# Algorithme de Backtrack :

# un outil de base pour résoudre de nombreux problèmes d'IA

Cours de GMIN101

ML Mugnier

## Rappel des cours précédents

- A de nombreux problèmes on peut associer un **espace de recherche** défini par :
  - un **état initial**
  - des **actions** permettant de passer d'un état à ses successeurs
  - la définition de ce qu'est un **état but**
  - éventuellement : une fonction de **coût** d'une action
- Résoudre le problème : déterminer une **séquence d'actions** (un chemin dans el graphe associé) menant de l'état **initial** à un état **but**, en minimisant éventuellement le coût de cette séquence
- On peut explorer un espace en construisant une arborescence (« **arbre de recherche** ») :
  - dont la racine est associée à l'état initial
  - avec différents types d'exploration : profondeur, largeur, mixte, ...

## Une famille de problèmes fréquente en IA

- Problèmes définis sous la forme **(X,D,C)** :
  - **X** : ensemble de **variables**
  - **D** : ensemble de **domaines** (valeurs possibles)
  - **C** : ensemble de **conditions** sur la compatiblité des valeurs que peuvent prendre simultanément des variables
- Une **assignation** sur Y ⊆ X est une application qui associe à chaque variable de Y une valeur de son domaine
- Elle est **totale** si Y = X
- Elle est **consistante** si elle satisfait les conditions qui portent sur les variables de Y (« **solution partielle** »)
- Une **solution** est une assignation totale consistante
- Exemple : **CSP** (« Constraint Satisfaction Problem »)

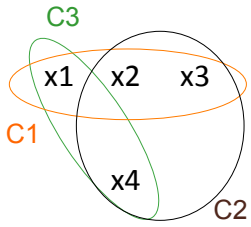
On va en voir d'autres !

## EXEMPLE DE CSP

### Réseau de contraintes

- Ensemble de variables **X**={x1,x2,x3,x4}
- Ensemble de contraintes **C**={C1,C2,C3}

  ⟶ **hypergraphe**




- Domaines des variables **D1**=**D2**=**D3**=**D4**={a,b}
  (**D** : union des Di)
- Définitions des contraintes

**C1**

| x1 | x2 | x3 |
|---|---|---|
| a | a | b |
| a | b | a |
| b | a | a |

**C2**

| x2 | x3 | x4 |
|---|---|---|
| a | b | a |
| b | a | b |

**C3**

| x1 | x4 |
|---|---|
| a | b |
| b | b |

Une assignation *f* sur Y ⊆ X est **consistante** si :
pour tout Ci sur (y1,…,yk), (f(y1), …, f(yk)) appartient à la définition de Ci

## Un espace de recherche particulier ...

- Un état contient une assignation (pas forcément totale)
- Etat initial : assignation vide
- Action : assigner une nouvelle variable
- Etat but : si assignation totale et consistante

- **Remarque 1** : le chemin qui permet d'atteindre une solution n'est pas important
- **Remarque 2** : on ne peut pas « réparer » une assignation inconsistante en l'étendant

→ ne considérer qu'une seule variable pour générer les successeurs d'un noeud

→ ne considérer que des états qui satisfont les conditions

tronquer l'arbre de recherche dès qu'on arrive à une assignation inconsistante

→ **algorithme de backtrack**

## ALGORITHME DE BACKTRACK (EXISTENCE D'UNE SOLUTION)

```
Fonction BacktrackingSearch() : Booléen  // accès aux données du problème
// retourne vrai ssi il existe une solution
   Début
      Prétraitements;
      retourner BT({});
   Fin
Fonction BT(Assignation a) : Booléen       // accès aux données du problème
   // retourne vrai ssi il existe une solution étendant a
Début
   si  |a| = |X|  alors retourner vrai;  //solution trouvée
   x ← ChoixVariableNonAssignée(a);
   pour tout v ∈ Domaine(x) faire
      si Consistant(a∪{(x,v)}) alors
            si BT(a∪{(x,v)}) alors retourner vrai
   retourner faux;
Fin
```

## IMPLÉMENTATION DES SOUS-FONCTIONS

**ChoixVariableNonAssignée ?**

1) si ordre « **statique** » sur les variables : la prochaine selon cet ordre
comment calculer un « bon » ordre ?
idée : détecter les échecs au plus tôt

(a) par une descente dans le graphe des variables
(b) par taille croissante des domaines
(c) par nombre décroissant de conditions sur les variables

2) si ordre « **dynamique** » sur les variables :
(b), puis (c) pour départager les ex-aequos

**Test Consistant ?**

Méthode simple si ordre statique sur les variables :

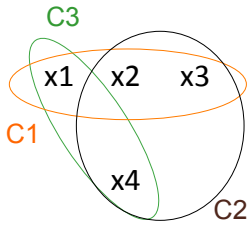
- **rang** d'une condition (contrainte) : rang de sa plus grande variable
- quand on assigne la **ième variable**, on vérifie les conditions de **rang i**.

## EXEMPLE DE CSP

### Réseau de contraintes

- Ensemble de variables **X**={x1,x2,x3,x4}
- Ensemble de contraintes **C**={C1,C2,C3}

⟶ **hypergraphe**




- Domaines des variables **D1**=**D2**=**D3**=**D4**={a,b}
(**D** : union des Di)
- Définitions des contraintes

**C1**

| x1 | x2 | x3 |
|---|---|---|
| a | a | b |
| a | b | a |
| b | a | a |

**C2**

| x2 | x3 | x4 |
|---|---|---|
| a | b | a |
| b | a | b |

**C3**

| x1 | x4 |
|---|---|
| a | b |
| b | b |

Une assignation *f* sur Y ⊆ X est **consistante** si :
pour tout Ci sur (y1,…,yk), avec {y1, …, yk} ⊆ Y,
(f(y1), …, f(yk)) appartient à la définition de Ci

## ALGORITHME DE BACKTRACK (CALCUL D'UNE SOLUTION)

```
Fonction BacktrackingSearch() : Assignation (dont « échec » ou « null »)
   // retourne une solution s'il en existe une, sinon échec
   Début
      Prétraitements;
      retourner BT({});
   Fin
Fonction BT(Assignation a) : Assignation (ou échec)
// retourne une solution s'il en existe une étendant a
Début
   si |a| = |X| alors retourner a; //solution trouvée
   x ← ChoixVariableNonAssignée(a);
   pour tout v ∈ Domaine(x) faire
      si Consistant(a∪{(x,v)}) alors
            Assignation b = BT(a∪{(x,v)})
            Si b ≠ échec alors retourner b
   retourner échec;
Fin
```

## ALGORITHME DE BACKTRACK (CALCUL DE TOUTES LES SOLUTIONS)

```
Fonction BacktrackingSearch() : Ensemble d'Assignation
// retourne l'ensemble des solutions
   Début
      Prétraitements;
      Ens → vide // ensemble de solutions
      BT(Ens, {}) // alimente Ens
      Retourner Ens
   Fin
Fonction BT(Ens d'Assignation Ens, Assignation a)
   // met dans Ens les solutions étendant a
Début
   si |a| = |X| alors ajouter a à Ens; //solution trouvée
   sinon
      x ← ChoixVariableNonAssignée(a);
      pour tout v ∈ Domaine(x) faire
         si Consistant(a∪{(x,v)}) alors BT(Ens, a∪{(x,v)})
Fin
```

## COLORATION DE GRAPHE

- Problème : étant donné un graphe G = (V,E) déterminer s'il existe une **k-coloration** de G
- k-coloration : assignation d'une couleur dans {1,..., k} à chaque sommet de V, telle que deux sommets adjacents n'aient pas la même couleur

→ Variables ?
→ Domaines ?
→ Conditions ?

- Variante : **colorer les arêtes**

## BD RELATIONNELLE (→ VUE LOGIQUE)

- **Schéma** de BD : ensemble de relations (avec leurs attributs)

*ex:* Film [titre, directeur, acteur]
Pariscope [salle, titre, horaire]
Coordonnées [salle, adresse, téléphone]

*On peut remplacer les attributs par une numérotation : 1,2,3*
→ Vue logique : Film, Pariscope, Coordonnées
sont des relations (prédicats) ternaires

- **Instance d'une relation** : ensemble de k-uplets
(k = arité de la relation)

→ Vue logique :
valeurs : constantes
instance de relation : ensemble d'atomes

- **Instance de BD** : ensemble des instances de relation

## Une instance de la relation Film

| *titre* | *directeur* | *acteur* |
|---|---|---|
| The trouble | Hitchcock | Green |
| The trouble | Hitchcock | Forsythe |
| The trouble | Hitchcock | MacLaine |
| The trouble | Hitchcock | Hitchcock |
| Cries and Whispers | Bergman | Anderson |

Vue logique :

{ film(t,h,g), film(t,h,f), film(t,h,m), film(t,h,h), film(c,b,a) }

## REQUÊTES CONJONCTIVES

***En SQL:*** « SELECT ... FROM ... WHERE conditions de jointure »

*Exemple :*

« trouver les noms des films où Hitchcok joue »

```
SELECT Film.Titre
FROM Film
WHERE Film.Acteur = « Hitchcock »
```

Vue logique ?

*trouver* x *tel que* Film(x,y,*h*)

*Exemple :*

« trouver les noms des salles dans lesquelles on joue un film de Bergman »

- Requête SQL ?
- Vue logique ?

*Exemple :*
« trouver les noms des salles dans lesquelles on joue un film de Bergman »

```
SELECT Pariscope.Salle
FROM Films, Pariscope
WHERE
    Films.Directeur = « Bergman »
    AND Films.Titre=Pariscope.Titre
```

Vue logique :
*trouver* z *tel que* Films(x,Bergman,y) ∧ Pariscope(z,x,t)

**Vue logique d'une requête conjonctive :**
ensemble d'atomes
+ une liste de « variables réponses » (à retourner comme réponse)

Si la liste des variables est vide, on a une requête booléenne

## RÉPONDRE À UNE REQUÊTE CONJONCTIVE

- Requête Q et base de données D : deux ensembles d'atomes
- Un **homomorphisme h** de Q dans D est une substitution des variables de Q par des constantes de D
  telle que :
  Pour tout atome p(x1,...,xk) de Q
  p(h(x1),...,h(xK)) est dans D

  On considère que si *xi* est une constante, *h(xi) = xi*
- Les **réponses** à Q dans D sont obtenues en prenant les images des variables réponses par les homomorphismes de Q dans D

## RECHERCHE D'HOMOMORPHISMES DE Q DANS D

**Représentation sous la forme (X,D,C) ?**

X = variables de Q

D = constantes de D

C = conditions à satisfaire par une assignation ?

Pour chaque atome p(x1,...,xk) de Q

p(h(x1),...,h(xK)) est dans D

Exemple : Q = { p(x,y,z), s(z,t) }
D = { p(a,b,c), s(b,a), s(c,b) }

- Dérouler l'algorithme de backtrack
  en prenant un ordre statique sur les variables : z t x y
- Quels sont les atomes à tester à chaque étape ?