

# Intelligence Artificielle : Résolution de problèmes par recherche dans les graphes

Support de cours L3 Informatique- Année 2020-2021

# Table des matières

<b>1</b>	<b>Introduction à l'Intelligence Artificielle</b>	<b>2</b>
I	Situation de la discipline . . . . .	2
II	Disciplines liées à l'IA . . . . .	2
III	Historique . . . . .	2
IV	Thèmes de recherche actuels . . . . .	2
<b>2</b>	<b>Représentation et résolution de problèmes</b>	<b>3</b>
I	Formalisation d'un problème par espace d'états . . . . .	3
II	Recherche dans un espace d'états : généralités . . . . .	3
III	Recherche informée et heuristiques . . . . .	4
1)	Stratégie Meilleur d'abord Gloutonne . . . . .	5
2)	Stratégie A* . . . . .	5
<b>3</b>	<b>Algorithmes adaptés à des parcours particuliers de graphes</b>	<b>7</b>
I	Résolution de problème et arbre ET/OU . . . . .	7
II	Problèmes de coloration de graphes . . . . .	8
III	Problèmes de satisfaction de contraintes . . . . .	10

# Chapitre 1

## Introduction à l'Intelligence Artificielle

### I Situation de la discipline

L'Intelligence Artificielle (IA) est à la fois une **science** et une **technique**.  
C'est une **science** qui cherche à

- observer,
- étudier,
- comprendre
- puis à modéliser et à simuler

les facultés intellectuelles (ou capacités cognitives) comme le raisonnement, la compréhension du langage naturel, la perception, . . . .

L'IA est aussi une **branche de l'informatique** qui conçoit des méthodes et des techniques pour faire réaliser par un ordinateur des tâches nécessitant de l'intelligence (i.e. qui semblaient intelligentes lorsqu'un agent humain, ou même un animal les réalisait), et traiter des problèmes (toujours de plus en plus difficiles) pour lesquels on ne connaît pas de méthode de résolution assurée.

### II Disciplines liées à l'IA

Philosophie, Mathématiques, Psychologie, Neurosciences, Ethologie, Informatique, Linguistique, Economie

### III Historique

1956 séminaire de Dartmouth (McCarthy, Minsky, Rochester, Shannon. Années 60 décollage (A\*, résolution automatique. Années 80 : systèmes experts et déception. Années 90 : pluridisciplinarité avec robotique, IHM, SI. 2016 : révolution du deep-learning

### IV Thèmes de recherche actuels

Représentation des connaissances, Résolution de problèmes, Formalisation des raisonnements, Algorithmes dédiés, Plannification et Décision, Apprentissage Automatique et Data Science, IHM intelligents, TAL et Web Sémantique.

## Chapitre 2

# Représentation et résolution de problèmes

### I Formalisation d'un problème par espace d'états

On s'intéresse à des problèmes pour lesquels il existe une description formelle, mais pas de méthode de résolution spécifique et dont l'environnement est supposé observable et déterministe.

Un problème est caractérisé par :

- la description d'un état possible
- un état initial
- une description d'un état-but (explicite ou implicite par ses propriétés)
- la description des opérateurs

L'ensemble de tous les états accessibles à partir de l'état initial s'appelle l'**espace de recherche**

La **résolution** d'un problème est la recherche (et la découverte si le problème admet une solution) d'un état-but dans l'espace de recherche.

### II Recherche dans un espace d'états : généralités

Nous utiliserons dans la suite le vocabulaire suivant :

- Un état est dit **exploré** (ou **développé**) si on a testé si c'est un état but et dans le cas négatif on a créé ses successeurs.
- Un état est **créé** lorsqu'il est produit par application d'un opérateur à un autre état. On calcule alors le code de sa représentation. Par convention, l'état initial est créé.
- Lors d'une exploration de l'espace de recherche, les états développés seront rangés dans l'ensemble appelé **Vus**, les états créés mais non encore développés seront rangés dans l'ensemble appelé **EnAttente**.

La mise en oeuvre d'une stratégie d'exploration produit un algorithme de recherche qui peut être évalué par différents critères :

**Complétude** Si le problème admet une solution, l'algorithme s'arrête en fournissant une solution.

```

Données :  $e_0$  : état initial, FilsEtat : produit les fils d'un état, TestBut : prédicat qui teste
             si un état est un état-but, classe : range les fils d'un état dans la liste EnAttente
Variables : EnAttente, Vus
                $e$  : état courant ,  $trouve$  : un booléen

début
  EnAttente  $\leftarrow (e_0)$ ; Vus  $\leftarrow ()$ 
  trouve  $\leftarrow$  FALSE
  tant que EnAttente non vide et non trouve faire
     $e \leftarrow$  choisirEtEnlever(EnAttente)
    Vus  $\leftarrow$  ajouter( $e$ , Vus)
    si TestBut( $e$ ) alors
      trouve  $\leftarrow$  TRUE
    sinon
      EnAttente  $\leftarrow$  classe(FilsEtat( $e$ ), EnAttente, Vus)
  si non trouve alors
    print(ECHEC)
  sinon
    retourner  $e$ 

```

Algorithme 1 : Algorithme de recherche général

**Admissibilité** Ce critère (encore appelé optimalité) s'applique à la recherche d'une solution avec coût. Une recherche est admissible si elle fournit une meilleure solution.

**Complexité temporelle** Temps nécessaire pour trouver une solution.

**Complexité spatiale** Place mémoire nécessaire pour effectuer l'exploration.

La complexité temporelle (resp. spatiale) est donc mesurée par le nombre d'états créés (resp. nombre maximal d'états conservés en mémoire).

Différentes stratégies d'exploration :

**Stratégie Largeur d'abord** La fonction *classe* ajoute les états à la fin de la liste EnAttente (qui est donc une file) et ChoisirEtEnlever prend le premier de la liste.

**Stratégie Profondeur d'abord** La fonction *classe* ajoute les états en tête de la liste EnAttente (qui est donc une pile) et ChoisirEtEnlever prend le premier de la liste

**Profondeur bornée** Profondeur d'abord avec une limite ( $l$ ) sur la profondeur (les états de profondeur  $l$  ne sont pas développés).

**Profondeur itérative** Profondeur bornée en essayant des profondeurs successives (1, 2, 3, ...) ( $\neq$  largeur d'abord car l'ordre est différent).

### III Recherche informée et heuristiques

La stratégie peut utiliser une heuristique, qui dépend du problème. On appelle heuristique tout procédé nous guidant dans la recherche de(s) solution(s).

**Données :**  $e_0$  : état initial, FilsEtat : produit les fils d'un état, TestBut : prédicat qui teste si un état est un état-but, classeSelonEval : range les fils d'un état dans la liste EnAttente selon l'évaluation

**Variables :** EnAttente, Vus  
 $e$  : état courant ,  $trouve$  : un booléen

**début**


```


EnAttente  $\leftarrow$  ( $e_0$ ) ; Vus  $\leftarrow$  ()
trouve  $\leftarrow$  FALSE
tant que EnAttente non vide et non trouve faire
     $e \leftarrow$  enleverTete(EnAttente)
    Vus  $\leftarrow$  ajouter( $e$ , Vus)
    si TestBut( $e$ ) alors
        trouve  $\leftarrow$  TRUE
    sinon
        EnAttente  $\leftarrow$  classeSelonEval(FilsEtat( $e$ ), EnAttente, Vus)
si non trouve alors
    print(ECHEC)
sinon
    retourner  $e$ 

```

### Algorithme 2 : Stratégie meilleur d'abord

La plupart des stratégies "meilleur d'abord" intègrent comme composante de la fonction  $f$  une fonction heuristique, généralement notée  $h$ , dont la valeur dépend uniquement de l'état, et non du déroulement de la recherche.

 : FilsEtat( $e$ ) crée tous les fils  $e'_i$  de  $e$ , en associant à chacun un pointeur vers leur père  $e$  et une valeur  $f(e'_i)$

 : Attention : l'insertion de tout fils  $e'$  dans EnAttente (classeSelonEval) se fait au bon endroit de la liste selon sa valeur et en respectant les deux optimisations (EnAttente sans doublon et utilisation de Vus) de la façon suivante :

- si  $e'$  n'est ni dans EnAttente ni dans Vus il est inséré dans EnAttente
- si  $e'$  est déjà dans EnAttente ou Vus (avec  $f_{prec}(e')$  son ancienne valeur),
  - si  $f(e') \geq f_{prec}(e')$  alors ne pas stocker le nouvel  $e'$ .
  - si  $f(e') < f_{prec}(e')$  alors supprimer l'ancien  $e'$  et insérer le nouveau  $e'$  EnAttente.

### 1) Stratégie Meilleur d'abord Gloutonne

La fonction d'évaluation  $f$  se réduit à la composante heuristique  $h$ , qui estime le coût minimal d'un chemin menant de l'état courant à un état-but.

### 2) Stratégie A\*

Un cas particulier très connu de stratégie meilleur d'abord est la stratégie A\*, dans laquelle la fonction  $f$  appliquée à un état donne une estimation du coût du chemin optimal menant de l'état initial à un état but en passant par l'état courant (coût de la solution optimale passant par l'état

courant).

Pour chaque état  $e$ , on définit :

- $g^*(e)$  le coût du chemin optimal menant de l'état initial à l'état  $e$
- $h^*(e)$  le coût d'un chemin optimal menant de l'état courant à un état-but
- $f^*(e) = g^*(e) + h^*(e)$

On obtient une estimation  $f$  de  $f^*$  par  $f = g + h$ , où  $g$  est une estimation de  $g^*$  et  $h$  une estimation de  $h^*$ . Un choix naturel pour  $g(e)$  est le coût du chemin menant de l'état initial à l'état courant  $e$ , dans l'arbre de recherche.

#### a) Caractéristiques d'une heuristique

- $h$  est une heuristique *monotone* ssi pour tout état  $u$  et pour tout état  $v$  fils de  $u$ , on a  $h(u) \leq h(v) + \text{coût}(u, v)$
- $h$  est une heuristique *coïncidente* ssi pour tout état-but  $e$ ,  $h(e) = 0$
- $h$  est une heuristique *minorante* (ou optimiste) ssi pour tout état  $e$ ,  $h(e) \leq h^*(e)$

**Propriété 1** Une heuristique coïncidente et monotone est minorante.

#### b) Propriétés d'un algorithme utilisant la stratégie $A^*$

**Théorème 1** Quand le nombre de successeurs d'un état est fini et que chaque arc porte un coût positif  $c(e_i, e_j) \geq \delta > 0$ , l'algorithme  $A^*$  est **complet** pour les graphes finis (et même pour les graphes infinis).

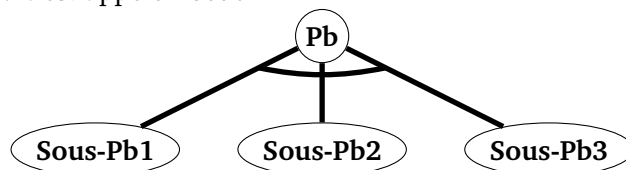
**Théorème 2** Quand le nombre de successeurs d'un état est fini (H1) et que chaque arc porte un coût positif  $c(e_i, e_j) \geq \delta > 0$  (H2), si l'heuristique  $h$  est minorante alors l'algorithme  $A^*$  est **admissible**.

## Chapitre 3

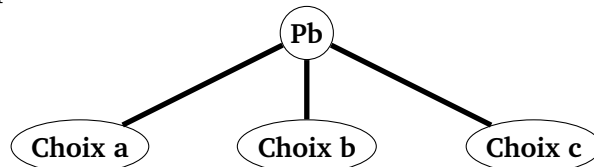
# Algorithmes adaptés à des parcours particuliers de graphes

### I Résolution de problème et arbre ET/OU

- Soit  $n$  un noeud associé à un problème dont on connaît une décomposition en conjonction de sous-problèmes.  $n$  est appelé noeud *ET*



- Soit  $n$  un noeud associé à un problème pour lequel on connaît un choix de méthodes de résolution.  $n$  est appelé noeud *OU*



L'arbre d'exploration est appelé arbre *ET/OU*.

#### Définition 1

- Un noeud terminal est résolu s'il correspond à un problème dont la solution est connue.
- Un noeud terminal est en échec s'il correspond à un problème sans solution et indécomposable.
- Un noeud *ET* est résolu si tous ses fils sont résolus [et si la contrainte est satisfaite].
- Un noeud *OU* est résolu si l'un au moins de ses fils est résolu.
- Un arbre *ET/OU* est résolu ssi sa racine est un noeud résolu.

**Définition 2 (Plan issu d'un noeud)** Etant donné un arbre *ET/OU*, et  $n$  un noeud de l'arbre :

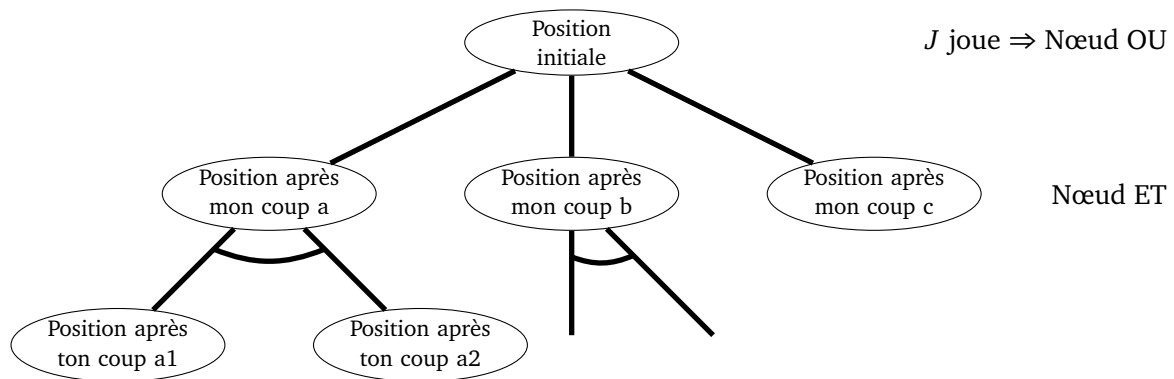
- Un plan partiel issu du noeud  $n$  est un sous-arbre de racine  $n$  qui contient au plus un fils de chacun de ses nœuds *OU*.
- Un plan complet issu du noeud  $n$  est un plan partiel issu de  $n$  qui contient exactement un fils de chacun de ses nœuds *OU*, tous les fils de chacun de ses nœuds *ET*.



**Définition 3 (Plan pour un arbre ET/OU)** Etant donné un arbre ET/OU :

- Un plan pour l'arbre ET/OU est un plan complet issu de sa racine.
- Une solution de l'arbre ET/OU est un plan pour l'arbre ET/OU dont toutes les feuilles sont des noeuds résolus et où toutes les contraintes sont satisfaites.

**Définition 4** Un arbre de jeu est un arbre de toutes les situations possibles obtenues en appliquant les règles du jeu. Un arbre ET/OU est un arbre de jeu dont les sommets sont de type ET ou OU, le type ET/OU est donné par rapport à un joueur de référence (OU= joueur de référence, ET= adversaire).



**Définition 5**

- Une stratégie partielle (pour J) est un sous-arbre de l'arbre ET/OU qui comprend la racine de l'arbre de jeu et au plus un fils de chacun des nœuds OU.
- Une stratégie complète (pour J) est un sous-arbre de l'arbre ET/OU qui comprend :
  - la racine de l'arbre de jeu
  - un fils de chacun des nœuds OU
  - tous les fils de chacun des nœuds ET.
 Une stratégie complète représente une façon de jouer pour J contre toute défense de l'adversaire.
- Une stratégie gagnante (pour J) est une stratégie complète dont toutes les feuilles sont gagnantes pour J (étiquette G). Une stratégie gagnante représente une façon de jouer pour J qui lui permet de gagner contre toute défense de l'adversaire.

**Propriété 2** Etant donné un arbre de jeu, il existe toujours une stratégie gagnante pour l'un des joueurs.

**Définition 6** Etant donné un arbre de jeu dont les feuilles sont évaluées,

- la valeur d'une stratégie complète est le minimum des valeurs des feuilles. Cette valeur est le gain garanti par la stratégie.
- la stratégie optimale est la meilleure des stratégies complètes. Sa valeur est donc le gain garanti maximum, ou meilleur gain garanti. C'est la valeur minimax de l'arbre.

**Propriété 3** La procédure Minimax fait "remonter" à la racine la valeur minimax de l'arbre.

## II Problèmes de coloration de graphes

Nous ne considérerons que des graphes simples (i.e. sans boucle).

**Définition 7 (Graphe complet)** Un graphe non orienté  $G = (X, U)$  est complet s'il existe une arête entre deux sommets distincts quelconques.

**Définition 8 (Clique et Stable)** Soit  $G = (X, U)$  un graphe non orienté. Une clique de  $G$  est un ensemble de sommets tel que le sous-graphe engendré est un sous-graphe complet de  $G$ . Un stable de  $G$  est un ensemble de sommets tel que deux sommets distincts ne sont pas adjacents.

On note  $\alpha(G)$  le cardinal du plus grand stable de  $G$ , appelé nombre de stabilité de  $G$ .

**Définition 9 (Coloration et Nombre chromatique)**

- Une coloration de  $G$  est une fonction associant à tout sommet de  $G$  une couleur, généralement un élément de l'ensemble d'indices des couleurs  $\{1, 2, \dots, n\}$ , telle que deux sommets adjacents n'aient pas la même couleur (où  $n$  est le nombre de sommets du graphe).
- Une  $k$ -coloration est une coloration dont les couleurs appartiennent à  $\{1, 2, \dots, k\}$ .
- Le nombre minimum de couleurs pour obtenir une coloration de  $G$  est appelé le nombre chromatique de  $G$ , noté  $\chi(G)$ . On dit que  $G$  est  $\chi(G)$ -chromatique.

**Propriété 4** Le nombre chromatique d'un graphe  $G$  est supérieur ou égal au cardinal de la plus grande clique de  $G$ .

**Propriété 5** Soit  $D$  le degré maximal des sommets du graphe  $G$ . Le nombre chromatique d'un graphe  $G$  est inférieur ou égal à  $D + 1$ .

**Propriété 6** Soit  $G$  un graphe d'ordre  $n$ .  $\lceil \frac{n}{\alpha(G)} \rceil \leq \chi(G) \leq n - \alpha(G) + 1$ .

**Algorithme de base (dit glouton)**

On colorie un sommet à la fois et on choisit pour le prochain sommet à colorer la plus petite couleur possible (c'est à dire non utilisée pour les sommets adjacents ou sommets voisins).

**début**

$coul \leftarrow 1$

**tant que** il reste des sommets à colorer **faire**

    extraire le premier sommet  $s$

$V \leftarrow$  liste des voisins de  $s$

$nouvCoul \leftarrow$  plus petite couleur non utilisée dans  $V$

    colorier  $s$  avec  $nouvCoul$

**si**  $nouvCoul > coul$  **alors**

$coul \leftarrow nouvCoul$

**Algorithme 3 : Algorithme de coloration glouton**

Notons que l'ordre des sommets est ici statique et quelconque.

On peut améliorer en classant les sommets par ordre de degré décroissant.

Une amélioration de l'algorithme précédent est l'algorithme DSatur, dans lequel le prochain sommet à colorer est choisi par ordre décroissant du degré de saturation. Soit  $x$  un sommet,  $Dsat(x)$  est le nombre de couleurs distinctes déjà utilisées pour les sommets adjacents à  $x$ .

On extrait un sommet  $s$  tel que  $Dsat(s)$  est maximal (en cas d'égalité choisir un sommet dont le nombre de voisins non coloriés est maximal).

**Théorème 3 (Théorème des 4 couleurs)** *Il est possible, en n'utilisant que quatre couleurs différentes, de colorer n'importe quelle carte découpée en régions connexes, de sorte que deux régions adjacentes (ou limitrophes), c'est-à-dire ayant toute une frontière (et non simplement un point) en commun reçoivent toujours deux couleurs distinctes.*

De manière tout à fait équivalente, on en déduit qu'il faut 4 couleurs pour la coloration des faces d'un polyèdre, ou des sommets d'un graphe planaire.

**Définition 10** *Un graphe planaire est un graphe qui a la particularité de pouvoir se représenter sur un plan sans qu'aucune arête (ou arc pour un graphe orienté) n'en croise une autre.*

### III Problèmes de satisfaction de contraintes

**Définition 11** *Un CSP est défini par un triplet  $(X, D, C)$  tel que :*

- $X = \{x_1, x_2, \dots, x_n\}$  est un ensemble de variables.
- $D$  est la fonction qui associe à chaque variable  $x_i$  son domaine fini  $D(x_i)$ .
- $C = \{C_1, C_2, \dots, C_k\}$  est un ensemble de contraintes. Chaque contrainte  $C_i$  est définie par un couple  $(v_i, r_i)$  où :
  - $v_i$  est une séquence de variables de  $X$  sur lesquelles porte la contrainte  $C_i$ . L'arité de  $C_i$  correspond à la longueur de  $v_i$ .
  - $r_i$  est une relation définie sur les domaines associés aux variables de  $v_i$ .  $r_i$  représente les  $m$ -uplets de valeurs autorisés pour les variables de  $v_i$ .

**Définition 12** *Soit  $(X, D, C)$  un CSP*

- Une affectation de  $Y \subseteq X$  est une application qui associe à chaque variable  $x_i$  de  $Y$  une valeur appartenant au domaine  $D(x_i)$ . On notera  $A = \{x_1 \leftarrow u_1, \dots, x_p \leftarrow u_p\}$  l'affectation qui associe la valeur  $u_1$  à la variable  $x_1$ , ..., la valeur  $u_p$  à la variable  $x_p$ .
- Une affectation de  $X$  est dite totale. Une affectation de  $Y \subset X$  est dite partielle.
- Une affectation  $A$  de  $Y$  satisfait la contrainte  $C_i = (v_i, r_i)$ , noté  $A \models C_i$ , ssi  $v_i \subseteq Y$  (toutes les variables de  $C_i$  sont instanciées dans  $A$ ) et  $A(v_i) \in r_i$  (la relation définie par  $C_i$  est vérifiée pour les valeurs des variables de  $C_i$  définies dans  $A$ ).
- Une affectation  $A$  de  $Y$  viole la contrainte  $C_i = (v_i, r_i)$  ssi  $v_i \subseteq Y$  et  $A(v_i) \notin r_i$ .

$A(v_i)$  dénote l'application de l'affectation  $A$  à la séquence de variables  $v_i$ .

**Définition 13** *Soit  $(X, D, C)$  un CSP*

- Une affectation de  $Y \subseteq X$  est consistante ssi pour toute contrainte  $C_i = (v_i, r_i)$  de  $C$  telle que  $v_i \subseteq Y$ , on a  $A \models C_i$  (c'est à dire  $A$  ne viole aucune contrainte de  $C$ ).
- Une solution  $S$  du CSP est une affectation totale consistante. On dit alors que  $S$  satisfait le CSP
- Le CSP est consistant ssi il admet au moins une solution.

#### Algorithme Backtrack

La fonction récursive Backtrack (BT) prend en argument une séquence  $V$  de variables (initialement  $V = X$ ) et une affectation partielle  $A$  (initialement vide).

```

début
  si  $V = \emptyset$  alors
    └ retourner VRAI ( $A$  est une solution, le CSP est consistant)
  sinon
    Soit  $x_i \in V$ 
    pour chaque  $u \in D(x_i)$  faire
      si  $A \cup \{x_i \leftarrow u\}$  est consistant alors
        └ BT( $V \setminus \{x_i\}, A \cup \{x_i \leftarrow u\}$ )

```

**Algorithme 4 : Algorithme Backtrack****Algorithme Anticipation (ou Forward-Checking)**

La fonction récursive FC prend en argument une séquence  $V$  de variables (initialement  $V = X$ ) et une affectation partielle  $A$  (initialement vide).

```

si  $V = \emptyset$  alors retourner VRAI ( $A$  est une solution, le CSP est consistant)
sinon
  Soit  $x_i \in V$ 
  pour chaque  $u \in D(x_i)$  faire
    (*sauvegarde des domaines des variables*)
    sauvegarder ( $V \setminus \{x_i\}$ )
    (*filtrage des domaines par rapport à  $A \cup \{x_i \leftarrow u\}$ *)
    bool  $\leftarrow$  VRAI
     $W \leftarrow V \setminus \{x_i\}$ 
    tant que bool et  $W \neq \emptyset$  faire
       $x_j \leftarrow pop(W)$ 
       $D(x_j) \leftarrow \{w \in D(x_j) : A \cup \{x_i \leftarrow u, x_j \leftarrow w\} \text{ est consistante}\}$ 
      si  $D(x_j) = \emptyset$  alors bool  $\leftarrow$  FAUX
    si bool alors FC( $V \setminus \{x_i\}, A \cup \{x_i \leftarrow u\}$ )

    (*restauration des domaines des variables*)
    restaurer ( $V \setminus \{x_i\}$ )

```

**Algorithme 5 : Algorithme Forward-Checking( $V, A$ )**