Université Libre de Bruxelles

INFO-F311
Intelligence Artificielle

Synthèse IA

Étudiants :
Rayan Contuliano Bravo

Enseignants :
T. Lenaerts

2 octobre 2023

Contents

1 Introduction **2**

2 Recherche **5**
2.1 Problème de recherches 5
2.2 Graphe d'espace d'état 6
2.3 Arbres de Recherches 6
2.3.1 Recherche dans un arbre de recherche 7
2.4 DFS 7
2.5 BFS 7
2.6 Iterative deepening 8
2.7 UCS 8

1 Introduction

Définition 1.1. Qu'est-ce que l'ia

L'intelligence artificielle est une branche de l'informatique qui crée des systèmes qui pensent de manière **rationnelle**

Définition 1.2. Décisions rationnelles

Penser de manière rationnelle signifie qu'on va se concentrer sur le **choix de décisions** qui *maximisent la probabilité* d'atteindre un objectif donné. On va faire agir les systèmes de manière **optimale**

Remarques:

1. Être rationel signifie donc **maximiser** l'utilité attendue.
2. On définis un objectif par son **utilité**.

Définition 1.3. Agent

Un agent est un système qui perçoit son environnement par des **capteurs** et agit sur celui-ci par des **effecteurs**.

Définition 1.4. Agent rationnel

Un agent rationnel est un agent qui agit de manière à maximiser son utilité attendue.

Remarque: Les **capteurs**, **effecteurs** et l'**environnement** permettent à l'agent de percevoir et d'agir sur le monde de manière **rationnelle**. L'**agent** est le système qui prend les décisions.

Définition 1.5. Fonction agent

La fonction agent est une fonction qui prend en entrée une séquence de perceptions et retourne une action. $f : \mathcal{P}^* \to \mathcal{A}$

Exemple:

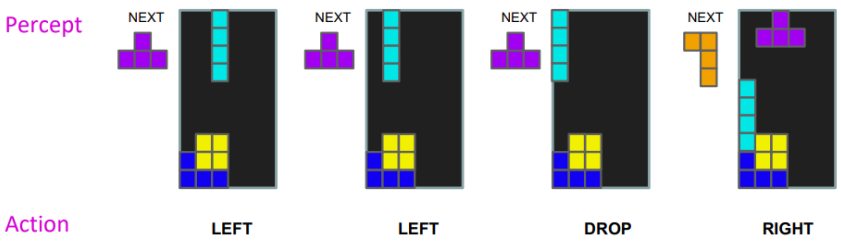

FIGURE 1.1 – Représentation fonction agent dans le jeu Tetris

Définition 1.6. Programe Agent

Un **programe agent** l est exécuté sur une **machine** M afin d'implémenter la fonction agent f.

Remarque: Les machines dans le monde réel sont **imparfaites** et **limitées** en temps et en mémoire.

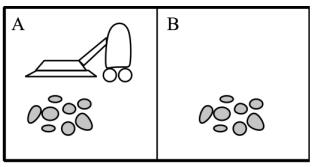

FIGURE 1.2 – Etat de l'environnement de l'aspirateur

Exemple: Nous pouvons représenter un aspirateur comme un agent qui perçoit son environnement par des capteurs et agit sur celui-ci par des effecteurs.

- **Perception** : capteurs qui détectent la saleté et sa localisation dans l'espace
- **Action** : effecteurs qui déplacent l'aspirateur dans l'espace et aspire ou non

En imaginant la situation en figure 1.2, nous pouvons définir la fonction agent de l'aspirateur comme suit :

TABLE 1 – Fonction agent de l'aspirateur

Sequence de perception	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [B, Clean]	Left
[A, Clean], [B, Dirty]	Suck
etc...	etc...

Pour que notre agent soit bien rationnel, il nous faut une manière de **mesurer** la **performance** de celui-ci. Pour cela, nous allons définir une **fonction de performance** qui va mesurer la qualité des actions de l'agent.

Exemple: On peut lui faire gagner des points ou bien lui en retirer en fonction d'une action

De cette manière, l'agent va savoir quelles actions lui permettent de **maximiser** son utilité attendue.

Afin de bien déterminer un environnement, les particularité de notre agents, il nous faut **avant toute chose** définir **PEAS**

Définition 1.7. PEAS

- **Performance** : mesure de la qualité des actions de l'agent
- **Environnement** : type d'environnement dans lequel l'agent va évoluer
- **Actuateurs** : les effecteurs de l'agent
- **Sensors** : les capteurs de l'agent

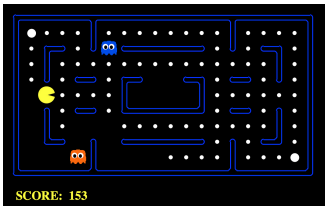

FIGURE 1.3 – Environnement Pacman

Exemple: Pour l'environnement Pacman de la figure 1.3, nous pouvons définir PEAS comme suit :

- **Performance** : -1/pas, +10/nourriture, +500/partie gagnées, -500/mort, +200/tuer un fantôme effrayé
- **Environnement** : labyrinthe **dynamique** de pacman
- **Actuateurs** : Haut, Bas, Gauche, Droite
- **Capteurs** : L'état entier visible

Définition 1.8. Types d'environnement

Il y a plusieur type d'environnement :

- **Mono-agent** : un seul agent
- **Multi-agent** : plusieurs agents qui maximisent leur **propre** tâche (coop ou concurentiel)
- **Déterministe** : l'état de l'env est déterminé **seulement** par les actions de l'agent
- **Stochastique** : l'environnement est non déterministe
- **Épisodique** : les actions de l'agent n'affectent pas les actions futures
- **Séquentiel** : les actions de l'agent affectent les actions futures
- **Dynamique** : l'environnement peut changer pendant que l'agent réfléchit
- **Statique** : l'environnement ne change pas pendant que l'agent réfléchit
- **Complètement observable** : les capteurs de l'agent perçoivent l'état complet de l'environnement
- **Partiellement observable** : les capteurs de l'agent perçoivent une partie de l'état de l'environnement
- **Discret** : un nombre fini d'états
- **Continu** : un nombre infini d'états
- **Connu** : l'agent connait les lois de l'environnement

Il existe plusieurs types d'agents qui répondent à des environnements plus complexes :

- **Agent réflexe simple** : l'agent choisit son action en fonction de la **dernière** perception
- **Agent réflexe basé sur un modèle** : l'agent choisit son action en fonction de la **dernière** perception et d'un **état interne**(dépend de l'**historique** des perceptions)
- **Agent fondés sur des buts** : l'agent choisit son action en fonction de la dernière perception ainsi que des infos relatives à l'objectif
- **Agent fondés sur l'utilité** : l'agent choisit son action en fonction de sa satisfaction par rapport à l'état résultant

2 Recherche

Définition 2.1. Agent de Plannification

Un agent de plannification font des **hypothèses** sur les conséquences des actions entreprises et utilise un **modèle** de l'environnement pour trouver un plan qui atteint son objectif.

Note:-

Un plan est une séquence d'actions qui mène à l'objectif.

2.1 Problème de recherches

Définition 2.2. Problème de Recherche

Un problème de recherche est défini par :

- **Ensemble d'État** S : Une situation dans lequel l'environnement peut être agencé
- **État initial** s_o : l'état dans lequel le problème commence
- **Actions** $A(s)$: les actions possibles dans l'état s
- **Modèle de Transition** $Result(s, a)$: la fonction qui définit les conséquences des actions
- **Solution** : Une séquence d'actions qui mène de l'état initial à l'état final
- **État final** : l'état que l'on veut atteindre

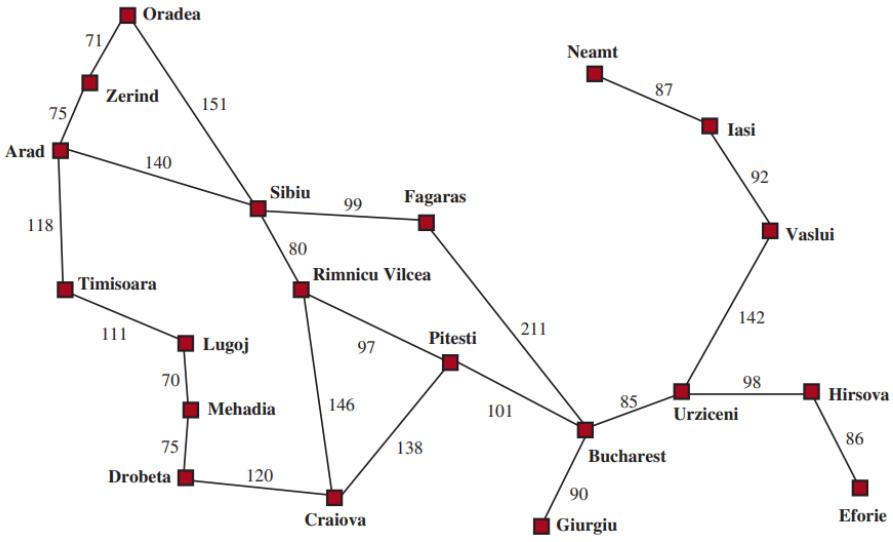

FIGURE 2.1 – Représentation simple de la Roumanie en graphe

Exemple: Voyage en Roumanie :

- **États** : les villes de Roumanie
- **État initial** : Arad
- **Actions** : les routes entre les villes adjacentes
- **Modèle de transition** : Atteindre une ville adjacente
- **Cout de l'action** : distance entre les villes
- **État final** : Bucharest

2.2 Graphe d'espace d'état

Définition 2.3. Graphe d'espace d'état

Un graphe d'espace d'état est un graphe qui représente les états et les actions possibles.

- **Noeuds** : les états
- **Arêtes** : les actions

L'état initial est le noeud racine et l'état final est un noeud (ou plusieurs ?).

⚠ Dans ce genre de graphe, chaque état n'est représenté qu'**une seule fois**.

Remarque: Il est fortement possible de ne pas pouvoir représenter un problème de recherche par un graphe d'espace d'état car il y a **trop d'états** ou que les états sont **continus**.

2.3 Arbres de Recherches

Définition 2.4. Arbre de Recherche

Un arbre de recherche est un arbre qui représente les états et les actions possibles.

- **Noeuds** : les états, plans pour arriver à ces états
- **Arêtes** : les actions
- **Enfants** : les états suivants (*succeseurs*)

L'état initial est le noeud racine et l'état final est un noeud (ou plusieurs ?). Les noeuds peuvent être représentés plusieurs fois, (**il est donc plus grand qu'un graphe d'espace d'état.**)

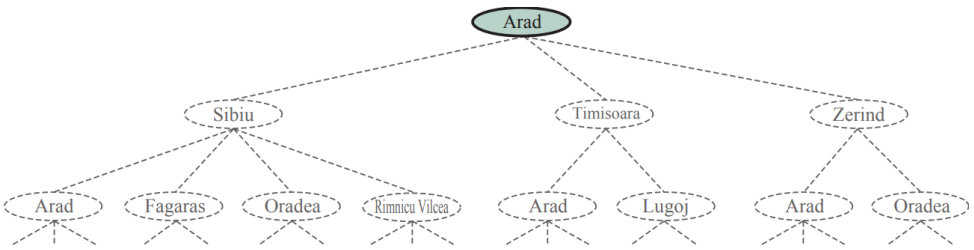

FIGURE 2.2 – Arbre de recherche de la figure 2.1

2.3.1 Recherche dans un arbre de recherche

Recherche dans un Arbre 1 Algorithme de recherche

```
function TREE-SEARCH(problème, stratégie)
    initialise un noeud avec l'état initial du problème
    loop
        if il ne peut plus y avoir d'état à explorer then
            return Erreur
        end if
        choisis un noeud non exploré selon la stratégie
        if le noeud est l'état final then
            return le plan qui mène à l'état final
        else
            Développe le noeud et ajoute ses enfants à l'arbre
        end if
    end loop
end function
```

Remarques:

1. La frontière est l'ensemble des noeuds non explorés de l'arbre
2. Pour développer un noeud de la frontiere, on le retire de la frontière et on l'ajoute à l'ensemble des noeuds explorés ses enfants sont ajoutés à la frontière

2.4 DFS

Définition 2.5. DFS

La recherche en profondeur (**DFS**) est une stratégie de recherche qui explore l'arbre en allant le plus loin possible dans une branche avant de revenir en arrière.

— **Frontière** : une pile (LIFO)
— **Stratégie** : on choisit le noeud le plus profond de la frontière

2.5 BFS

Définition 2.6. BFS

La recherche en largeur (**BFS**) est une stratégie de recherche qui explore l'arbre en allant le plus large possible dans une branche avant de revenir en arrière.

— **Frontière** : une file (FIFO)
— **Stratégie** : on choisit le noeud le moins profond de la frontière

DFS est meilleur que **BFS** dans les cas suivant :
— Si il y a des limitations de mémoire

BFS est meilleur que **DFS** dans les cas suivant :
— Si on veut trouver la solution la plus courte

2.6 Iterative deepening

Note:-

L'idée est d'avoir les avantages mémoire de **DFS** et la solution optimale de **BFS**

Définition 2.7. Iterative deepening

L'exploration itérative en profondeur (**Iterative deepening**) est une stratégie de recherche qui explore l'arbre en faisant une recherche en profondeur avec une limite de profondeur de 1, puis 2, puis 3, etc. jusqu'à ce que la solution soit trouvée.

2.7 UCS

Définition 2.8. UCS

La recherche par coût uniforme (**UCS**) est une stratégie de recherche qui explore l'arbre en allant le plus loin possible dans une branche avant de revenir en arrière.

- **Frontière** : une file de priorité
- **Stratégie** : on choisit le noeud ayant le plus petit coût de la frontière

Pour ananlyser un algorithme, on va utilser ces différentes propriétés :

- **Complet** : l'algorithme trouve toujours une solution si elle existe
- **Optimal** : l'algorithme trouve toujours la solution optimale (avec le plus petit coût)
- **Complexité en temps** : Combien de temps l'algorithme prend pour trouver une solution
- **Complexité en espace** : Combien de mémoire l'algorithme prend pour trouver une solution

TABLE 2 – Comparaison stratégie d'exploration

Critère	Largeur	Cout uniforme	Profondeur	Profondeur itérative
Complet	Oui	Oui	Non	Oui
Optimal	Oui	Oui	Non	Oui
Temps	$O(b^d)$	$O(b^{\frac{C^*}{\epsilon}})$	$O(b^m)$	$O(b^d)$
Espace	$O(b^d)$	$O(b^{\frac{C^*}{\epsilon}})$	$O(bm)$	$O(bd)$