

# Algorithmique pour IA et Jeux

## Tronc commun informatique

Vincent Puyhaubert

PC\* Joffre

28 février 2023

# Table des matières

- 1 Introduction à la théorie des jeux
  - Jeux d'accessibilité sur un graphe
  - Algorithme min-max
- 2 Algorithmes d'apprentissage
  - Apprentissage supervisé
  - Apprentissage non supervisé

# Exemples

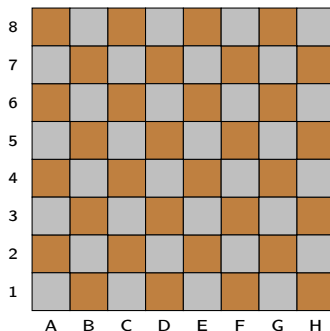
Dans tous les exemples qui suivent, deux joueurs jouent à tour de rôle.

- Ford-Boyard (jeu de Nim)
  - ▶ 20 batonnets sont disposés sur une table.
  - ▶ A son tour, un joueur retire entre 1 et 3 batonnets de la table.
  - ▶ Le perdant est celui qui retire le dernier batonnet.
- Jeu de Marienbad
  - ▶ On dispose de  $p$  tas d'allumettes
  - ▶ A son tour, chaque joueur peut retirer un nombre quelconque d'allumettes, mais dans un seul tas.
  - ▶ Le gagnant est celui qui retire la dernière allumette.
- Jeux à possibilité de match nul
  - ▶ Morpion
  - ▶ Puissance 4
  - ▶ Échecs

# Exemples

## • Jeu de Wythoff

- ▶ On joue sur un échiquier de taille quelconque, sur lequel se déplace une reine.
- ▶ Les seuls déplacements autorisés sont vers la gauche, vers le bas, ou en diagonale en bas à gauche, de longueur arbitraire.
- ▶ Le gagnant est celui qui place la reine dans le coin en bas à gauche.



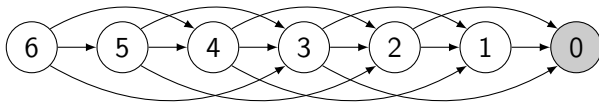
# Jeux à deux joueurs (programme officiel)

Le programme officiel limite l'étude aux jeux à deux joueurs avec les caractéristiques suivantes :

- Les deux joueurs jouent à tour de rôle (exit Shifumi)
- Les deux joueurs ont à chaque instant la même information sur l'état actuel du jeu (exit les jeux de cartes)
- Les coups jouables ne dépendent que de l'état actuel du jeu, ni du joueur (jeu impartial), ni des coups précédents (jeu sans mémoire).
- Dans une position donnée, une décision amène toujours à la même nouvelle position (jeu sans hasard).

# Modélisation

- Les jeux sont modélisés par un graphe  $(V, E)$  orienté fini (l'arène).
- Une position est un sommet du graphe
- Une arête  $(i, j)$  entre deux sommets  $i$  et  $j$  signifie que l'on peut jouer un coup depuis la position  $i$  pour arriver en position  $j$ .
- Une position est dite finale s'il elle n'est l'extrémité gauche d'aucune arête (on ne peut plus jouer de coup). Parmi elles, on distingue
  - ▶ les positions finales gagnantes (ou perdantes suivant le choix)
  - ▶ les positions finales de match nul
- Pour assurer qu'un jeu se finit, on ajoute généralement une hypothèse d'acyclicité au graphe.

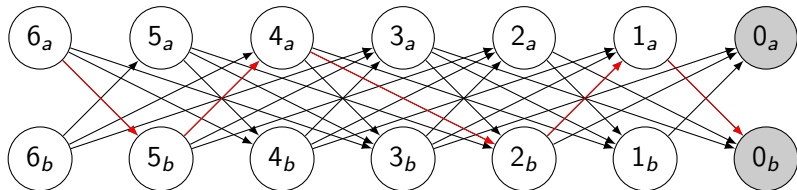


# Modélisation

- Il est préférable de dupliquer le graphe de manière à créer deux ensembles de sommets disjoints :
  - ▶ l'ensemble  $V_a$  sommets contrôlés par le joueur  $A$
  - ▶ l'ensemble  $V_b$  des sommets contrôlés par le joueur  $B$

Il y a trois types de positions finales : celles gagnantes pour  $A$ , celles gagnantes pour  $B$  et les positions de match nul.

- Chaque coup joué par un joueur amène sur un sommet contrôlé par l'autre joueur. Le graphe est dit bi-parti.



- Une partie d'origine  $s$  est un chemin dans le graphe bi-parti dont la première position est  $s$ . Elle est gagnée par le joueur  $J \in \{A, B\}$  si elle termine dans une position finale gagnante contrôlée par  $J$ .

# Stratégie et positions gagnantes

## Definition

- Une stratégie pour le joueur  $A$  est une application  $f : V'_a \subset V_a \longrightarrow V_b$  telle que pour tout  $s \in V'_a$ ,  $(s, f(s)) \in E$ .
- Une partie  $s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_n$  est dite jouée suivant  $f$  lorsque pour tout  $k \in \llbracket 0; n-1 \rrbracket$ , si  $s_k \in V'_a$ , alors  $s_{k+1} = f(s_k)$ .
- Une stratégie est dite gagnante pour le joueur  $A$  depuis une position  $s$  si toute partie jouée suivant cette stratégie et passant par  $s$  est gagnante pour ce joueur.

Intuitivement,

- Une stratégie est une fonction qui indique quel coup jouer depuis chaque élément d'un sous-ensemble de position.
- Elle est gagnante depuis  $s$  si elle permet de gagner à partir de la position  $s$  quels que soient les coups joués par son adversaire.

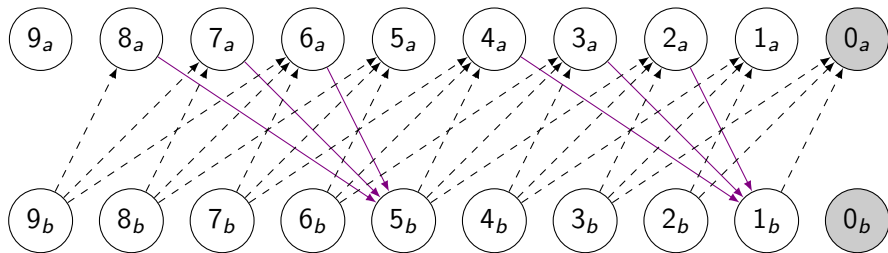
# Stratégie et positions gagnantes

## Exemple

Dans le jeu des batonnets de Ford-Boyard, on pose

$$g : k = (4q + r) \longrightarrow \begin{cases} 4q + 1 & \text{si } r \in \{2, 3, 4\} \\ \_ & \text{sinon} \end{cases}$$

Alors, l'application  $f : k_a \mapsto g(k)_b$  est une stratégie gagnante depuis tout sommet de  $S_a$  qui n'est pas de la forme  $(4q + 1)_a$ .



# Stratégie et positions gagnantes

## Definition

Une position  $s$  est dite gagnante pour le joueur  $J$  s'il existe une stratégie gagnante pour  $J$  depuis  $s$ .

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 8 | G | G | G | G | P | G | G | G |
| 7 | G | G | G | G | G | G | G | G |
| 6 | G | G | G | P | G | G | G | G |
| 5 | G | G | G | G | G | G | G | P |
| 4 | G | G | G | G | G | P | G | G |
| 3 | G | P | G | G | G | G | G | G |
| 2 | G | G | P | G | G | G | G | G |
| 1 | P | G | G | G | G | G | G | G |
|   | A | B | C | D | E | F | G | H |

# Stratégie et positions gagnantes

## Definition

Une position  $s$  est dite gagnante pour le joueur  $J$  s'il existe une stratégie gagnante pour  $J$  depuis  $s$ .

Plus généralement, si  $s$  est une position quelconque,

- Si  $s$  est contrôlée par  $A$  et qu'au moins un coup amène à une position gagnante pour  $A$ ,  $s$  est également gagnante pour  $A$ .
- Si elle est contrôlée par  $B$  et que **tous** les coups amènent à une position gagnante pour  $A$ ,  $s$  est gagnante pour  $A$ .

## Attracteurs

On se donne  $G = (V, E)$  un graphe bi-parti acyclique représentant un jeu à 2 joueurs. On note  $V = V_a \cup V_b$  où  $V_a$  (resp.  $V_b$ ) sont les positions contrôlées par  $A$ . On note  $W_a$  l'ensemble des positions finales gagnantes de  $A$ , puis on définit par récurrence :

- $\mathcal{A}_0 = W_a$
- $\forall j \in \mathbb{N}, \quad \mathcal{A}_{j+1} = \mathcal{A}_j \cup \{v \in V_a \mid \exists v' \in \mathcal{A}_j \mid (v, v') \in E\}$   
 $\cup \{v \in V_b \mid \forall v' \in \mathcal{A}_j \mid (v, v') \in E\}$

L'attracteur de  $W_a$  est défini par

$$\mathcal{A} = \bigcup_{j \in \mathbb{N}} \mathcal{A}_j$$

### Theorem

- *Le joueur  $A$  possède une stratégie gagnante depuis tout sommet de  $\mathcal{A}$ .*
- *Le joueur  $B$  possède une stratégie pour ne pas perdre (ie gagner ou faire match nul) depuis tout sommet de  ${}^c\mathcal{A}$ .*

# Attracteurs

Remarques :

- On peut bien entendu définir de la même manière l'attracteur  $\mathcal{B}$  du joueur  $B$ .
- Un jeu est dit déterminé si  $\mathcal{A} \cup \mathcal{B} = V$ , c'est-à-dire si depuis n'importe quelle position, l'un des deux joueurs a une stratégie gagnante.
- Si le graphe de jeu est acyclique, fini et sans position de match nul, alors il est déterminé.

# Attracteurs

Algorithme de calcul de l'attracteur :

- On va parcourir le graphe en remontant depuis les positions gagnantes  $W_a$  de  $A$  (parcours dans le graphe transposé).
- On initialise au préalable pour chaque sommet de  $V_2$  son degré sortant (nombre de coups jouables).
- Si on découvre un sommet de  $V_1$  depuis un élément de  $\mathcal{A}$ , on l'ajoute à  $\mathcal{A}$ .
- Si on découvre un sommet de  $V_2$  depuis un élément de  $\mathcal{A}$ , on diminue de 1 le nombre de coups jouables depuis ce sommet, et on ajoute le sommet à  $\mathcal{A}$  si ce nombre atteint 0.
- On relance le parcours à chaque ajout d'un sommet dans  $\mathcal{A}$  depuis le sommet en question.

# Attracteurs

## Algorithme de calcul de l'attracteur :

**Entrée :** Un graphe  $G = (V, E)$  donné par listes d'adjacence, une partition de  $V$  en deux ensembles  $V_1 \cup V_2$ ,  
un ensemble  $W$

**Sortie :** L'attracteur associé à  $W$

$\mathcal{A} \leftarrow \emptyset$ ;

Pour chaque sommet  $v$ , calculer  $n_v$ , le degré sortant de  $v$  dans  $G$ ;

Calculer  ${}^tG$ , graphe transposé de  $G$ ;

**Fonction** *parcours*( $u$ ) :

**si**  $u \notin \mathcal{A}$  **alors**

$\mathcal{A} \leftarrow \mathcal{A} \cup \{u\}$ ;

**pour** tout voisin  $v$  de  $u$  dans  ${}^tG$  **faire**

$n_v \leftarrow n_v - 1$ ;

**si**  $v \in V_1$  ou  $n_v = 0$  **alors**

                parcours( $v$ )

**pour** tout sommet  $u$  de  $W$  **faire**

    parcours( $u$ )

# Algorithme min-max

Limites de la méthode précédente :

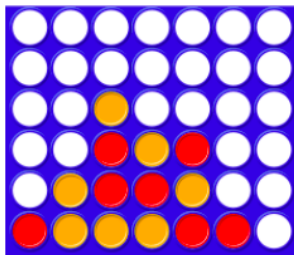
- Quand le graphe associé au jeu est très gros (jeu d'échec), il n'est pas possible de calculer l'attracteur sur une machine (pas assez de mémoire, ni de temps de calcul).
- Qui plus est, le graphe peut comporter des cycles ce qui n'assure pas la finitude d'une partie.

Nouvelle approche :

- On ne va travailler que sur une partie du graphe, en explorant uniquement les voies qui semblent le plus à même de faire gagner la partie.
- On a recours à une heuristique pour évaluer dans quelle mesure une position nous est favorable ou pas.

## Exemple d'heuristique (puissance 4)

On attribue à chaque case de la grille une valeur égale au nombre de rangées de 4 cases auxquelles elle appartient. Pour un joueur donné, on additionne les points des cases qu'il occupe, puis on soustrait celles occupées par son adversaire.

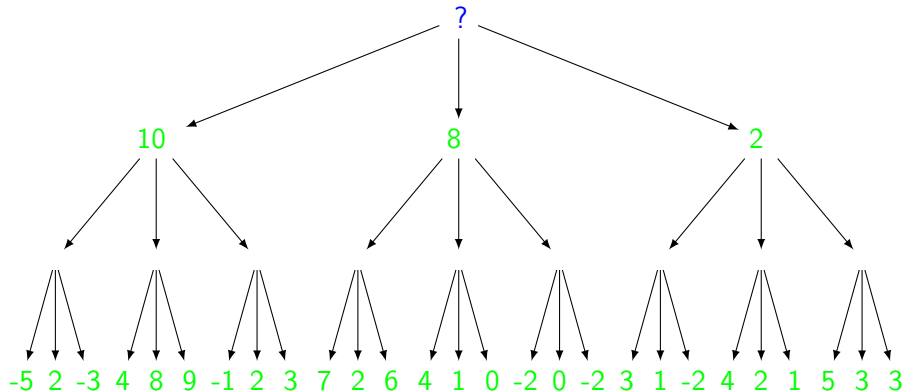


|   |   |    |    |    |   |   |
|---|---|----|----|----|---|---|
| 3 | 4 | 5  | 7  | 5  | 4 | 3 |
| 4 | 6 | 8  | 10 | 8  | 6 | 4 |
| 5 | 8 | 11 | 13 | 11 | 8 | 5 |
| 5 | 8 | 11 | 13 | 11 | 8 | 5 |
| 4 | 6 | 8  | 10 | 8  | 6 | 4 |
| 3 | 4 | 5  | 7  | 5  | 4 | 3 |

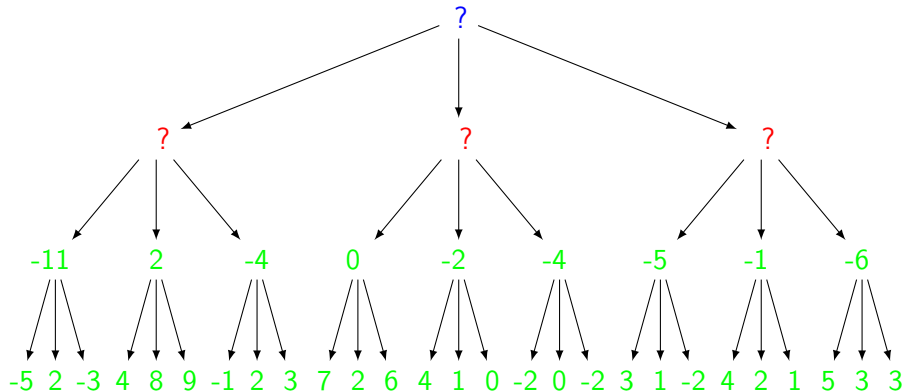
Points du joueur jaune :

$$4 + 5 + 7 + 6 + 8 + 13 + 11 - 3 - 5 - 4 - 8 - 10 - 11 - 11 = 2$$

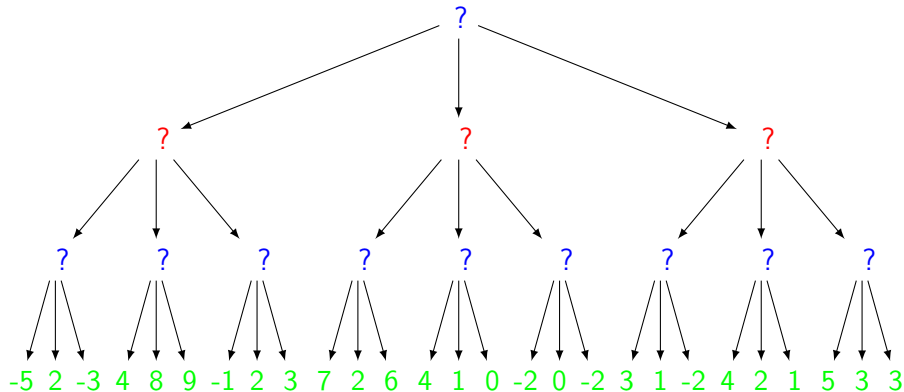
# Exemple d'optimisation de partie



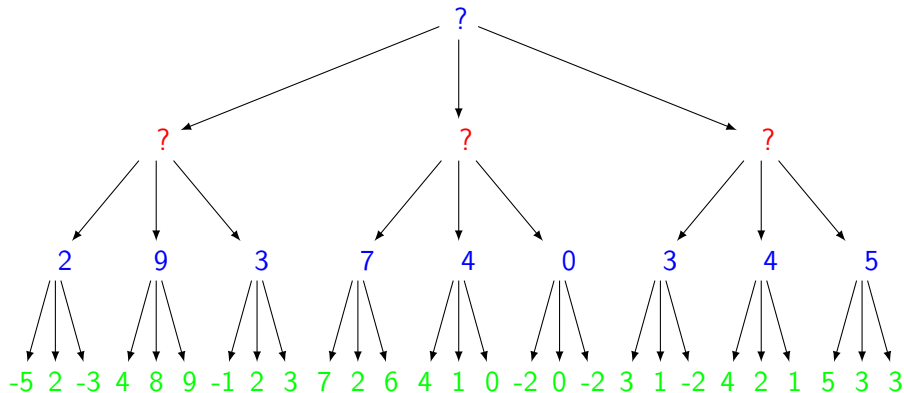
# Exemple d'optimisation de partie



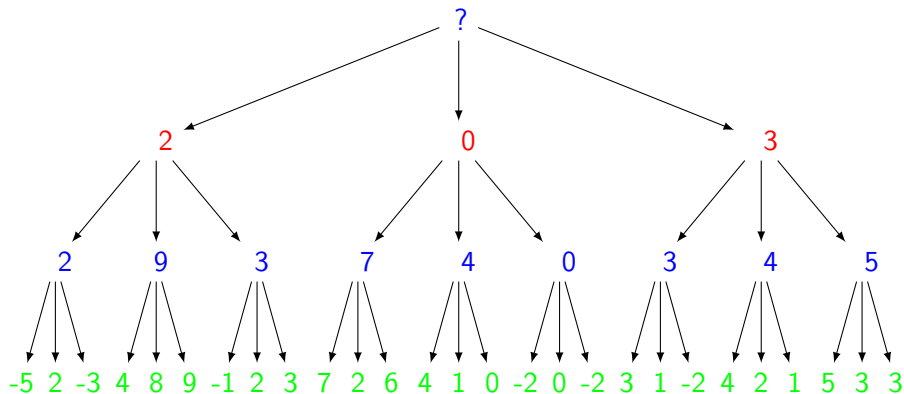
# Exemple d'optimisation de partie



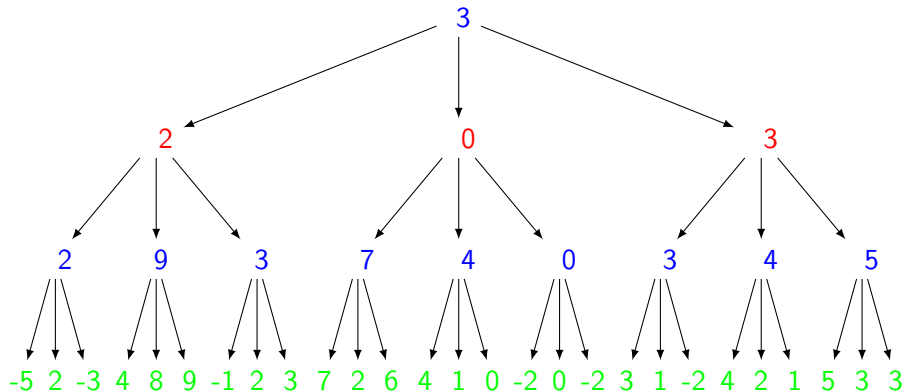
# Exemple d'optimisation de partie



# Exemple d'optimisation de partie



# Exemple d'optimisation de partie



# Algorithme min-max

## Algorithme min-max :

**Entrée :** Un nœud  $n$  d'un arbre associé à un jeu Min-Max, un entier  $p \geq 0$  (profondeur de recherche), une fonction  $\mathcal{S}$  de score sur les feuilles, une heuristique  $\mathcal{H}$  sur les autres nœuds.

**Sortie :** Une estimation de  $\mathcal{E}(n)$ , score sur le nœud  $n$  de l'arbre.

**si**  $n$  est une feuille **alors**

  └ Renvoyer  $\mathcal{S}(n)$

**si**  $p = 0$  **alors**

  └ Renvoyer  $\mathcal{H}(n)$

**si**  $n$  est un nœud max **alors**

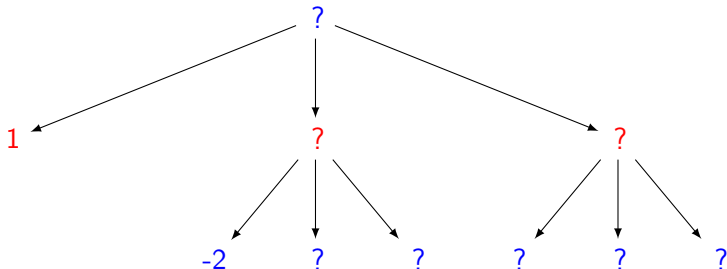
  └ Renvoyer  $\max\{\text{Minimax}(f, p-1, \mathcal{S}, \mathcal{H}) \mid f \text{ fils de } n\}$

**sinon**

  └ Renvoyer  $\min\{\text{Minimax}(f, p-1, \mathcal{S}, \mathcal{H}) \mid f \text{ fils de } n\}$

## Elagage $\alpha - \beta$ (hors programme)

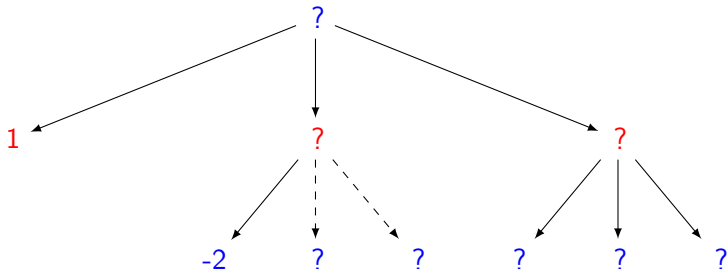
- La branche de gauche a été explorée entièrement, la valeur maximale pour le joueur bleu étant 1.
- La branche milieu-gauche a été explorée et la valeur  $-2$  a été remontée.



- Dans ces conditions, on peut éviter d'explorer les deux sous branches milieu et droite du sous-arbre du milieu (coupure  $\alpha$ ).

## Elagage $\alpha - \beta$ (hors programme)

- La branche de gauche a été explorée entièrement, la valeur maximale pour le joueur bleu étant 1.
- La branche milieu-gauche a été explorée et la valeur  $-2$  a été remontée.



- Dans ces conditions, on peut éviter d'explorer les deux sous branches milieu et droite du sous-arbre du milieu (coupure  $\alpha$ ).

# IA et apprentissage

L'intelligence artificielle, c'est quoi ?

Un ensemble de techniques permettant à un ordinateur d'effectuer des tâches habituellement effectuées sans difficulté par les humains.

Exemples :

- reconnaître des lettres, des symboles, des sons
- traduire des phrases

L'apprentissage, c'est quoi ?

Des méthodes permettant à un ordinateur qui est déjà capable d'effectuer une certaine tâche d'améliorer ses performances avec l'expérience.

Deux types d'apprentissage :

- Apprentissage supervisé :

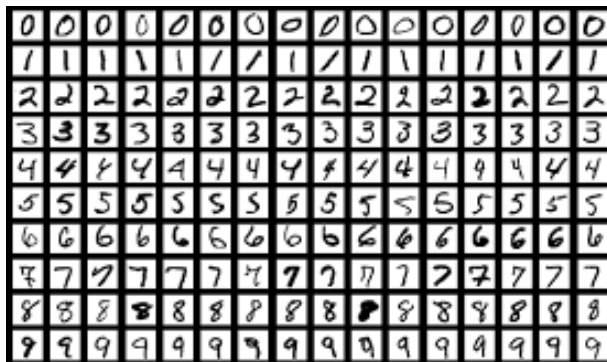
- ▶ On fournit à l'ordinateur une grosse quantité d'exemples, où chaque élément est donné avec la catégorie à laquelle il appartient.
- ▶ La machine construit elle-même les paramètres qui lui permet de ranger chaque objet dans sa catégorie.
- ▶ Une fois l'apprentissage terminé, la machine doit être capable de traiter de nouveaux exemples.

- Apprentissage non supervisé :

- ▶ On fournit à l'ordinateur une grosse quantité d'exemples
- ▶ La machine construit elle-même ses propres catégories !

# Reconnaissance de chiffres

Extrait d'une base d'exemple :



2 0 2 3

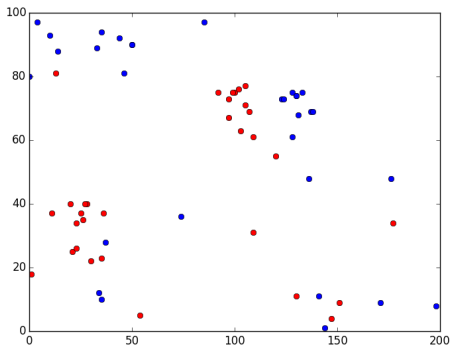
A identifier :

# Algorithme des $k$ plus proches voisins

- On construit une fonction de distance entre deux objets.
- Etant donné une donnée  $x$  à identifier, à l'aide d'un échantillon de données exemples :
  - ▶ On trie les exemples de l'échantillon par ordre croissant de distance à la donnée  $x$ .
  - ▶ On attribue à la donnée  $x$  la catégorie majoritaire parmi les  $k$  données exemples les plus proches.

# Algorithme des $k$ plus proches voisins

Exemple simplifié :



- On dispose d'une image dont une partie des pixels est coloriée.
- On souhaite colorier le reste de l'image de façon cohérente (respect des zones majoritairement bleues / rouges).

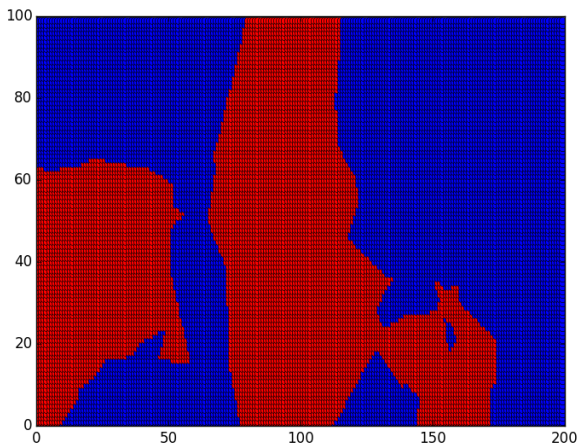
# Algorithme des $k$ plus proches voisins

Fonction de coloration d'un pixel :

```
def d(p1, p2):  
    return sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)  
  
def couleur(X, p, k):  
    L = [(d(p, c[0]), c[1]) for c in X]; L.sort()  
    nb = 0  
    for i in range(k):  
        if L[i][1] == 'r':  
            nb += 1  
        else:  
            nb -= 1  
    if nb > 0:  
        return 'r'  
    else:  
        return 'b'
```

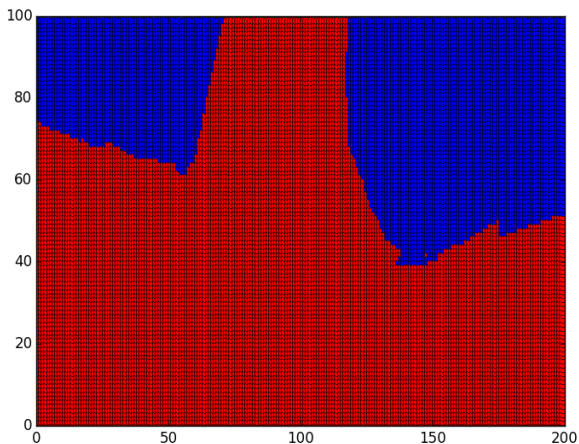
# Algorithme des $k$ plus proches voisins

Résultat pour  $k = 3$  :



# Algorithme des $k$ plus proches voisins

Résultat pour  $k = 7$  :



# Matrice de confusion

On dispose d'un algorithme  $\mathcal{A}$  de classification en  $p$  classes et d'un ensemble de données  $X$  tests.

- Chaque donnée  $x \in X$  appartient à une unique classe réelle  $C_i$  pour un certain  $i \in \llbracket 1; p \rrbracket$ .
- L'algorithme  $\mathcal{A}$  attribue à chaque  $x \in X$  un indice  $j \in \llbracket 1; p \rrbracket$  correspondant à une classe estimée .

Note :  $p = 2$  dans l'exemple précédent (deux classes : bleu ou rouge).

## Definition

La matrice de confusion est la matrice dont la case d'indices  $(i, j)$  contient le nombre de données appartenant à la classe réelle  $i$  et placée dans la classe  $j$  par  $\mathcal{A}$ .

- Plus la matrice de confusion est proche d'une matrice diagonale, plus l'algorithme de classification est efficace.
- La matrice de confusion permet d'adapter certains paramètres de l'algorithme pour améliorer son efficacité.

# Algorithme des $k$ moyennes

Problème (plus complexe) :

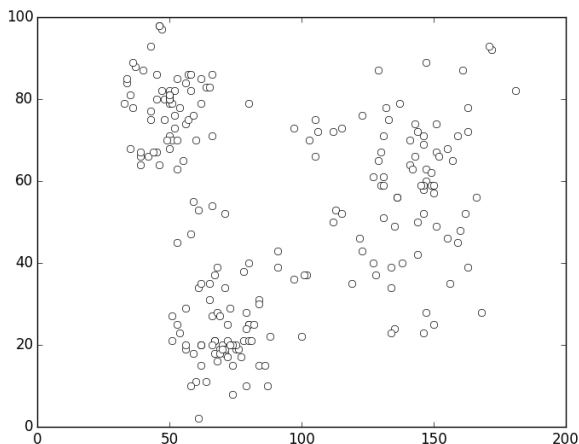
- On dispose toujours d'un échantillon de données exemples.
- Les données ne sont plus rangées par catégories !

Objectif :

- l'ordinateur doit reformer des catégories, en regroupant dans un même groupe les données qui lui semblent proches.

# Algorithme des $k$ moyennes

Exemple simplifié (encore) :



# Algorithme des $k$ moyennes

Objectif :

- Regrouper les points du nuage par sous-ensembles de points proches.
- Pour simplifier le travail, le nombre  $k$  de sous-groupes va être choisi à l'avance.

Outils formels : Etant donné un sous-ensemble  $C$  de points,

- On appelle barycentre de l'ensemble le point

$$\mu_C = \frac{1}{\text{card}(C)} \sum_{x \in C} x$$

- On appelle moment d'inertie de l'ensemble le réel

$$m_C = \sum_{x \in C} \|x - \mu_C\|^2$$

# Algorithme des $k$ moyennes

## Definition

Problème du partitionnement : Etant donné un ensemble de points  $C$  et un entier  $k$ , déterminer une partition de  $C$  en  $k$  sous-ensembles  $C_1, \dots, C_k$  qui minimisent la quantité

$$m = \sum_{i=1}^k m_{C_i}$$

# Algorithme des $k$ moyennes

Le nombre de partitions d'un ensemble à  $n$  éléments est donné par le nombre de bell

$$B_n = \sum_{k=0}^{+\infty} \frac{k^n}{k!} \underset{n \rightarrow +\infty}{\sim} \frac{1}{\sqrt{n}} \left[ \frac{n}{W(n)} \right]^{n+1/2} e^{n/W(n)-n-1}$$

où  $W$  est la fonction de Lambert définie par la réciproque de la restriction de  $x \mapsto xe^x$  à  $] -1; +\infty[$ .

- Une recherche exhaustive est inenvisageable.
- On se rabat sur un algorithme glouton.
  - ▶ Ne garantit pas de trouver la solution optimale
  - ▶ Procède par choix localement optimaux pour proposer une solution proche de l'optimal (sans garantie)
  - ▶ Converge bien plus rapidement

# Algorithme des $k$ moyennes

Pseudo-code :

- 1 on choisit aléatoirement  $k$  centres  $\mu_1, \dots, \mu_k$  parmi les  $n$  points.
- 2 on répartit les  $n$  points dans  $k$  classes  $C_1, \dots, C_k$  : chaque point  $x$  est mis dans l'ensemble  $C_j$  où  $j$  est l'indice du point le plus proche de  $x$  parmi  $\mu_1, \dots, \mu_k$ .
- 3 on calcule les  $k$  barycentres  $\mu'_1, \dots, \mu'_k$  des classes  $C_1, \dots, C_j$ .
- 4 Si  $(\mu_1, \dots, \mu_k) \neq (\mu'_1, \dots, \mu'_k)$ , remplacer  $(\mu_1, \dots, \mu_k)$  par  $(\mu'_1, \dots, \mu'_k)$  et reprendre à l'étape 2.

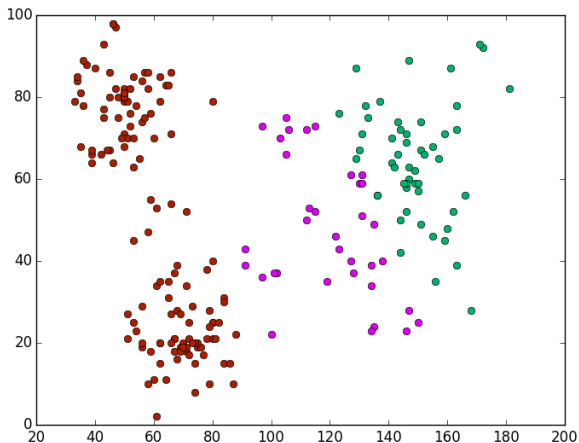
Points essentiels :

- Il y a un nombre fini de partitions  $(C_1, \dots, C_k)$  distinctes.
- (Admis) Si  $(\mu_1, \dots, \mu_k) \neq (\mu'_1, \dots, \mu'_k)$ , alors la somme  $m$  des moments d'inertie de la nouvelle répartition est strictement inférieure à celle de l'ancienne répartition.

L'algorithme termine !

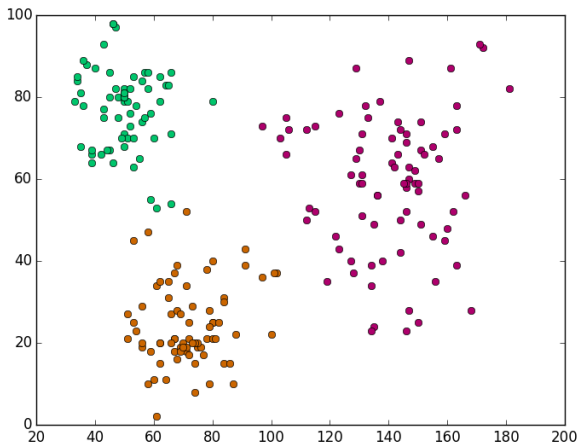
# Algorithme des $k$ moyennes

Deux exemples de résultats :



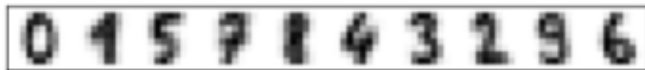
# Algorithme des $k$ moyennes

Deux exemples de résultats :



# Application : reconnaissance de chiffres

- 1797 images de chiffres de 0 à 9.
- barycentres des 10 classes renvoyées par l'algorithme des plus proches moyennes :

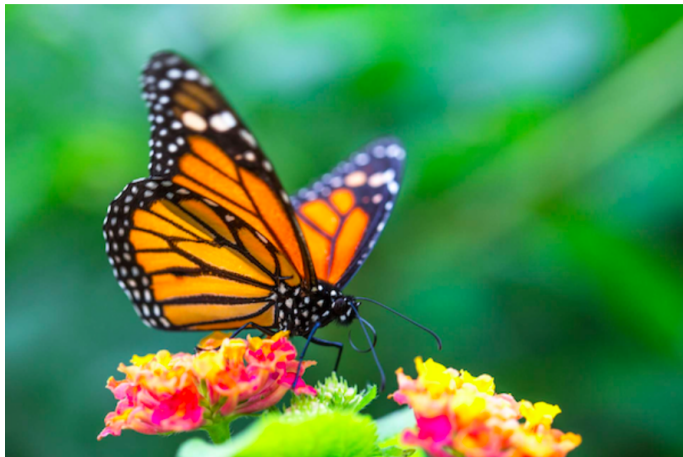


- Examen des classes : pourcentage d'image correspondant au chiffre de la classe :

| Classe | 0  | 1  | 5  | 7  | 8  | 4  | 3  | 2  | 9  | 6  |
|--------|----|----|----|----|----|----|----|----|----|----|
| %      | 99 | 60 | 91 | 85 | 45 | 98 | 87 | 85 | 56 | 97 |

# Application : compression d'image

Image de départ :  $256^3 \approx 16$  millions de couleurs



Objectif : réduire à 16 couleurs !

# Application : compression d'image

Utilisation de l'algorithme des  $k$  moyennes :

- Chaque pixel est vu comme un élément de  $\mathbb{R}^3$ .
- On applique l'algorithme des  $k$  moyennes pour répartir ces pixels en 16 sous-groupes de couleurs proches.
- On calcule le barycentre de chaque groupe, puis on remplace chaque pixel du groupe par le barycentre de son groupe.

# Application : compression d'image

Image compressée

