Résolution de problèmes (Problem-Solving)

1 - Introduction

2 - Graphes de résolution

3 - Recherche informée et Exploration

4 - Jeux à deux joueurs

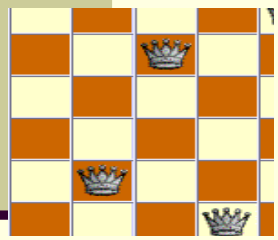
Résoudre un problème

- Le formaliser : Choix d'une représentation et spécification formelle de l'énoncé ;
  - Logiques,
  - Formalisation suivant des types de règles,
  - ...
- Le résoudre : Résolution par la machine de l'énoncé formalisé et construction de la solution
  - Moteur d'inférence des systèmes experts
  - STRIPS
  - ...

Énoncés de type combinatoire

- Se définit par un ensemble de solutions potentielles + ensemble de contraintes;
- Objectif : solution respectant les contraintes;

- Exemples :



  - Problème des 8 reines (Comment placer 8 reines sur un échiquier de 8x8 de telle manière qu'aucun couple de reines ne soit en attaque ?)
  - SEND + MORE = MONEY,
  - Sudoku.

Énoncés de type buts / opérateurs de décomposition

- Problème à résoudre (un but),
- Une liste de problèmes que l'on sait résoudre (problèmes primitifs),
- Des opérateurs de décomposition de problèmes en sous-problèmes;

- Exemples :
  - Analyse grammaticale d'une phrase

Énoncés de types états / opérateurs de changement d'états

- Un état initial,
- Des états finaux à atteindre ou leurs caractérisations,
- Des opérateurs de type <conditions, effets>,
- Objectifs : découvrir un enchaînement d'opérateurs permettant de passer de l'état initial à un état final;

- Exemple :
  - Jeu de Taquin

Analyse du problème : jeu du Taquin

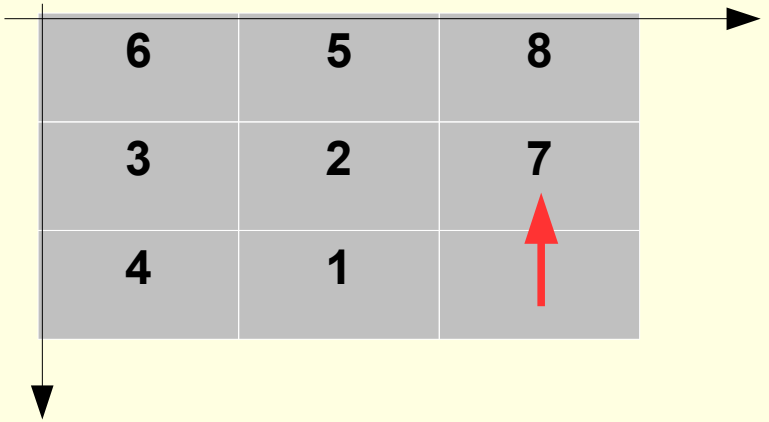
- États et opérateurs
  - État : matrice 3*3 + coordonnées de la case vide (x,y)
  - Opérateurs : HAUT, BAS, GAUCHE, DROITE.
- Opérateur HAUT (= se déplacer vers le haut)

Pré-condition : y > 1

Effets : M(x, y) = (x, y-1)

M(x, y-1) = vide

y = y - 1



Résolution de problèmes (Problem-Solving)

1 - Introduction

2 - Graphes de résolution

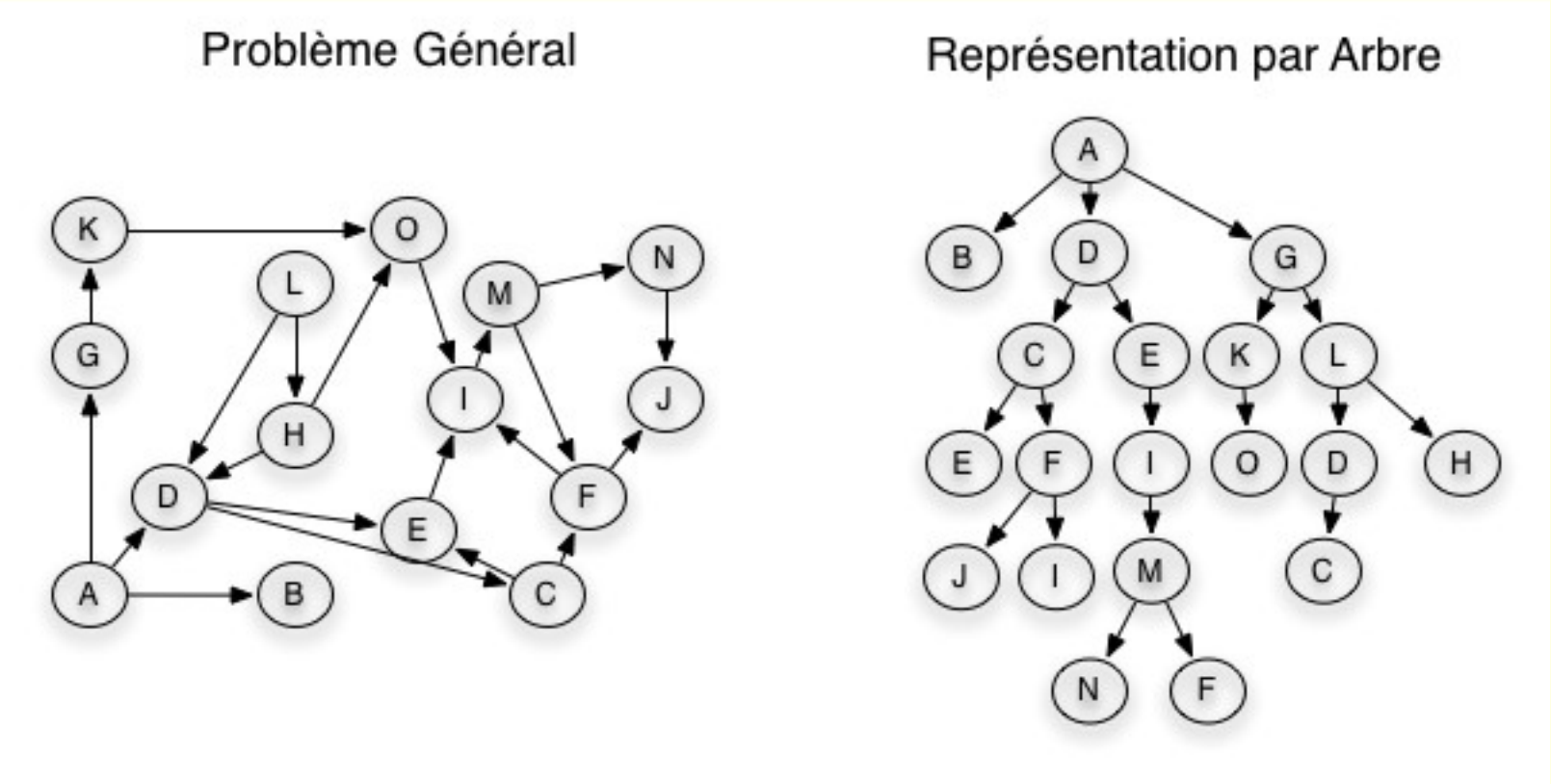
3 - Recherche informée et Exploration

4 - Jeux à deux joueurs

Graphes sous-jacents à ce type de problèmes

- On peut alors développer un graphe d'états = graphe de résolution:
  - à partir de l'état initial, on applique tous les opérateurs dont les conditions d'application sont vérifiées et on recommence sur les états obtenus;
  - les arcs sont étiquetés par le nom de l'opérateur appliqué;
- Si on précise qu'on ne passe qu'une seule fois par un même état, ce graphe est alors un DAG (graphe acyclique orienté) :
  - la racine est l'état initial ;
  - les feuilles sont des états finaux ou des impasses.
- Résoudre le problème revient à chercher un chemin dans ce graphe menant de l'état initial à un état final.

Graphe et arbre



Graphe de résolution et graphe de recherche

- Ce graphe est en général trop important pour être développé complètement en mémoire à un instant donné.

  => Il va falloir mettre en place des stratégies ou des heuristiques pour décider des parties les plus intéressantes du graphe.

- Définition : La partie du graphe de résolution qui est explorée lors de la recherche est nommée : le graphe de recherche.

- Une propriété importante d'une recherche :
  - Complexité : la coût de la recherche, en particulier la taille du graphe de recherche par rapport au graphe de résolution.

Complexité

Elle est exprimée en utilisant les quantités suivantes:

- le facteur de branchement (b) : le nombre maximum de successeurs à un nœud;
- la profondeur du nœud but (d) le moins profond;
- la longueur maximale d'un chemin dans l'espace d'états (m).

- Définitions :
  - Complexité de temps : le nombre de nœuds générés pendant la recherche.
  - Complexité d'espace : le nombre maximum de nœud en mémoire.

Recherche de solution dans un arbre

- Simuler l'exploration de l'espace d'états en générant des successeurs pour les états déjà explorés.
- Nœud de recherche
  - État : l'état dans l'espace d'état;
  - Nœud parent : Le nœud dans l'arbre de recherche qui a généré le nœud en cours;
  - Action : L'action qui a été appliquée au parent pour générer ce nœud;
  - Coût du chemin : Le coût du chemin à partir de l'état initial jusqu'à ce nœud;
  - Profondeur : Le nombre d'étapes dans le chemin à partir de la racine (de l'état initial).

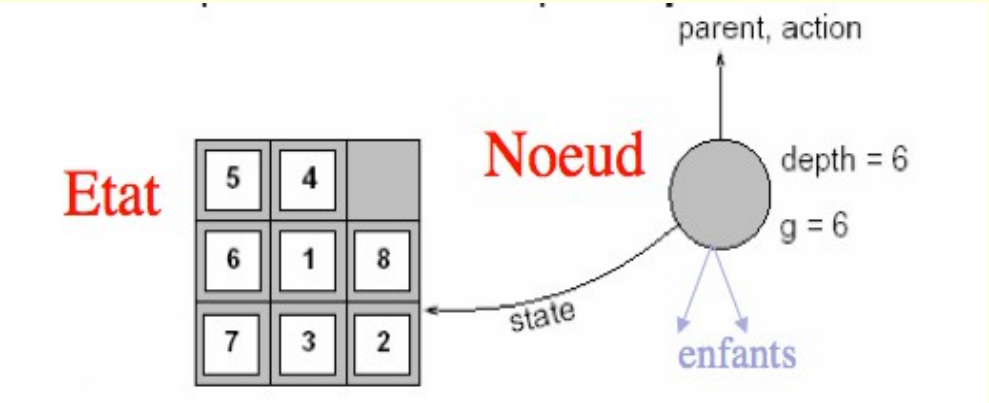
La recherche dans un arbre

Idée de base de la recherche dans un arbre :

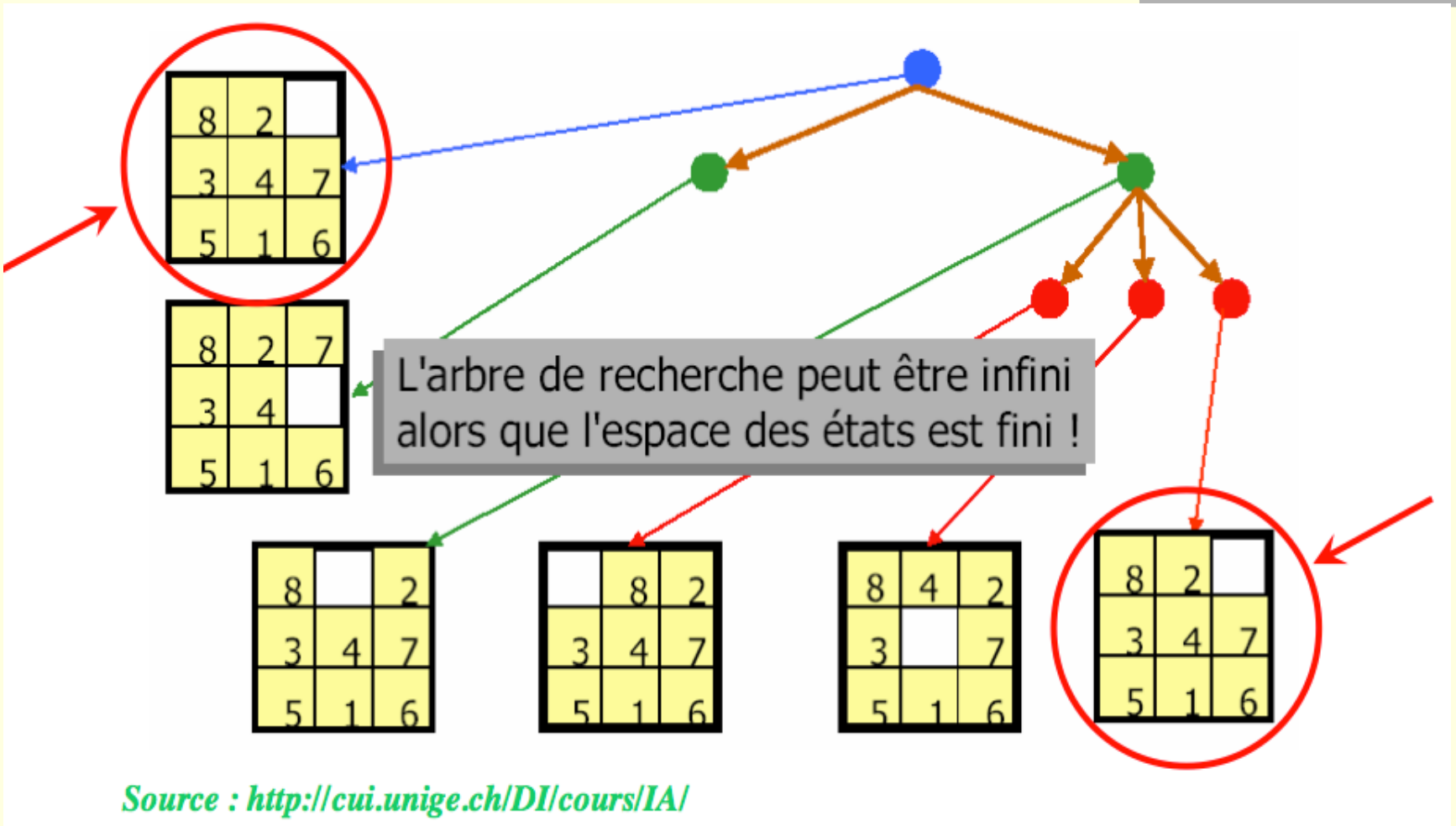
- Exprimer toutes les possibilités;
- Rechercher la meilleure possibilité en parcourant l'arbre;

États vs nœuds

- Un état est une représentation d'une configuration réelle.
- Un nœud est un élément d'une structure de données constitutive d'un arbre de recherche ; il possède les champs suivants:
  - état, parent, (enfants), profondeur, coût du chemin g(x)
  - ces notions n'existent pas pour les états



États vs nœuds : exemple du Taquin



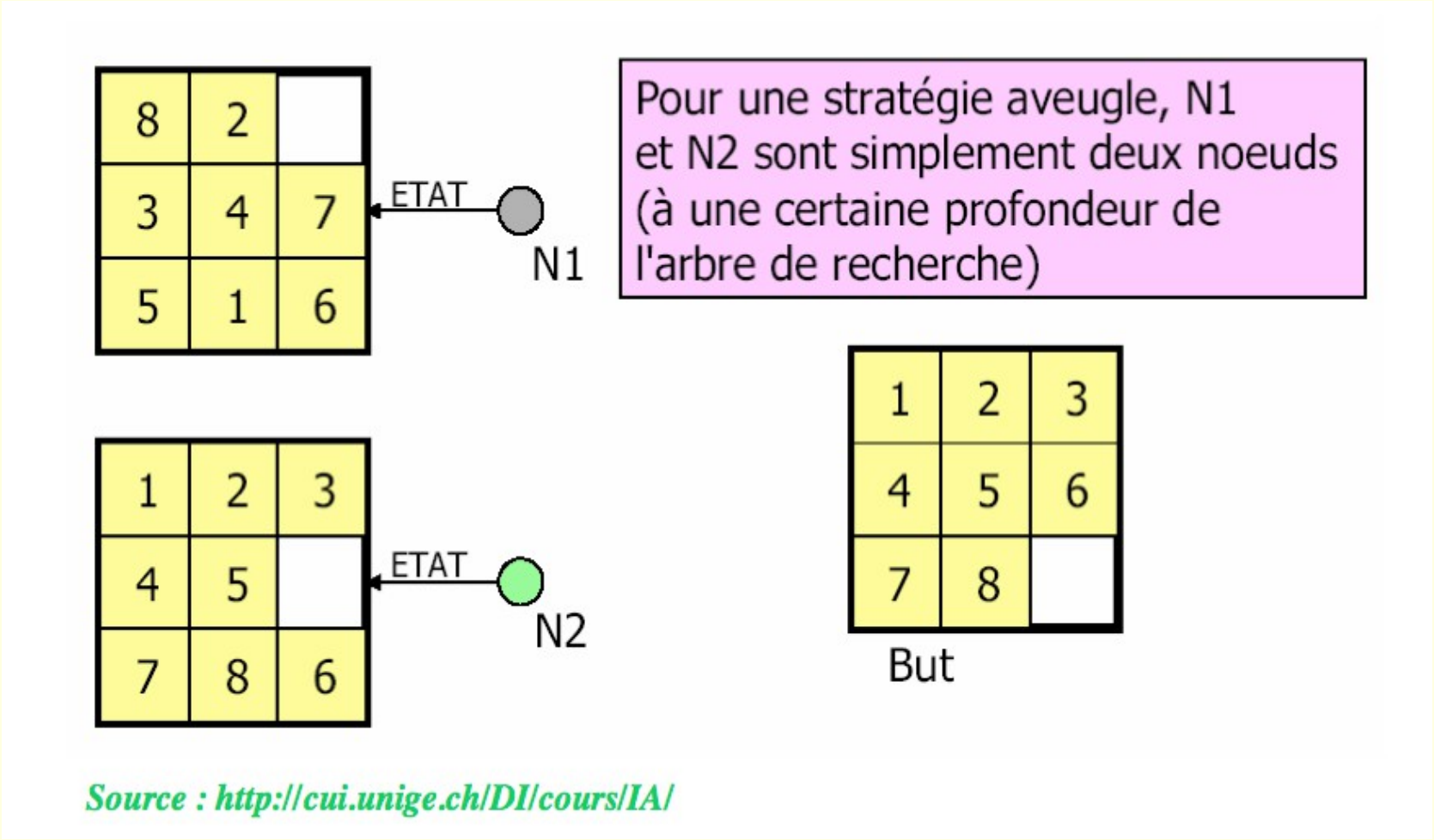
Stratégies de Recherche

Définition : La stratégie de recherche détermine l'ordre de développement des nœuds.

Stratégies aveugles : n'exploitent aucune information contenue dans un nœud donné;

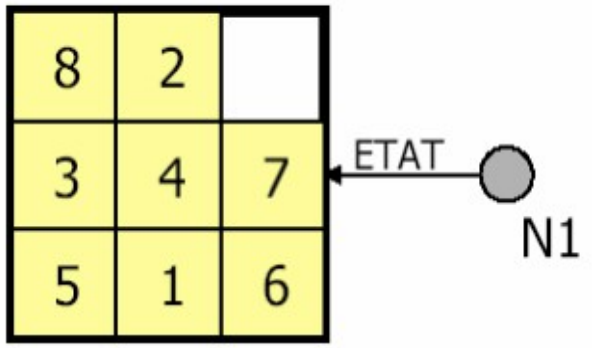
Stratégies heuristiques : exploitent certaines informations pour déterminer si un nœud est "plus prometteur" qu'un autre;

Exemple Taquin: stratégie aveugle

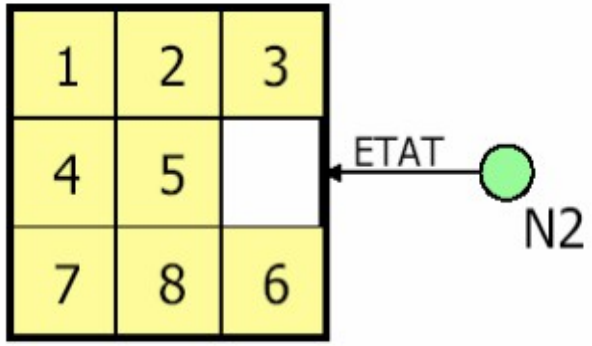


Exemple Taquin: stratégie heuristique

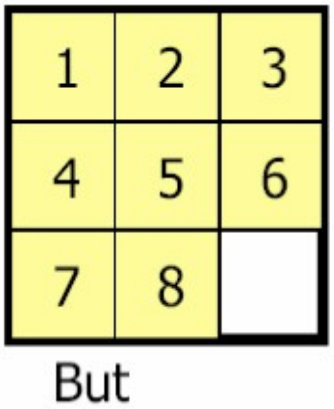
7 pions mal placés



1 pion mal placé



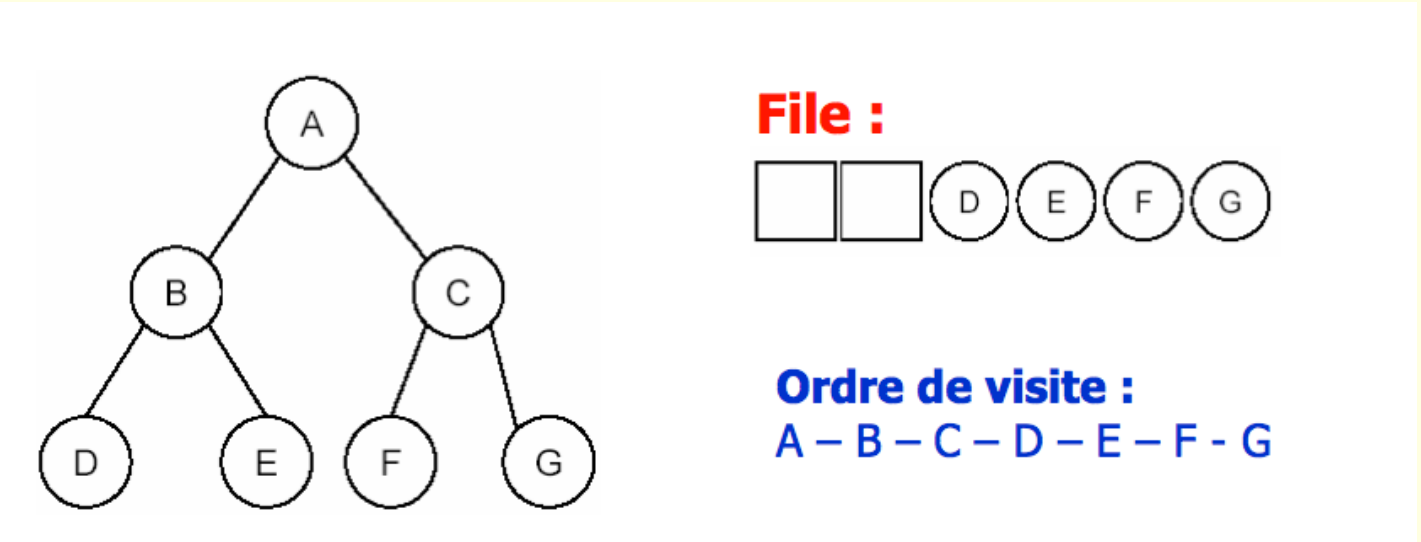
Pour une stratégie heuristique comptant le nombre de plaquettes mal placées, N2 est plus prometteur que N1



Source : http://cui.unige.ch/DI/cours/IA/

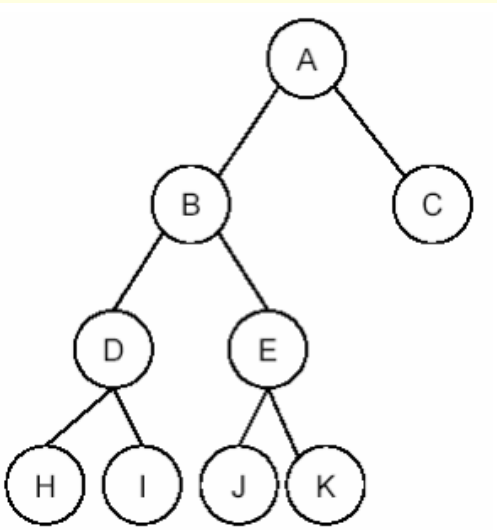
Stratégies de recherche aveugle

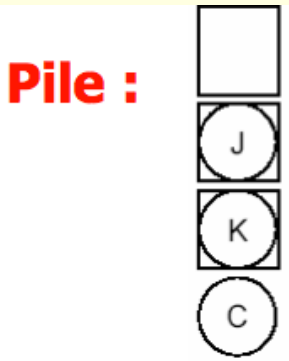
- Largeur d'abord (Breath-first)
  - Développement de tous les nœuds au niveau i avant de développer les nœuds au niveau i + 1.
  - Les nouveaux successeurs vont à la fin.



Stratégies de recherche aveugle

- Profondeur d'abord (Depth-first)
  - Développe le nœud le plus profond. Implémenté à l'aide d'un pile.
  - Les nouveaux nœuds générés vont sur le dessus.

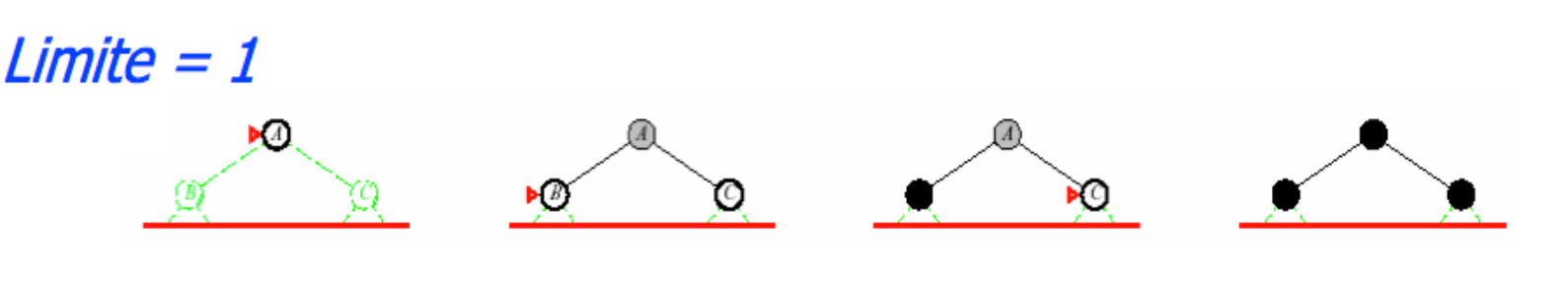


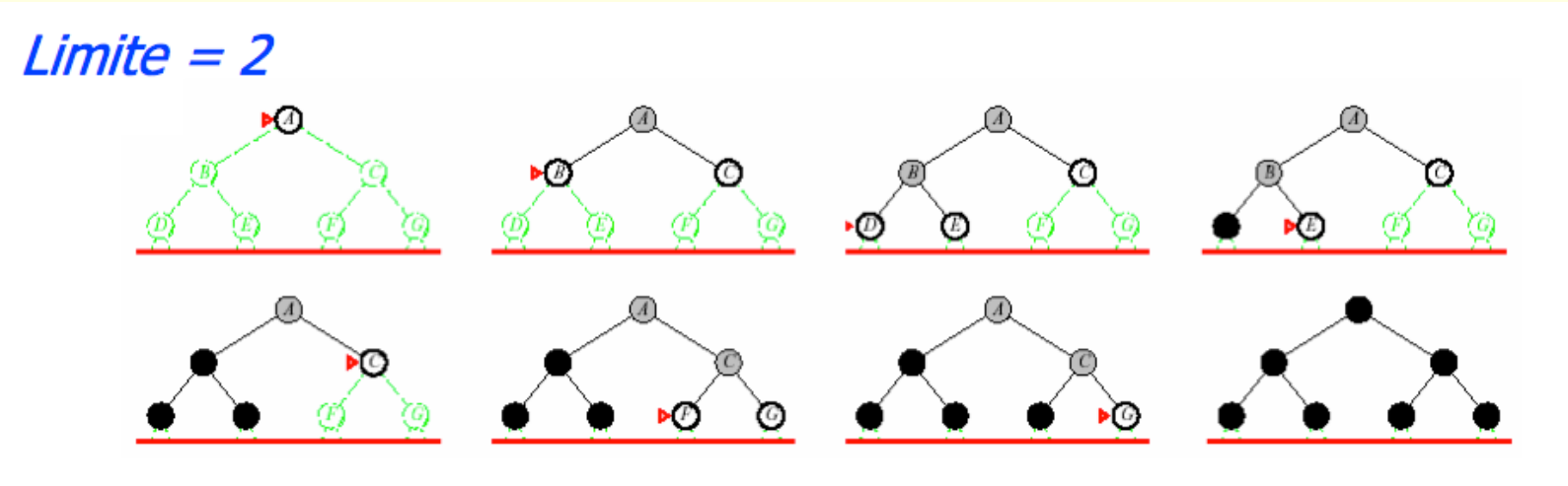


Ordre de visite :
A – B – D – H – I – E – J – K - C

Stratégies de recherche aveugle

- Profondeur itérative (Iterative deepening)
  - Profondeur limitée
  - Modifier le seuil de la limite progressivement





Résolution de problèmes (Problem-Solving)

1 - Introduction

2 - Graphes de résolution

3 - Recherche informée et Exploration

4 - Jeux à deux joueurs

Limites des recherches aveugles

- Les algorithmes de recherche aveugle 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 améliore presque toujours le processus de recherche.

Fonction heuristique

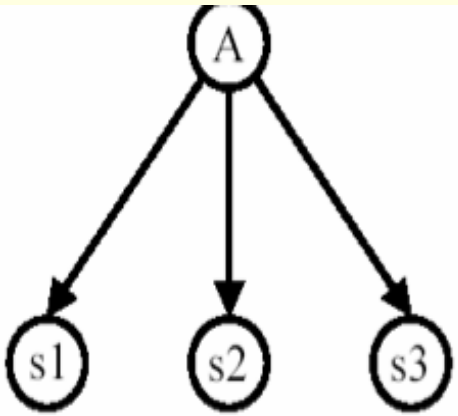
Une fonction heuristique

$$H : E \rightarrow R$$

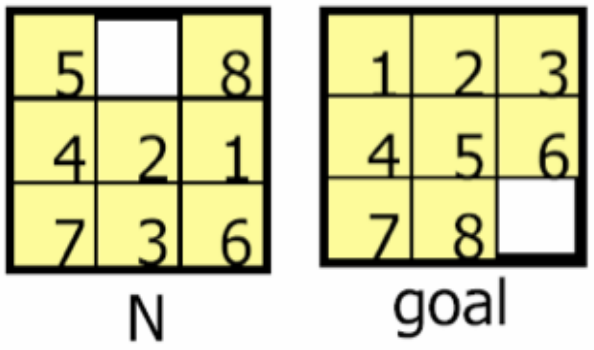
fait correspondre à un état s Є E (espace d'états) un nombre h(s) Є R qui est généralement une *estimation du rapport coût/bénéfice* qu'il y a à étendre le chemin en passant par s.

Exemple :

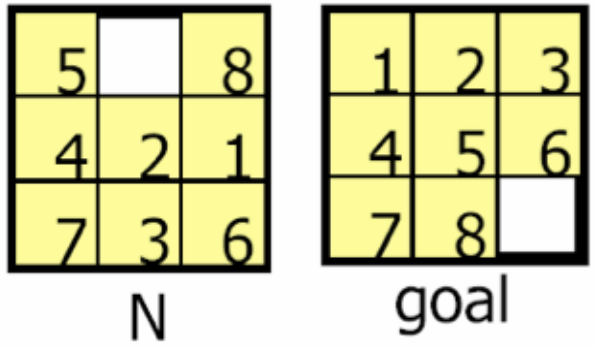
- Contrainte : $h(solution) = 0$
- Le nœud A a 3 successeurs pour lesquels :
  - $h(s1) = 0.8$ $h(s2) = 2.0$ $h(s3) = 1.6$
- La poursuite de la recherche par *s1* est heuristiquement la meilleure.



Exemple d' heuristiques



- h(N) = nombre de plaquettes mal placées = 6



- h(N) = somme des distances (Manhattan) de chaque plaquette à sa position finale

  = 3 + 1 + 3 + 0 + 2 + 1 + 0 + 3

  = 13

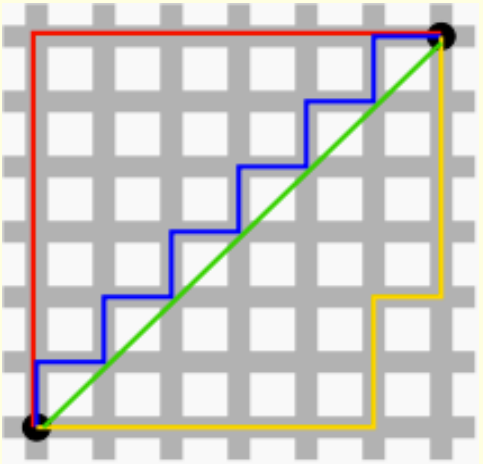
Recherche meilleur-d'abord

- Combinaison entre recherche en profondeur et en largeur.
- Exploitation des nœuds dans l'ordre (croissant/décroissant) de leurs valeurs heuristiques : aller vers le nœud le plus prometteur.

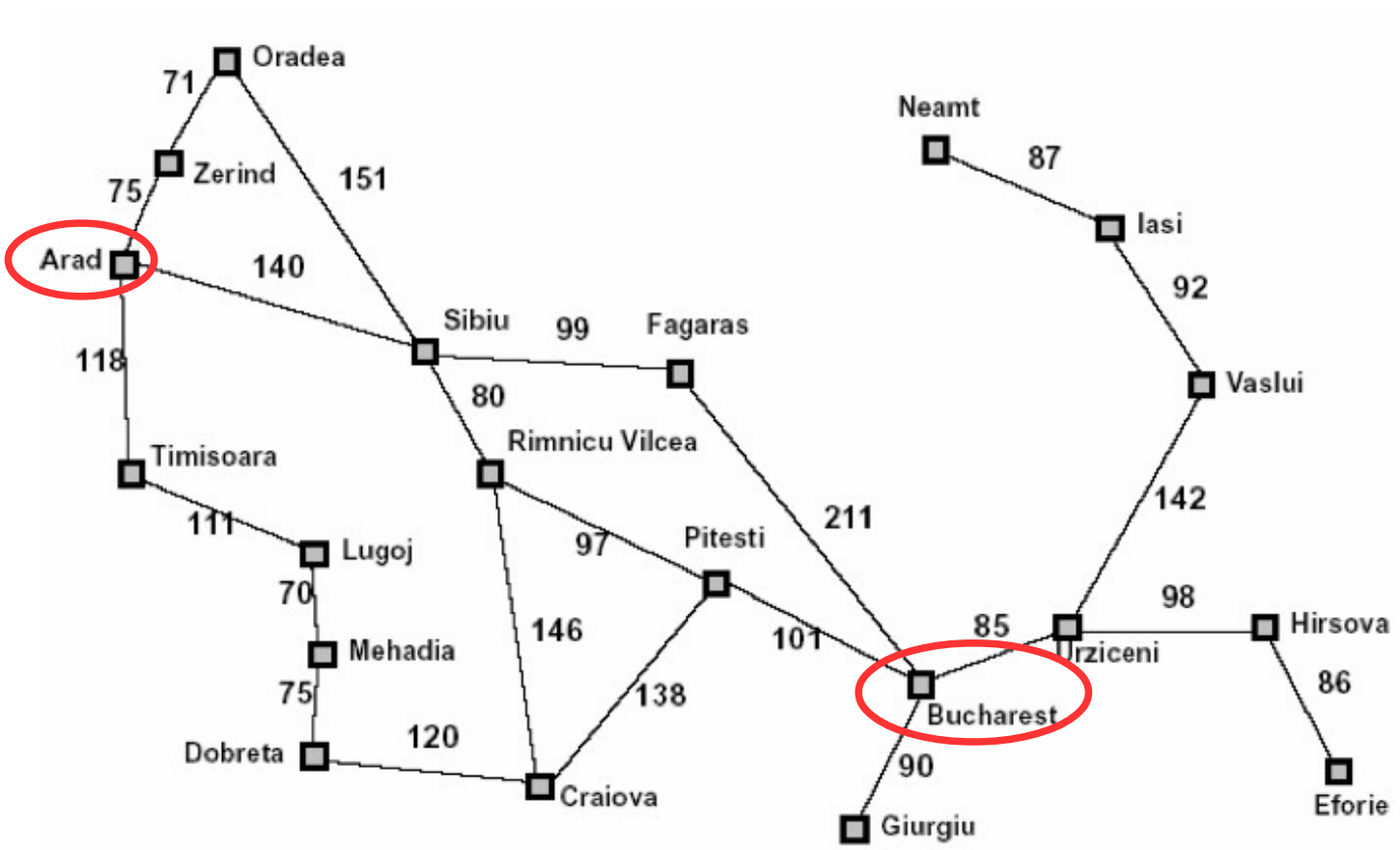
- 2 exemples de la recherche meilleur-d'abord :
  - Recherche avare
  - A*

Recherche avare

- Fonction heuristique :
  - h(n) = estimation du coût du nœud n au but
- Recherche avare : minimiser le coût estimé pour atteindre le but .
- Le nœud qui *semble* être le *plus proche du but* sera étendu / développé en priorité.
- Fonctions heuristiques classiques :
  - Distance à vol d'oiseau
  - Distance « Manhattan »



Exemple : voyage en Roumanie (plus court chemin)



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Recherche avare

- Complétude : non,
  - Peut rester pris dans une boucle,
  - Est complet si espace de recherche est fini et si vérification d'absence de boucle ;
- Optimal : non
  - Ne considère pas la distance parcourue ;
- Susceptible de faux départ
  - Exemple : Iasi – Fagaras : passage par Neamt même si c'est une impasse ;

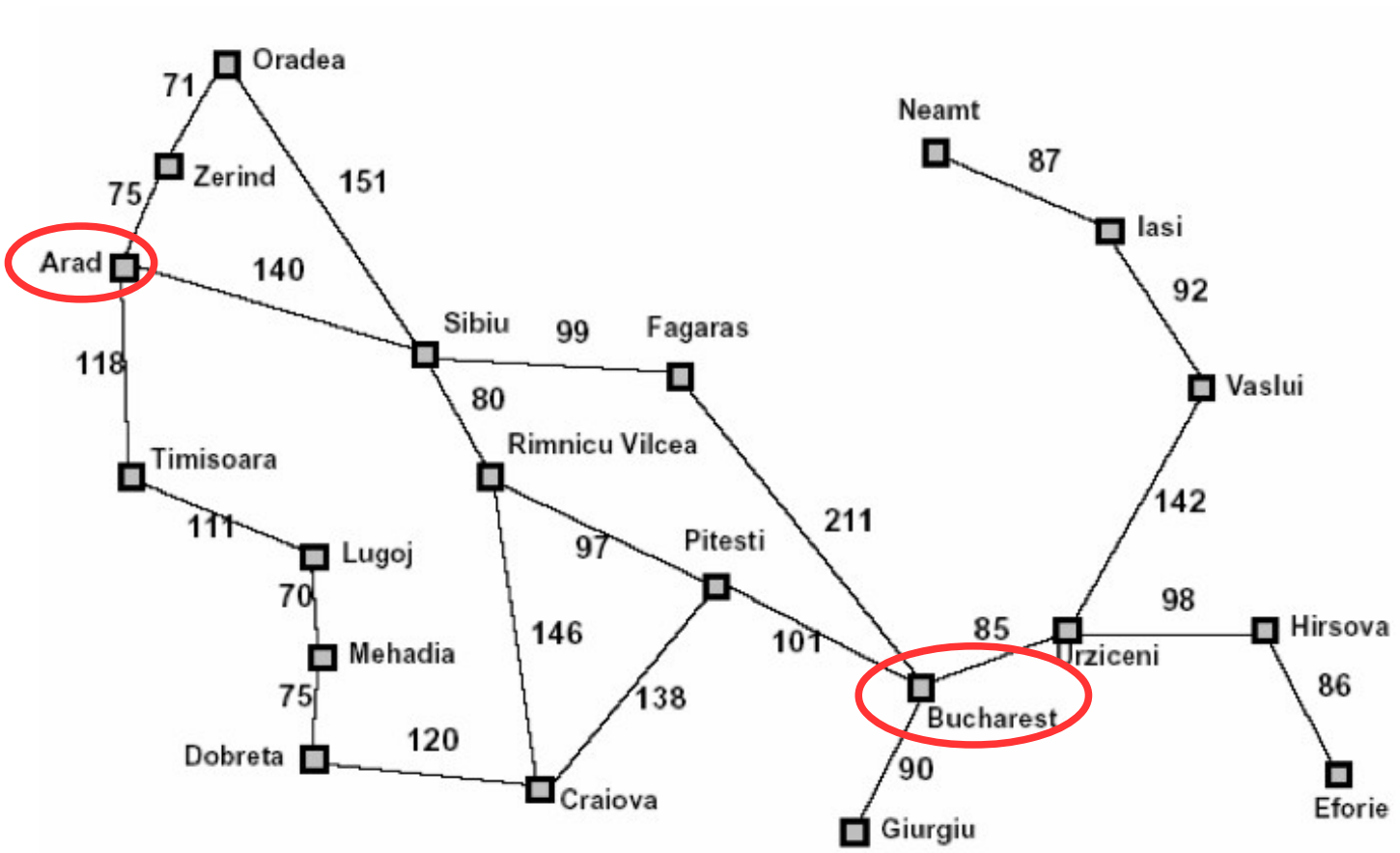
Algorithme A*

- Définition d'une fonction f capable d'évaluer un nœud n et définie comme suit :
  - f(n) = g(n) + h(n) avec
    - g(n) le coût du meilleur chemin connu pour atteindre n depuis l'état initial ;
    - h(n) l'estimation du coût du chemin entre n et un état final ;
    - f(n) est donc une estimation du coût du meilleur chemin passant par n.

- On note f*, g* et h* les coûts optimaux.

  Remarque: h(n) garde la même valeur alors que g(n) varie en cours de recherche.

Exemple A*: voyage en Roumanie (plus court chemin)



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Propriétés de A*

- Complétude : oui, sauf si nombre infini de nœuds ;
- Optimal : oui ;
- Complexité en temps : exponentielle ;
- Complexité en espace : garde tous les nœuds en mémoire ;

Algorithme A*

```
Algorithme A*
Début
  ouverts = { état initial }
  fermés  = vide
  succès  = faux
  Tant que (ouverts non vide) et (non succès) faire
    choisir n dans les ouverts tel que f(n) est minimum
    Si est_final(n) Alors succès=vrai
    Sinon ouverts = ouverts privé de n
          fermés  = fermés + n
          Pour chaque successeurs s de n faire
            Si (s n'est ni dans ouverts ni dans fermés)
            Alors
              ouverts = ouverts + s
              père(s) = n
              g(s) = g(n) + coût(n,s)
            Sinon
              Si (g(s) > g(n) + coût(n,s)) Alors
                père(s) = n
                g(s) = g(n) + coût(n,s)
                Si (s se trouve dans fermés) Alors
                  fermés  = fermés  - s
                  ouverts = ouverts + s
                Fin si
              Fin si
            Fin si
          Fin pour
    Fin si
  Fin TQ
Fin
```

Résolution de problèmes (Problem-Solving)

1 - Introduction

2 - Graphes de résolution

3 - Recherche informée et Exploration

4 - Jeux à deux joueurs

Qu'est-ce qu'un jeu à deux joueurs ?

- Deux joueurs adverses;
- Alternance des coups jusqu'à obtention d'un état terminal {gagnant, perdant, nul};
- Chacun veut maximiser ses gains et minimiser ceux de l'adversaire (hypothèse pour la modélisation);
- Connaissance parfaite du jeu à chaque instant;

- Exemples:
  - Jeu de Nim (jeu des allumettes)
  - Tic-Tac-Toe
  - Échecs
  - Dames
- Contre-exemples:
  - Jeux de hasard (dès)
  - Jeux de cartes

Pourquoi étudier les jeux ?

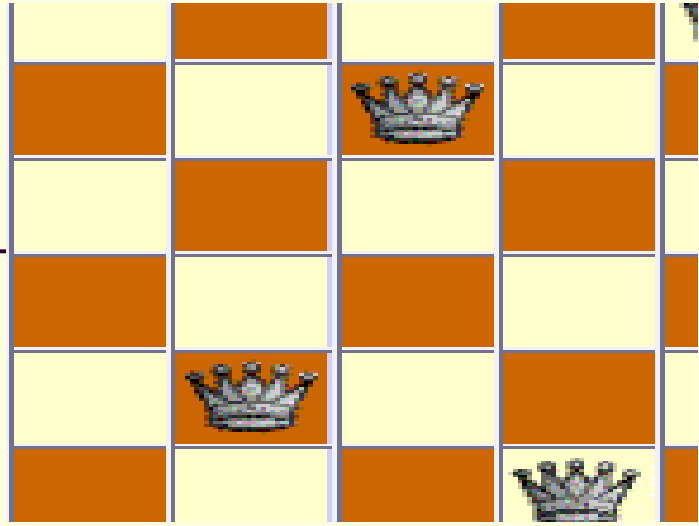
- Jeux:
  - Micro-mondes pour l'étude de stratégie;
  - Limiter le monde réel à un monde réduit et réaliser un système qui agit sur ce monde réduit.

- Intérêts:
  - Univers limité;
  - Règles spécifiées mais riches en possibilité de déduction;

Étapes dans la réalisation du jeu

1. Définir la modélisation/représentation du jeu;
2. Gérer la visualisation du jeu ;
3. Modélisation des règles du jeu ;
4. Créer le moteur qui va permettre de lancer le jeu et de faire jouer un joueur contre un joueur artificiel ;
5. Améliorer le « joueur » artificiel : min-max, alpha-beta,…

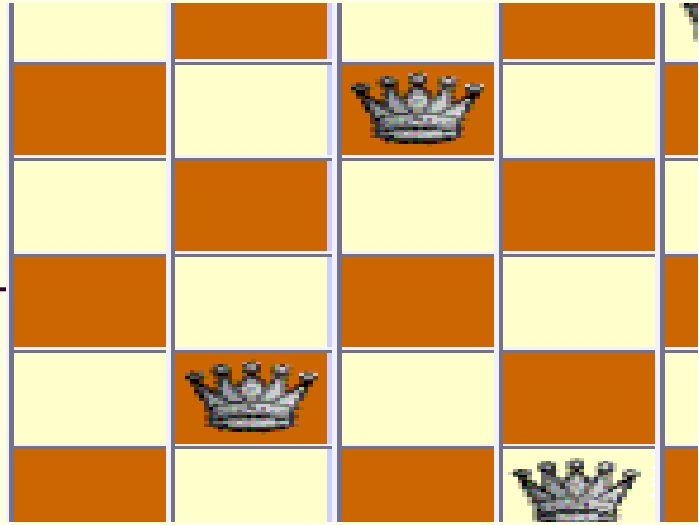
Exemple des 8 reines



- Comment placer 8 reines sur un échiquier de 8x8 de telle manière qu'aucun couple de reines ne soit en attaque ?

- Rappel : La reine est la pièce la plus mobile et la plus puissante du jeu d'échecs. Elle se déplace d'un nombre quelconque de cases dans toutes les directions (lignes/colonnes/diagonales).

Exemple des 8 reines modélisation du problème



- Solution 1 : Tableau à 2 dimensions
  grille[num_ligne][num_colonne]
  avec grille[5][2] = "reine";

- Solution 2 : Liste de couples de coordonnées
  Reines = [[1,1], [2,7], [3,5], [4,8], [5,2], [6,4], [7,6], [8,3]]

- Solution 3 : Liste de numéros de colonne (lignes forcément différentes)
  - la position de Xi dans la liste correspond au numéro de ligne de la reine;
  - la valeur de Xi correspond au numéro de la colonne.
    Reines = [1,7,5,8,2,4,6,3]

  Trouver la solution = Trouver une liste de nombres de 1 à 8 respectant certaines conditions.

  Il n'existe pas une solution meilleure *a priori*

  mais la solution 3 évite le superflu

Exemple le Tic-Tac-Toe

- Grille de 3*3 avec soit :
  - **-** case vide
  - **x** pour joueur 1
  - **o** pour joueur 2

- Modélisation du problème
  - Solution 1: liste de listes [[x,-,-],[-,o,-],[-,-,-]]

  - Solution 2 : liste à 9 éléments [x,-,-,-,o,-,-,-,-]

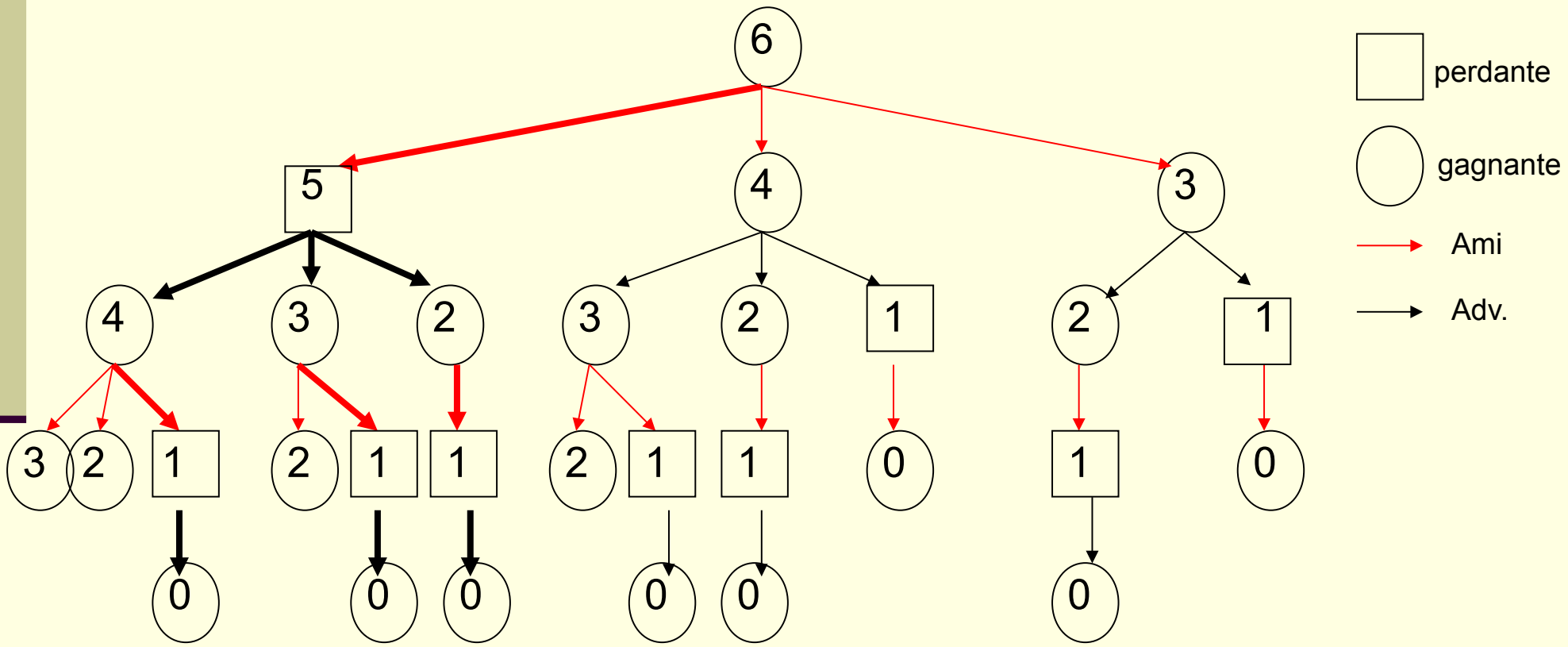
  ➔ quelle solution choisir ?

Exemple le Jeu de Nim

- 21 allumettes,
- 2 joueurs,
- allumettes sur la même ligne.
- Règles:
  - Prendre 1, 2 ou 3 allumettes;
  - Celui qui prend la dernière a perdu;

Exemple le Jeu de Nim

- Graphe d'états = Arbre de jeu = permet de représenter les différentes positions possibles et leurs liens en fonction des coups joués



Arbre de Jeu *versus* Arbre de Recherche

- Arbre de jeu ➔ explosion combinatoire
  - Morpion : arbre de jeu : 362 880 positions
  - Othello : 2^{60} positions

- Arbre de recherche :
  - sous-ensemble de l'arbre de jeu construit dynamiquement;
  - positions terminales ne sont pas forcément celles du jeu

    ➔ Limiter la profondeur de l'arbre de recherche

Exploration dans les arbres de jeux

- **Racine** (prof 0) : position de départ;
- Nœuds de **profondeur paire** : positions dans lesquelles c'est au joueur 1 de jouer (AMI) ;
- Nœuds de **profondeur impaire** : positions dans lesquelles c'est au joueur 2 de jouer (ADV) ;
- Les **arcs** issus d'un nœud quelconque représentent les différents coups possibles qui peuvent être joués à partir d'un nœud;
- Les **feuilles** : positions gagnantes, perdantes ou bloquées.

Fonction d'évaluation

- Cette fonction permet d'évaluer une position non terminale et d'estimer qui peut gagner sans développer de sous-arbres.
- Soit un nœud n (ie une position), par convention,
  - f : N→] -∞ , + ∞[
  - f(n) = -∞ si n est perdant pour AMI,
  - f(n) = +∞ si n est gagnant pour AMI,

➔ Problème : trouver une bonne fonction d'évaluation

MinMax

- Maximiser la valeur de la fonction d'évaluation de la situation courante pour AMI et minimiser celle de l'adversaire

- L'idée est de développer complètement l'arbre de jeu, de noter chaque feuille avec sa valeur, puis de faire remonter ces valeurs avec l'hypothèse que chaque joueur choisit le meilleur coup pour lui :
  - AMI choisit le coup amenant à l'état de plus grande valeur ;
  - ENNEMI choisit le coup amenant à l'état de plus petite valeur (pour l'autre)

MinMax pour le Tic-Tac-Toe

- Exemple de fonction d'évaluation:

$$V = V1 - V2,$$

Avec :

V1 est la somme :

- nb d'occurrences de 3 'X' en ligne/col/diag multiplié par 10;
- nb d'occurrences de 2 'X' en ligne avec la 3ième position vide multiplié par 4;
- nb d'occurrences d'1 'X' en ligne avec 2 autres positions vides;

V2 la même formule mais pour les 'O'

Algorithme DecisionMinMax

```
Algorithme DecisionMinMax (e,J)
{ Décide du meilleur coup à jouer par le joueur J dans la situation e }
Début
  Pour chaque coup de CoupJouables(e,J) faire
    valeur[coup] = ValeurMinMax(Applique(coup,e),J,false)
  Fin pour
  retourner (coup tel que valeur[coup] est maximale)
Fin

Algorithme ValeurMinMax (e,J,EstUnEtatMax)
{ Calcule la valeur de e pour le joueur J selon que e EstUnEtatMax ou pas }
Début
  Si PartieGagnante(e,J) Alors retourner(+1)
  Sinon Si PartiePerdante(e,J) Alors retourner(-1)
        Sinon Si PartieNulle(e,J) Alors retourner(0)
              Sinon
                 vals = vide
                 Pour s parmi les successeurs de e faire
                    ajouter ValeurMinMax(s,J,not(EstUnEtatMax))) à vals
                 Fin pour
                 Si EstUnEtatMax
                 Alors retourner(maximum de vals)
                 Sinon retourner(minimum de vals)
              Fin si
        Fin si
  Fin si
Fin
```

Limite du MinMax

- En pratique, l'algorithme Min-Max n'est pas utilisé en l'état : il est rarement possible de développer l'ensemble de l'arbre de jeu :
  - limiter la profondeur d'exploration,

    problème : effet d'horizon, on ne voit pas s'il y a un coup gagnant à la profondeur suivante ;

  - effectuer des coupes dans l'arbre de jeu :

    algorithme alpha-beta

1997 : *Deep Blue* bat Kasparov

- Recherche sur 14 niveaux de profondeur ;
- Explore plus de 1 milliard de combinaisons par coup à raison de +200 millions de position/seconde ;
- Méthode d'évaluation très sophistiquée ;
- Des méthodes non divulguées pour étendre certaines recherches à plus de 40 niveaux...

Références

Russell & Norvig « Artificial Intelligence, a modern approach », 2003.

http://www.grappa.univ-lille3.fr/~torre/Enseignement/Cours/Intelligence-Artificielle/

http://www1.ifi.auf.org/personnel/Alain.Boucher/cours/