Robotique 2

Raja Chatila Raja.Chatila@laas.fr

LAAS-CNRS, Toulouse
ISIR-UPMC, Paris

Introduction

- Robotique: *"lien intelligent entre la perception et l'action"*
- Déjà vu (M1):
  - Fonctions sensori-motrices
  - Navigation (cartographie, localisation)
  - Planification des mouvements
- Ce cours:
  - Qu'est ce qu'un agent intelligent?
  - Comment planifier des actions pour atteindre des objectifs?
  - Comment apprendre de nouvelles capacités?

Contenu du cours

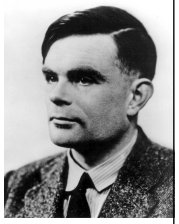
- Qu'est-ce que l'IA?
- Agents intelligents; Architectures.
- Algorithmes de recherche dans les espaces d'états
- Représentations des connaissances
- Logique des propositions et des prédicats du 1er ordre
- Planification d'actions
- Raisonnement dans l'incertain, Processus décisionnels de Markov (MDP)
- Apprentissage et apprentissage par renforcement

3

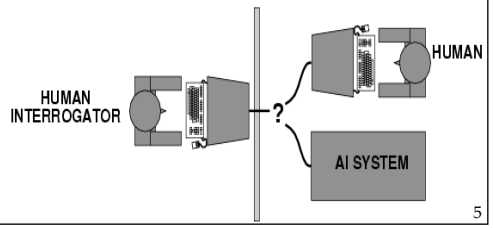
Qu'est-ce que l'IA?

4

Historique



- Alan Turing (1950) "*Computing machinery and intelligence*": Les machines peuvent-elles penser?
- Ou : Les machines peuvent-elles se comporter "intelligemment"?
- Test: le jeu de l'imitation (test de Turing).
- A prévu qu'en 2000, une machine pourrait avoir une chance sur trois de faire illusion pendant 5 minutes.



Naissance de l'IA

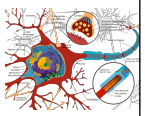
- Conference à Dartmouth College (NH, USA) 1956.
- Définition: Programmes informatiques qui résolvent des problèmes habituellement résolus par des processus mentaux de haut niveau chez les humains (John McCarthy)

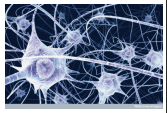
Deux approches

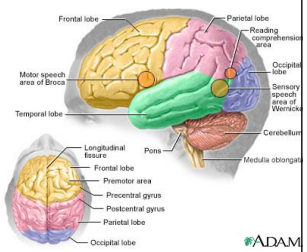
- **Imiter** l'intelligence humaine. Les machines (ordinateurs, robots) doivent *raisonner*/*agir* comme les humains.
- Agir **rationnellement**. Les machines (ordinateurs, robots) doivent *raisonner*/*agir* rationnellement.

Penser comme les humains

- Sciences cognitives et neurosciences cognitives. Théorie de l'esprit.
- Modèles de fonctionnement du cerveau.
- Validation à partir de prévision et d'études/tests du comportement humain et à partir de données neurologiques (IRM,...).







Penser et agir rationnellement

- Aristote et les règles du raisonnement correct. Syllogismes: "Tous les hommes sont mortels; Socrate est un homme; donc Socrate est mortel"
- La *logique mathématique (XIXe, XXe* siècles): règles et notations. A permis la mécanisation du raisonnement.
- Mais: tout comportement intelligent n'est pas forcément issu d'un raisonnement logique (ex: mécanismes sensori-moteurs).
- Agir rationnellement: "Faire la chose correcte". *Optimiser un critère donné compte tenu de l'information disponible.*

Domaines de l'IA

- Résolution de problèmes - recherche de solutions
- Représentation des connaissances
- Raisonnement logique; agents logiques
- Planification d'actions
- Architectures intelligentes
- Raisonnement sur l'incertain
- Raisonnement sur le temps
- Apprentissage
- Langues naturelles
- Perception
- Robotique

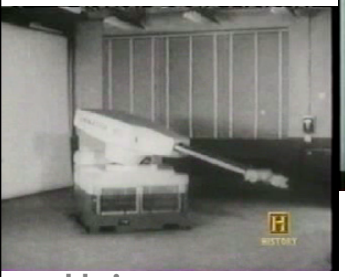
Situation de l'IA

- Les technique de l'IA sont diffusives (et invisibles) en informatique.
- Exemples marquants: Deep Blue vs. Garry Kasparov (1997); Jeopardy! (2011)
- Preuve de théorèmes
- Systèmes experts
- Planification de missions spatiales ou militaires
- Robotique

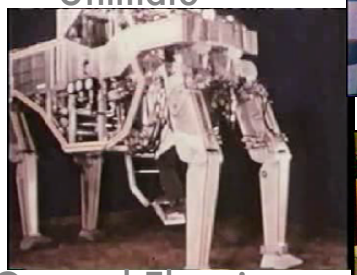
La robotique

- Paradigme de l'IA
- Accent mis sur la perception et le mouvement
- **Machines matérielles** dans le **monde réel**. Importance de la matérialisation (embodiment) et de la mise en situation.
- Importance du "lien intelligent" entre perception et action. Intégration de toutes les fonctions.
- Robots: agents rationnels/agents réactifs

Les débuts de la Robotique



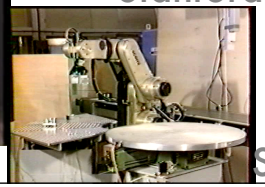
Unimate



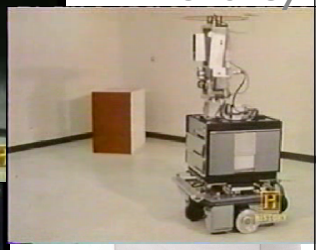
General Electric

Turtle

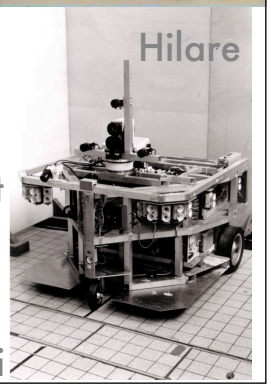
Stanford Cart



Scemi



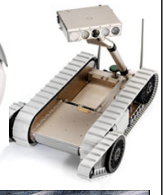
Shakey

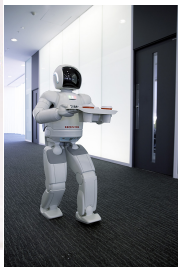


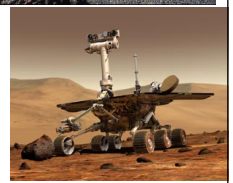
Hilare

13

Robots







14

Le robot: principales fonctions

15

Agents; Architectures

16

Agents

- Système possédant des capteurs et des actionneurs, percevant et agissant dans un environnement.
  - **Fonction**(s) *f: Percepts ---> Actions*
  - Mise en oeuvre d'une architecture organisant des fonctions.
  - Les actions modifient les percepts futurs.
- Agent *rationnel:* agent effectuant les actions de manière à maximiser une mesure de performance compte-tenu de possibilités d'action, de ses percepts et de ses connaissances.

Un robot interactif

Agents

- Agents: environments, Perception, Actions
- Rationalité, connaissances
- Mesure de performance
- Catégories d'agents

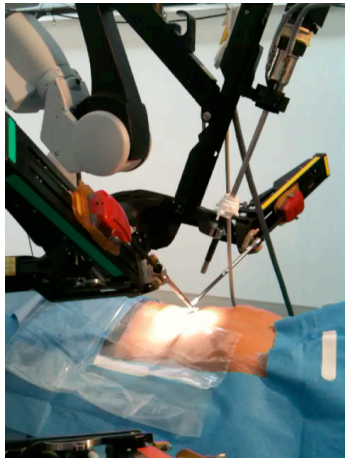
19

Conception

- La conception d'un agent (architecture, capteurs, effecteurs) dépend de:
  - l'environnement dans lequel il va agir,
  - des tâches qu'il doit y effectuer.
  - des performances qu'il doit atteindre

20

Différents agents



Environnements

- Totalement vs. partiellement observable: l'agent possède une connaissance complète/partielle de l'état du monde à chaque instant à travers ses capteurs.
- Deterministe vs. stochastique: L'évolution de l'état du monde est entièrement déterminée par l'état courant et l'action effectuée par l'agent ou incertaine. (*Stratégique* : si d'autres agents agissent).
- Episodique vs. séquentiel: épisodes atomiques {perception- une action choisie en fonction de la perception}.
- Statique vs. dynamique: L'environnement n'évolue pas pendant que l'agent délibère.
- Discret vs. continu: Nombre de percept et d'actions fini (et souvent faible).
- Mono-agent (vs. multiagent): L'agent est le seul acteur dans l'environnement.

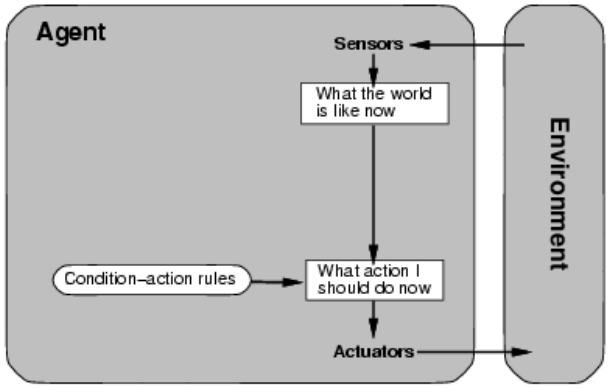
Exemples d'Environments

	Jeu d'échecs avec pendule	Jeu d'échecs sans pendule	Taxi autonome
Observable	OUI	OUI	Partiel
Déterministe	Stratégique	Stratégique	NON
Episodique	NON	NON	NON
Statique	Semi	OUI	NON
Discret	OUI	OUI	NON
Mono-agent	NON	NON	NON

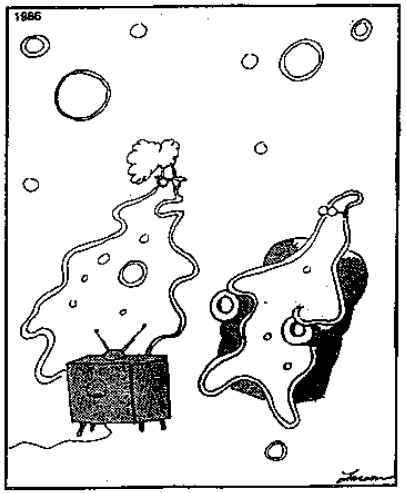
Types d'agents

- Quatre catégories (généralité et complexité croissantes):
  - Agents réflexifs simple
  - Agents réflexifs avec modèle
  - Agents orientés objectif
  - Agents orientés utilité.

Agent réflexif



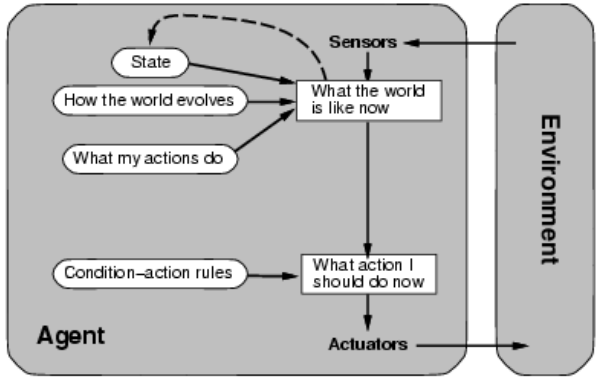
Agent réflexif



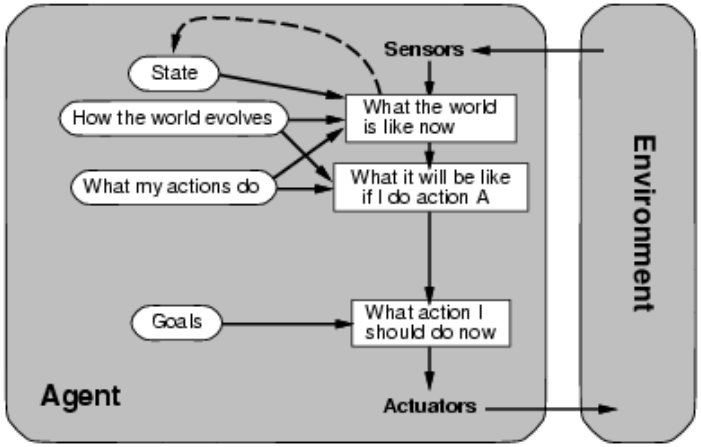
"Stimulus, response! Stimulus, response! Don't you ever *think*?"

Agent réflexif avec modèle

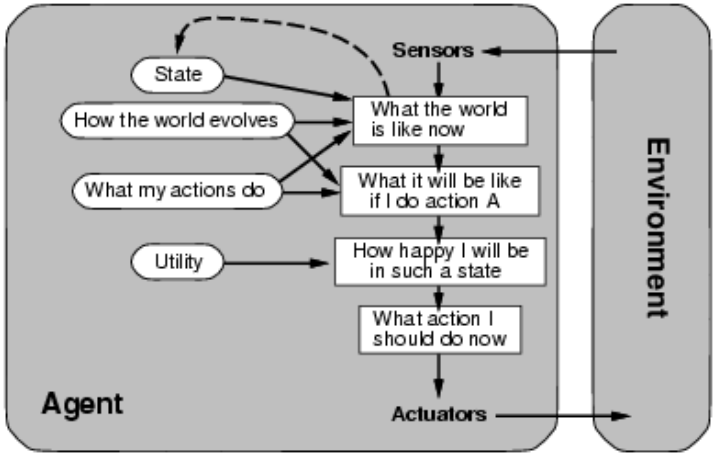
- Avec Modèle: état, évolution



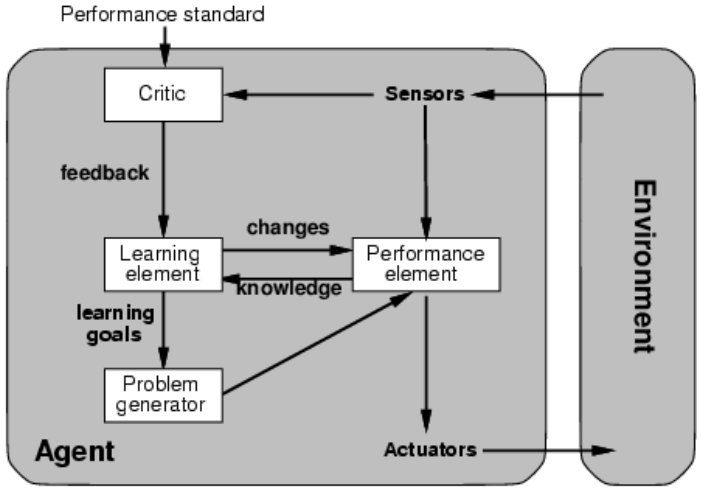
Agent orienté but



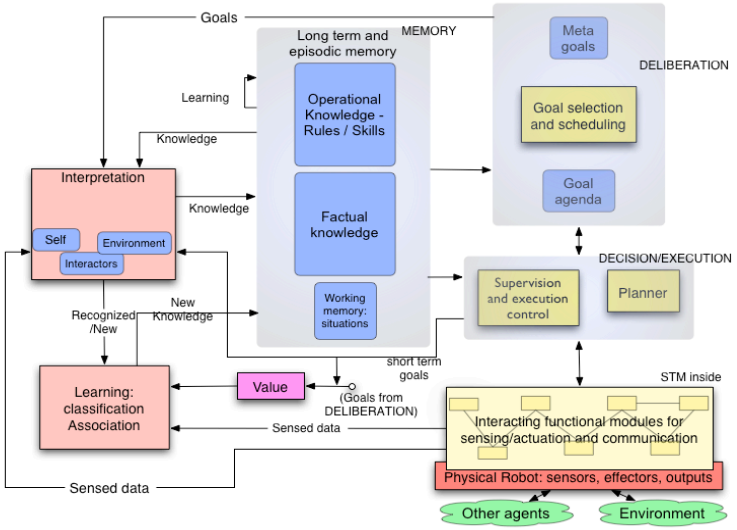
Agent orienté utilité



Agent apprenant



Agent cognitif



31

Résolution de problèmes
Algorithmes de Recherche

32

Plan

- Résolution de problèmes
- Formulation en espace d'états
- Algorithmes de recherche

33

Définition

- Un problème est défini par:

1. Un état initial
2. Un ensemble d'Actions ou une fonction successeur S(x) = ensemble de paires {action-état}
3. Test d'atteinte de but
   - explicite: être à un endroit donné, tenir un objet.
   - implicite: mat du roi adverse.
4. Coût (additif) du chemin entre état initial et but (somme de distances, nombre d'actions). Coût d'une étape positif ou nul

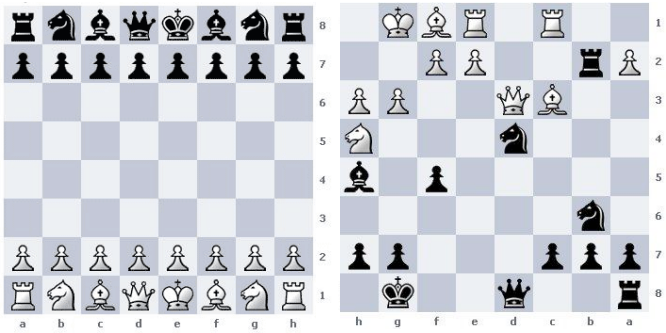
Une solution est une séquence d'actions joignant l'état initial à un état but.

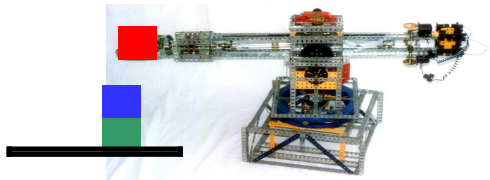
34

Espace d'états

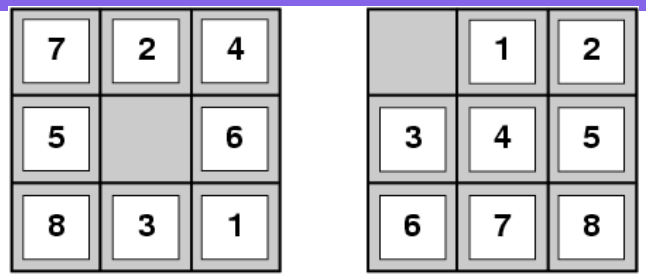
- Abstraction du monde réel: états (être dans une ville, dans une pièce, tenir un objet, position d'un objet, ...)
- Abstraction des actions réelles: actions plus ou moins complexes (prendre, poser aller-à, grimper, ...)

Exemple d'états





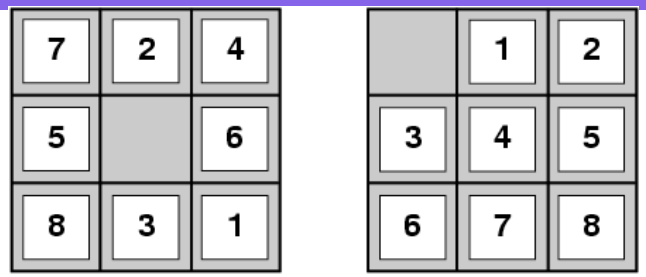
Exemple: jeu de taquin



- Etats?
- Actions?
- Test état but?
- Coût?

37

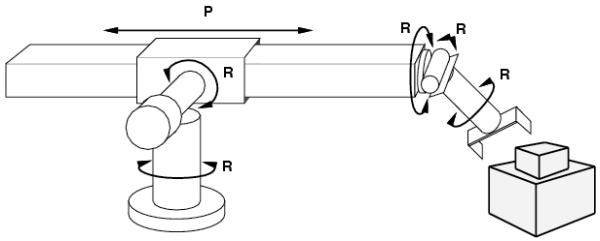
Exemple: jeu de taquin



- Etats? emplacement des carreaux
- Actions? g, d, h, b **du carreau vide**
- Test état but?positions finales des carreaux
- Coût? 1 par mouvement.

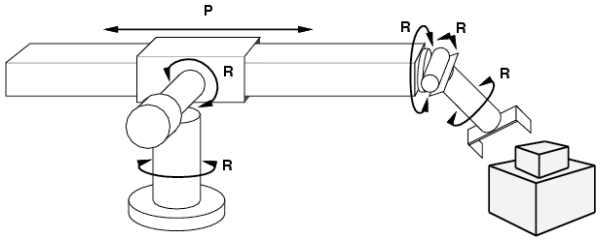
38

Exemple: robot manipulateur



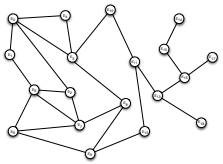
- Etats?
- Actions?
- Test état but?
- Coût?

Exemple: robot manipulateur



- Etats? configuration du bras, objets à assembler
- Actions? mouvements du bras
- Test état but? objet assemblé
- Coût? temps; nombre d'actions

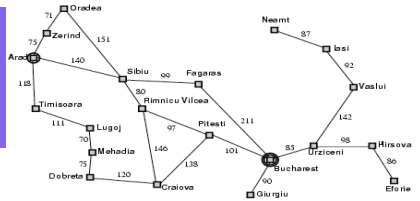
Algorithmes de recherche



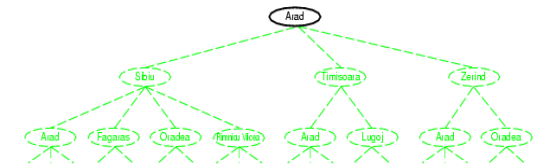
- Espace d'état: **graphe**
- Recherche arborescente: exploration de l'espace d'états par génération des successeurs des états explorés
-

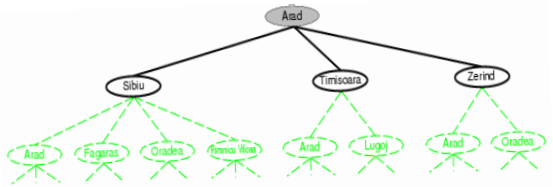
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
```


Exemple

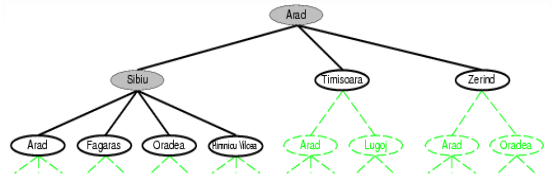


Graphe d'états

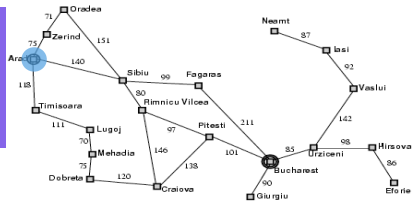




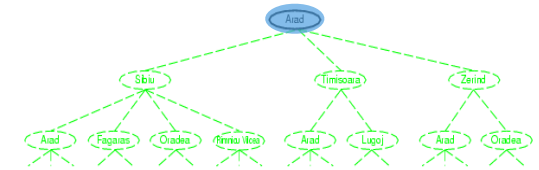
Arbre

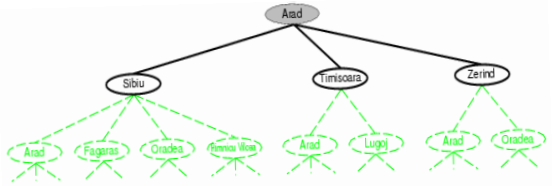


Exemple

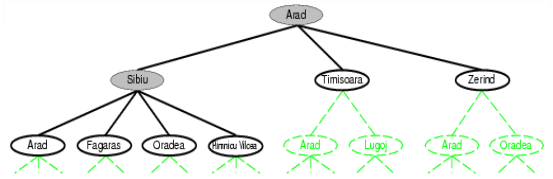


Graphe d'états



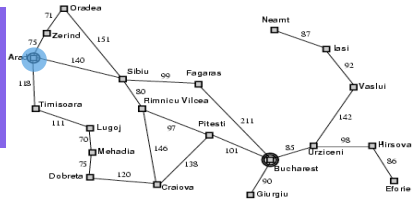


Arbre

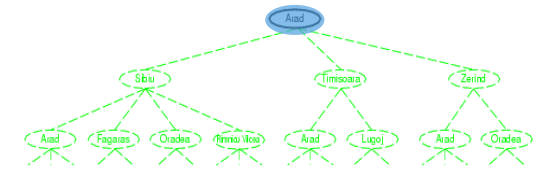


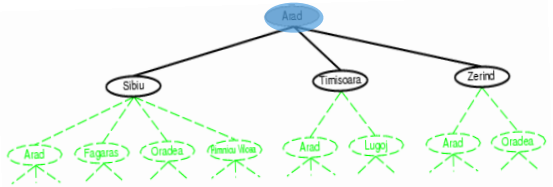
42

Exemple

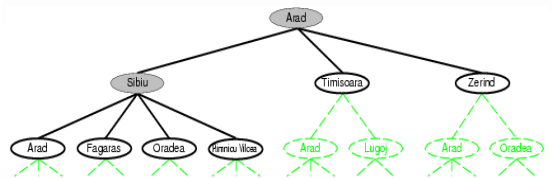


Graphe d'états



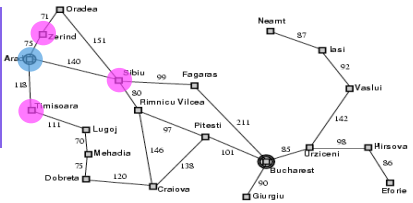


Arbre

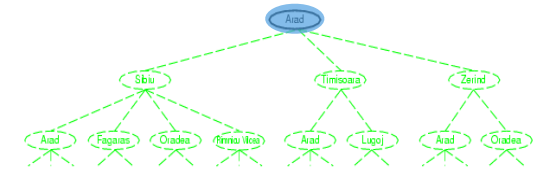


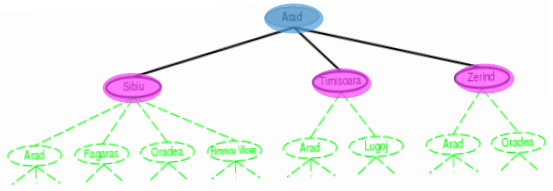
43

Exemple

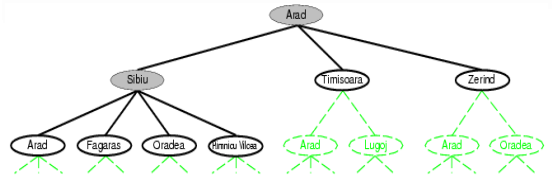


Graphe d'états



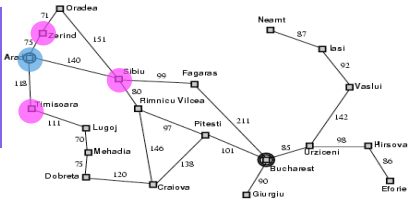


Arbre

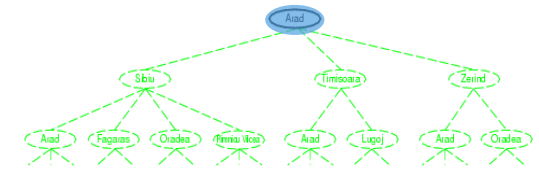


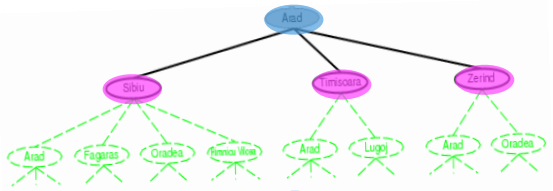
43

Exemple

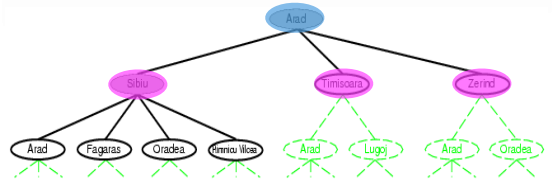


Graphe d'états



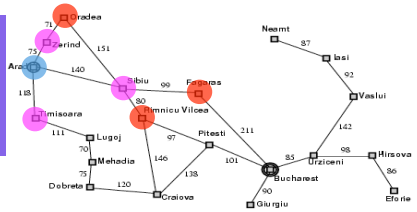


Arbre

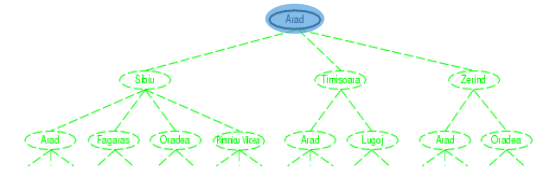


44

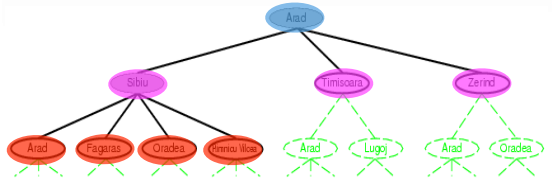
Exemple



Graphe d'états



Arbre



44

Exemple

Graphe d'états

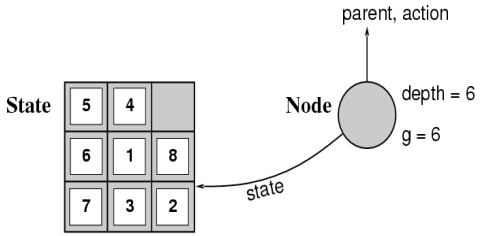
Arbre

44

44

Etats et Noeuds

- Un état est une représentation de la situation matérielle
- Un noeud est une *structure de données* constitutive de l'arbre de recherche, incluant l'état, le noeud parent, l'action, le coût du chemin g(x), la profondeur (itération).
- Le développement d'un noeud crée de nouveaux noeuds, avec les champs remplis.
-



Stratégies de recherche

- Une stratégie de recherche est définie par l'ordre de développement des noeuds. *Algorithme de recherche*.
- Une stratégie est évaluée par les propriétés de:
  - complétude : si une solution existe, est-elle toujours trouvée?
  - complexité en temps: nombre de noeuds générés
  - complexité en espace: nombre maximal de noeuds en mémoire
  - optimalité: la solution trouvée est-elle de moindre coût?

Complexité

- La complexité en temps et en espace sont mesurées par :
  - b: facteur de branchement (nombre d'états successeurs d'un état donné) maximal de l'arbre de recherche
  - d: profondeur de la solution de moindre coût
  - m: profondeur maximale de l'espace de recherche
- Fournie en ordre de grandeur de la taille du problème : O(.)

47

Algorithme général

```
Search( S, G_test, Criteria)/ /* Espace d'états fourni
implicitement par l'état initial et la fonction successeur */ /
Open = { S }; Closed = { };
repeat
        if (empty(Open)) return fail;
        select Node from Open using Criteria;
        if (Goal_test(Node)) return Node;
        for each Child of Node do
            if (Child not in Closed)
                Open ← Open + { Child };
        Closed ← Closed + { Node };
```


48

Algorithmes de recherche

- Algorithmes non informés
  - Largeur d'abord
  - Profondeur d'abord
- Algorithmes informés
  - Meilleur d'abord
  - A*

Algorithmes non informés

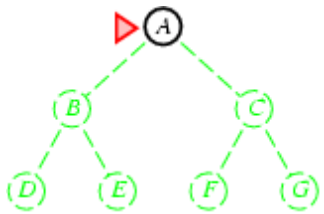
- Aucune connaissance sur le problème à part l'espace d'état et le coût de développement.

50

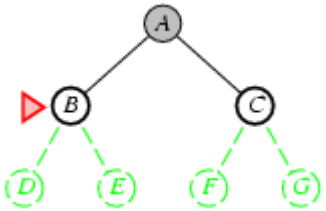
Largeur (Breadth)

Recherche systématique - critère : distance au prédécesseur

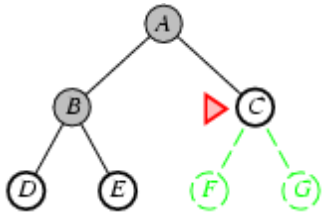
```
Search( Start, Goal_test, Criteria )
    Open: fifo_queue;
    Closed: hash_table;
    enqueue(Start, Open);
    insert(Start, Closed)
    repeat
        if (empty(Open)) return fail;
        Node = dequeue(Open);
        if (Goal_test(Node)) return Node;
        for each Child of Node do
            if (not find(Child, Closed))
                enqueue(Child, Open)
                insert(Child, Closed)
```

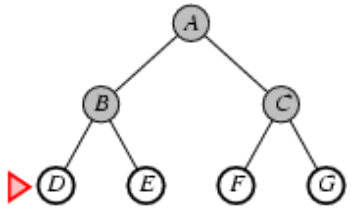
1



2



3



4

Propriétés

- Complet? si b est fini
- Complexité en Temps: $1+b+b^2+b^3+\ldots +b^d+ b(b^d-1) = O(b^{d+1})$
- Complexité en Espace: $O(b^{d+1})$ (garde tous les noeuds en mémoire)
- Optimalité si le coût est uniforme pour chaque pas (coût = 1)

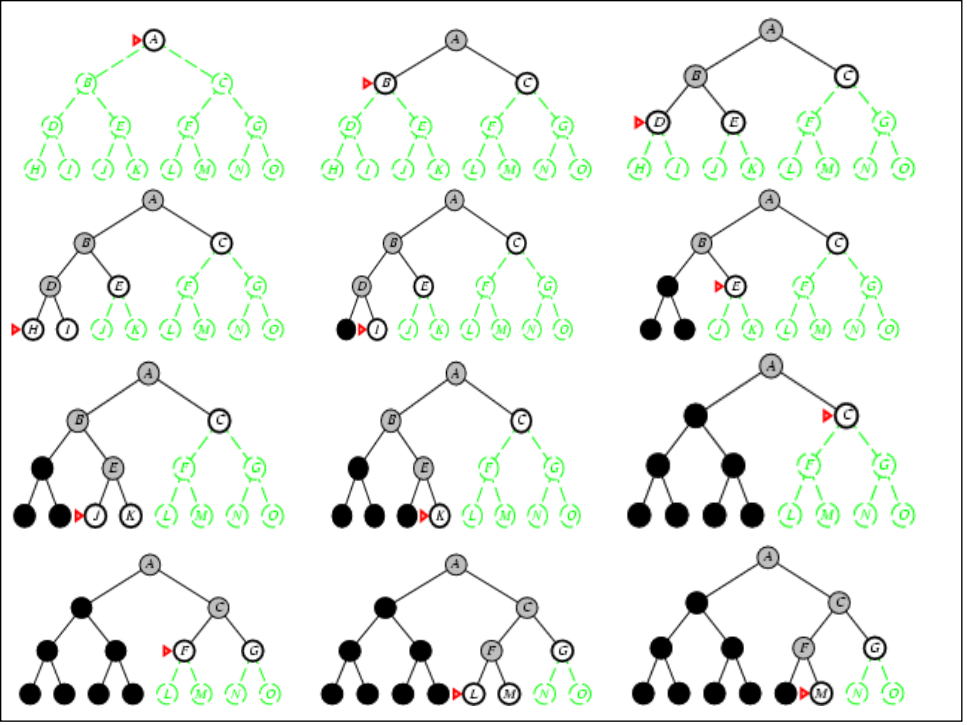
53

Profondeur (Depth)

Critère: distance la plus longue du départ

```
Search( Start, Goal_test, Criteria )
    Open: stack;
    Closed: hash_table;
    push(Start, Open);
    insert(Start, Closed)
    repeat
        if (empty(Open)) return fail;
        Node = pop(Open);
        if (Goal_test(Node)) return Node;
        for each Child of node do
            if (not find(Child, Closed))
                push(Child, Open)
                insert(Child, Closed)
```


54



Propriétés

- Complétude Non: échoue dans les espaces de profondeur infinie et les espaces avec cycles (il y a des versions améliorées). Complétude dans espaces finis.
- Complexité en temps: $O(b^m)$: très mauvais si $m >> d$
  - Peut être plus rapide que "largeur"selon la densité de la structure de l'espace.
- Complexité en espace? $O(bm)$, (linéaire).
- Optimalité: Non.

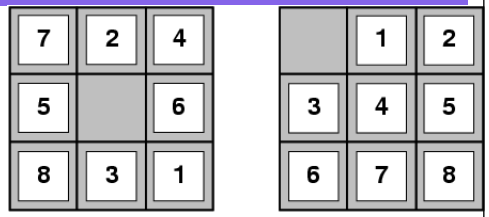
Stratégies informées

- Fonction d'évaluation *f(n)* pour choisir le noeud à développer
- Utilisation d'une *heuristique*. Objectif: réduction de la complexité.

Heuristique

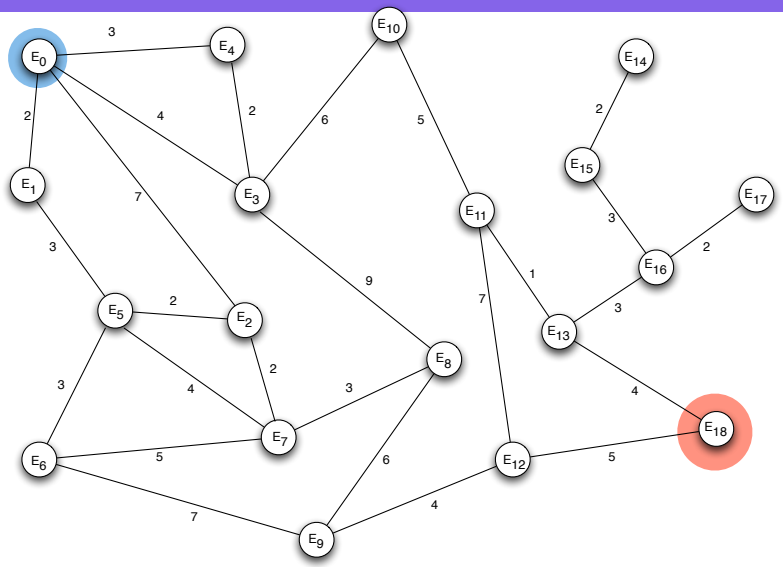
- Valeur numérique associée à un état donné et égale à l'*estimation* du coût restant pour atteindre un état but.
- Plusieurs fonctions possibles pour un problème donné. Souvent spécifique au problème
- Doit être à la fois discriminante (pour être utile) et simple à calculer (pour ne pas augmenter la complexité).

Exemples d'heuristique

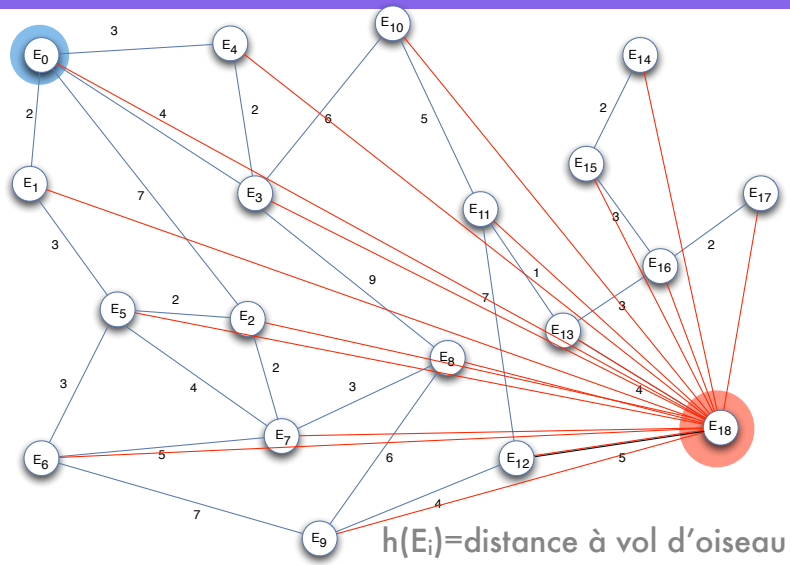


- Jeu de taquin:
  - $h_1(E)$: nombre de carreaux ne se trouvant pas à leur emplacement final. $h_1(E_0)=8$
  - $h_2(E)$: Distance Manhattan (nombre de cases entre l'emplacement courant et l'emplacement but pour chaque carreau). $h_2(E_0)=2+2+3+2+3+3+1+2=18$

Exemple d'heuristique



Exemple d'heuristique



h(Ei)=distance à vol d'oiseau au but

Propriétés

- Une heuristique est **admissible** si pour tout noeud n, **h(n) ≤ h*(n)** où h*(n) est le vrai coût pour atteindre le but à partir du noeud n. Ne surestime jamais le véritable coût.
- Un heuristique est **consistente** si pour tout noeud n et son successeur n', h(n) ≤ c(n,n') + h(n'), où c(n,n') est le coût de production de n' à partir de n.

Meilleur d'abord (Best first)

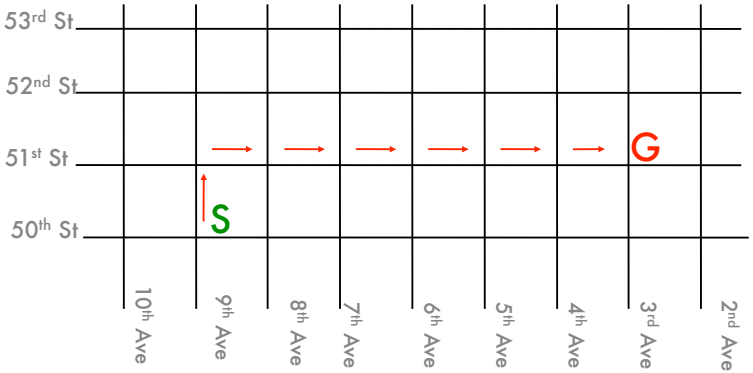
Critère: coût heuristique

```
Search( Start, Goal_test, Criteria )
    Open: priority_queue;
    Closed: hash_table;
    enqueue(Start, Open, heuristic(Start));
    repeat
        if (empty(Open)) return fail;
        Node = dequeue(Open);
        if (Goal_test(Node)) return Node;
        for each Child of node do
            if (not find(Child, Closed))
                enqueue(Child, Open, heuristic(Child))
                insert(Child, Closed)
```

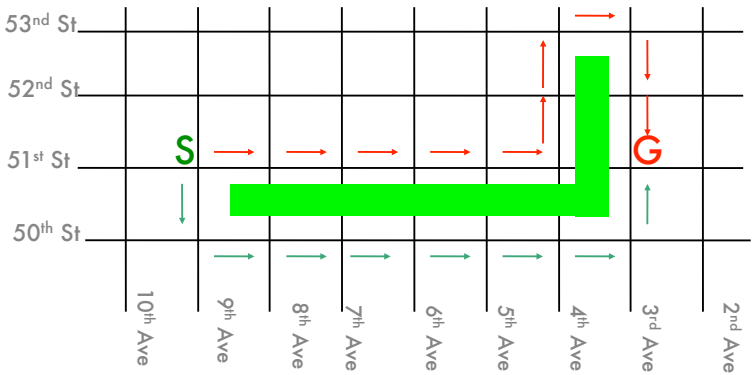

63

Exemple de Best-First

heuristique: Distance Manhattan (Δx+Δy)



Non-Optimalité de Best-First



A*

Critère de choix d'un noeud: minimiser

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

$g(n)$= distance depuis l'état initial à n

$h(n)$= heuristique : estimation de la distance de n au but

66

Algorithme A* (*S*, G)

S: noeud origine; *G*: but; Variables : k(*p*,*n*),g(*n*), h(*n*), f(*n*)=g(*n*)+h(*n*), *nbest*

Début

Open = {*S*}; Closed = { };

Répéter

Si (vide(Open)) alors **échec**;

Sinon

Choisir *nbest* /*tête de Open*/;

Open ← Open - {*nbest*};

Closed ← Closed + {*nbest*};

Si *nbest* = G, Fournir le chemin : la séquence du noeud initial à G. **Exit**

Sinon

Développer *nbest*; Production de **Successeurs**(*nbest*) ensemble des noeuds successeurs de *nbest*.

Pour tout *x* dans Successeurs(*nbest*) qui n'est pas dans Closed

Si *x* n'est pas dans Open alors:

Open ← Open + {*x*};

Sinon

Si g(*nbest*) + k(*nbest*,*x*) < g(*x*) alors

Modifier parent de *x* à *nbest* ;

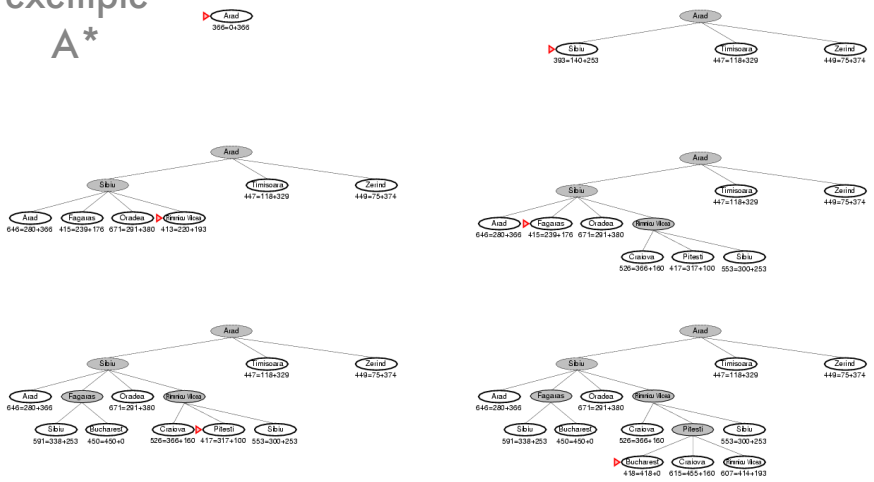
Remplacer g(*x*) par g(*nbest*) + k(*nbest*,*x*) ;

Finsi

Ordonner Open par ordre croissant de f; si égalité, par ordre décroissant de g

Fin

exemple A*



Propriétés de A*

- Si h(n)=0 pour tout n, A* est l'algorithme de Dijkstra
- A* est optimal si l'heuristique est admissible: **h(n) ≤ h*(n)**, pour tout noeud n, où h*(n) est le vrai coût pour atteindre le but à partir de n.
- Complet
- Exponentiel en temps. Polynômial si heuristique minorante.

Propagation de la recherche dans A*