

# Intelligence Artificielle

## Résolution de problèmes

Laurent.Bougrain@loria.fr



# Bibliographie

## Résolution de problèmes, jeux et planification

- LAURIERE J.L. : Intelligence artificielle : résolution de problèmes par l'homme et la machine, eds Eyrolles, 1987
- LAURIERE J.L. : Intelligence artificielle : représentation des connaissances, eds Eyrolles, 1988
- PEARL J. : Heuristique : stratégies de recherche intelligente pour la résolution de problèmes par ordinateur, eds Cepadues, 1990
- PITRAT J. : Metaconnaissance : futur de l'intelligence artificielle, ed. Hermes, 1990
- PITRAT J. : Penser autrement l'informatique, ed. Hermes, 1993

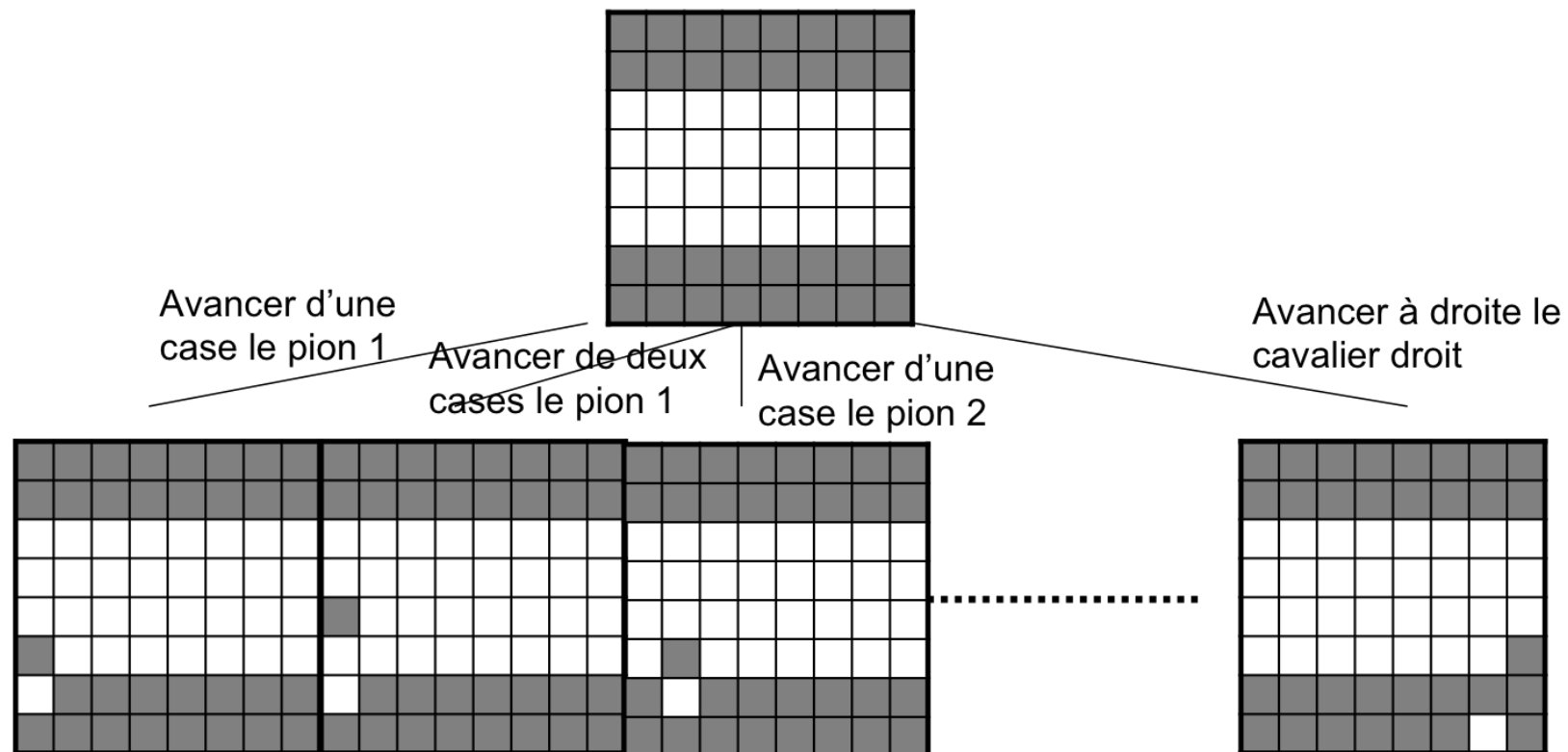
# Plan

- 1 Mécanisme de résolution de problèmes
- 2 Méthodes de recherche de solutions
  - Critères d'évaluation
  - Recherche simple (largeur, profondeur, profondeur limitée, profondeur itérative, uniforme, bi-directionnelle)
  - Recherche avec heuristique (meilleur, gourmande, A, A\*, IDA\*, SMA\*, AO\*)
- 3 Les jeux
- 4 Autres types de recherche (hill-climbing, recuit simulé, algorithmes génétiques)

# Mécanisme de résolution de problèmes

- Les algorithmes de recherche sont un mécanisme de résolution de problèmes général et puissant très utilisé en IA.
  - Il se déroule dans un espace appelé espace d'états.
  - Il recherche une séquence d'actions (appelée chemin) menant à une solution.
  - Dans un état donné, il explore toutes les actions possibles (parfois, seule une partie des actions est envisagée).
- Espace d'états
  - Il est décrit par un **état initial** et par un **ensemble d'actions** possibles (opérateurs) ayant un coût donné.
  - Un état est une représentation qui permet de connaître la situation actuelle sans ambiguïté et ne tient compte que des informations pertinentes pour résoudre le problème.
- Test-but : fonction applicable à chaque état qui détermine si cet état est solution.
- Coût-chemin : détermine le coût d'une succession d'actions.
- La meilleure solution est la résolution la moins coûteuse.

# Exemple : le jeu d'échecs



# Le problème des cruches

## Enoncé :

Soient deux cruches de 3 et 4 litres, et de l'eau en quantité illimitée. Comment faire pour obtenir un volume de deux litres ?

## Travail à réaliser :

- 1 Choisir un formalisme permettant de représenter les états possibles
- 2 Identifier, dans ce formalisme, l'état initial et le(s) état(s) final(ux)
- 3 Définir l'ensemble des opérateurs en précisant les prémisses, les contraintes et les conséquences
- 4 Etablir la liste des actions permettant d'atteindre une solution

## Variante :

Soient trois cruches non graduées de 3, 5 et 8 litres, et de l'eau en quantité illimitée. Comment faire pour obtenir un volume de 4 litres dans les cruches de 5 et 8 litres, si au départ les cruches de 3 et 5 litres sont vides et celle de 8 litres est pleine ?

# Le problème des cruches

1. Choisir un formalisme permettant de représenter les états possibles
2. Identifier, dans ce formalisme, l'état initial et le(s) état(s) final(ux)

# Le problème des cruches

3. Définir l'ensemble des opérateurs en précisant les prémisses, les contraintes et les conséquences

# Le problème des cruches

4. Etablir la liste des actions permettant d'atteindre une solution

# Le problème des cruches

## 5. Arbre de recherche

# Problèmes classiques

- Les cruches
- Le taquin
- Les tours de Hanoï
- Le loup, la chèvre et le chou
- Les missionnaires et les cannibales
- Le voyageur de commerce
- La crypt-arithmétique
- Les mots croisés
- Les huit reines
- Les dominos
- La fausse pièce
- Le nim
- Les jeux d'échecs, de dames, ...
- Les blocs

# Problèmes réels

- Recherche de parcours
  - itinéraires automatiques, guidage routier, planification de routes aériennes, routage sur réseaux informatiques...
- Robotique
  - assemblage automatique, navigation autonome...
- Planification et ordonnancement
  - horaires, organisation de tâches, allocation de ressources...
- Configuration de circuits VLSI

# Le problème du loup, de la chèvre et du chou

## Enoncé :

Un loup, une chèvre et un chou se trouvent sur la rive gauche d'une rivière. Comment les faire traverser de l'autre côté à l'aide d'une barque ne contenant que deux places (dont une pour le fermier) sachant que si le loup et la chèvre restent sur la même rive la chèvre sera mangée et que si la chèvre et le chou restent ensemble c'est le chou qui sera mangé ?

## Travail à réaliser :

- 1 Choisir un formalisme permettant de représenter les états possibles
- 2 Identifier, dans ce formalisme, l'état initial et le(s) état(s) final(ux)
- 3 Définir l'ensemble des opérateurs en précisant les prémisses, les contraintes et les conséquences
- 4 Etablir la liste des actions permettant d'atteindre une solution

# Le problème des missionnaires et les cannibales

## Enoncé :

Trois missionnaires et trois cannibales sont sur un même côté d'une rivière qu'ils doivent traverser à l'aide d'une barque à deux places nécessitant au moins une personne pour la manipuler. Si sur un côté (et la barque est toujours considérée comme étant sur un des côtés) il y a strictement plus de cannibales que de missionnaires, ces derniers sont mangés. Comment faire traverser tout le monde sans que les missionnaires soient mangés ?

## Travail à réaliser :

- 1 Choisir un formalisme permettant de représenter les états possibles
- 2 Identifier, dans ce formalisme, l'état initial et le(s) état(s) final(ux)
- 3 Définir l'ensemble des opérateurs en précisant les prémisses, les contraintes et les conséquences
- 4 Etablir la liste des actions permettant d'atteindre une solution

# Le problème de la traversée du pont

## Enoncé :

Quatre personnes doivent traverser un pont en 17 minutes. Chacune d'entre elles marche à une vitesse maximale donnée. Une personne peut traverser le pont en 1 minute, une autre en 2 minutes, une autre en 5 minutes et la dernière en 10 minutes. Il est impossible de traverser le pont sans torche, et ces quatre personnes n'ont en tout qu'une torche. Le pont ne peut supporter que le poids de 2 personnes. On cherche à déterminer l'ordre dans lequel ces quatre personnes doivent traverser.

## Travail à réaliser :

- 1 Choisir un formalisme permettant de représenter les états possibles
- 2 Identifier, dans ce formalisme, l'état initial et le(s) état(s) final(ux)
- 3 Définir l'ensemble des opérateurs en précisant les prémisses, les contraintes et les conséquences
- 4 Etablir la liste des actions permettant d'atteindre une solution

# Le problème des tours de Hanoï

## Enoncé :

Dans ce solitaire, on dispose d'anneaux de tailles toutes différentes et à tout moment, au dessus de tout anneau, il n'y a que des anneaux plus petits. Comment amener tous les disques à l'unique position légale où tous les disques se trouvent sur l'axe de gauche à celle où tous se trouvent sur l'axe de droite ?

## Travail à réaliser :

- 1 Choisir un formalisme permettant de représenter les états possibles
- 2 Identifier, dans ce formalisme, l'état initial et le(s) état(s) final(ux)
- 3 Définir l'ensemble des opérateurs en précisant les prémisses, les contraintes et les conséquences
- 4 Etablir la liste des actions permettant d'atteindre une solution

# Le problème de l'atelier

## Enoncé :

On considère une chaîne de production de voitures :

La chaîne de production est composée de 4 ateliers A, B, C et D.

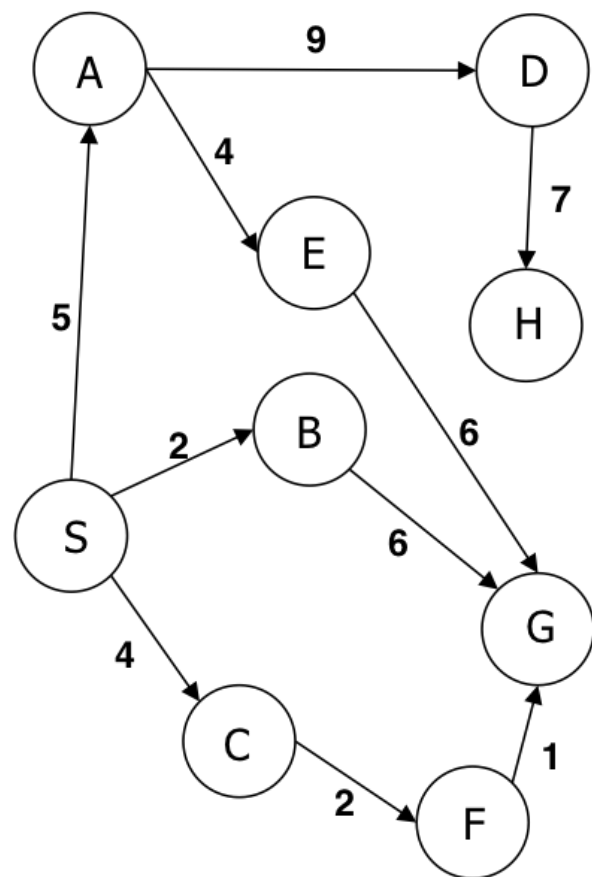
Chaque voiture doit passer successivement par l'atelier A, puis le B, puis le C et enfin le D.

Chaque atelier a une capacité de production à l'heure, qui dépend du nombre d'opérateurs qui lui sont affectés : l'atelier A peut traiter 5 voitures par opérateur et par heure, l'atelier B, 3, l'atelier C, 2 et l'atelier D, 3.

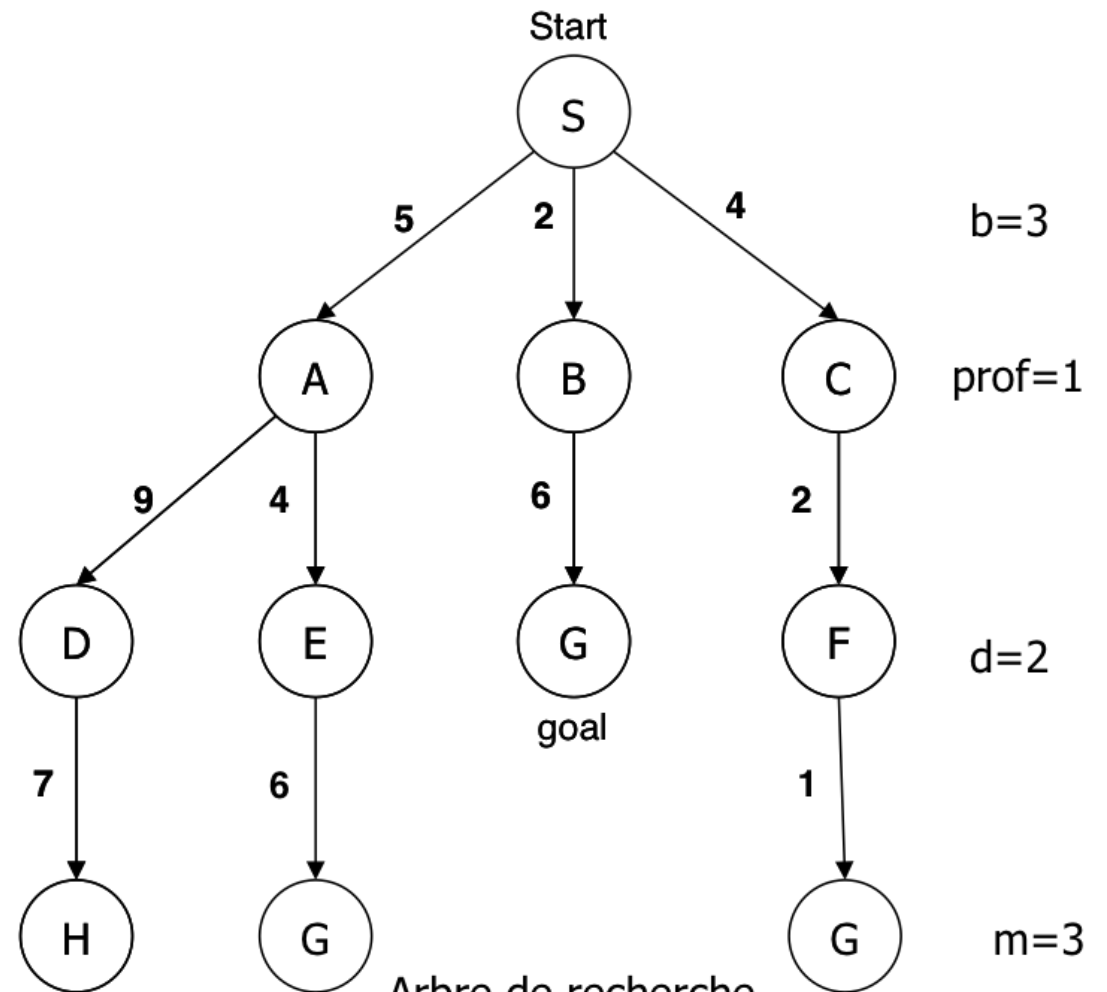
il y a 4 opérateurs : l'opérateur 1 travaille de 8 h à 13 h, l'opérateur 2 de 8h à 13h, l'opérateur 3 de 9h à 14h et l'opérateur 4 de 10h à 15h.

au départ, la chaîne de production est dans un état initial donné par le nombre de produits en attente pour chaque atelier, par exemple: à 8 heures, il y a 30 voitures en attente de l'atelier A, et 0 voiture en attente des autres ateliers.

# L'espace d'états et l'arbre de recherche

**Du monde**

Espace d'états

**De la recherche**

Arbre de recherche

# Méthodes de recherche

- Critères d'évaluation
- Méthodes de recherche simple (aussi appelée non informée)
  - en largeur
  - en coût uniforme
  - en profondeur
  - en profondeur limitée
  - par approfondissement itératif
  - bi-directionnelle
- Méthodes de recherche heuristique
  - du meilleur d'abord
    - gourmande
    - Algorithmes A, A\*, IDA\*, SMA\*

# Critères d'évaluation

Les différentes méthodes de recherche sont évaluées selon les critères suivants:

- Complétude (trouve une solution si elle existe)
- Optimalité (trouve la meilleure solution)
- Complexité en temps
- Complexité en espace mémoire

Ces critères vont dépendre de :

- $b$  = facteur de branchement maximum de l'arbre de recherche
- $d$  = profondeur à laquelle se trouve le meilleur nœud-solution
- $m$  = profondeur maximum de l'espace de recherche (parfois)

# Idée de base

- Exploration simulée de l'espace d'états en générant les états successeurs des états déjà explorés
- fonction recherche-generale (*probleme*, *strategie*) rend *solution* ou *echec*
  - initialiser l'arbre de recherche par l'*état initial* du problème
  - boucle
    - si il n'y a pas de candidats pour l'expansion alors rendre *echec*
    - sinon choisir une *feuille* pour l'expansion en fonction de la *strategie*
    - si le *nœud* contient un *état final* alors rendre la *solution*
    - sinon étendre le *nœud* et ajouter les *nœuds fils* dans l'arbre
  - fin
- Les méthodes de recherche diffèrent les unes des autres selon l'ordre dans lequel les nœuds sont explorés.

# Recherche en largeur d'abord

- Stratégie : Etudier parmi la liste des états à étudier (nommée OUVERT) celui qui est le moins profond (le premier en cas d'égalité)
- Implantation : Insérer les états successeurs à la fin de la liste des états à étudier
- fonction recherche-largeur (*probleme*) rend *solution* ou *echec*
- recherche-generale (*probleme*, *insere-a-la-fin*)

# Exemple de la complexité en largeur d'abord

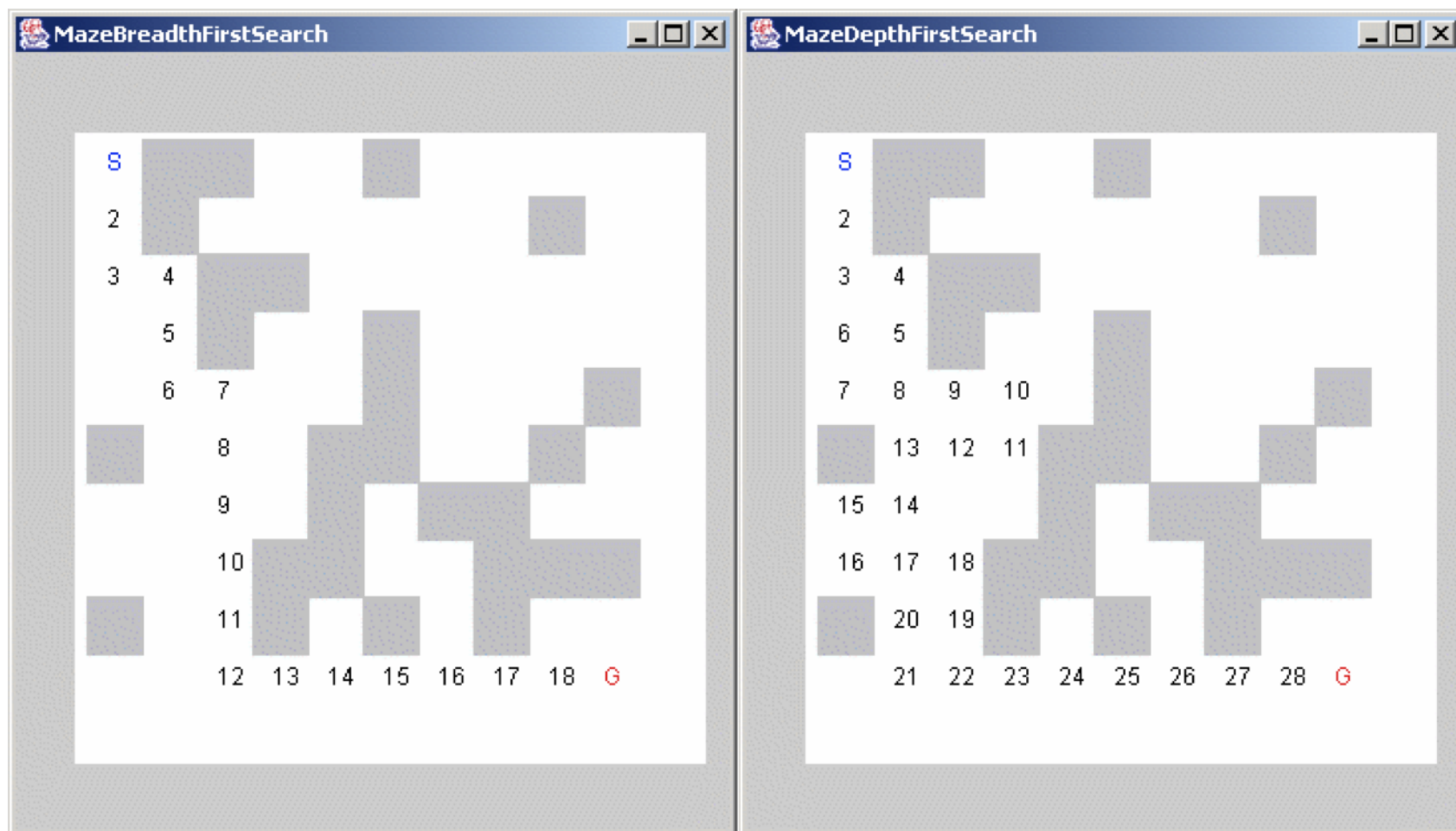
| Profondeur | Nœud      | Temps      | Mémoire    |
|------------|-----------|------------|------------|
| 0          | 1         | 1 ms       | 100 octets |
| 1          | 11        | 11 ms      | 1.1 Ko     |
| 2          | 111       | 111 ms     | 11.1 Ko    |
| 4          | $10^4$    | 11 s       | 1 Mo       |
| 6          | $10^6$    | 18 minutes | 111 Mo     |
| 8          | $10^8$    | 31 heures  | 11 Go      |
| 10         | $10^{10}$ | 128 jours  | 1 To       |
| 12         | $10^{12}$ | 35 ans     | 111 To     |
| 14         | $10^{14}$ | 3500 ans   | 11 111 To  |

# Recherche en profondeur

- Stratégie : étendre le nœud le plus profond
- Implantation : insertion des successeurs en tête de la file d'attente
- fonction recherche-profondeur (*probleme*) rend *solution* ou *echec*
- recherche-generale (*probleme*, *insere-en-tete*)

# Comparaison recherche en largeur d'abord Vs Profondeur d'abord

Ordre des actions : gauche, droite, haut, bas



# Recherche en profondeur limitée

- Stratégie : étendre le nœud le plus profond si sa profondeur d'exploration  $L$  est inférieure à la limite sinon passe au suivant
- Implantation : insertion des successeurs en tête de la file d'attente si  $L < \text{limite}$
- fonction recherche-profondeur-limite (*probleme, limite*) rend *solution* ou *echec*
- recherche-generale (*probleme, insere-en-tete-si-inf-limite*)

# Recherche par approfondissement itératif

- Stratégie : Essayer toutes les valeurs possibles de limite à partir de  $\text{limite} = 1$
- Implantation : insertion des successeurs en tête de la file d'attente
- si  $L < \text{limite}$  avec  $L$  variant de 0 à l'infini
- fonction recherche-profondeur-iteratif (*probleme*) rend *séquence de solutions*
- pour *limite* 1 à  $\infty$  faire
- *resultat* recherche-profondeur-limite (*probleme*, *limite*)
- si *resultat cutoff* alors rendre *resultat*
- fin
- Le problème avec la recherche en profondeur limitée est de fixer la bonne valeur de limite
- Il combine les avantages de la recherche en largeur et en profondeur
  - optimal et complet comme la recherche en largeur
  - économe en espace comme la recherche en profondeur
- C'est l'algorithme de choix si l'espace de recherche est grand et si la profondeur est inconnue

# Recherche bi-directionnelle

- Stratégie : effectuer une recherche par chaînage avant à partir de l'état initial et une recherche par chaînage arrière à partir de l'état final. Arrêter la recherche lorsque les deux processus se rencontrent

# Recherche en coût uniforme

- Stratégie : étendre le nœud de coût le plus faible
- Implantation : insertion des successeurs dans l'ordre croissant des coûts de chemin
- fonction recherche-coût-uniforme (*probleme*) rend *solution* ou *echec*
- recherche-generale (*probleme, insere-dans-ordre-des-couts*)

# Comparaison

| Critères   | Largeur                        | Coût uniforme | Profondeur                         | Profondeur limitée    | Profondeur itérative           | Bi-directions |
|------------|--------------------------------|---------------|------------------------------------|-----------------------|--------------------------------|---------------|
| Complétude | Oui                            | Oui           | Non<br>sauf esp. fini<br>acyclique | Oui,<br>si $L \geq d$ | Oui                            | Oui           |
| Optimalité | Oui,<br>si coûts<br>identiques | Oui           | Non                                | Non                   | Oui,<br>si coûts<br>identiques | Oui           |
| Temps      | $b^d$                          | $b^d$         | $\underline{b^m}$                  | $\underline{b^L}$     | $b^d$                          | $b^{d/2}$     |
| Mémoire    | $b^d$                          | $b^d$         | $\underline{bm}$                   | $\underline{bL}$      | $bd$                           | $b^{d/2}$     |

## II- Recherche heuristique

Les algorithmes de recherche simple n'exploitent aucune information concernant la structure de l'arbre de recherche ou la présence potentielle de nœuds-solution pour optimiser la recherche

La plupart des problèmes réels sont susceptibles de provoquer une explosion combinatoire du nombre d'états possibles

Un algorithme de recherche heuristique utilise l'information disponible pour rendre le processus de recherche plus efficace.

Une information heuristique est une règle ou une méthode qui presque toujours améliore un processus de décision.

# Fonction heuristique

## Définition

Une fonction heuristique  $h : E \rightarrow \mathbb{R}$  ( $E$ , espace d'états) fait correspondre à un état  $s \in E$  un nombre  $h(s) \in \mathbb{R}$  qui est (généralement) une estimation du rapport coût/bénéfice qu'il y a à étendre le chemin courant en passant par  $s$ .

$h(s) \geq 0$ , pour tout état  $s$

$h(s) = 0$ , implique que  $s$  est un état-but

$h(s) = \infty$ , implique que  $s$  est un état ne permettant pas d'atteindre un but

Une heuristique est spécifique à un problème. Elle utilise les connaissances du domaine afin d'orienter la recherche pour la rendre plus efficace.

# Qualités des heuristiques

## Facteur de branchement effectif

soit  $N$  = nombre total d'états produits pour obtenir la solution

soit  $d$  = profondeur à laquelle la solution a été trouvée

alors  $b^*$  est le facteur de branchement effectif d'un arbre fictif parfaitement équilibré tel que :

$$N = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

Une bonne fonction heuristique aura une valeur de  $b^*$  proche de 1

## Dominance

Soient deux fonctions heuristiques  $h_1(n)$  et  $h_2(n)$

Si  $h_2(n) \geq h_1(n)$ ,  $\forall n$  alors on dit que  $h_2$  domine  $h_1$  et produira une recherche plus efficace

## Admissibilité

Une fonction heuristique est admissible si elle ne surestime jamais le coût réel i.e. si  $\forall n$ ,  $h(n) \leq h^*(n)$  avec  $h^*(n)$  = coût réel depuis  $n$  au but.

# Recherche du meilleur d'abord

## Stratégie

Etendre le nœud le plus prometteur au regard d'une fonction d'évaluation du coût total  $f(n)$  qui tient compte de l'heuristique.

## Implantation

insérer les successeurs en ordre croissant de leur valeur  $f(n)$ .

## Fonctionnement

Combinaison entre la recherche en profondeur et en largeur

P: solution trouvée sans avoir à calculer tous les nœuds

L: ne risque pas de rester dans un cul-de-sac

## Cas particuliers

recherche gourmande

algorithmes A et A\*

# Recherche gourmande

## Stratégie

Minimiser le coût estimé pour atteindre le but.  $f(s) = h(s)$

## Implantation

Toujours enlever le plus grand morceau du coût restant pour atteindre le but

## fonctions heuristiques classiques

- distance à vol d'oiseau
- distance de Manhattan

Relativement efficace bien que pas toujours optimal

# Les algorithmes A et A\*

La **recherche gourmande** minimise le coût estimé  $h(n)$  du nœud  $n$  au but réduisant ainsi considérablement le coût de la recherche, mais il n'est pas optimal et pas complet. La **recherche en coût uniforme** minimise le coût  $g(n)$  depuis l'état initial au nœud  $n$ , il est optimal et complet, mais pas très efficace.

## Fonctionnement

Combiner la recherche gourmande et la recherche en coût uniforme, et rechercher le coût total  $f(n)=g(n)+h(n)$  du chemin passant par le nœud

## Théorème

Si A utilise une fonction heuristique admissible alors A est optimal. Dans ce cas, l'algorithme est nommé A\*.

## Stratégie

étendre le nœud dont la valeur  $f(s) = g(s) + h(s)$  est la plus faible

# Algorithme A (et A\*)

Soient  $s_0$  l'état initial, **OUVERTS** la liste des noeuds à étudier, **FERMES** la liste des noeuds étudiés et **tete(OUVERTS)** la fonction qui renvoie le premier élément de la liste OUVERTS

$\text{OUVERTS} \leftarrow \{s_0\}$ ,  $\text{FERMES} \leftarrow \emptyset$ ,  $\text{pere}(s_0) \leftarrow -$ ,  $g(s_0) \leftarrow 0$ ,  $s \leftarrow s_0$

**Tant que**  $\text{OUVERTS} \neq \emptyset$  et  $\text{tete}(\text{OUVERTS})$  non terminal **Faire**  
supprimer  $s$  de  $\text{OUVERTS}$  et le mettre dans  $\text{FERMES}$

**Pour tout** noeud  $s'$  successeur de  $s$  **Faire**

**Si**  $s' \notin (\text{OUVERTS} \cup \text{FERMES})$  **ou**  $g(s') > g(s) + \text{cout}(s, s')$  **Alors**

$g(s') \leftarrow g(s) + \text{cout}(s, s')$

$f(s') \leftarrow g(s') + h(s')$

$\text{pere}(s') \leftarrow s$  ranger  $s'$  dans  $\text{OUVERTS}$  dans l'ordre  $f$  croissant

**Si**  $s' \in \text{FERMES}$  **Alors**

$\text{FERMES} \leftarrow \text{FERMES} \cup \{s'\}$

**Fin pour**

**Si**  $\text{OUVERTS} = \emptyset$  **Alors**  $s \leftarrow \text{tete}(\text{OUVERTS})$

**Fin tant que**

**Si**  $\text{OUVERTS} = \emptyset$  **Alors** le problème n'admet pas de solution

**Sinon** le chemin( $s$ ) est solution (avec  $\text{chemin}(s) : \text{chemin}(\text{pere}(s)), s$ )

# Variantes de A\*

Pour certains problèmes réels, l'espace de recherche est très grand. Même les méthodes de recherche avec heuristique connaissent des problèmes de place mémoire.

2 variantes de A\* économes en place mémoire :

- **IDA\* = A\* avec approfondissement itératif (IDS + A\*)**

- Algorithme de type Profondeur d'abord utilisant une limite de recherche basée sur  $f(n) = g(n) + h(n)$
- Au départ  $\text{lim} = f(\text{état initial})$ . Les nœuds  $n$  où  $f(n) > \text{lim}$  ne sont pas pris en considération.
- Si un état final n'est pas atteint, l'algorithme est appliqué de nouveau avec pour limite la valeur  $f(n)$  la plus petite rencontrée pour les successeurs non évalués.

- **SMA\* = A\* avec gestion de la mémoire**

- La taille de la liste des nœuds à étudier est limitée pour limiter l'espace mémoire
- Si un successeur a une valeur  $f(n)$  inférieure à celles stockées, il est inséré et le nœud avec la valeur  $f(n)$  la plus grande est retiré.

## En résumé

- **La recherche du meilleur d'abord** est une méthode de recherche générale où les nœuds de coût minimum sont explorés en premier.
- **La recherche gourmande** utilise une estimation du coût minimum au but pour réduire le temps de recherche mais il n'est ni complet ni optimal.
- **La recherche A** combine la recherche en coût uniforme et la recherche gourmande :  $f(n) = g(n) + h(n)$ . L'heuristique ne doit jamais surestimer le coût réel i.e qu'elle doit être admissible pour que l'algorithme soit optimal (on parle alors de A\*).
- A\* est complet, optimal et très rapide mais demande beaucoup de mémoire.
- IDA\* et SMA\* réduisent l'espace mémoire utilisé
- La qualité de l'heuristique influence la durée de la recherche