielle

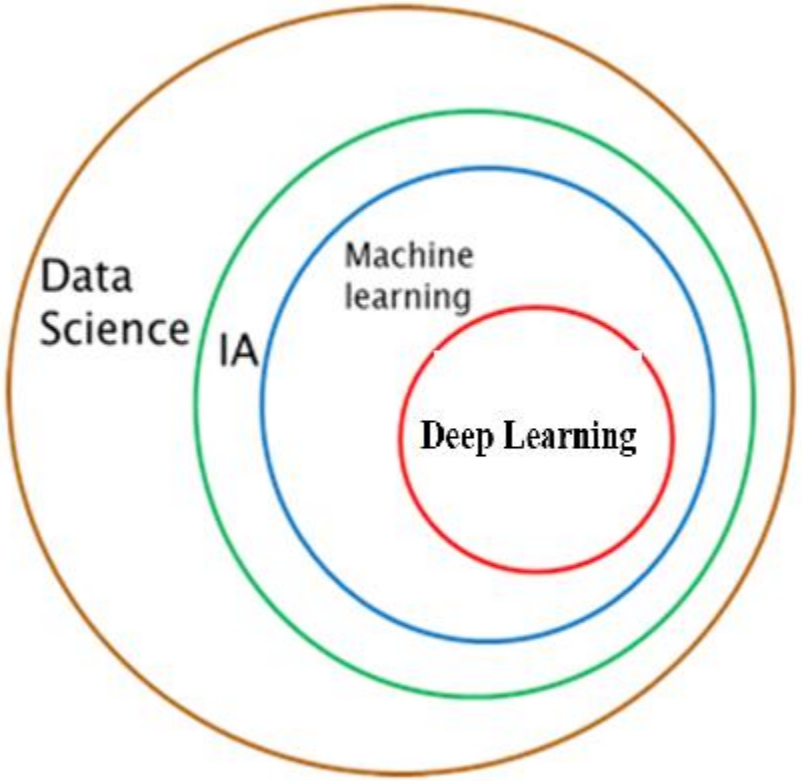
- Le **Deep Learning**, qui est une discipline au sein même du Machine Learning, cherche à entraîner des modèles extrêmement **complexes** avec des **milliards de données (Big Data)** dans le but de surpasser les performances humaines.
- Le **Deep Learning** s'applique souvent sur des quantités de données beaucoup plus importantes que le Machine Learning. Il apprend de cette masse d'exemples et obtient dans certains cas de bien meilleurs résultats que les disciplines traditionnelles d'intelligence artificielle.

- Le **Machine Learning et Le Deep Learning** sont aujourd'hui des domaines en plein essor, ceci grâce à l'émergence des objets connectés (IoT) en 2007 (lancement de l'IPhone) qui fournissent aujourd'hui une grande quantité de données (**Big Data**) pour entraîner les modèles des **Data Scientists.**



l’Intelligence Artificielle

- Nous produisons chaque jour de nombreuses données comme les emails, les photos, etc.
- Le **Big Data**, c’est l’ensemble de ces données massives.
- L’i**ntelligence artificielle** et la **Data Science** sont deux disciplines qui sont utilisées conjointement, notamment pour mettre en place du **Machine Learning** ou du **Deep Learning.**
- La **data science**, c’est le champ scientifique globale de toutes ces disciplines.



l’Intelligence Artificielle

- Une des particularités de **l’intelligence artificielle** de ce point de vue est qu’elle se situe non pas dans la continuité d’un seul domaine mathématique, mais est au contraire à l’intersection de nombreuses, si ce n’est de toutes les problématiques classiques des mathématiques. Qu’elles concernent:

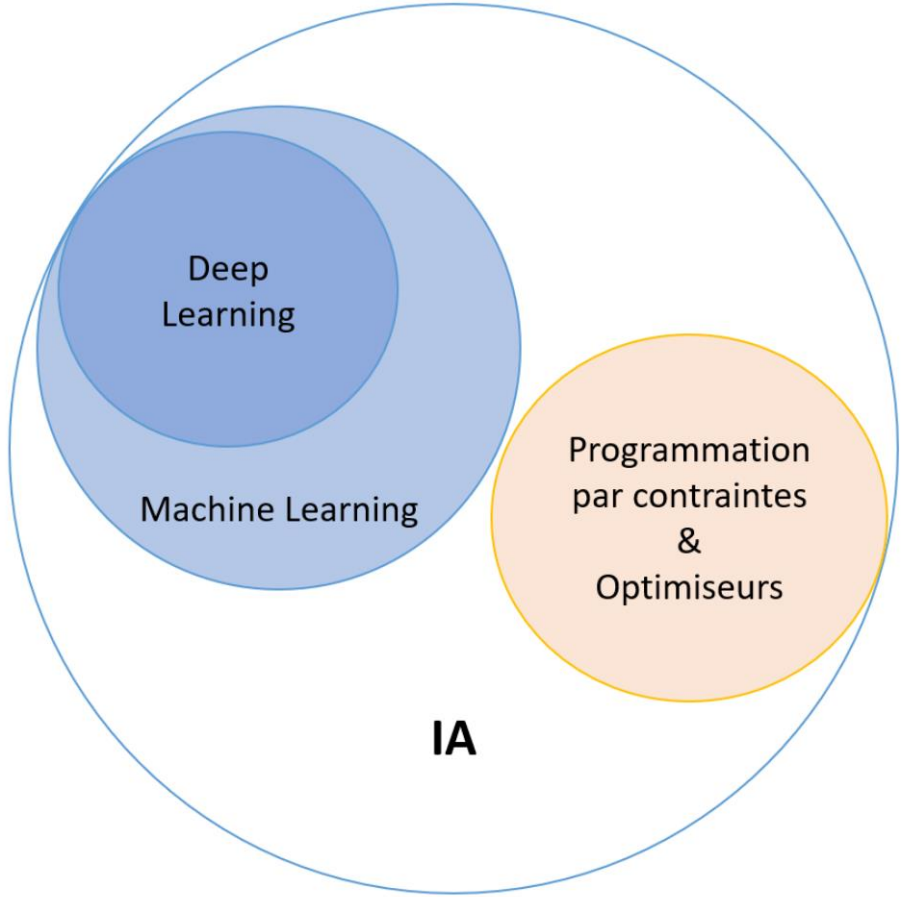
➢ Les **mathématiques appliquées**, comme les statistiques et les probabilités bien sûr, également, l’optimisation, les équations stochastiques et aux dérivées partielles, la théorie des jeux, etc.,

➢ Les **mathématiques fondamentales** (logique , algèbre, etc.)

➢ Les **mathématiques discrètes** (graphes, réseaux, combinatoires)

l’Intelligence Artificielle

Les utilisations de l’IA aujourd’hui peuvent être regroupées en 3 catégories principales : **l’identification, la prédiction** et la **génération** de données.
Dans le cas de l’identification, l’IA est aujourd’hui utilisée pour tout ce qui a trait à la reconnaissance faciale, reconnaissance de texte (OCR) ainsi que de la détection vocale, ou bien de la traduction ; de manière générale, tout ce qui a trait au traitement de l’image et du son peut être lié à l’IA. Ce sont dans le cadre de ces applications que l’on utilise le plus souvent le deep learning.
Dans le cas de la prédiction, l’IA peut servir dans nombre infini de situations, comme par exemple le calcul prévisionnel du CA d’une entreprise, la prédiction de trafic automobile, ou bien la prévision météorologique.



Représentation des différents types d'intelligence artificielle

Optimisation

Optimisation

- **Euler dit** « il n’y a rien au monde qui ne se réalise sans la volonté de **minimiser** ou **maximiser** quelque chose

- L'**optimisation** est une branche des mathématiques cherchant à **modéliser**, à **analyser** et à **résoudre** analytiquement ou numériquement les problèmes qui consistent à **minimiser** ou **maximiser** une fonction.

Notion d’optimisation combinatoire

- **Processus d’optimisation**

1. Modélisation:

- La définition l’espace de recherche (l’ensemble de solutions possible)
- La formulation mathématique : variables de décision, fonction objective, contraintes.

2. Résolution: application d'une méthode d'optimisation

3. Implémentation: la mise en ouvre de la solution

Notion d’optimisation combinatoire

- **L'optimisation combinatoire** occupe une place très importante en informatique et en Mathématique.
- Vise à **trouver la meilleure solution**, d’un problème donné, parmi un ensemble d’alternatives possibles.
- De nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation.
- Utilisée dans différents secteurs industriels:
  - Télécommunications, Electronique, Mécanique,
  - Chimie, biologie,
  - Transport, ….
- Plusieurs techniques **exactes** et **approchées** proposées pour résoudre les problèmes d’optimisation.

Notion d'optimisation combinatoire

L'optimisation:

- Une branche des Mathématiques cherchant à modéliser et à résoudre analytiquement ou numériquement les problèmes de **minimisation** ou de **maximisation** d'une fonction sur un ensemble.

Plus formellement:

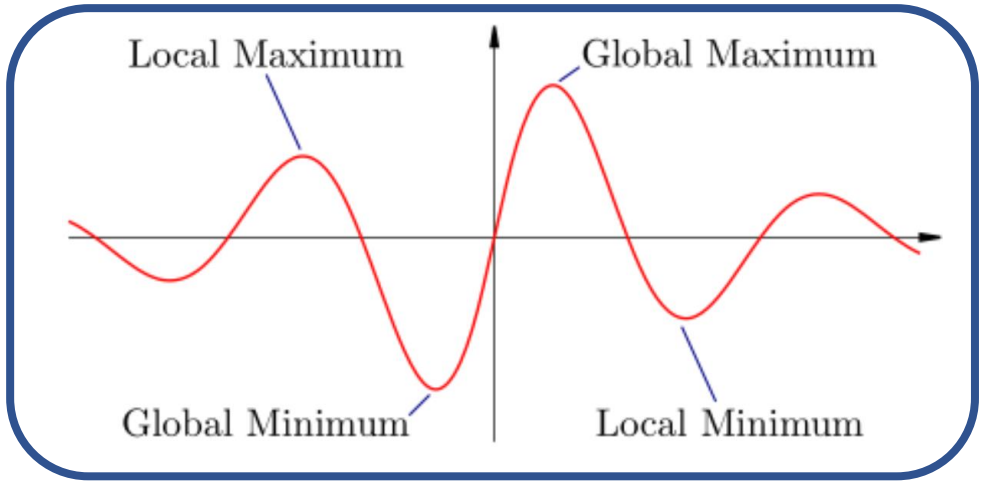
- Étant donnée une fonction $f: I \rightarrow R$, dite fonction objective, fonction de coût, l'optimisation consiste à trouver des valeurs minimum ou de maximum **x*** ,tel que:

$$f(x^*) = \min_{x \in I} f(x)$$ ⟸ **Dans le cas d'un problème de minimisation**

ou

$$f(x^*) = \max_{x \in I} f(x)$$ ⟸ **Dans le cas d'un problème de maximisation**

Notion d'optimisation combinatoire



- On dit que ***f* admet un maximum global en** $x_0 \in I$ si

$\forall x \in I \quad f(x) \leq f(x_0).$

Ce maximum $f(x_0) = \max f(I)$ est aussi noté $\max_{x \in I} f(x)$.

- On dit que ***f* admet un minimum global en** $x_1 \in I$ si

$\forall x \in I \quad f(x_1) \leq f(x).$

Ce minimum $f(x_1) = \min f(I)$ est aussi noté $\min_{x \in I} f(x)$.

Notion d’optimisation combinatoire

- L’optimisation combinatoire:

➢ Spécialisation de l’optimisation (cas particulier)

➢ consiste à:

  - Trouver une **solution optimale** à partir d’un ensemble fini et discret de solutions.

Ou plus formellement:

  - **Minimiser** ou **maximiser** une fonction sur un ensemble fini et discret (potentiellement très grand)

Notion d'optimisation combinatoire

- Un problème d'optimisation combinatoire peut être défini par un quadruplet (**S**, **f**, **C**, **s***) où :
- **S**: représente **l'espace de recherche** ou **l'ensemble de solutions** possibles.

- **f : S→R** est la **fonction objective** à optimiser.

  ➢ Permet de définir une relation d'ordre entre chaque pair de solutions.
  ➢Une fonction de **coût** , **distance**, **temps** ….

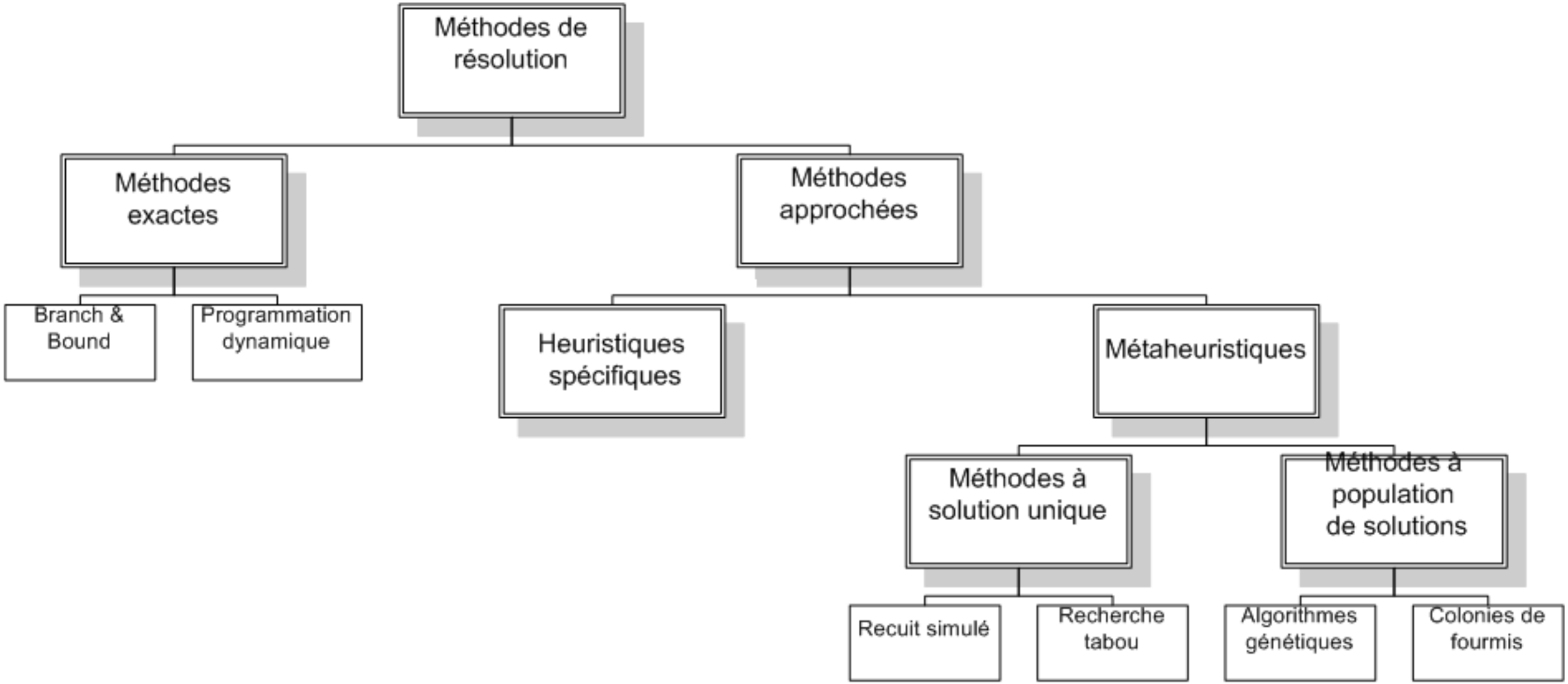
Notion d'optimisation combinatoire

- **C** : **L'ensemble de contraintes**.
  - Définit des conditions sur l'espace d'états que les variables doivent satisfaire.
  - Souvent des contraintes d'inégalité ou d'égalité ($x_1>0$).
  - Permettent en général de limiter l'espace de recherche (solutions réalisables).
- **s***: **l'optimum global**
  - **Minimum globale** ou le **maximum globale** de la fonction **f**
  - **s*** vérifie l'inégalité suivante:

    - Dans le cas de **minimisation**: $\forall s \in S, \quad f(s^*) \leq f(s)$
    - Dans le cas de **maximisation**: $\forall s \in S, \quad f(s^*) \geq f(s)$

Notion d'optimisation combinatoire

- L'optimisation combinatoire occupe une place très importante en recherche opérationnelle.
- Son importance se justifie par:
  - La grande difficulté des problèmes d'optimisation (**NP- Difficile**)
  - De nombreuses applications pratiques pouvant être formulées sous la forme d'une problèmes d'optimisation combinatoire
- L'importances de ces problèmes
  - De nombreuse méthodes de résolution ont été développées

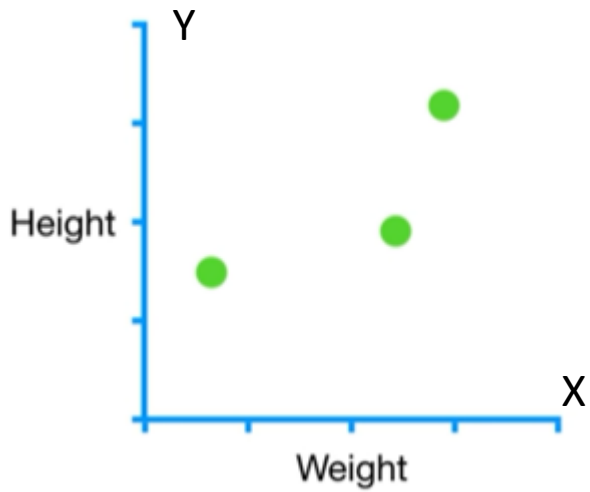

Méthodes de résolution
Méthodes exactes
Méthodes approchées
Branch & Bound
Programmation dynamique
Heuristiques spécifiques
Métaheuristiques
Méthodes à solution unique
Méthodes à population de solutions
Recuit simulé
Recherche tabou
Algorithmes génétiques
Colonies de fourmis

DataSet

Nous allons voir comment développer un **modèle** de Machine Learning à partir d'un **Dataset** à une seule variable x . On aura donc un **Dataset** avec **m=3** exemples et **1** « ***feature*** » variable.

Notre **Dataset** représente la taille en fonction du poids. Ce **Dataset** nous pourrait nous donner le nuage de point suivant :

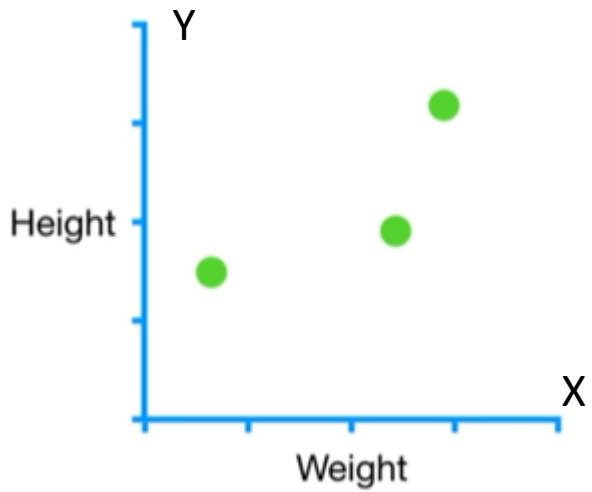
Weight (x)	Height (y)
0,5	1,4
2, 3	1,9
2,9	3,2



Modèle

En voyant notre nuage de point, on dirait clairement qu'il suit une **tendance linéaire**, voilà pourquoi nous allons développer un modèle… **linéaire** ! Notre modèle $\boldsymbol{f(x) = ax + b}$

Weight (x)	Height (y)
0,5	1,4
2, 3	1,9
2,9	3,2



Fonction cout

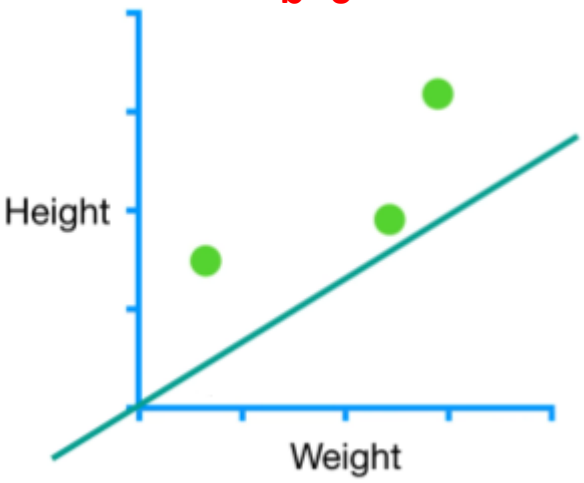
objet	Weight (x)	Height (y)
Obj1	0,5	1,4
Obj2	2, 3	1,9
Obj3	2,9	3,2

$$y = ax + b$$

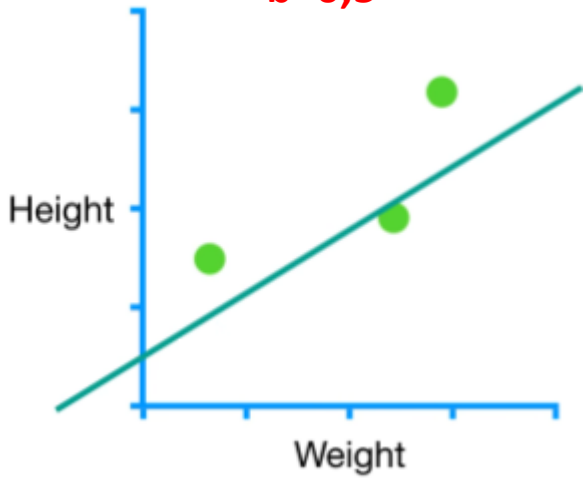
Nous allons voir comment le ML peut ajuster la ligne (le modèle) aux données en trouvant les valeurs optimales des paramètres a et b du modèle
Tout d'abord nous commençons par fixer la valeur de **a** et nous essayons de voir comment le ML va trouver la valeur optimale de **b**

Pour commencer on fixe par exemple **a=0,64** et on change la valeur de **b**

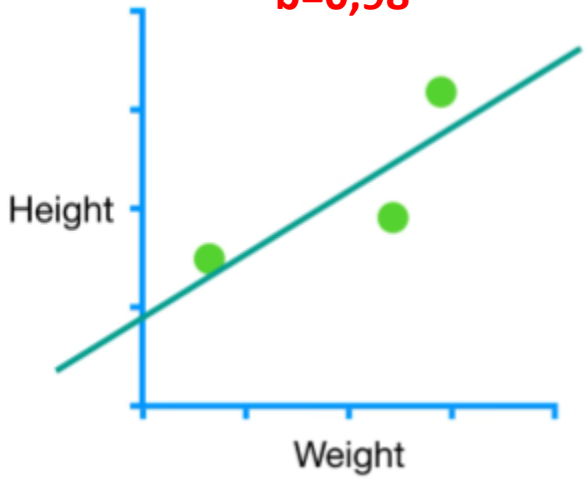
b=0



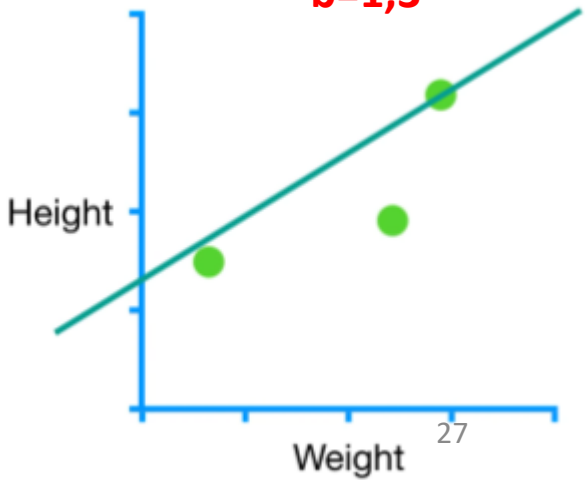
b=0,5



b=0,98

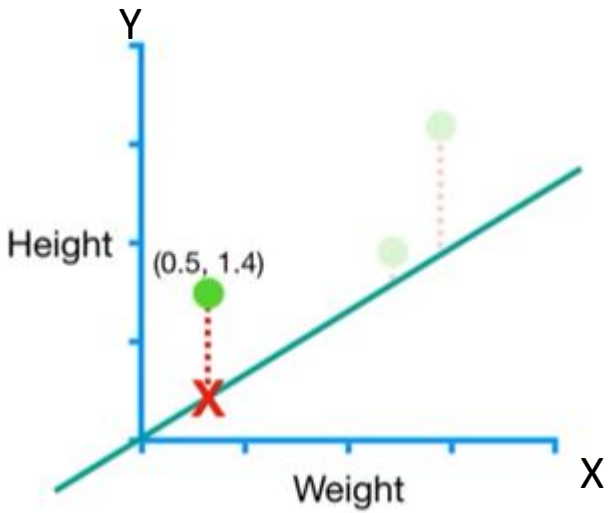


b=1,3



Fonction cout

objet	Weight (x)	Height (y)
Obj1	0,5	1,4
Obj2	2, 3	1,9
Obj3	2,9	3,2



a=0,64 et b=0 , calculons l'erreur

- Le **premier point** de données représente l'objet 1 avec un poids **0,5** et une taille **1,4**
- Sur la ligne de prédiction (le modèle), on a pour un objet de poids **x=0,5** la taille prédite **f(x) = 0,64*x + 0 = 0,64*0,5 + 0 = 0,32**
- **L'erreur= (taille observée – taille prédite)= (1,4-0,32)=1,1**

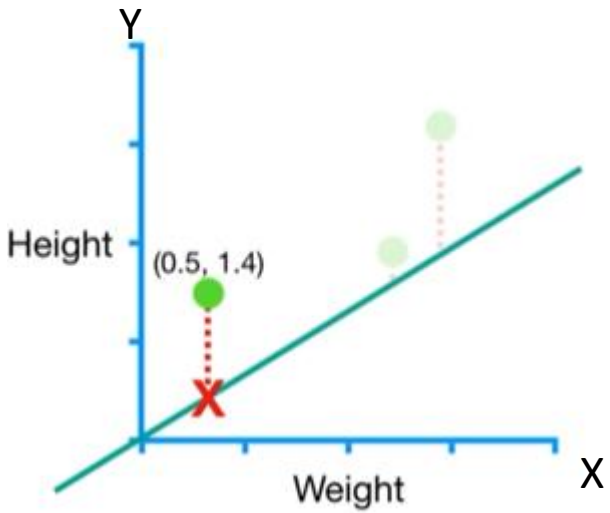
- **EQ=$1,1^2$**

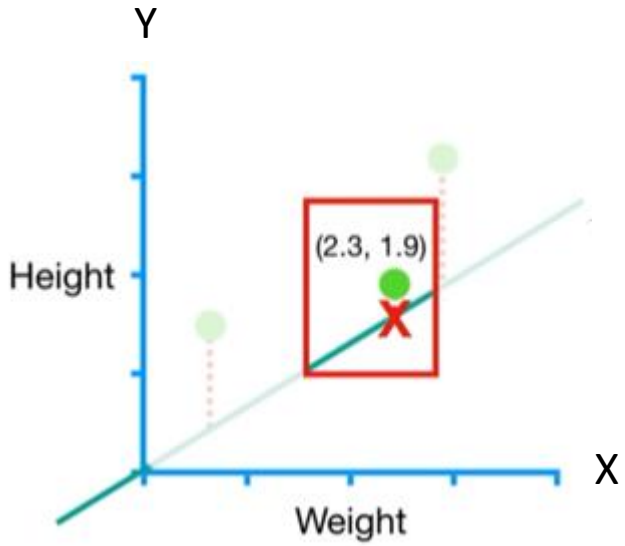
Fonction cout

objet	Weight (x)	Height (y)
Obj1	0,5	1,4
Obj2	2, 3	1,9
Obj3	2,9	3,2

- Le **deuxième point** de données représente l'objet 2 avec un poids **2,3** et une taille **1,9**
- Sur la ligne de prédiction (le modèle) on a pour un objet de poids **x=2,3** la taille prédite **f(x) = 0,64*x + 0 = 0,64*2,3 + 0 =1,5**
- **L'erreur= (taille observée – taille prédite)= (1,9-1,5)=0,4**

- **EQ=$1,1^2 + 0,4^2$**

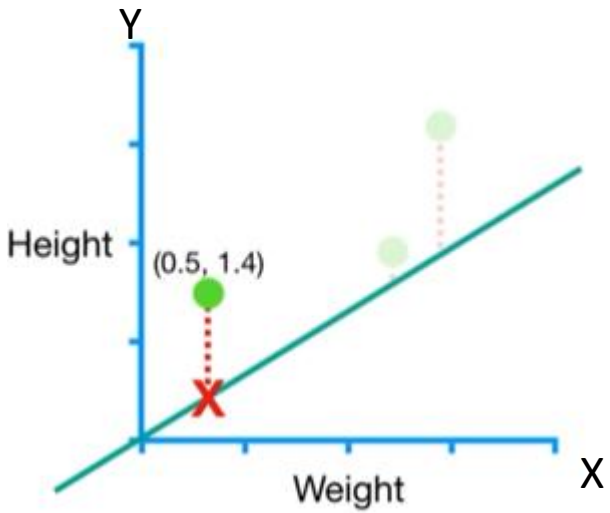


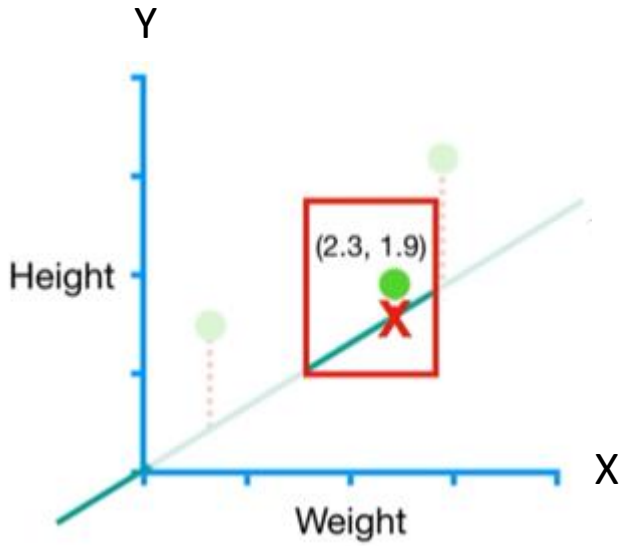


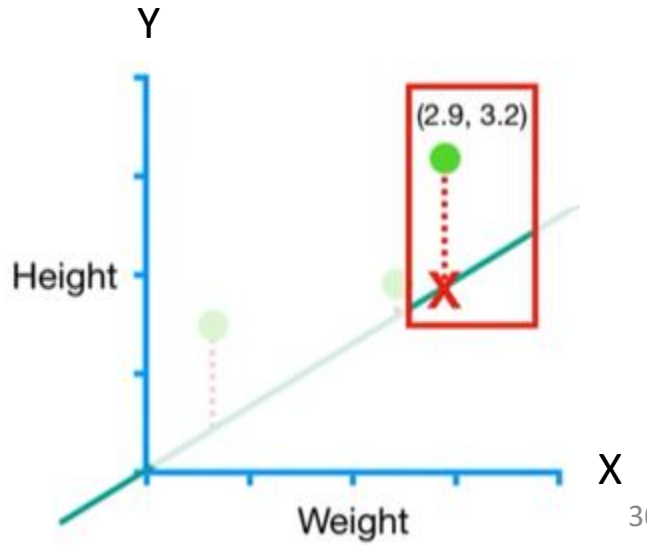
Fonction cout

objet	Weight (x)	Height (y)
Obj1	0,5	1,4
Obj2	2, 3	1,9
Obj3	2,9	3,2

- Le troisième point de données représente l'objet 3 avec un poids **2,9** et une taille **3,2**
- Sur la ligne de prédiction (le modèle) on a pour un objet de poids **x=2,9** la taille prédite **f(x) = 0,64*x + 0 = 0,64*2,9 + 0 =1,8**
- **L'erreur= (taille observée – taille prédite)= (3,2-1,8)=3,1**
- **EQ=$1,1^2 + 0,4^2 + 3,1^2 =$ 4,6**

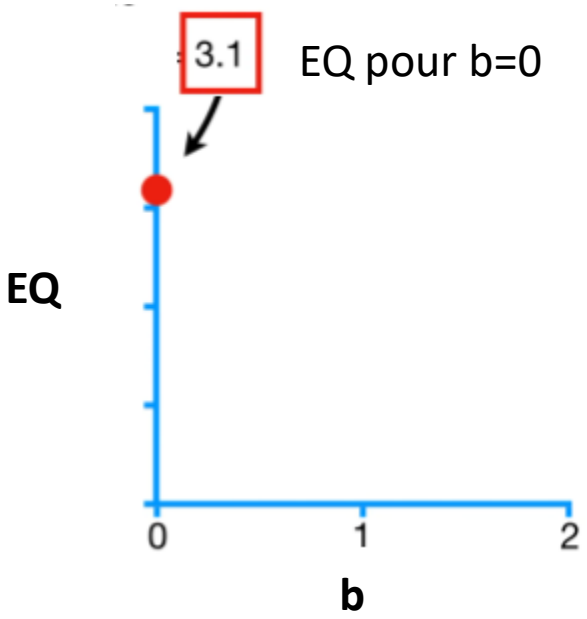






Fonction cout

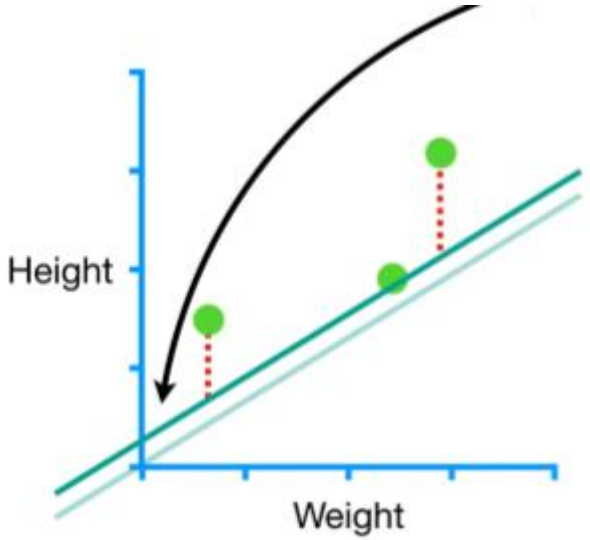
- **EQ=$1{,}1^2 + 0{,}4^2 + 3{,}1^2 = 3{,}1$**



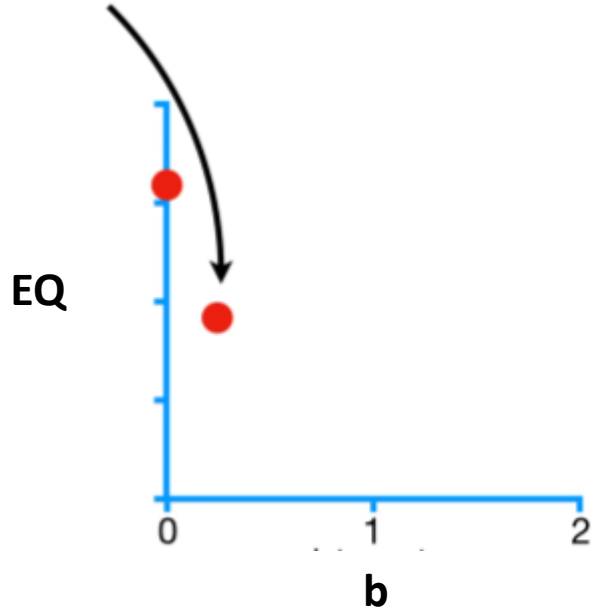
Le graphe représente la somme d'Erreur Quadratique en fonction de b

Fonction cout

Le modèle lorsque **b=0,25**

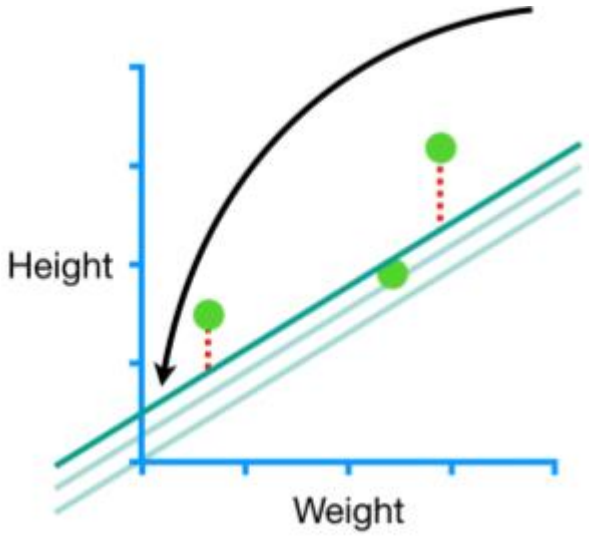


Nous obtenons EQ ici

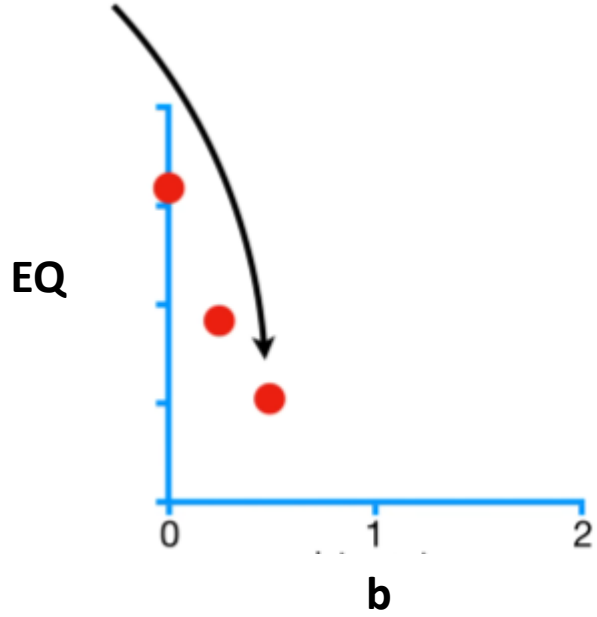


Fonction cout

Le modèle lorsque **b=0,5**

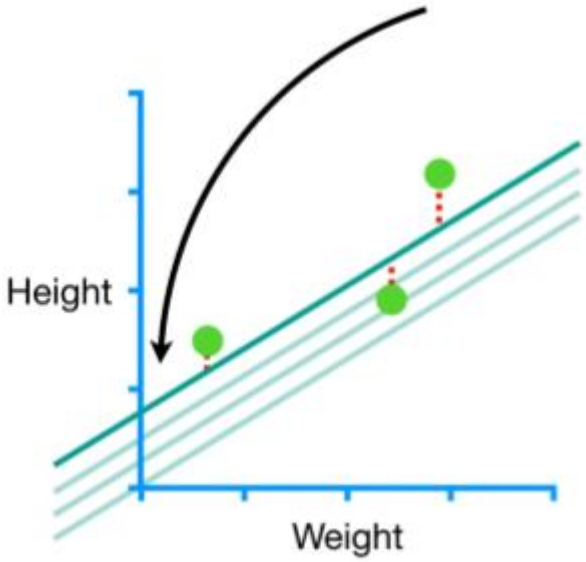


Nous obtenons EQ ici

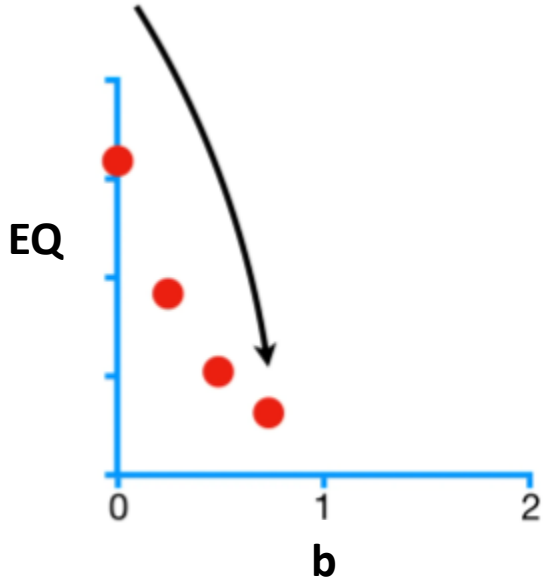


Fonction cout

En augment la valeur de b

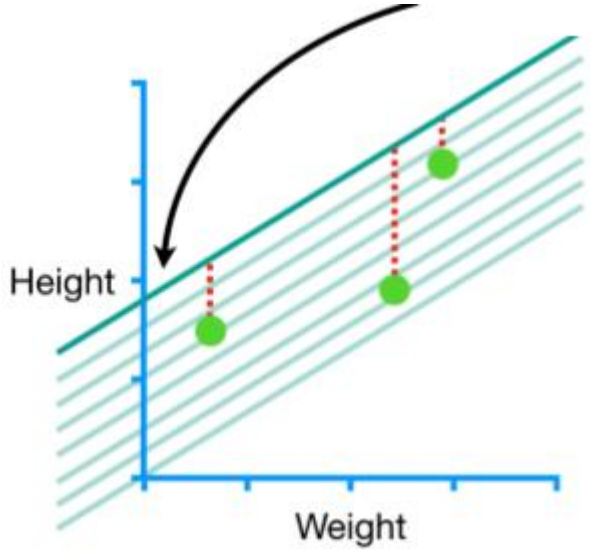


Nous obtenons Les EQ suivantes

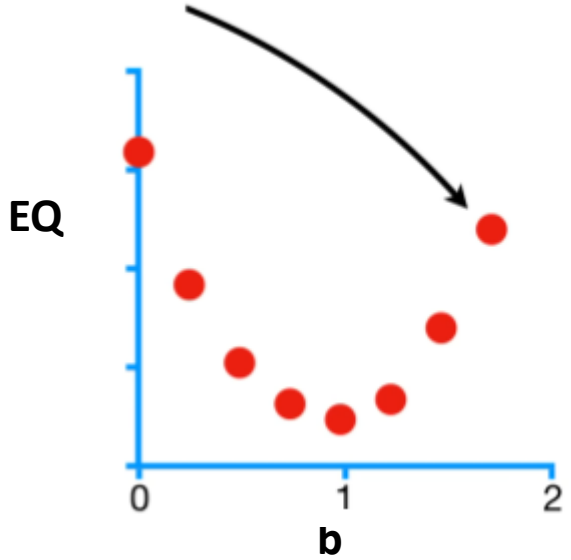


Fonction cout

En augment encore les valeurs de **b**



Nous obtenons Les EQ suivantes

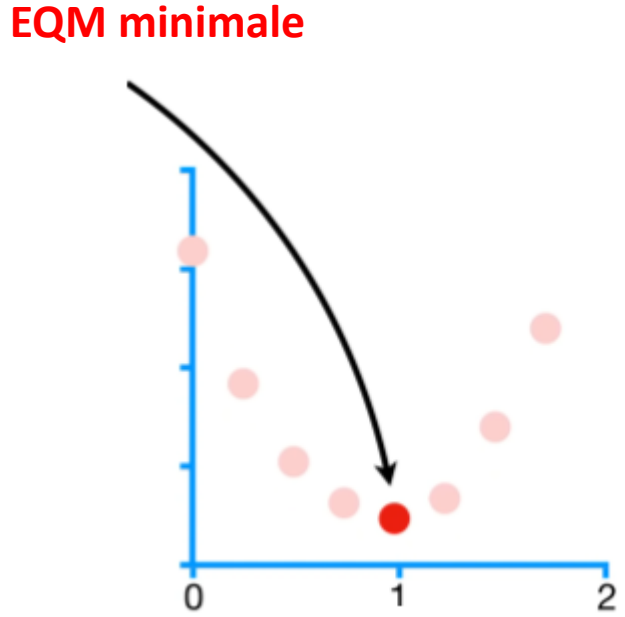


4. L'Algorithme d'apprentissage

En Supervised Learning, la machine cherche les **paramètres** (a et b) de modèle $ax^{(i)} + b$ qui **minimisent** la **Fonction Coût** $J(a,b) = \frac{1}{2m}\sum_{i=1}^{m}[ax^{(i)} + b - y^{(i)}]^2$.

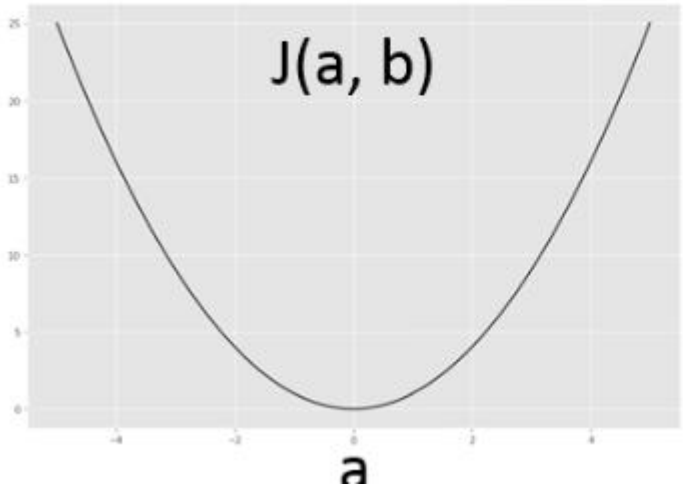
C'est ça qu'on appelle **l'apprentissage**.

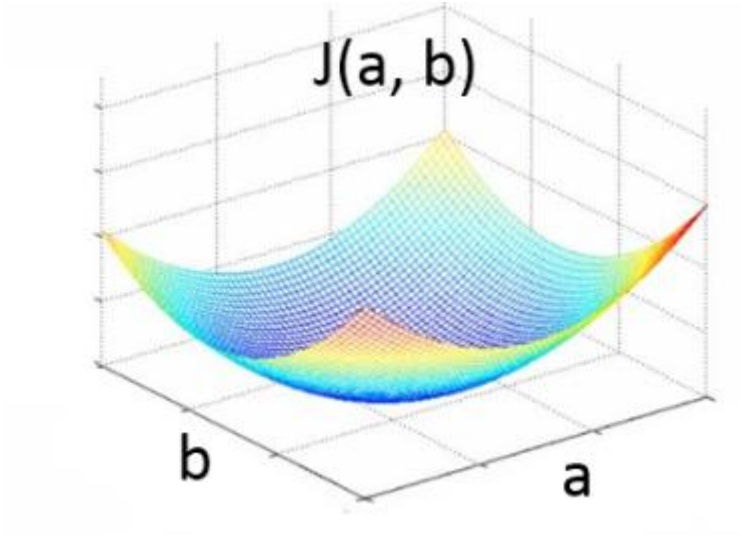
Cette phrase est **très importante**. C'est l'essentiel de ce qu'il faut comprendre en Machine Learning.



4. L'Algorithme d'apprentissage

- La **Fonction Coût** $J(a,b) = \frac{1}{2m}\sum_{i=1}^{m}[ax^{(i)} + b - y^{(i)}]^2$ ne dépend que de deux paramètres : **a** et **b** .
- Quand on l'observe, elle a l'apparence d'une vallée bien lisse. On dit que c'est une fonction **convexe**. Cette propriété est très importante pour s'assurer de converger vers le minimum avec l'algorithme de la **descente de gradient**





4. L'Algorithme d'apprentissage

- On dit que la machine apprend quand elle trouve quels sont les paramètres du modèle qui **minimisent** la fonction Coût.
- On cherche donc à développer un **algorithme de minimisation**. Il existe un paquet de méthodes de minimisation (méthode des **moindres carrés**, méthode de **Newton**, **Gradient Descent**, **Simplex**, etc.)
- Le **Gradient Descent**, que nous allons décortiquer en détail dans les prochains slides, permet de converger progressivement vers le minimum de n'importe quelle fonction **convexe** (comme la Fonction Coût) en suivant la direction de la pente (le gradient) qui descend, d'où son nom de Gradient Descent.

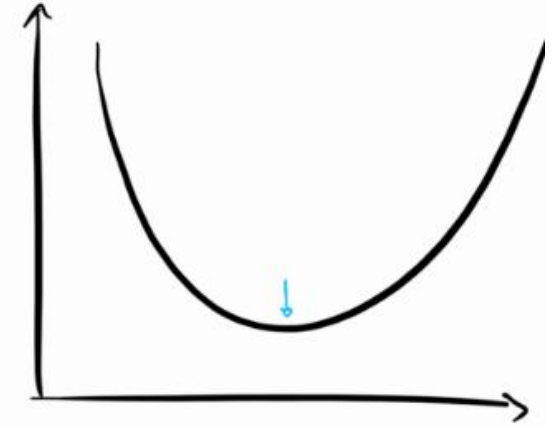
L'Algorithme d'apprentissage: Descente de gradient

- La **Descente de Gradient**, (ou ***Gradient Descent*** en anglais) est un des algorithmes **les plus importants** de tout le Machine Learning et de tout le Deep Learning.
- Il s'agit d'un algorithme **d'optimisation** extrêmement puissant qui permet d'entraîner les modèles de **régression linéaire**, **régression logistiques** ou encore les **réseaux de neurones**.
- Si vous vous lancez dans le Machine Learning, il est donc **impératif** de comprendre en profondeur l'algorithme de la descente de gradient

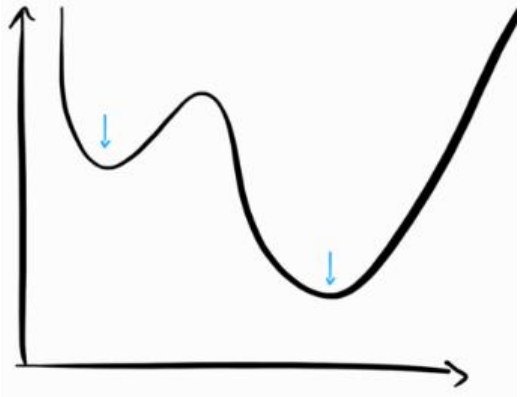
L'Algorithme d'apprentissage: Descente de gradient

- La Descente de Gradient est un algorithme **d'optimisation** qui permet de trouver le **minimum** de n'importe quelle fonction **convexe** en convergeant progressivement vers celui-ci.

- *Rappel: Une fonction **convexe** est une fonction dont l'allure ressemble à celle d'une belle vallée avec au centre un **minimum global**. A l'inverse, une fonction **non-convexe** est une fonction qui présente plusieurs **minimums locaux** et l'algorithme de descente de gradient ne doit pas être utilisé sur ces fonctions, au risque de se bloquer au premier minima rencontré.*



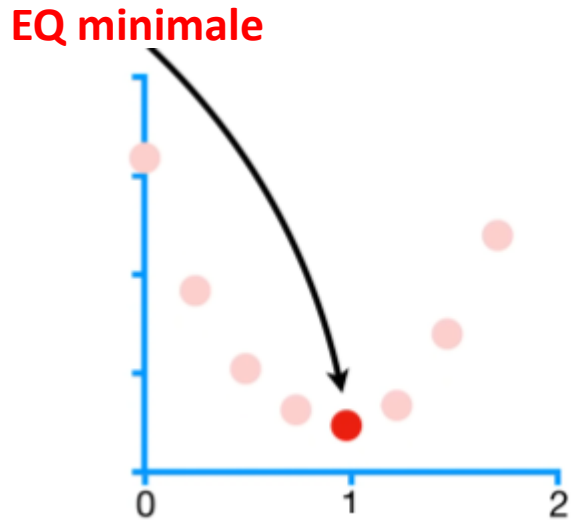
Fonction Convexe

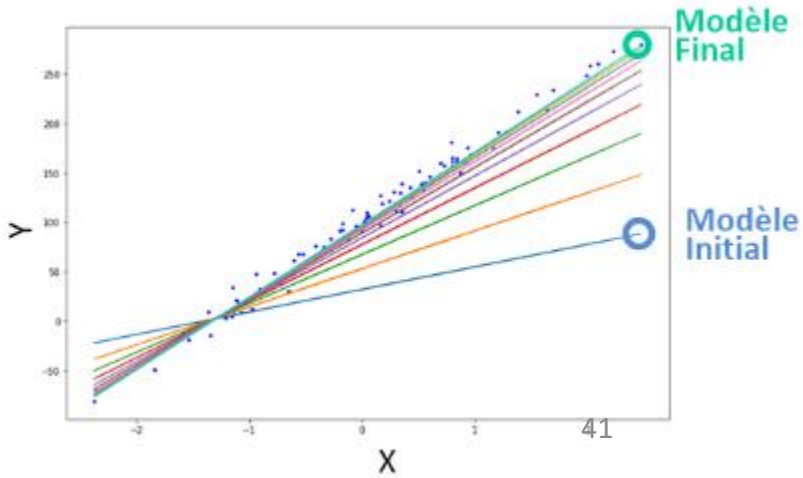


Fonction Non-Convexe

L'Algorithme d'apprentissage: Descente de gradient

- En Machine Learning, on va utiliser l'algorithme de la Descente de Gradient dans les problèmes **d'apprentissage supervisé** pour **minimiser la fonction coût**, qui justement est une fonction **convexe** (par exemple l'*erreur quadratique moyenne*).
- C'est grâce à cet algorithme que la machine apprend, c'est-à-dire trouve le meilleur modèle. En effet, rappelez-vous que minimiser la fonction coût revient à trouver les paramètres **a, b , c , etc**. qui donnent les plus petites erreurs entre notre modèle et les points du Dataset.





L'Algorithme d'apprentissage: Descente de gradient

❑Trouver le minimum

Étape 1 : Calcul de la dérivée de la Fonction Coût

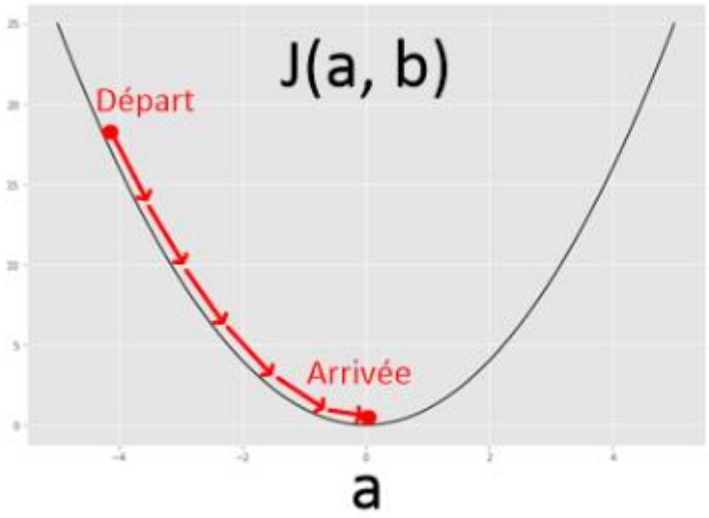
- Nous partons d'un point initial aléatoire puis nous mesurons la valeur de la pente en ce point. Et comment mesure-t-on une pente en mathématique ? En calculant la **dérivée** de la fonction !

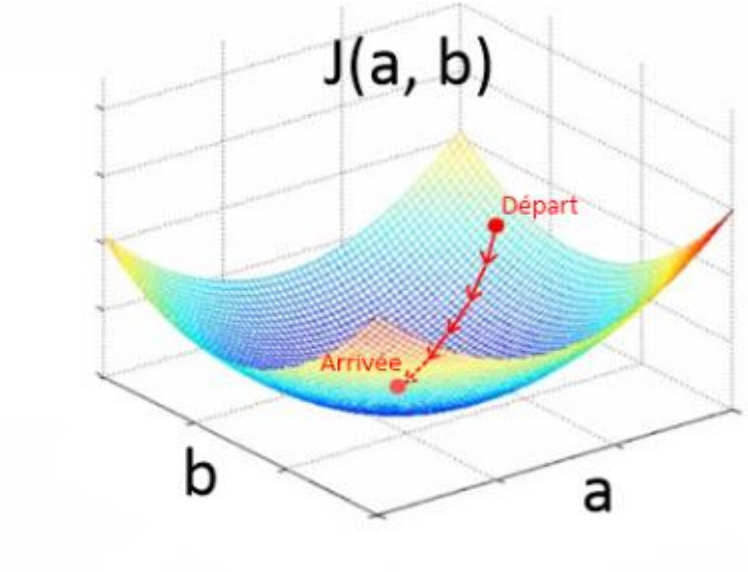
Étape 2 : Mise à jour des paramètres du modèle

- On progresse ensuite d'une certaine distance dans la direction de la pente qui descend, On appelle cette distance ***Learning Rate***, que l'on pourrait traduire par ***vitesse d'apprentissage***.
- Cette opération a pour résultat de modifier la valeur des paramètres (**a** et **b**) de notre modèle

L'Algorithme d'apprentissage: Descente de gradient

- En répétant ces deux étapes en boucle, l'algorithme de **Gradient Descent** est donc un algorithme **itératif**. Pour l'illustrer sur un graphique, je vais prendre l'exemple de la fonction coût **J(a,b)** que l'on a développé pour une régression linéaire. L'algorithme permet de trouver la valeur idéale pour les paramètres **a** et **b**.



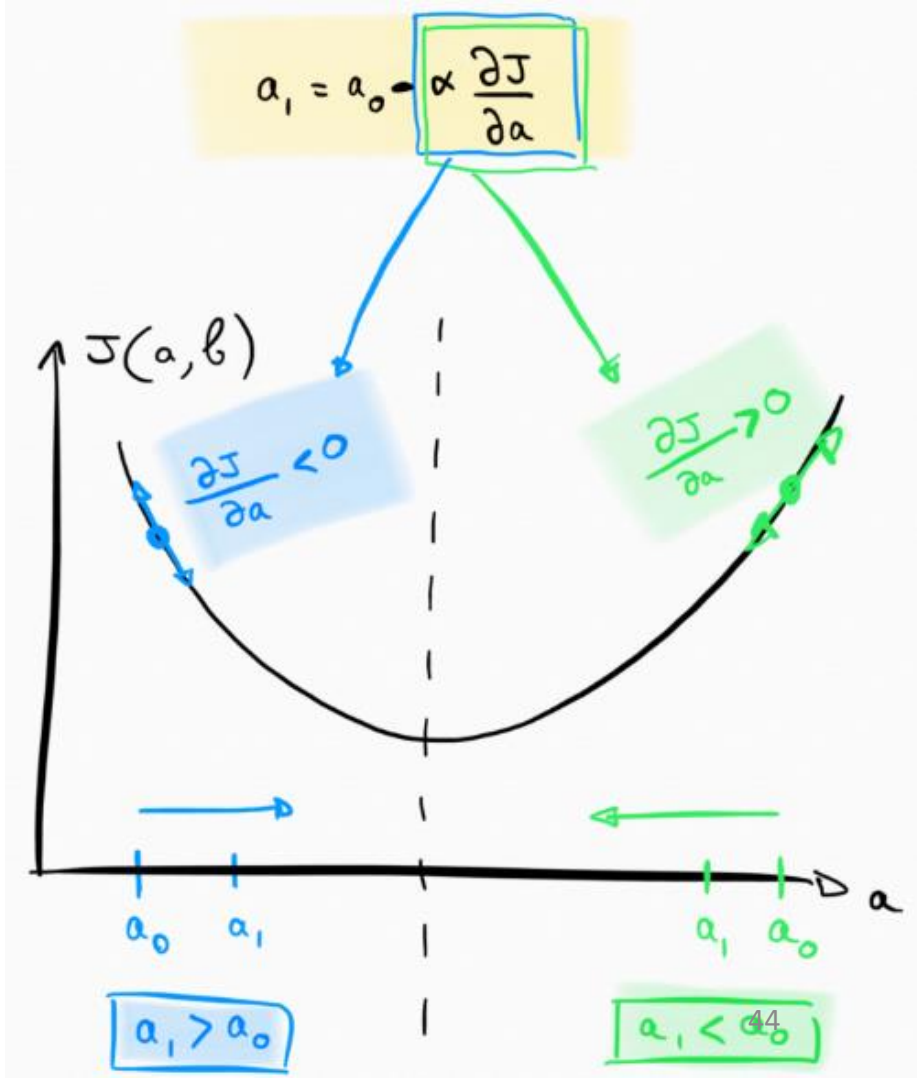


L'Algorithme d'apprentissage: Descente de gradient

❖Itération i=0

on prend une valeur aléatoire pour a_0
nous calculons la position a_1 en appliquant la formule :

$$a_1 = a_0 - \alpha * \frac{\partial J(a,b)}{\partial a}$$



En général

- Itération suivante
- $a_{k+1} = a_k - \alpha * \frac{\partial J(a,b)}{\partial a}$

L'Algorithme d'apprentissage: Descente de gradient

- **Descente de gradient** est un algorithme itératif à chaque itération on calcule paramètres (**a** et **b**) de notre modèle
- On progresse avec une distance α (***Learning Rate)*** dans la direction de la pente qui descend,

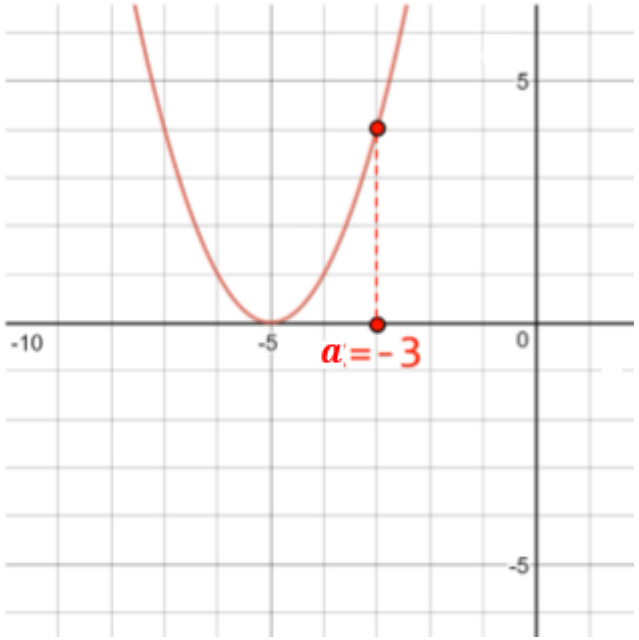
$$a_{k+1} = a_k - \alpha * \frac{\partial J(a,b)}{\partial a}$$

$$b_{k+1} = b_k - \alpha * \frac{\partial J(a,b)}{\partial b}$$

L'Algorithme d'apprentissage: Descente de gradient

Exemple :

- Considérons la fonction suivante $J(a) = (a+5)^2$
- Commençons par un point aléatoire **a=-3** et nous nous déplaçons dans la direction négative pour chercher le minimum
- Nous calculons la dérivée de la fonction
- $\frac{dJ}{da} = 2(a+5)$



Nous déplaçons dans la direction négative un pas $\alpha = 0,01$ (learning_rate)

L'Algorithme d'apprentissage: Descente de gradient

Exemple :

Itération 0: calculer a_0

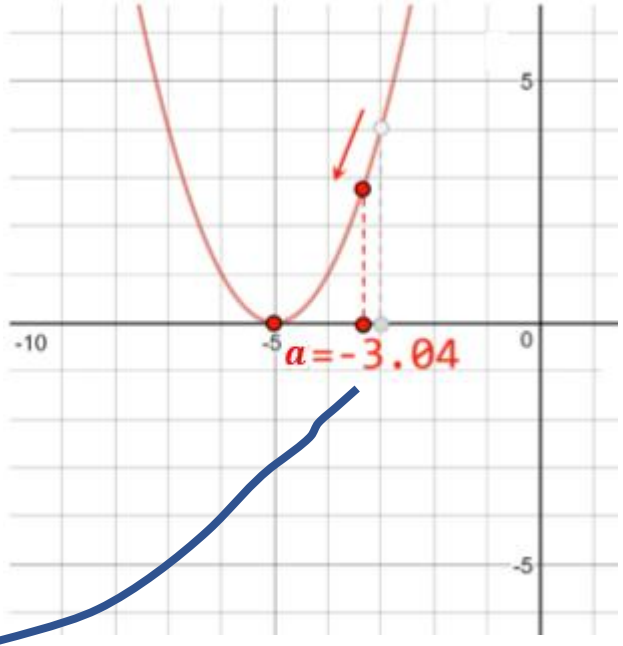
$a_0 = -3$, $\frac{dJ}{da} = 2(a+5)$ et learning_rate $\alpha = 0,01$

Itération 1: calculer a_1

$$a_1 = a_0 - \alpha * \frac{dJ}{da}$$

$$a_1 = a_0 - \alpha * 2(a_0 + 5)$$

$$a_1 = -3 - 0,01 * 2(-3+5) = -3;04$$



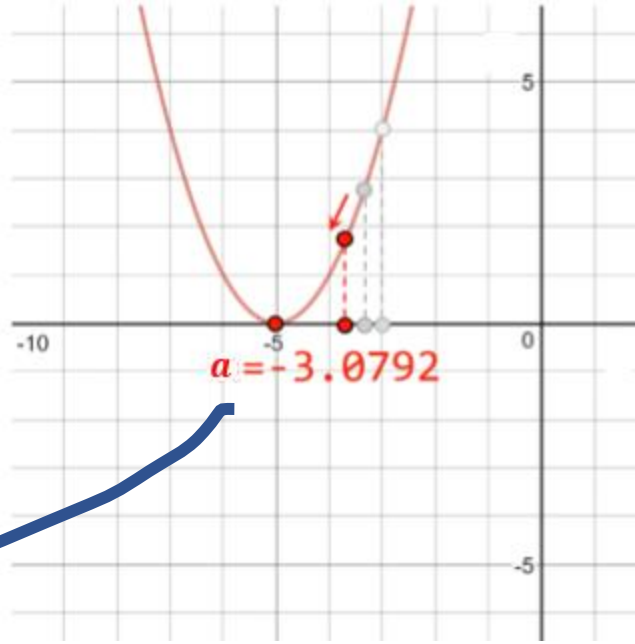
L'Algorithme d'apprentissage: Descente de gradient

Exemple :

Itération 2: calculer a_2

$$a_2 = a_1 - \alpha * 2(a_1 + 5)$$

$$a_2 = -3,04 - 0,01 * 2(-3,04 + 5) = \boxed{-3,0792}$$

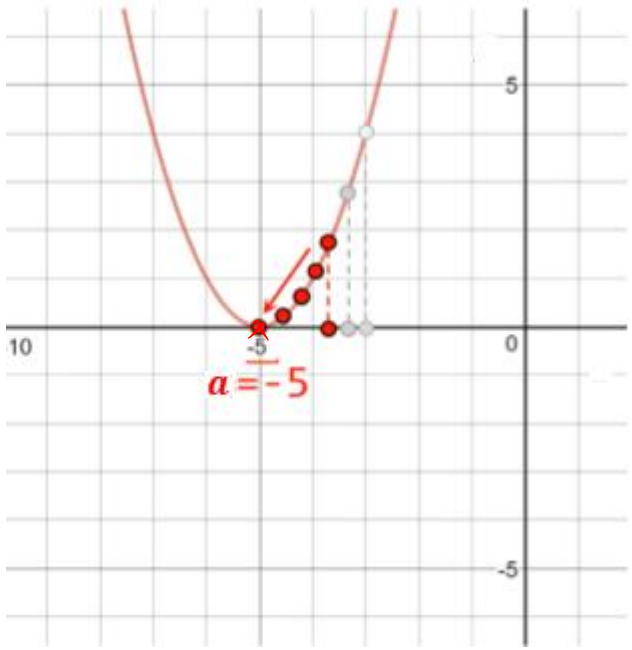


L'Algorithme d'apprentissage: Descente de gradient

Exemple :

Itération k: $a_k = -5$ **nous calculons** a_{k+1}

- $a_{k+1} = a_k - \alpha * 2(a_k + 5)$
- $a_{k+1} = -5 - 0,01 * 2(-5 + 5) = -5$



Exemples de Problèmes d’optimisation

Notion d’optimisation combinatoire

Exemples de Problèmes d’optimisation

- Le voyageur de commerce
- Le problème de sac à dos
- Routage dans les réseaux télécoms
- Affectation de fréquence en téléphonie
- Routage de véhicules

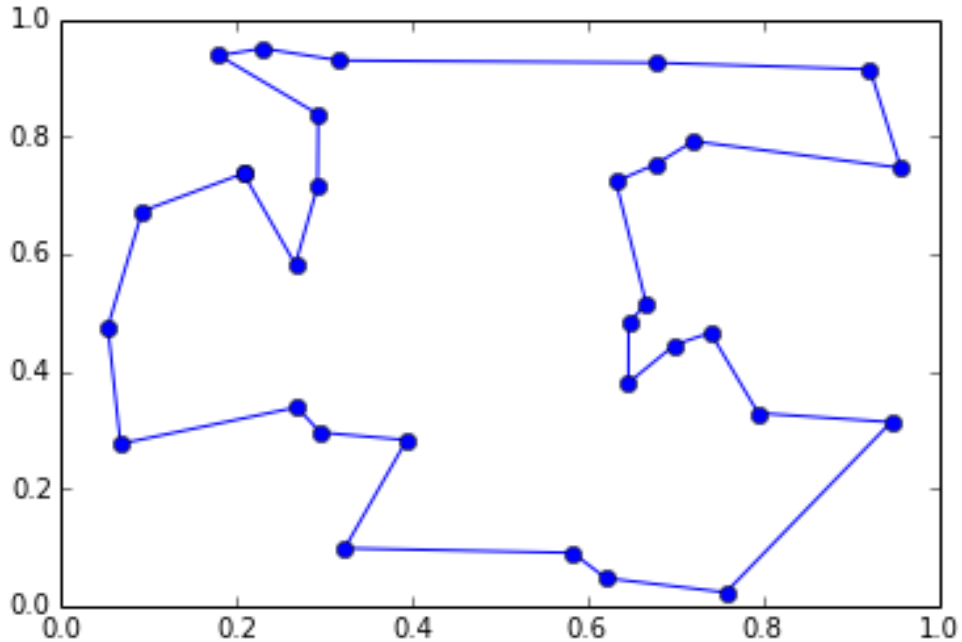
Exemples de Problèmes d'optimisation

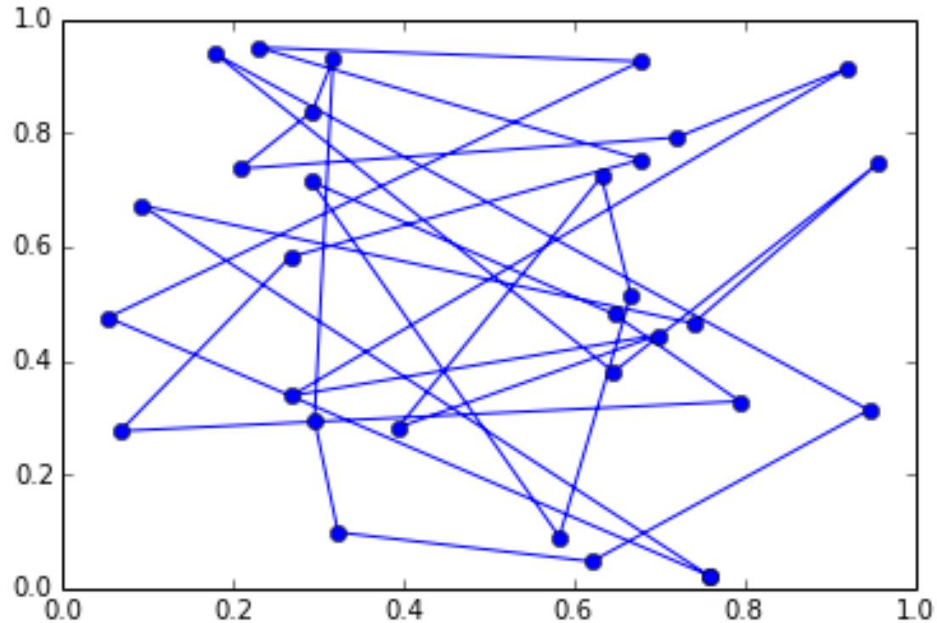
Problème de Voyageur du Commerce (PVC)

Traveling Salesman Problem (TSP)

- **Données :**
  - N villes,
  - Une matrice de distances **D = (d_{ij})**

- **Problème** : trouver un chemin passant une fois et une seule par chaque ville et minimisant la distance totale parcourue
- **S =N!** : ensemble des partitions (de solutions) d'un ensemble à N éléments.
- **s*** : le plus court chemin
- **f(x)**: la longueur 'distance' (ou coût) du chemin x.


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Exemples de Problèmes d'optimisation

PVC

- **Méthode exhaustive**:
- générer tous les trajets possibles, calculer leurs distances, choisir le trajet ayant la distance minimale.

N: le nombre des villes	Le nombre des chemins possibles
3	6
4	24
5	120
6	720
7	5040
8	40320
10	3628800
20	2432902008176640000
40	815915283247897734345611269596115894272000000000

Le nombre des chemins possibles = **N!**

Problème de complexité factorielle

Exemples de Problèmes d'optimisation

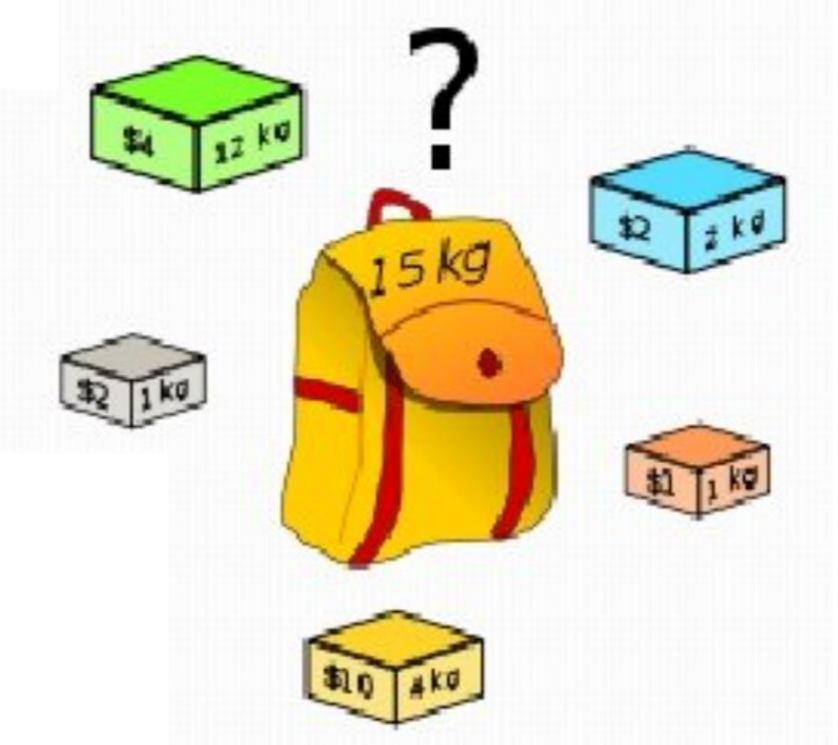
- **Problème de complexité exponentielle**

Taille des données	Le nombre des solutions possibles
1	**2**.71
2	**7**.38
3	**20**.08
4	**54**.59
10	**22026**.46
15	**3269017**.37
20	**485165195**.40
30	**10686474581524**
50	**5.1847055285871E+21**

Exemples de Problèmes d’optimisation

Problème de Sac à Dos

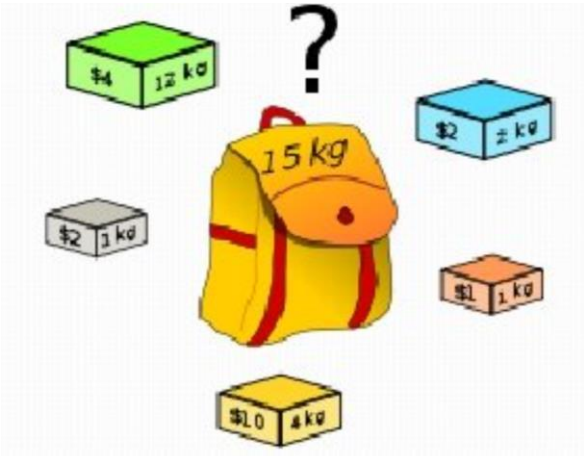
- Dans ce problème nous avons.
  - Un ensemble de N objets.
  - Chaque objet O a un poids et une valeur spécifiés.
  - Le sac à dos a une capacité (le poids total maximum).



Exemples de Problèmes d’optimisation

Problème de Sac à Dos

- Le problème de décision du sac à dos consiste à trouver un sous-ensemble des objets dont le poids total est au plus égal à la capacité du sac.
- Le problème d’optimisation pour le problème de sac à dos est de remplir le sac à dos avec comme objectif la maximisation de la bénéfice.



Problèmes d’optimisation

- Les problèmes **NP-complets d'optimisation combinatoire** sont caractérisés par une **complexité exponentielle ou factorielle**, par conséquent ; il est impossible d'énumérer toutes les solutions possibles car cela dépasse la capacité de calcul de n'importe quel ordinateur.
- Il est donc **très difficile** de trouver la solution optimale. Pour pallier à ces problèmes, les chercheurs ont introduit des méthodes approchées appelées **heuristiques / méta-heuristiques.**

Méthodes de trajectoire

- Les **méthodes de recherche locale** passent d'une solution à une autre dans l'espace des solutions candidates (l'espace de recherche) qu'on note **S**, jusqu'à ce qu'une solution considérée comme optimale soit trouvée ou que le temps imparti soit dépassé.

- La méthode de recherche locale la plus élémentaire est la **méthode de descente.**

Méthodes de trajectoire

La méthode de descente

- La **méthode de la descente** consiste à partir d'une solution **S** à choisir une solution **S'** dans un voisinage de **S**, telle que **S'** améliore la solution.
- La recherche s'arrête donc au premier **minimum** (ou **maximum**) **local** rencontré, c'est là son principal **défaut**.

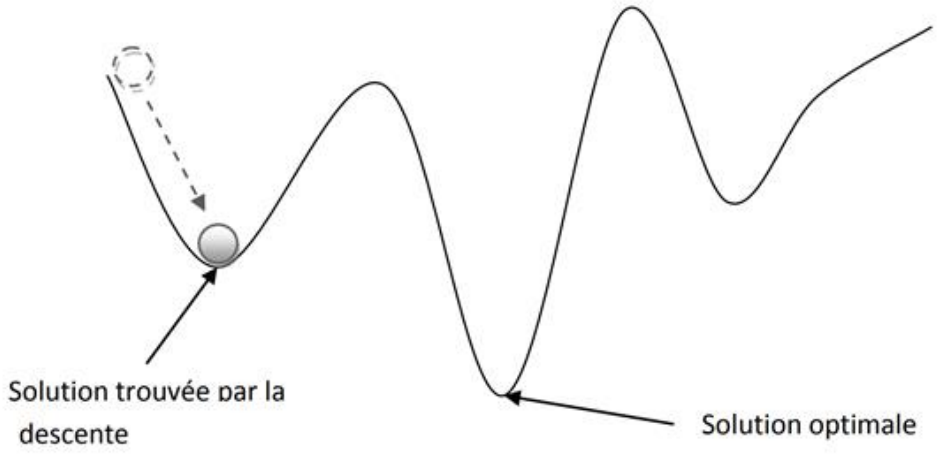


Figure : blocage d'une heuristique classique dans un minima local

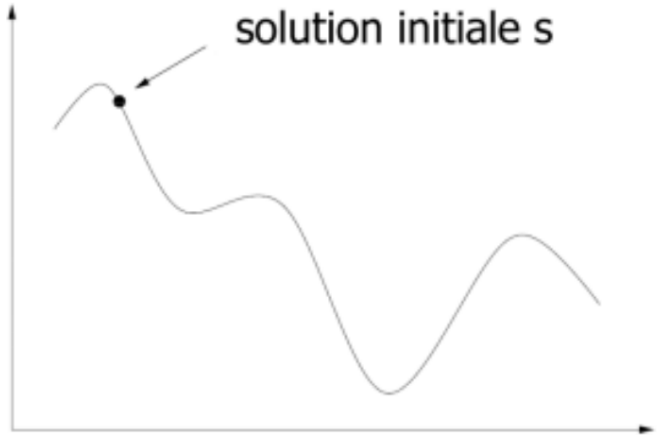
Méthodes de trajectoire

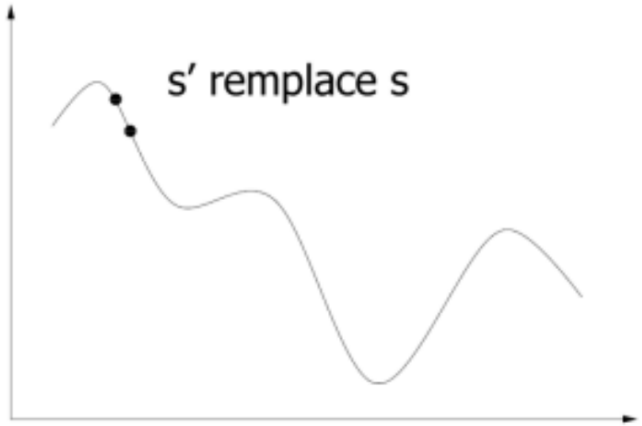
La méthode de descente

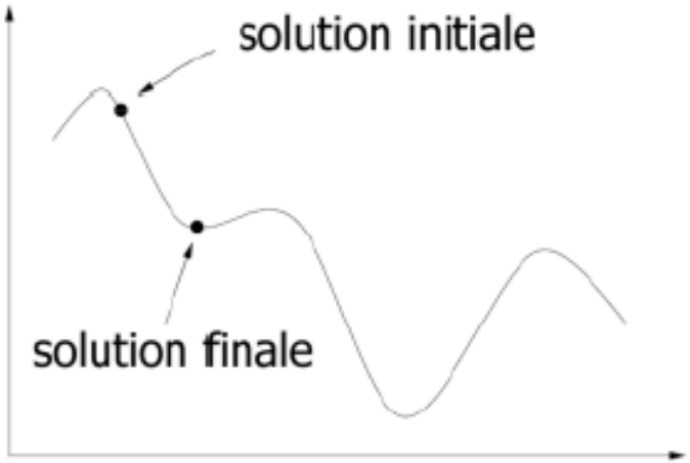
- Pour un problème de minimisation d'une fonction f , la méthode descente peut être décrite comme suit

Algorithme La méthode de descente

1. Solution initiale s ;
2. **Repeter** :
3. Choisir s' dans un voisinage $N(s)$ de s ;
4. Si $f(s') < f(s)$ alors $s := s'$;
5. **jusqu'à** ce que $f(s') \geq f(s)$, $\forall s' \in N(s)$.







Méthodes de trajectoire

La méthode de descente

- Inconvénient:

➢L’inconvénient majeur de la méthode de descente est son arrêt au premier minimum local rencontré.

- Avantage:

➢La simplicité de mise en œuvre

Méthodes de trajectoire

La méthode de descente

- Pour améliorer les résultats, on peut relancer plusieurs fois l'algorithme mais la performance de cette technique décroît rapidement.
- Ce qui a poussé les chercheurs à proposer de nouvelles méthodes générales (applicables à la plupart des problèmes d'optimisation) appelées **métaheuristiques**, dont la méthode du **recuit simulé** ; conçu pour rechercher un optimum global parmi plusieurs minimas (ou maximas) locaux.

Métaheuristiques

- Une recherche locale peut être piégée dans un minimum local.

- Face aux difficultés rencontrées par les **heuristiques** pour avoir une solution réalisable de bonne qualité pour des problèmes d'optimisation difficiles, les **métaheuristiques** ont fait leur apparition.

- Les **méta-heuristiques** peuvent échapper à ces minimums en construisant une suite de solution, mais dans laquelle la fonction objective peut temporairement augmenter.

Métaheuristiques

- Les algorithmes de **métaheuristique** sont plus complets et complexes qu'une simple **heuristique**, et permettent généralement **d'obtenir** une **solution de très bonne qualité** pour des problèmes issus des domaines de la recherche opérationnelle ou de l'ingénierie dont on ne connait pas de méthodes efficaces pour les traiter ou bien quand la résolution du problème **nécessite un temps élevé** ou une **grande mémoire de stockage.**

Métaheuristiques

- La plupart des **métaheuristiques utilisent des processus aléatoires et itératifs** comme moyens de rassembler de l'information, d'explorer l'espace de recherche et de faire face à des problèmes comme l'explosion combinatoire.
- Plusieurs d'entre elles sont souvent inspirées par des systèmes naturels dans de nombreux domaines tels que :
  - **La biologie:** **algorithmes évolutionnaires et génétiques**
  - **La physique:** **recuit simulé**
  - **L'éthologie:** **algorithmes de colonies de fourmis**

Métaheuristiques

- Les **métaheuristiques** peuvent être classées de nombreuses façons. On peut distinguer celles qui travaillent avec une **population de solutions** de celles qui ne manipulent qu'une **seule solution** à la fois.
- Les méthodes qui tentent **itérativement** d'améliorer une solution sont appelées **méthodes de recherche locale** ou **méthodes de trajectoire**.
- Ces méthodes construisent une **trajectoire** dans **l'espace des solutions** en tentant de se diriger vers des solutions optimales. Les exemples les plus connus de ces méthodes sont : La **recherche Tabou** et le **Recuit Simulé**.
- Les **algorithmes génétiques**, l'optimisation par **essaim de particules** et Les **algorithmes de colonies de fourmis** présentent les exemples les plus connus des méthodes qui **travaillent avec une population**.

Métaheuristiques

- Les **heuristiques** classiques ne sont pas très satisfaisantes pour résoudre les problèmes d'optimisation, car les solutions générées ne sont pas de bonne qualité. **L'intelligence artificielle** s'est donc tournée vers la nature pour créer de nouvelles méthodes: plus générales et plus efficaces. On peut dire que le recuit simulé est une bonne solution pour trouver des solutions acceptables à certains problèmes **NP difficile**, en particulier le problème du **voyageur de commerce**.

Recuit simulé

Recuit en physique - définition

- Le **recuit** d'une pièce métallique ou d'un matériau est un procédé correspondant à un cycle de chauffage. Celui-ci consiste en une étape de montée graduelle en température suivie d'un refroidissement contrôlé. Cette procédure, courante en sciences des matériaux, permet de modifier les caractéristiques physiques du métal ou du matériau étudié.

Recuit simulé

Historique

- La méthode de recuit simulé réalisées par **Metropolis et al. (1953)** pour **simuler** l'évolution de ce processus de **recuit physique (Metropolis53)**.
- Elle a été mise au point par trois chercheurs de la société IBM,. **Kirkpatrick**, **C.D. Gelatt** et **M.P. Vecchi en 1983** au Etats-Unis, et indépendamment par **V. Černy** en 1985 en Slovaquie.
- **Inspirer par la physique** statistique et les refroidissement des métaux.
- Cette métaheuristique est basée sur une technique utilisée depuis longtemps par les **métallurgistes** qui, pour obtenir un alliage sans défaut, faisant alterner les cycles de **réchauffage** (ou de **recuit**) et de **refroidissement lent** des métaux.

Recuit simulé

Historique

- L’utilisation pour la résolution des **problèmes d'optimisation combinatoire** est beaucoup plus récente. Le **recuit simulé** est la **première méta-heuristique** qui a été proposée. C’est une **méthode de recherche locale**, utilisant une stratégie pour **éviter les minima locaux**.

- La méthode du **recuit simulé**, conçu pour rechercher un **optimum global** parmi plusieurs minimums locaux.

Recuit simulé

Définition

- La méthode du **recuit simulé** (***Simulated Annealing)*** s'inspire du processus du recuit physique. Ce processus utilisé en métallurgie pour améliorer la qualité d'un solide cherche un état d'énergie minimale qui correspond à une structure stable du solide.
- **En partant d'une haute température** à laquelle le solide est devenu liquide, la phase de refroidissement conduit la matière liquide à retrouver sa forme solide par une **diminution progressive de la température**. Chaque température est maintenue jusqu'à ce que la matière trouve un équilibre thermodynamique. Quand la température tend vers zéro, seules les transitions d'un état à un état d'énergie plus faible sont possibles.

le but du recuit simulé

- Explorer l’espace d’état de manière aléatoire afin **d’éviter les minimums locaux**.
- Diminuer **progressivement** la température **T** pour stabiliser l’algorithme sur un minimum global.
  - Si le refroidissement est trop rapide, il y a un **risque de rester bloqué dans un minimum local** (configuration sous-optimale).

Principe du recuit simulé

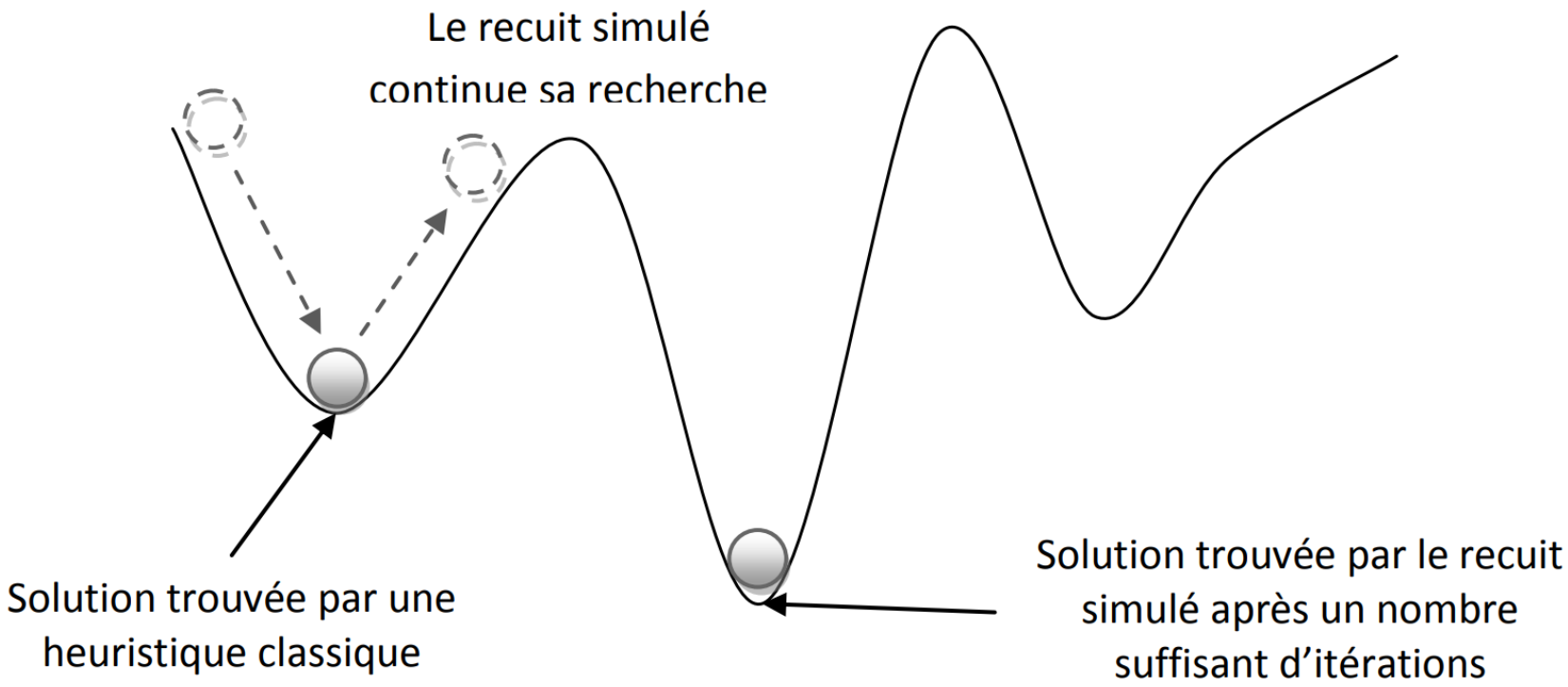
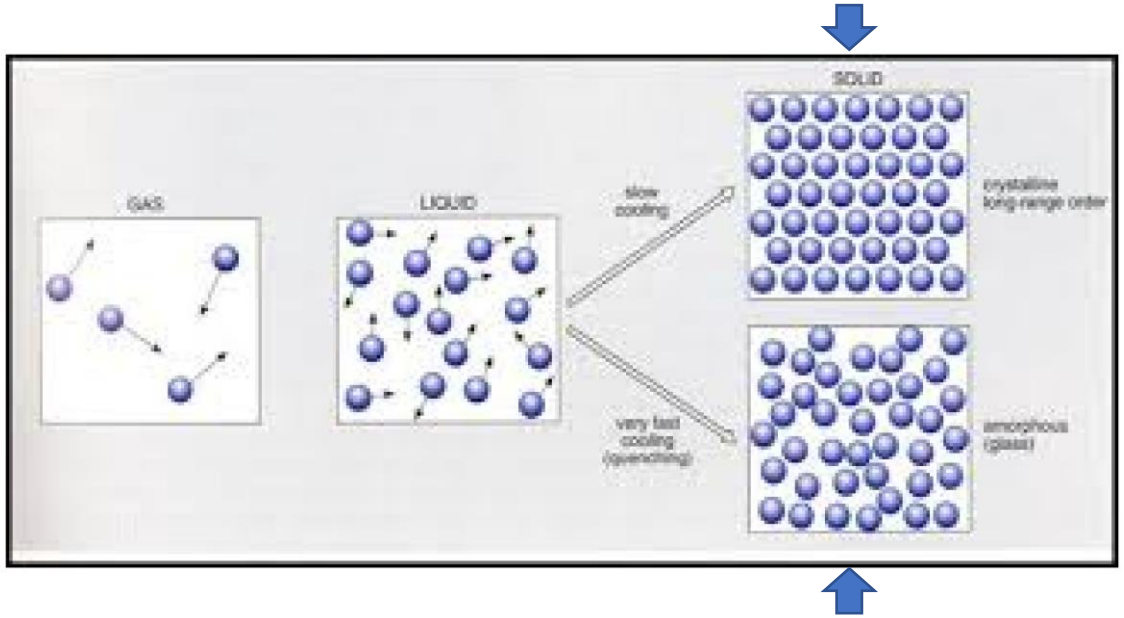


Figure : comparaison entre le recuit simulé et une heuristique classique

Principe du recuit simulé

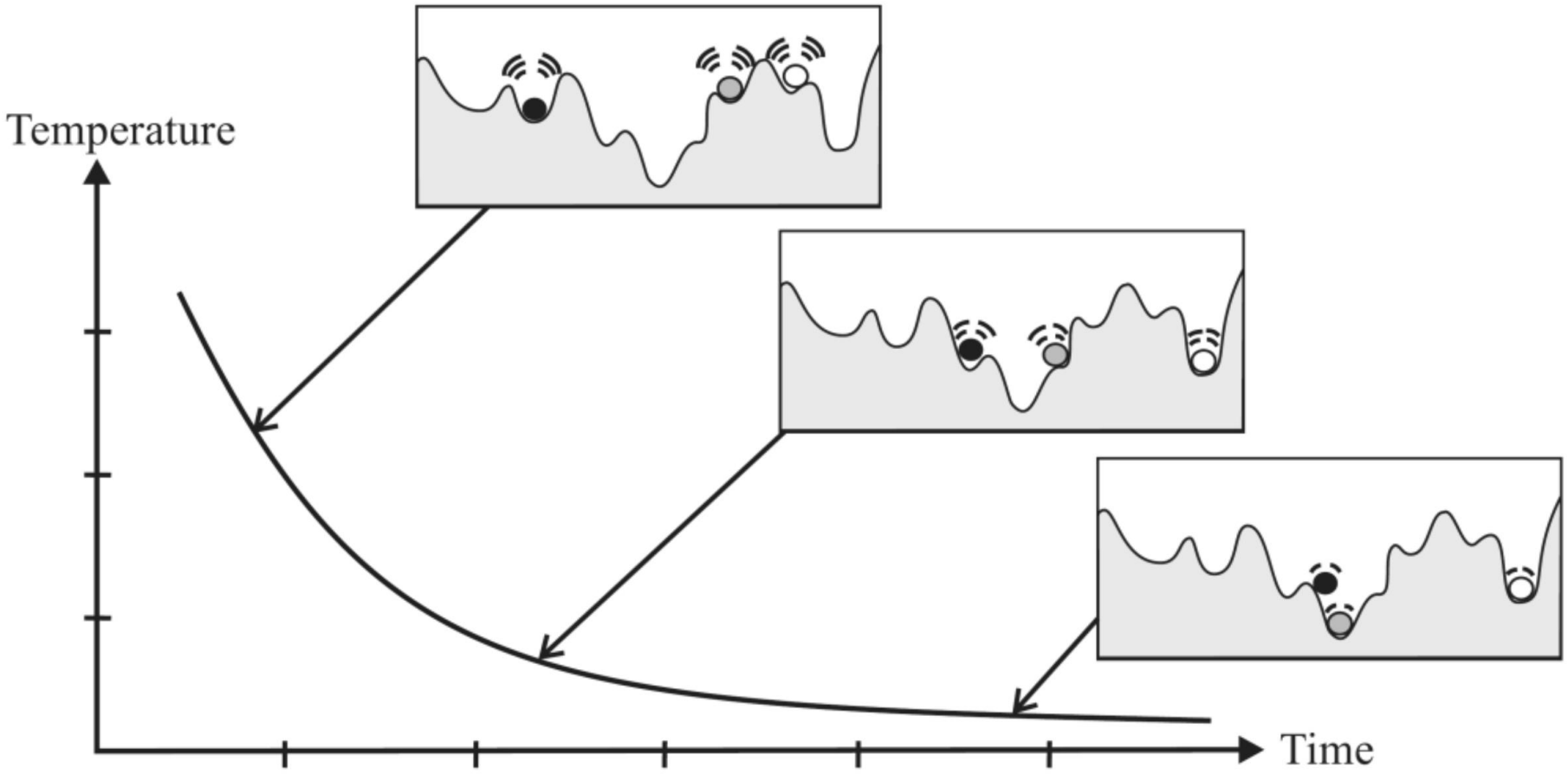
- **Inspirée de la physique:** On **chauffe** très fort le métal puis on le **refroidit lentement** pour obtenir un alliage sans défaut- ou dans la fabrication du verre.
- **Idée**: au départ **température haute**, état instable. Il y a assez d'énergie pour changer de configuration; on "laisse les choses" s'organiser petit à petit et au fur et à mesure, on **refroidit**.

refroidissement trop rapide



refroidissement lent

Principe du recuit simulé



Principe du recuit simulé

- Le principe du **recuit simulé** est de parcourir de manière **itérative** l'espace des solutions. On part avec une **solution notée s_0 initialement** générée de manière ***aléatoire*** dont correspond une énergie initiale **E_0**, et une **température initiale T_0 généralement élevée**.
- A chaque **itération** de l'algorithme, un changement élémentaire est effectué sur la solution, cette modification fait varier l'énergie du système **ΔE**.
- **Si cette variation est négative** (la nouvelle solution améliore la fonction objective, et permet de diminuer l'énergie du système), elle est **acceptée**.
- **Si la solution trouvée est moins bonne** que la précédente alors elle sera **acceptée avec une probabilité P** calculée suivant le critère de **Metropolis** suivant :

$$p = e^{-\frac{\Delta E}{T}}$$

Principe du recuit simulé

- **L’algorithme de Metropolis:**
- L’algorithme de Metropolis (1953) simule l’évolution du système vers l’équilibre. L’équilibre thermodynamique correspond à l’état d’énergie minimale $\min \boldsymbol{E(x)}$
- Dans l'algorithme de Metropolis, on part d'une configuration donnée, et on lui fait subir une modification aléatoire. Si cette modification fait diminuer la fonction objectif (ou énergie du système), elle est directement acceptée ; Sinon, elle n’est pas acceptée qu’avec une probabilité égale à $\boldsymbol{e^{-\frac{\Delta E}{T}}}$, cette règle est appelé critère de **Metropolis**.

Principe du recuit simulé

L'Algorithme de Recuit Simulé

Le recuit simulé applique **itérativement** l'algorithme de **Metropolis**, pour engendrer une séquence de configurations qui tendent vers l'équilibre thermodynamique :

1. Choisir une température de départ **T** et une solution initiale $\mathbf{s = s_0}$
2. Générer une solution **aléatoire** dans le voisinage de la solution actuelle. **s→s'**, $s' \in V(s)$
3. On calcule la variation de coût $\Delta f = f(s') - f(s_0)$
4. Si $\Delta f \leq 0$, le coût diminue et on effectue la transformation améliorante $s := s'$
5. Si $\Delta f > 0$, le coût remonte. On calcule une probabilité d'acceptation $p = e^{-\frac{\Delta f}{T}}$, puis on tire au sort β dans $[0,1]$. Si $\beta \leq p$, la transformation est déclarée **acceptée**, bien qu'elle dégrade le coût, et on fait $s := s'$. Sinon, la transformation est **rejetée** : on garde s_0 pour l'itération suivante.
6. répéter 2 et 3 jusqu'a ce que l'équilibre statistique soit atteint ;
7. décroitre la température et répéter jusqu'a ce que le système soit gelé.

Principe du recuit simulé

❑L’état initiale de l’algorithme

- La **solution initiale** s_0 peut être **prise au hasard** dans l'espace des solutions possibles.

- À cette solution correspond une énergie initiale $E=E_0$. Cette énergie est calculée en fonction du critère que l'on cherche à optimiser.

- Une température initiale $T=T_0$ **élevée** est également choisie.

Principe du recuit simulé

❑ Paramètre de température

- Le paramètre **T** (température) est un réel positif.
- La température permet de contrôler l'acceptation des dégradations :
  - Si **T** est **grand**, les dégradations sont **acceptées avec une probabilité plus grande**.
  - A la limite, quand **T** **tend vers l'infini**, tout voisin est **systématiquement accepté**.
  - Inversement, pour **T=0**, une dégradation n'est **jamais acceptée**.
- La température varie au cours de la recherche : T est élevée au début, puis diminue et finit par tendre vers 0.

Principe du recuit simulé

- Le choix de la température initiale dépend de la qualité de la solution de départ.
- Si cette solution est choisie aléatoirement, il faut prendre une température relativement élevée. On utilise souvent la règle suivante :

$$T_{k+1} = T_k * e^{(-\frac{a}{\tau})}$$

- où τ un paramètre assez grand qui exprime la diminution lente de la température de l'itération k à k + 1.

```
# diminution de la température
t1=10
k=0
print('t1=',t1)
for i in range(50):
    tau=1e4

    t2=t1*np.exp(k/-tau)
    k+=1
    print('t2=',t2)
```

Principe du recuit simulé

Convergence

- L'acceptation de solutions moins bonnes permet d'explorer l'espace des solutions.
- Le système peut s'extraire d'un minimum local si le nombre d'essais est suffisant.
- La convergence vers un minimum global nécessite :
  - une **température** initiale **élevée** → pour autoriser l'accès à tous les états
  - une **décroissance** de température **lente** → pour échapper au minima locaux
  - un nombre d'essais **élevé**

Principe du recuit simulé

- ***Avantages***

– La méthode du recuit simulé a l'avantage d'être souple vis-à-vis des évolutions du problème et facile à implémenter,

– Contrairement aux méthodes de descente, SA évite le piège des optima locaux,

– Excellents résultats pour un nombre de problèmes complexes.

Principe du recuit simulé

- **Inconvénients**

– Nombreux tests nécessaires pour trouver les bons paramètres,

– Définir les voisinages permettant un calcul efficace de ΔE

Recuit simulé appliqué pour le PVC

- La méthode du **Recuit Simulé** permis de résoudre des problèmes très complexes du type « **Voyageur de Commerce (PVC)** » où les méthodes déterministes sont rapidement piégées dans des minimums locaux. Le but est alors de trouver le circuit hamiltonien de coût minimal dans un graphe. L'énergie représentera la distance totale à parcourir, et un état du système représentera le chemin entre les villes.
- L'algorithme va donc tenter de minimiser la longueur totale du chemin, en modifiant l'ordre des villes à parcourir.

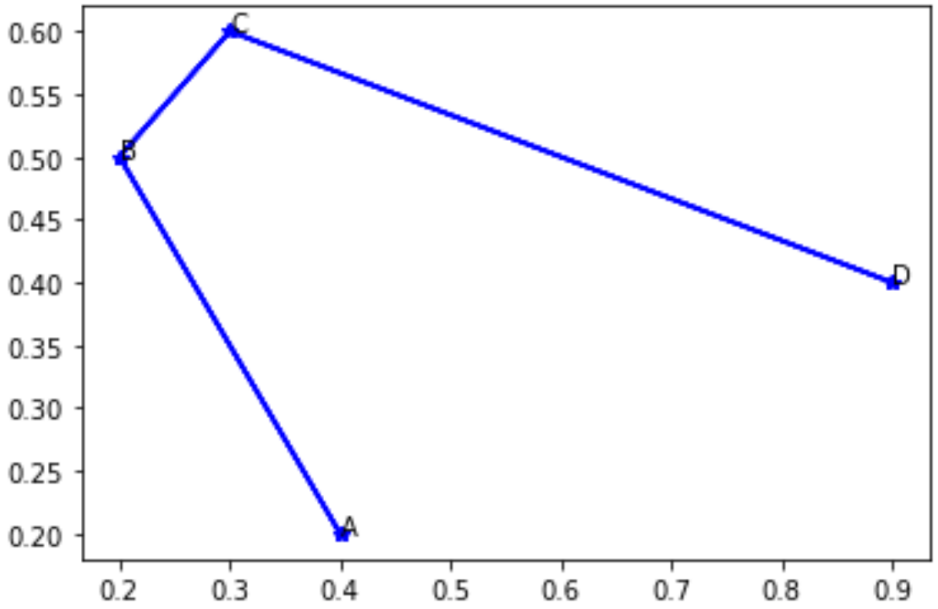
```
#génération aléatoire des villes
import numpy as np
import matplotlib.pyplot as plt
N=4
chemin=np.arange(N)
print('Chemin=', chemin)
x= np.array([0.4, 0.2, 0.3, 0.9])
y= np.array([0.2, 0.5, 0.6, 0.4])

print('X_chemin', x[chemin])
print('Y_chemin', y[chemin])
annotations=["A","B","C","D"]
for i, label in enumerate(annotations):
    plt.annotate(label, (x[i], y[i]))
    plt.plot(x,y, marker='*', color='b')
```

La solution la plus simple est de parcourir les villes dans l'ordre :

Chemin	0	1	2	3

	x_A	x_B	x_C	x_D
x	0,4	0,2	0,3	0,9

	y_A	y_B	y_C	y_D
y	0,2	0,5	0,6	0,4



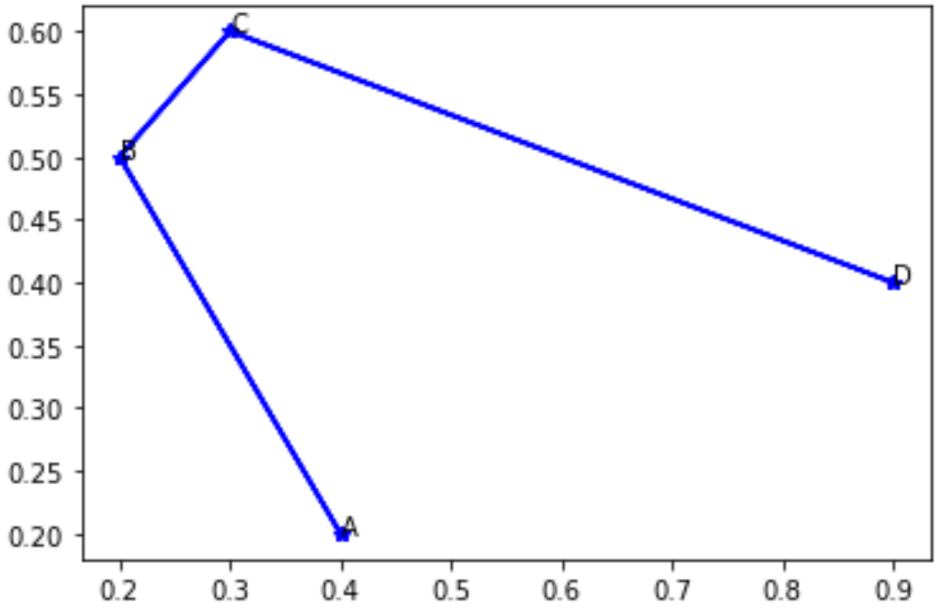
#La fonction objective égale à la somme des distances entre les villes

$$\mathbf{d}(A,B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

$$\mathbf{d}(B,C) = \sqrt{(x_B - x_C)^2 + (y_B - y_C)^2}$$

$$\mathbf{d}(C,D) = \sqrt{(x_C - x_D)^2 + (y_C - y_D)^2}$$

$$\mathbf{d}(D,A) = \sqrt{(x_C - x_D)^2 + (y_C - y_D)^2}$$



$$\textbf{Energie} = d_T = \mathbf{d}(A,B) + \mathbf{d}(B,C) + \mathbf{d}(C,D) + \mathbf{d}(D,A)$$

Calcul de la fonction objective - Etape 1:

numpy.column_stack() La fonction est utilisée pour empiler des tableaux 1-D sous forme de colonnes dans un tableau 2-D.

La fonction **numpy.roll()** utilisée pour dérouler les éléments du tableau le long d'un axe spécifié. (axis=0: le long des lignes). Fondamentalement, ce qui se passe, c'est que les éléments du tableau d'entrée sont décalés. L'élément de la première position est roulé à la dernière position

Chemin	A	B	C	D
	0	1	2	3

	x_A	x_B	x_C	x_D
x	0,4	0,2	0,3	0,9

	y_A	y_B	y_C	y_D
y	0,2	0,5	0,6	0,4

xy=np.column_stack((x,y))

	x	y
A	0,4	0,2
B	0,2	0,5
C	0,3	0,6
D	0,9	0,4

XY=np.roll(xy,-1,axis=0)

	X	Y
B	0,2	0,5
C	0,3	0,6
D	0,9	0,4
A	0,4	0,2

Calcul de la fonction objective - Etape 2:

Soustraction de xy-XY

	x	y
A	0,4	0,2
B	0,2	0,5
C	0,3	0,6
D	0,9	0,4

−

	X	Y
B	0,2	0,5
C	0,3	0,6
D	0,9	0,4
A	0,4	0,2

=

$x_A - X_B$ $y_A - Y_B$

xy-XY=

0,2	-0,3
-0,1	0,1
-0,6	0,2
0,5	0,2

Le carré de (xy-XY) :

$(x_A - X_B)^2$ $(y_A - Y_B)^2$

(xy-XY)**2=

x	y
0,04	0,09
0,01	0,01
0,36	0,04
0,25	0,04

Calcul de la fonction objective - Etape 3:

$(x_A - X_B)^2$

$(y_A - Y_B)^2$

(xy-XY)**2

x	y
0,04	0,09
0,01	0,01
0,36	0,04
0,25	0,04

np.sum((xy-XY)**2,axis=1)

0,13
0,02
0,4
0,29

$(x_A - X_B)^2 + (y_A - Y_B)^2$

p=np.sqrt(np.sum((xy-XY)**2,axis=1))

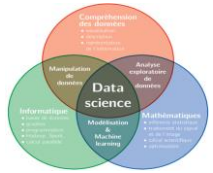
$d(A,B)$	0,360
$d(B,C)$	0,141
$d(C,D)$	0,632
$d(D,A)$	0,538

$\sqrt{(x_A - X_B)^2 + (y_A - Y_B)^2}$

Energie=d_{totale}=np.sum(p) 1.672

Générer les voisins (les chemins)

```
#générer les permutations
import numpy as np
import itertools as it
import math

N=3
chemin=np.arange(N)
print('Chemin=', chemin)0
M=list(it.permutations(chemin))
M=np.array(M)
print('la taille du tableau= ',len(M))
print(N,'!= ',math.factorial(N) )
print(M)
```



La **complexité est factorielle**, donc il est impossible d'énumérer toutes les solutions possibles car cela dépasse la capacité de calcul de n'importe quel ordinateur.

Il est donc **très difficile** de trouver la solution optimale. Pour pallier à ces problèmes, les chercheurs ont introduit des méthodes approchées appelées **heuristiques / méta-heuristiques.**

Générer les voisins (les chemins)

- La fonction **randint()** est utilisée pour générer des entiers aléatoires entre une plage spécifiée.
- Pour générer une liste de nombres aléatoires avec cette fonction, nous pouvons utiliser la méthode de compréhension de liste avec la boucle for

```
import random
x = random.randint(0,10)
print(x)
```

```
import random
x = [random.randint(0, 9) for p in range(0, 10)]
print(x)
```

Résultat:

[6, 2, 5, 5, 9, 3, 3, 8, 8, 7]

Problème!! -- générer un chemin contant même ville plusieurs fois

Problème de PVC : trouver un chemin passant une fois et une seule par chaque ville et minimisant la distance totale parcourue

Les algorithmes génétiques

Introduction

- Les algorithmes génétiques sont des algorithmes d'optimisation s'appuyant sur des techniques dérivées de la génétique et de l'évolution naturelle. Leur but est d'obtenir une solution approchée à un problème d'optimisation lorsqu'il n'existe pas de méthode exacte pour le résoudre en un temps raisonnable.
- Les algorithmes génétiques utilisent la notion de sélection naturelle, et l'appliquent à une population de solutions potentielles au problème donné, Où la population de solutions potentielles évolue au cours du temps, en s'échangeant des caractéristiques subissant la loi de l'évolution naturelle qui élimine les mauvais individus selon un critère bien déterminé ; ils ont commencé en 1970 avec J. Holland [24], et sont développés par la suite, par ses étudiants et ses collègues en s'inspirant de la théorie de l'évolution par la sélection naturelle de Darwin [25].

Les algorithmes génétiques

Un AG est définit, par la donnée des quatre éléments de base suivants :

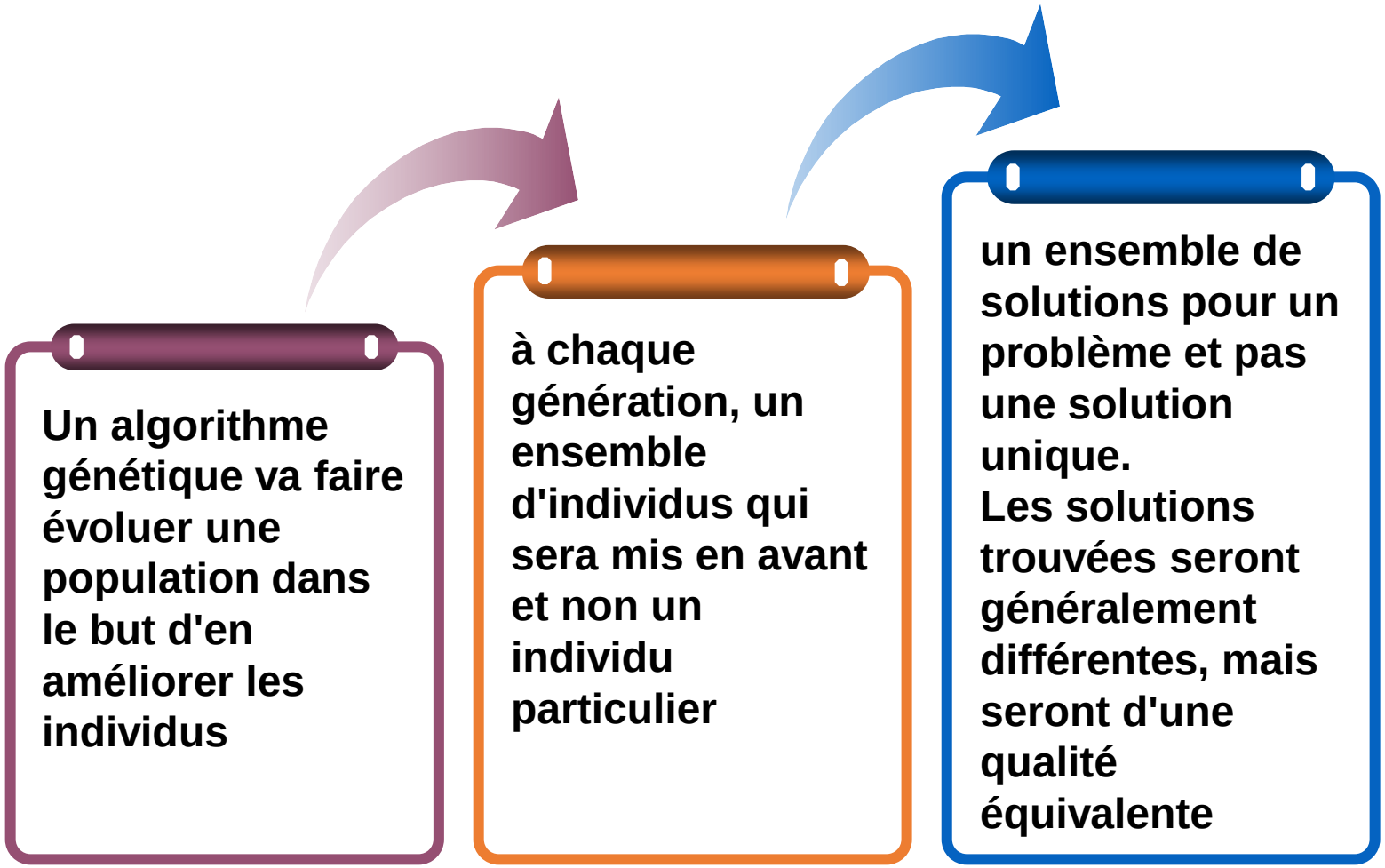
- **Individu/chromosome/séquence** : une solution potentielle du problème qui correspond à une valeur codée de la variable (ou des variables) en considération ;

- **Population** : un ensemble de chromosomes ou de points de l'espace de recherche (donc des valeurs codées des variables) ;

- **Environnement** : l'espace de recherche (caractérisé en termes de performance correspondant à chaque individu possible) ;

- **Fonction de performance** : la fonction - positive - que nous cherchons à maximiser/ minimiser car elle représente l'adaptation de l'individu à son environnement.

Les algorithmes génétiques



Les algorithmes génétiques

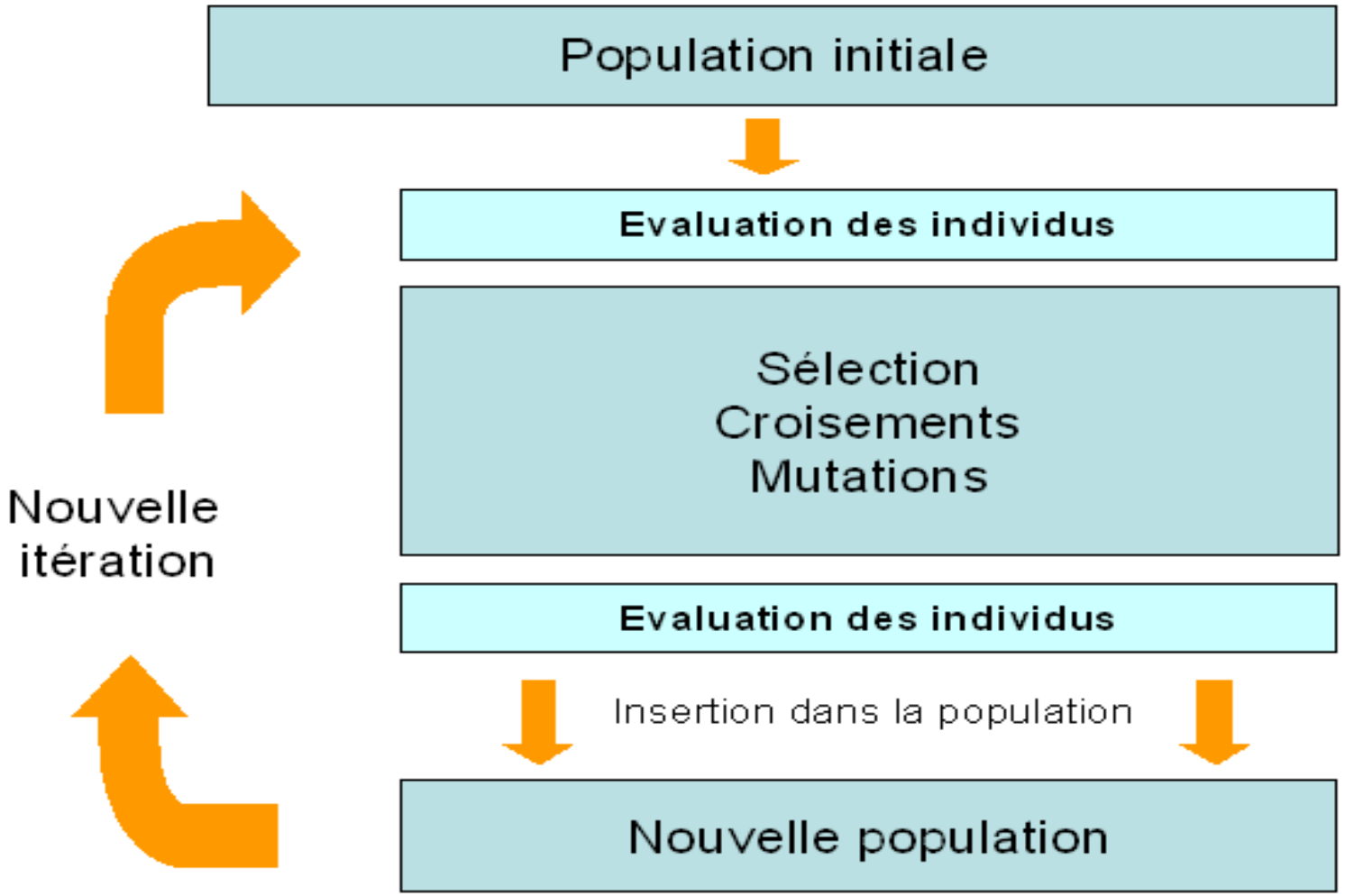
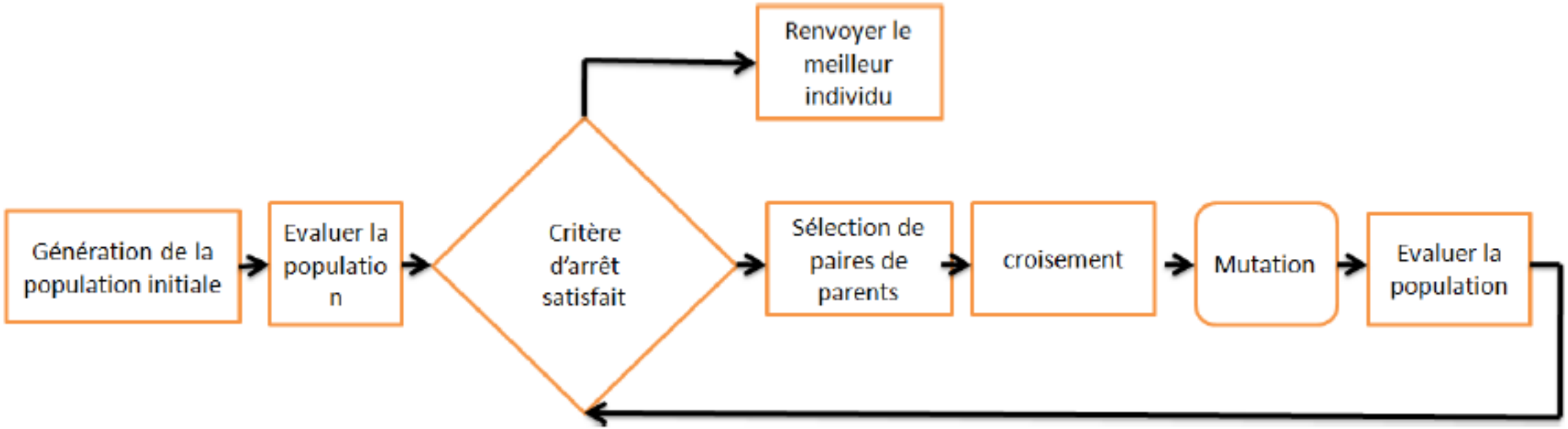
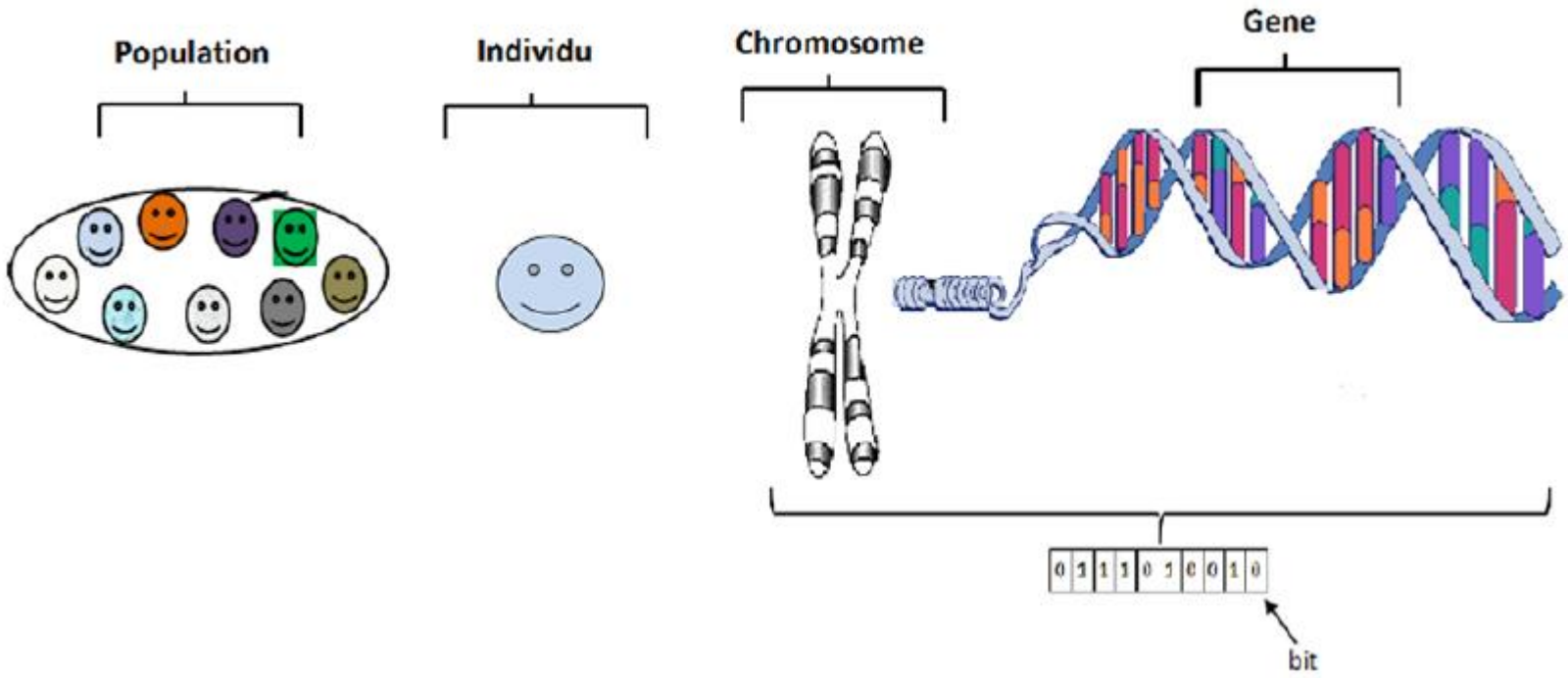


Schéma général d'un algorithme génétique

Les algorithmes génétiques



Les algorithmes génétiques



Codage des individus

- Avant de commencer la résolution d'un problème, il faut d'abord penser à coder les paramètres : un gène correspond à une variable d'optimisation, un ensemble de gène correspond à un chromosome, un individu à un ou plusieurs chromosomes et une population est un ensemble d'individus.
- Un chromosome est une suite de bits (formée des zéros et des uns), appelé aussi chaîne binaire. Une chaîne binaire de longueur n peut représenter 2^n états de variable.

Création de la population initiale

- L'AG démarre alors avec une population composée par un tirage aléatoire de Ni individus (chaînes, chromosomes) dans le codage retenu.
- Avec un codage binaire, selon la taille de la chaîne, on effectue alors pour chaque individu **n** tirages dans **{0, 1},** selon une loi de probabilité (la loi uniforme est couramment retenue).
- On obtient la nouvelle population en remplaçant les plus mauvais individus de la population courante par leurs fils, reproduits par la phase précédente, s'ils sont mieux qu'eux.

Création de la population initiale

- La nouvelle population est composée uniquement d'enfants. Nous laissons disparaître tous les individus de la population courante. L'inconvénient de cette approche est le risque de perdre le meilleur candidat.

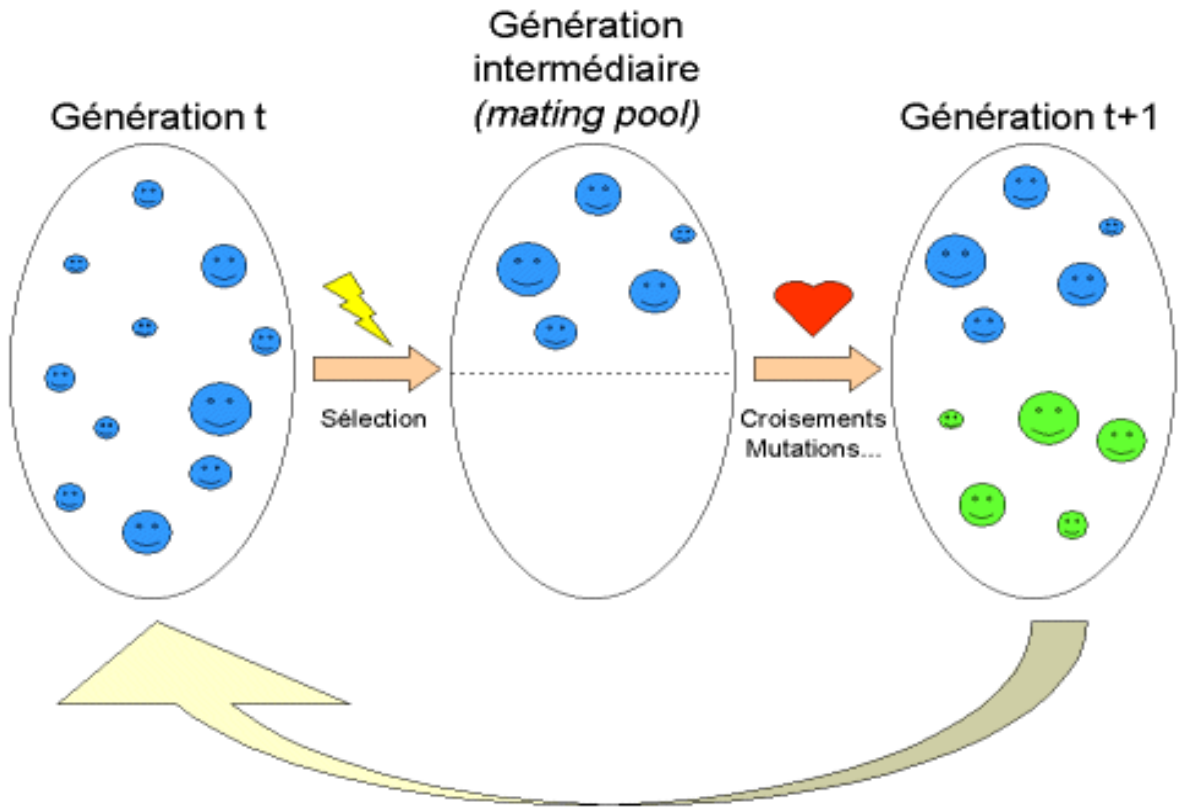
- Les enfants remplacent d'une façon régulière les individus les moins adaptés de la génération courante.

- La nouvelle génération est composée des **n** meilleurs individus de la population intermédiaire formée d'enfants et de parents.

Les opérations de reproduction

Lors de la création des individus nous utilisons trois opérateurs:

- La sélection
- Le croisement
- La mutation

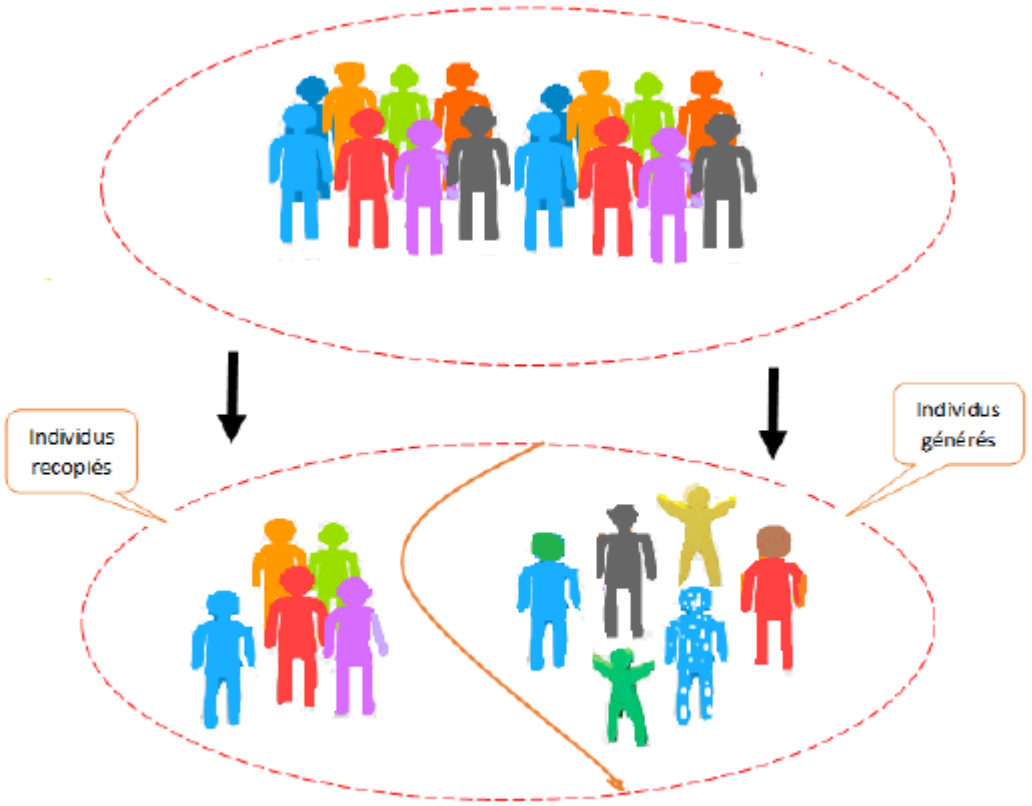


Opérateur de Sélection

Opérateur de sélection

- Le processus de sélection a pour but d'améliorer l'exploration locale. Ainsi les meilleurs individus trouvés seront copiés dans la population courante pour assurer le passage des meilleurs gènes à leurs enfants par héritage.

- Cet opérateur génétique garantit la non-disparition du meilleur individu trouvé au cours des différentes générations.



Sélection des individus.

Opérateur de sélection

Il existe plusieurs méthodes pour la reproduction.

On citera à titre d'exemple :

- La sélection par roulette ou proportionnelle
- La sélection par tournoi.
- La sélection par le rang.
- La sélection uniforme.

Sélection proportionnelle RWS (Roulette Wheel Selection)

- C'est le mode de sélection le plus classique introduit dès l'origine par Goldberg.
- Cette technique de sélection utilisée dans l'algorithme de base, consiste à faire un tirage aléatoire avec une roue de loterie où chaque individu se voit attribuer une place plus ou moins grande selon son adaptation.

Sélection par Roulette.

Sélection proportionnelle RWS (Roulette Wheel Selection)

- La méthode RWS exploite la métaphore d'une roulette de casino, qui comporterait autant de cases que d'individus dans la population. On attribue à chaque individu une section de surface, sur la roulette, proportionnelle à sa performance
- Plus la performance d'un individu est élevée par rapport à celle des autres individus de la même population, plus il a une chance d'être reproduit dans la population.
- Les individus ayant une grande performance relative ont donc plus de chance d'être sélectionnés.

Sélection par Tournoi

- C'est la méthode la plus facile à mettre en œuvre. Il s'agit de sélectionner aléatoirement deux ou plusieurs individus. Nous comparons leur valeur d'adaptation et l'individu ayant la plus forte valeur d'adaptation est sélectionné pour faire la reproduction.
- Si l'on choisit de nombreux participants au tournoi, cela encouragera les individus les plus forts car un individu moyen ou faible aura moins de chance d'être prise. Le tournoi entre seulement deux individus favorise moins les individus forts.

Sélection uniforme

- C'est une technique très simple qui consiste à sélectionner un individu aléatoirement de la population **P** . La probabilité p_i pour qu'un individu soit sélectionné est dénie par :

$$p_i = \frac{1}{N_i}$$

- Où N_i c'est la taille de la population.

Sélection par rang

- Cette méthode est très semblable au tirage à la roulette, sauf que les cases de la roulette ne sont plus proportionnelles à la fitness des individus, mais à leur rang dans la population. Le meilleur individu a le rang le plus élevé, le dernier a un rang de 1.

Sélection par élitiste

- Cette méthode de sélection permet de mettre en avant les meilleurs individus de la population. Ce sont donc les individus les plus prometteurs qui vont participer à l'amélioration de notre population. Cette méthode a l'avantage de permettre une convergence (plus) rapide des solutions, mais au détriment de la diversité des individus.
- On prend en effet le risque d'écarter des individus de piètre qualité, mais qui aurait pu apporter de quoi créer de très bonnes solutions dans les générations suivantes.
- Une autre possibilité relevant aussi du domaine de l'élitisme, pour s'assurer que les meilleurs individus feront effectivement partie de la prochaine génération, est tout simplement de les sauvegarder pour pouvoir les rajouter à coup sûr dans la population suivante.

Opérateur de croisement

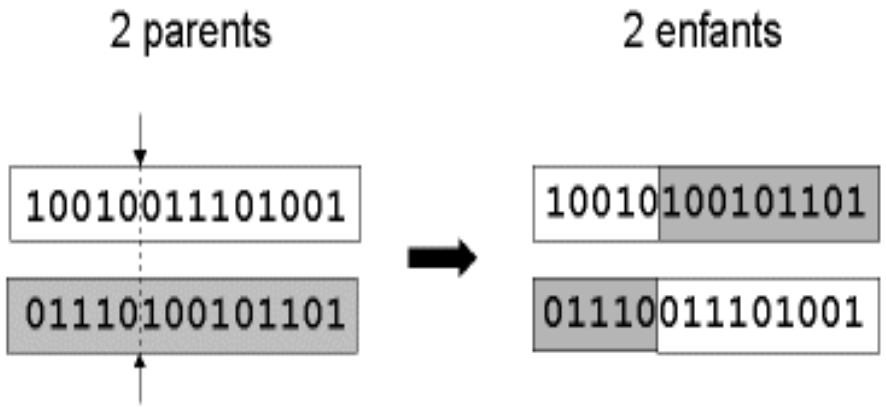
Opérateur de croisement

- L'opérateur de croisement permet la création de nouveaux individus selon un processus fort simple.
- Il permet donc l'échange d'information entre les chromosomes (individus) par le biais de leur combinaison.
- La population qui résulte de la sélection est divisée en deux sous-population de taille **N/2** et chaque couple formé par un membre provenant de chaque sous-population participe à un croisement avec une probabilité donnée (la probabilité de croisement noté $\mathbf{p_c}$, souvent supérieur à **60%**).

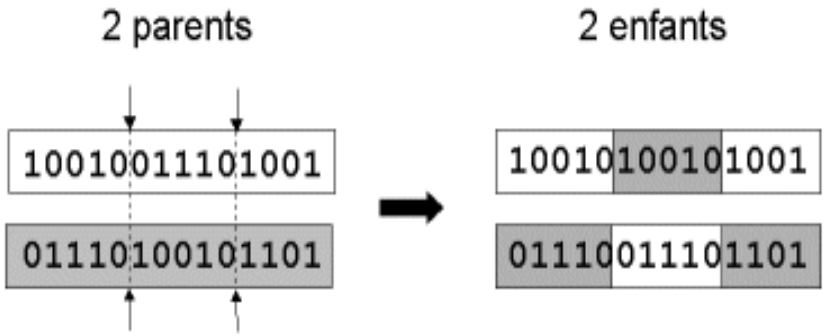
Opérateur de croisement

- Plusieurs méthodes de croisement sont développées et utilisées

➢A un point:



➢A deux points:



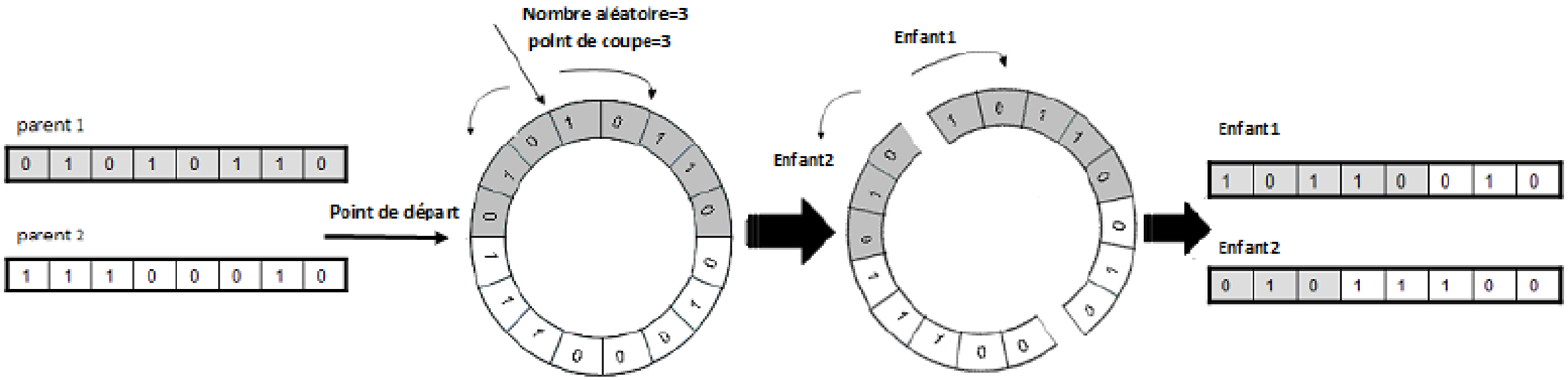
Opérateur de croisement

❑Croisement RC (Ring Crossover)

- C'est un type récent de croisement, se présente comme suit :
- Les deux parents, tels que parent 1 et parent 2 sont pris en compte pour le processus de croisement
- Les chromosomes des parents sont d'abord combinés avec une forme d'anneau.
- Puis, un point de coupe est choisi aléatoirement.
- Les enfants sont créés avec un nombre aléatoire généré en tout point de l'anneau en fonction de la longueur des deux chromosomes combinés. En ce qui concerne le point de coupe, tandis que l'un des enfants est créé dans la direction dans le sens horaire, l'autre est créée dans la direction de sens antihoraire.

Opérateur de croisement

❑Croisement RC (Ring Crossover)

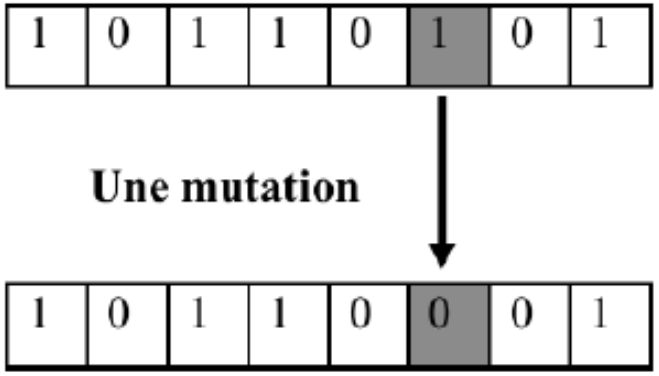


Exemple de croisement RC.

Opérateur de mutation

Opérateur de mutation

- La mutation de quelques gènes de l'individu lui permet d'avoir de nouveaux gènes qu'il n'a pas hérité de ses parents.
- Cet opérateur agit sur un individu. Il consiste à choisir d'une manière aléatoire un ou plusieurs gènes (bits) et à modifier leurs valeurs. Chaque bit d'un individu a une probabilité pour qu'il subisse une mutation, notée pi. La mutation se fait comme suit : Pour chaque bit de l'individu générer un réel aléatoire r, r 2 [0; 1], avec si r pm le bit sera inversé, sinon garder le bit.



Exemple de mutation.

- **Exercice :**

- Soit la **population1** initiale suivante de 6 individus

Ind1	**1**	**0**	**0**	**1**	**1**	**1**
Ind2	**0**	**1**	**0**	**0**	**1**	**0**
Ind3	**1**	**1**	**1**	**0**	**1**	**1**
Ind4	**1**	**0**	**1**	**1**	**0**	**0**
Ind5	**0**	**0**	**1**	**0**	**0**	**0**
Ind6	**0**	**1**	**0**	**1**	**1**	**1**

1. Former la **population2** de la **génération2** par les opérations de reproduction de l'algorithme génétique
2. Soit la fonction objective suivante $\boldsymbol{wi = \sum_{i=1}^{N} x_i \;\; avec\; x_i \neq 0}$ **(la fonction objective représente le poids de l'individus c-à-d le nombre de positions différente de zéro. Exemple le poids de l'individu 1 : w1=4)**

a. **En considérant qu'on traite un problème de minimisation trouver l'optimum pour chaque génération**