Chapitre 2 : L'IA et la résolution des problèmes

Pr. Mustapha BOURAHLA, Département d'Informatique, Université de M'Sila, Contact : mustapha.bourahla@univ-msila.dz

Janvier 2020

v 1.0

Table des matières

Introduction 3

I - **Types de domaines** 4

II - **Complexité** 5

III - **Représentation des problèmes** 6

IV - **Raisonnement** 8

V - **Méthodes de recherche aveugle** 9

VI - **Méthodes de recherche heuristique** 11

VII - **Qu'avez-vous retenu ?** 16

Conclusion 18

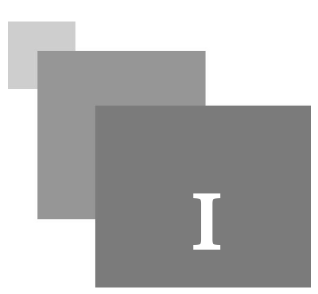
Solutions des exercices 19

Introduction

- **La résolution de problèmes** est le processus qui consiste à sélectionner et ordonnancer des **actions élémentaires** en séquences afin d'atteindre des **buts donnés** en respectant les **contraintes** d'un environnement donné.
- **Composition du processus** : un état initial (le problème à résoudre), un état final (la solution au problème), un ensemble d'actions élémentaires permettant de passer d'un état à un autre.
- **Objectif du processus** : Trouver une séquence d'actions à exécuter pour passer de l'état initial à l'état final
- **Complexité du processus** : A partir d'un état donné, on peut effectuer plusieurs actions élémentaires différentes qui débouchent sur des états différents.
- La méthode qui consiste à essayer successivement toutes les possibilités jusqu'à trouver une solution est une méthode de recherche fortement **combinatoire**
- On est souvent amené à choisir entre plusieurs choix ou à choisir la meilleur solution : **problème d'optimisation combinatoire**
- Apport de l'IA : L'IA apporte une solution à ce problème :
    - Les **heuristiques** : méthodes permettant de choisir en premier les actions ayant le plus de chance d'aboutir

©Université de M'Sila

Types de domaines



Types de domaines

- La démonstration de théorèmes
- Les jeux de stratégies (échec, dame)
- La planification :
  - robotique
  - allocations de ressources
  - emploi du temps

Exemple : Exemple de problème - Planification de chaîne de production

- Chaîne de production : Atelier A, Atelier B, Atelier C, Atelier D
- Capacité de production :
  - 5v/h 3v/h 2v/h 3v/h
  - Op1 : 8h-13h
  - Op2 : 8h-13h
  - Op3 : 9h-13h
  - Op4 : 10h-15h
- Ressources 30v en attente à 8h 0v en attente à 8h 0v en attente à 8h 0v en attente à 8h
- État initial
- Établir l'emploi du temps indiquant heure par heure quels opérateurs sont affectés à chaque atelier, de telle sorte que le maximum de voitures soient passées par les 4 ateliers à la fin de la journée

Exemple : Exemple de problème - Le voyageur de commerce

- Un voyageur de commerce partant d'une ville veut se rendre successivement dans un certain nombres de villes sans repasser deux fois dans la même ville et en revenant finalement à la ville de départ.
- On connaît les distances de ville à ville. Quel trajet doit-il effectuer de façon à minimiser la distance parcourue ?
- **EXPLOSION COMBINATOIRE** : $(n-1)!/2$ configurations
- configurations pour n=10 est 362880/2

Complexité

Complexité d'un algorithme

- Complexité est une mesure de temps (**complexité en temps**) ou d'espace mémoire (**complexité en espac**) utilisée par un algorithme en fonction de la taille des données
- La complexité en temps d'un algorithme est le nombre d'étapes de calcul qu 'il nécessite, et sa complexité en espace est la taille de mémoire nécessaire pour stocker les données intermédiaires au cours de son exécution
- Ces grandeurs sont indiquées en fonction de la taille des données du problème à résoudre, et généralemen à une constante de proportionnalité près
- Seul l'ordre de grandeur du nombre d'opérations, noté O, est utilisé pour exprimer la complexité : $O(f(n))$

Principales classes de complexité d'algorithmes

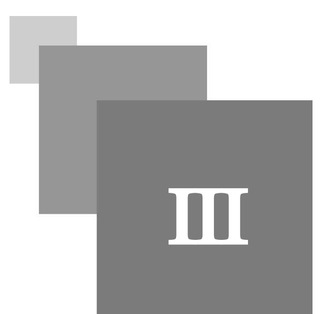
- **complexité logarithmique** : $O(log(n))$ exemple : recherche dichotomique dans un tableau trié
- **complexité linéaire** : $O(n)$ exemple : recherche du minimum d 'une liste de n éléments, calcul du produit scalaire de deux vecteurs de n éléments
- **complexité polynomiale** : $O(n^k)$ (k : une constante > 1) exemple : multiplication de deux matrices carrées d'ordre n : $(O(n^3))$
- **complexité exponentielle** :$O(a^n)$ (n : une constante > 1) exemple : voyageur de commerce

Complexité d'un problème

- La complexité (maximale) du meilleur algorithme connu pour le résoudre
- Classification des problèmes

  1. La **classe P** Classe des problèmes qu 'on peut résoudre en temps polynomial
  2. La **classe E** Classe des problèmes qu 'on peut résoudre en temps exponentiel
  3. La **classe NP** Classe des problèmes qu 'on peut vérifier en temps polynomial et résoudre en temps exponentiel (problèmes non déterministes polynomiaux)

Représentation des problèmes



Espace d'états

- Un problème est défini par un **espace d'états** qui est l'ensemble des états possibles de ce problème
- Exemples d 'états :
    - Une configuration de pièces dans un jeu d 'échecs
    - Un calendrier partiel ou complet des matches de tennis
    - Une partie de preuve ou une preuve complète d'un démonstrateur de théorèmes

Opérateurs

- Les états d'un problème sont reliés les uns aux autres par des **opérateurs** qui transforment un état dans un autre
- Exemples d'opérateurs :
    - Un coup légal aux échecs qui transforme la configuration des pièces dans une autre
    - L'ajout d'un match de tennis dans le problème de planification
    - L'application d'une règle d'inférence dans le démonstrateur de théorèmes

But

- Dans la résolution d'un problème, on a un **but** qui décrit ce que l'on cherche.
- Un but est un sous-ensemble d'états de l'espace d'états
- Exemples de buts :
    - Dans le jeu d'échecs, le but est l'ensemble des configurations de pièces qui mettent votre adversaire **échec et mat**
    - Pour le démonstrateur de théorèmes ce sont toutes les preuves dont **la dernière ligne** correspond la formule à prouver

Graphe d'états et espace de recherche

- En IA, on utilise les **graphes** pour représenter l'espace d'états d'un problème (graphe d'états)
- Chaque **nœud** représente un état, chacun des **arcs** représente un opérateur (une transition) faisant passer d'un état à un autre
- Le but est un nœud particulier du graphe
- La résolution d'un problème revient à explorer le graphe dans la recherche du **nœud but** (l'espace de recherche)

Réduction de problèmes et graphe ET/OU

- Lorsque on a un problème principal qui peut se décomposer en sous problèmes plus faciles à résoudre, résoudre ce problème principal revient à résoudre tous les **sous problèmes** qu'ils le composent
- La technique qui consiste à résoudre un but en résolvant ses sous-buts est appelé **réduction de problème**
- Certains buts ne peuvent être atteints que si leurs sous-buts immédiats le sont tous (**nœud ET**)
- Les autres buts sont atteints si l'un au moins de leurs sous-buts immédiats est atteint (**nœud OU**)
- Le graphe formé de nœuds ET et OU est appelé **graphe ET/OU**

Composition d'un système de résolution

- Des structures de données organisés en arbre ou en graphe
- Des opérateurs définis par leurs conditions d'application et leur action
- Une structure de contrôle mettant en œuvre la stratégie de recherche dans l'arbre ou le graphe (méthodes de recherche ou de résolution)

Raisonnement

Caractéristiques

1. Le système doit-il trouver une solution à partir de ce qu'il connaît du problème ou peut-il y avoir une interaction avec l'utilisateur
2. Faut-il se contenter d'une solution sous-optimale mais plus facilement et rapidement accessible
3. Peut-on annuler une action décidée au cours de la réflexion une fois qu'elle a été exécutée
4. Est-ce qu'on peut prévoir exactement l'effet d'une action sur l'état courant du problème
5. Est-ce que le problème peut se décomposer en sous problèmes plus faciles à résoudre et est-ce que les sous problèmes sont indépendants ou non.
6. Quelles sont les connaissances minimales requises pour aboutir à la solution
7. Est-ce qu'une solution est garantie ?, est-ce que la recherche terminera ? quelle est la complexité de la recherche en temps et en espace ?

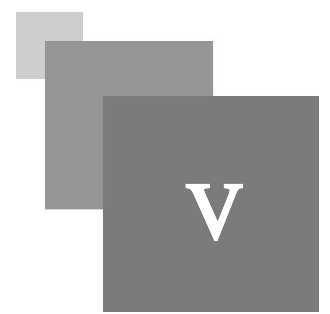
Méthodes de recherche

- **Méthodes aveugles** : énumération exhaustive de tous les états de l'espace de recherche
    1. Recherche en profondeur
    2. Recherche en largeur
    3. Recherche en profondeur limitée
    4. Recherche en approfondissement itératif
- **Méthodes heuristiques** : construction de **chemins minimaux** dans l 'exploration de l 'espace de recherche basée sur des critères, des principes, ou des méthodes permettant de faire **les choix les plus efficaces** pour atteindre le but fixé
    1. Recherche en profondeur ordonné
    2. Recherche du meilleur d 'abord

Critères d'évaluation des méthodes de recherche

- Complétude : solution garantie si elle existe
- Optimalisé : meilleur solution garantie
- Complexité en espace : espace mémoire nécessaire pour effectuer la recherche
- Complexité en temps : temps nécessaire pour effectuer la recherche
- La complexité est mesurée selon les 3 critères :
    1. b facteur de branchement (nombre de descendants/nœuds) maximum du graphe d'états
    2. d profondeur à la quelle se trouve le (meilleur) nœud solution
    3. m profondeur maximale du graphe

Méthodes de recherche aveugle



Profondeur d 'abord (LIFO)

Le principe est que la priorité est donnée aux nœuds de niveaux les plus profonds du graphe de recherche

```
1 Algorithme :
2 Début
3    empiler la racine
4    répéter Si sommet de pile <> nœud but Alors
5         dépiler le premier élément
6         empiler ses fils s'il en a du plus à droite au plus à gauche
7      Sinon ne rien faire Jusqu'à ce que la pile soit vide ou sommet pile = nœud
  but
8    la recherche aboutit si le nœud but a été atteint.
9 Fin
```

Propriétés :

- Complétude : Non si m tends vers l'infini
- Complexité en temps : $O(b^m)$
- Complexité en espace : $O(b * m)$
- Optimalité : Non

Largeur d'abord (FIFO)

Le principe est que la priorité est donnée aux nœuds de niveaux les moins profonds du graphe de recherche

```
1 Algorithme :
2 Début
3      empiler la racine
4      répéter Si sommet de pile <> nœud but Alors
5           dépiler le premier élément
6           empiler ses fils s'il en a du plus à gauche au plus à droite
7         Sinon ne rien faire Jusqu'à ce que la pile soit vide ou sommet pile =
  nœud but
8      la recherche aboutit si le nœud but a été atteint.
9 Fin
```

Propriétés :

1. Complétude : Non si b tends vers l'infini
2. Complexité en temps : $O(b^d)$
3. Complexité en espace : $O(b^d)$
4. Optimalité : Non

Exemple:

- Si $b = 10$
- production de nœuds : 1000 /s
- espace mémoire : 100 octet/noeud

Profondeur limitée

Principe : idem que profondeur d 'abord avec une limite de profondeur d 'exploration L

```
1 Algorithme :
2 Début
3     empiler la racine
4     répéter Si sommet de pile <> nœud but Alors
5         dépiler le premier élément
6         empiler ses fils s'il en a du plus à droite au plus à gauche
7       Sinon ne rien faire Jusqu'à ce que la pile soit vide ou sommet pile = nœud
  but
8             ou limite profondeur atteinte
9     La recherche aboutit si le nœud but a été atteint.
10 Fin
```

Propriétés :

- Complétude : Oui si $L \geq d$
- Complexité en temps : $O(b^L)$
- Complexité en espace : $O(b * L)$
- Optimalité : Non

Approfondissement itératif

Le principe est la recherche en profondeur, limitée à la profondeur 1, puis 2, ... jusqu'à l'obtention d'une solution (ou l'échec de la recherche)

Propriétés :

- Complétude : Oui
- Complexité en temps : $O(b^d)$
- Complexité en espace : $O(b * d)$
- Optimalité : Non

Conclusions

- Les recherches en largeur garantissent de trouver la solution (si elle existe) et de trouver le chemin le plus court au but
- Les recherches en largeur demandent un espace mémoire qui augmente exponentiellement, car elles gardent tous les enfants au même niveau
- Les recherches en profondeur sont très efficaces dans les cas où le but est loin de la racine, et les branche sont arrangées de gauche à droite par probabilité de solution
- Les recherches en profondeur peuvent se perdre dans une branche sans fin et parfois se retrouve en boucl infinie
- Les recherches aveugles sont caractérisées par une taille souvent rédhibitoire des arborescences et une trè forte complexité des algorithmes

Méthodes de recherche heuristique

VI

Fonction heuristique

$h : E \rightarrow Natural$ un état $e \in E$ (espace d'états) --> un nombre $h(e) \in Natural$:

- $h(e)$ est (généralement) une estimation du rapport coût/bénéfice qu'il y a à étendre le chemin courant en passant par e.
- contrainte : $h(solution) = 0$

Exemple de fonction heuristique :

- Problème des 8 reines
- Heuristique : laisser un plus grand nombre de cases non attaquables dans la partie restante de l'échiquier pour garder le plus de choix possibles pour les ajouts de reines futures

Profondeur ordonnée

Le principe est l'approfondissement des branches jusqu'à leur extrémité, dans l'ordre inverse de leur distance au but

```
1 Algorithme :
2 Début
3      empiler la racine
4      répéter Si sommet de pile <> nœud but Alors
5           dépiler le premier élément
6           empiler ses fils s'il en a dans l 'ordre inverse de leur distance au but
7         Sinon ne rien faire Jusqu'à ce que la pile soit vide ou sommet pile = 
  nœud but
8      la recherche aboutit si le nœud but a été atteint
9 Fin
```

Meilleur premier

Le principe est dès que l'on rencontre un chemin meilleur qu'auparavant, c'est celui là que l 'on approfondit

Heuristique : proximité du but

```
1 Algorithme :
2 Début
3    empiler la racine
4    répéter Si sommet de pile <> nœud but Alors
5         dépiler le premier élément
6         empiler ses fils s'il en a dans la file et trier la pile entière par ordre
```

```
7           croissant des distances calculées au but
8         Sinon ne rien faire Jusqu'à ce que la pile soit vide ou sommet pile = nœud
  but
9   la recherche aboutit si le nœud but a été atteint
10 Fin
```

Propriétés :

Complétude : Non

Complexité en temps : $O(b^m)$

Complexité en espace : $O(b^m)$

Optimalité : Non

*Algorithme A**

Le principe est d'améliorer l'algorithme du **Meilleur premier** en calculant le plus court chemin pour passer d'un état initial à un état final. Soient :

- n_0 : la racine du graphe
- G : le nœud solution
- $c(n_i, n_j)$: le coût pour aller du nœud n_i au nœud n_j

On définit

- $g(n) = c(n_0, n)$: le coût pour aller du nœud racine n_0 au nœud n
- $h(n) = c(n, G)$: l 'estimation heuristique du coût pour aller du nœud n au nœud solution G

On obtient alors la fonction d 'évaluation du coût du chemin en passant par le nœud n : $f(n) = g(n) + h(n)$

Définitions :

Heuristique consistante : Pour tout x, pour tout y descendant de x : $h(x) \leq h(y) + c(x, y)$

Heuristique admissible : Pour tout x, $h(x) \leq h^*(x)$ où $h^*(x)$ est la valeur optimale de x à G

```
1 Algorithme A*:
2 Début
3   Initialement, la file des chemins partiels contient le chemin d'ordre zéro,
4   de longueur nulle et reliant la racine à nulle part
5   Repeter
6     Si le premier chemin de la file n'aboutit pas au but Alors le supprimer
7      de la file, former les nouveaux chemins obtenus en prolongeant d'un niveau
8      le chemin supprimé, insérer ces nouveaux chemins dans la file le coût
9      de chaque chemin étant la somme de la distance déjà parcourue et
10     d'une estimation minorante de la distance restant à parcourir.
11     trier la file par coûts croissants si deux chemins ou plus atteignent
12     le même noeud, ne conserver que celui ayant la longueur minimale
13    Finsi
14  Jusqu'à ce que la file soit vide ou que le noeud but soit atteint
```

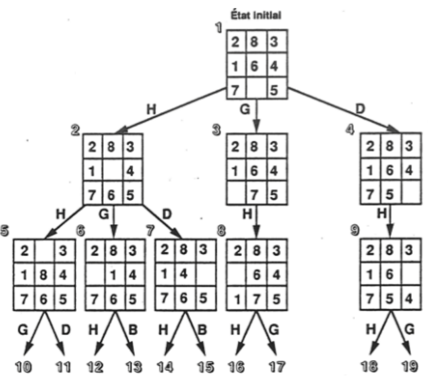
Propriétés :

- Complétude : Oui
- Complexité : linéaire si h est consistante
- Optimalité : Oui si h est admissible

Exemple : Prenons un exemple pour clarifier ces idées

- Le taquin à 8 tuiles est un casse-tête formé de tuiles numérotées de 1 à 8 que l'on peut déplacer dans une grille 3 x 3 comportant une case vide.
- Partant d'une situation initiale où les tuiles sont en désordre, le but du jeu est de les replacer en cercle dan l'ordre suivant : 1, 2 et 3 sur la première ligne, 8, case vide et 4 sur la deuxième et 7, 6 et 5 sur la troisième.
- Ayant décrit l'état final et l'état initial donnés dans l'énoncé du problème, il reste à constater à la figure qu'il y a quatre opérateurs possibles :

1. H : déplacer la tuile du haut dans la case vide;
2. G : déplacer la tuile de gauche dans la case vide;
3. D : déplacer la tuile de droite dans la case vide;
4. B : déplacer la tuile du bas dans la case vide



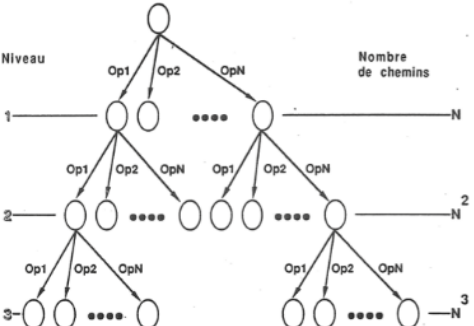
Une démarche de résolution du problème du taquin à 8 tuiles

Exemple : Une approche mécanique

Une façon classique de programmer la solution de ce problème consiste :

- à appliquer à l'état initial tous les opérateurs dans l'ordre H, G, D, B;
- à appliquer à chacun des quatre états à leur tour tous les opérateurs qui ne produisent pas une configuration déjà rencontrée;
- à procéder ainsi jusqu'à ce qu'une solution ait été atteinte ou que tous les chemins possibles aient été générés.

Cette approche mécanique a l'avantage d'être systématique. Toutefois, dès que le problème est quelque peu difficile, il faut abandonner l'idée d'une énumération exhaustive de tous les chemins.



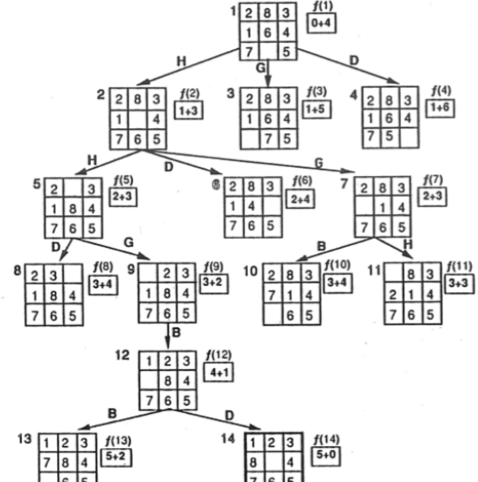
L'explosion combinatoire du nombre de chemin

- Règle générale :
    - Si à un état, on peut appliquer N opérateurs, après a niveaux, on aura généré un nombre de chemins égal à N^a.
- Voilà très précisément le problème de l'explosion combinatoire ou exponentielle évoqué précédemment e comme le montre la figure.
- Par exemple, au début d'une partie d'échecs, s' il y a 30 coups possibles en moyenne dans chaque position après 10 coups joués, il y aura $30^{10} = 590490000000000$ chemins qui auront été générés!
- Voilà qui peut excéder rapidement la capacité même d'un ordinateur très puissant.
- Il faut alors trouver des moyens d'orienter la recherche dans des directions fructueuses, en augmentant « l'intelligence » du programme.

Exemple : Une approche heuristique

- Une façon d'augmenter l'intelligence d'un programme est de le doter d'une méthode d'évaluation des états qui associe à chacun une mesure de sa probabilité de conduire vers une solution, un peu comme le ferait un joueur humain.
- Pour chaque état, le programme examinera les valeurs des états successeurs obtenus en appliquant tous le: opérateurs possibles et choisira de développer uniquement l'état ayant la meilleure valeur.
- Une telle méthode est basée sur le raisonnement suivant : « moins il y a de tuiles mal placées après l'application de l'opérateur, plus je suis sur la bonne voie; par contre, si j'ai joué beaucoup de coups depuis le début, je dois avoir fait fausse route ».
- Sur la figure, on a donné des numéros aux différents états E et on calcule pour chacun la fonction d'évaluation heuristique suivante :
  $f(E) =$ Nombre de coups depuis l'état initial + Nombre de tuiles mal placées.
- Voyons de plus près comment on utilise une telle fonction.
- Dans la figure nous avons utilisé l'algorithme A*.
- On considère cet algorithme comme une méthode générale pour faire une recherche dans un arbre des possibilités.
- Rendu à un état quelconque dans l'arbre, on examine tous les opérateurs applicables et on choisit de n'appliquer que celui qui produira la plus petite valeur de la fonction d'évaluation.

Un exemple de fonction heuristique



Cette méthode est efficace dans la mesure où la fonction d'évaluation mesure bien la valeur de chacun des états. Dans le problème du taquin, elle va presque droit au but et trouve une solution en développant seulement 12 états intermédiaires.

Qu'avez-vous retenu ?

Exercice [solution n°1 p.19]

Choisir par glissement de la réponse correcte

construction de chemins minimaux dans l 'exploration de l 'espace de recherche

énumération exhaustive de tous les états de l'espace de recherche

Méthodes aveugles	Méthodes heuristiques

Exercice

[*solution* n°2 p.19]

Choisir par glissement de la bonne réponse

Meilleur solution garantie

branchement maximum du graphe d'états

temps nécessaire pour effectuer la recherche

profondeur à la quelle se trouve le nœud solution

Solution garantie si elle existe

profondeur maximale du graphe

espace mémoire nécessaire pour effectuer la recherche

Complétude	Optimalisé	Complexité en espace	Complexité en temps	Critère b	Critère d	Critère m

Conclusion

Ce deuxième chapitre a présenté un aperçu sur l'apport de l'IA pour la résolution des problèmes

©Université de M'Sila

Solutions des exercices

> **Solution** n°1

Exercice p. 16

Choisir par glissement de la réponse correcte

Méthodes aveugles	Méthodes heuristiques
énumération exhaustive de tous les états de l'espace de recherche	construction de chemins minimaux dans l 'exploration de l 'espace de recherche

> **Solution** n°2

Exercice p. 17

Choisir par glissement de la bonne réponse

Complétude	Optimalisé	Complexité en espace	Complexité en temps	Critère b	Critère d	Critère m
Solution garantie si elle existe	Meilleur solution garantie	espace mémoire nécessaire pour effectuer la recherche	temps nécessaire pour effectuer la recherche	branchement maximum du graphe d'états	profondeur à la quelle se trouve le nœud solution	profondeur maximale du graphe

©Université de M'Sila