Chapitre 3 : Représentation des problèmes d'Intelligence Artificielle et leur résolution

1. Définition d'un problème

Un problème est défini par la donnée de cinq éléments qui sont :

1- Un état initial
2- Un ensemble d'actions
3- Une fonction successeur qui définit le résultat de l'exécution d'une action dans un état
4- Un ensemble d'états

On peut voir un problème comme **un graphe orienté** où les nœuds sont les états accessibles depuis l'état initial et les arcs sont des actions. **Une solution est un chemin** de l'état initial vers un état but. Une solution est dite **optimale** si son coût est **le minimum des coûts des autres solutions**.

2. Résolution d'un problème

La résolution de problème est une activité humaine en général subordonnée à un objectif et la réalisation de cet objectif pose problème.

Résoudre un problème c'est chercher un chemin qui permet d'aller d'une situation initiale à une situation finale (but).

La résolution de problème est l'intersection de divers domaines (figure 3.1)

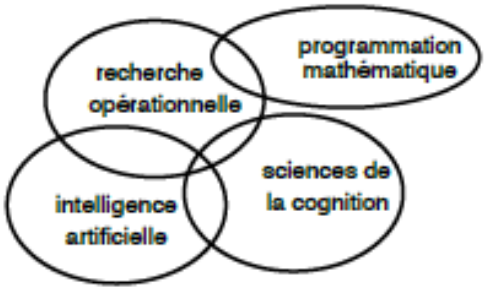


Figure 3.1 : Domaines connexes à la résolution de problèmes

Pour résoudre un problème dans sa globalité, il faut suivre une méthodologie qui assure que la solution fonctionne dans tous les cas. Une méthode rigoureuse consiste à :

- Poser correctement le problème : Définition du problème.
- Analyser la structure du problème.
- Définir une stratégie de résolution à partir de l'analyse.

En général, l'objectif est d'atteindre un **but précis** en appliquant une **séquence d'opérateurs** à une **situation initiale**.

Pour analyser la structure du problème, il faut essayer de trouver la représentation adéquate à ce problème.

En Intelligence Artificielle, il existe 2 types de représentation : la représentation en **espaces d'états** et celle en **réduction de problèmes**.

3. Représentation en espaces d'états

Dans cette représentation, on utilise 2 types d'entités : les états et les opérateurs :

- Les états constituent les conditions d'un problème à chaque étape de sa résolution et les opérateurs permettent de passer d'un état à un autre.

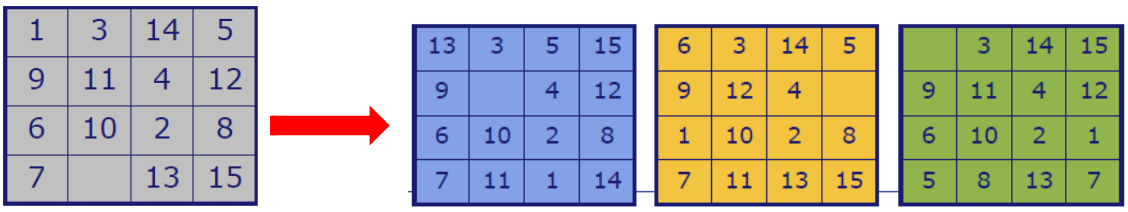
On part du principe que tous les problèmes étudiés sont composés d'un ensemble de situations ou objets que l'on peut décrire de façon univoque à l'aide d'un ensemble de variables appelés variables d'états du problème.

L'état d'un problème est alors l'ensemble des valeurs prises par les variables à un instant donné.

L'espace d'état d'un problème est l'ensemble de tous les états possibles de ce problème.

On peut considérer que la résolution d'un problème est la découverte d'un état du problème ayant des caractéristiques données (**solution**).

Exemple :
Un état est une configuration d'un tableau 4x4, on distingue, un état initial et un ou plusieurs états finaux.



Pour représenter le problème, il faut définir:

- Les états du problème = ensemble d'états réels.
- L'objectif à atteindre: solution = ensemble de chemins-solutions dans le monde réel.
- Les opérateurs de transformations (abstrait) = combinaison d'actions réelles (représentation par graphe)

Il existe plusieurs types de problème :

- *Déterministe accessible* : problème à état unique, état exact connu et les effets des actions connus.

- *Déterministe* inaccessible : problème à états multiples, un ensemble parmi plusieurs ensembles d'états, les effets des actions connus.

- *Non déterministe inaccessible* : problème contingent, (perception limitée, algorithmes plus complexes).

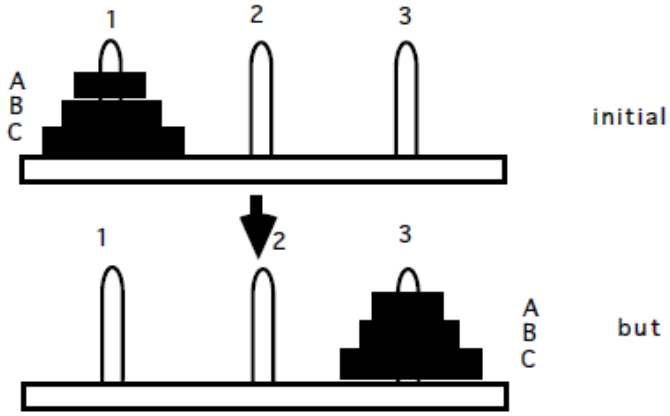
Voir l'exercice récapitulatif pour bien saisir la formulation de problème

4. Représentation en réduction de problèmes

Cette représentation est définie par :

- Une description du problème initial.
- Un ensemble d'opérateurs transformant les problèmes en sous-problèmes.
- Un ensemble de descriptions de problèmes primitifs.

Un exemple de problèmes qui se représente en réduction de problèmes est la **Tour de Hanoi.**



Reconstituer en position 3, la tour initialement construite en position 1.

On ne peut utiliser les positions 1,2,3 qu'en déplaçant un disque à la fois, en le prenant au sommet d'une tour pour le placer sur une position vide ou au sommet d'une tour, sur un disque de taille supérieure.

Le problème : déplacer une pile de taille n d'un piquet 1 vers un piquet 3 peut être remplacé par les 3 sous-problèmes suivants :

- Déplacer une pile de taille n-1 du piquet 1 vers le piquet 2
- Déplacer une pile de taille 1 du piquet 1 vers le piquet 3
- Déplacer une pile de taille n-1 du piquet 2 vers le piquet 3

5. Méthodes de recherche dans un espace d'états

5.1.Méthodes aveugles

- *Recherche en largeur*

Le parcours en largeur est un algorithme de recherche très simple : le système étant dans un état donné, on engendre tous les états qui peuvent être obtenus à partir de cet état avec les opérateurs dont on dispose, il y a succès quand l'état engendré est l'état final recherché, sinon on mémorise l'ensemble des états engendrés et on réitère pour chacun d'eux, tour à tour, le même procédé.

Voir l'exemple scanné du cours

- *Recherche en profondeur*

Le parcours en profondeur suit le chemin courant le plus longtemps possible. Le principe de cette recherche est comme suit :

Le système étant dans un état donné, engendrer tous les états pouvant en être issus.

- Il y a succès si l'un des descendant est l'état final recherché.
- Sinon, choisir un de ces états, mémoriser les autres et réitérer le procédé avec l'état choisi
- En cas d'échec, recommencer avec un des états mémorisés

Le parcours en profondeur n'est pas complet parce que l'algorithme peut continuer sur un chemin infini, ignorant complètement un état but qui se trouve sur un autre chemin. Si par contre, nous n'avons qu'un nombre fini de chemins possibles (ce qui n'est pas souvent le cas), le parcours en profondeur sera complet. L'algorithme n'est pas optimal: il n'y a rien qui garantit que le premier état but trouvé sera le bon.

Voir l'exemple scanné du cours

Le principal avantage du parcours en profondeur reste sa faible complexité en espace.

5.2.Méthodes heuristiques

Les algorithmes que nous avons vus font une recherche exhaustive de tous les chemins possibles, ce qui les rend inefficaces voire inutilisables sur les problèmes de grande taille. Dans cette section, nous présentons les algorithmes de recherche heuristiques qui utilisent des informations supplémentaires pour pouvoir mieux guider la recherche. Tout algorithme de recherche heuristique dispose d'une fonction d'évaluation f qui détermine l'ordre dans lequel les nœuds sont traités : la liste de nœuds à traiter est organisée en fonction des valeurs des nœuds, avec les nœuds de plus petite valeur en tête de liste. A priori, il n'y a pas vraiment de restriction sur la nature de la fonction d'évaluation, mais souvent elle a comme composante une fonction heuristique h où h(n) = coût estimé du chemin de moindre coût reliant n à un état but.

Notons que la fonction heuristique prend un nœud en entrée mais sa valeur ne dépend que de l'état associé au nœud. Et bien sûr, h(n) = 0 si n est un état but.

Une heuristique est une information qui peut être utilisée pour réduire l'espace d'états. Une méthode de recherche qui utilise une heuristique s'appelle méthode heuristique.

Les points sur lesquels l'heuristique peut s'appuyer sont :

- Déterminer le nœud suivant à développer au lieu d'effectuer un parcours en largeur d'abord ou en profondeur d'abord.
- Déterminer quels sont les successeurs à générer au lieu de générer tous les successeurs possibles.
- Décider quels sont les nœuds à ignorer.

Voir l'exemple scanné du cours

*Recherche avec l'algorithme A**

Nous appelons g(n) le coût du chemin entre l'état initial et n, la fonction d'évaluation utilisée par la recherche A* est donnée par la formule suivante :

$$f(n) = g(n) + h(n)$$

Comme g(n) est le coût réel associé au chemin entre l'état initial et n et que h(n) est une estimation du coût du chemin entre n et un état but, la fonction d'évaluation f donne une estimation du coût de la meilleure solution passant par le nœud n.

L'algorithme de recherche A* est complet et optimal s'il y a un nombre fini de successeurs et si nous plaçons une certaine restriction sur la fonction heuristique h. Il faut que la fonction h soit admissible, c'est à dire que la valeur h(n) ne doit jamais être supérieure au coût réel du meilleur chemin entre n et un état but.

La complexité de la recherche A* dépend de la fonction heuristique en question. En général, la complexité en temps et en espace est grande, ce qui rend la recherche A* mal adaptée pour les problèmes de grande taille. Pour pallier cet inconvénient, plusieurs autres algorithmes heuristiques moins gourmands en mémoire ont été proposés.