Introduction
à
l'intelligence artificielle

(recherche dans les graphes)

Antoine Cornuéjols

AgroParisTech

antoine.cornuejols@agroparistech.fr

Plan

Résolution de problème par

Exploration de graphes

2. Recherche dans les graphes : plan

1. Quels problèmes ? Quelles résolutions ?
2. Notion de graphe de recherche
3. Techniques de recherche non informée
   - En **largeur** d' abord
   - En **profondeur** d' abord
   - En **profondeur itérative**
4. Techniques de recherche informée
   - En **meilleur d'abord**
   - **A***
5. Graphes ET/OU
6. Techniques par satisfaction de contraintes

2. Spécification de problèmes (1)

- Quels **problèmes** ?
  - ***Problèmes*** *de classes primaires*
  - *Démonstration de* ***théorèmes***
  - ***Convaincre*** *un interlocuteur*
  - *Choix d'un* ***circuit optimal*** *pour le prochain voyage en Indonésie*
  - ***Retrouver*** *à l'aéroport quelqu'un que l'on n'a jamais vu*
  - *...*

- **Démarche** générale
  1. Passage d'un énoncé informel à une **spécification précise**
  2. **Recherche** d'une solution à l'intérieur du cadre défini par ces spécifications

2. Spécification de problèmes : types d'énoncés (2)

- ## Enoncés de type **combinatoire**

  Trouver dans un ensemble (espace) X donné, les éléments (points) x satisfaisants un ensemble de contraintes $K(x)$

  Ex : Pb des 8 reines, cryptarithmétique (SEND + MORE = MONEY)

- ## Enoncés avec **opérateurs** de changement d'états

  À partir d'un état initial donné, d'un critère objectif et d'un ensemble d'opérateurs de changement d'états, trouver une suite d'opérateurs permettant de passer de l'état initial à un état objectif

  Ex : Pb des missionnaires et des cannibales, les tours de Hanoï, le jeu du taquin

- ## Enoncés avec **opérateurs** de décomposition de pbs en ss-pbs

  Étant donné un problème (ou but), des opérateurs de décomposition du pb en ss-pbs, des problèmes dits primitifs (dont on connaît immédiatement la solution), trouver des opérateurs à appliquer pour décomposer le problème initial en un ensemble de ss-pbs primitifs

  Ex : Problèmes de planification, Tours de Hanoï, intégration symbolique

2. Résolution de problèmes

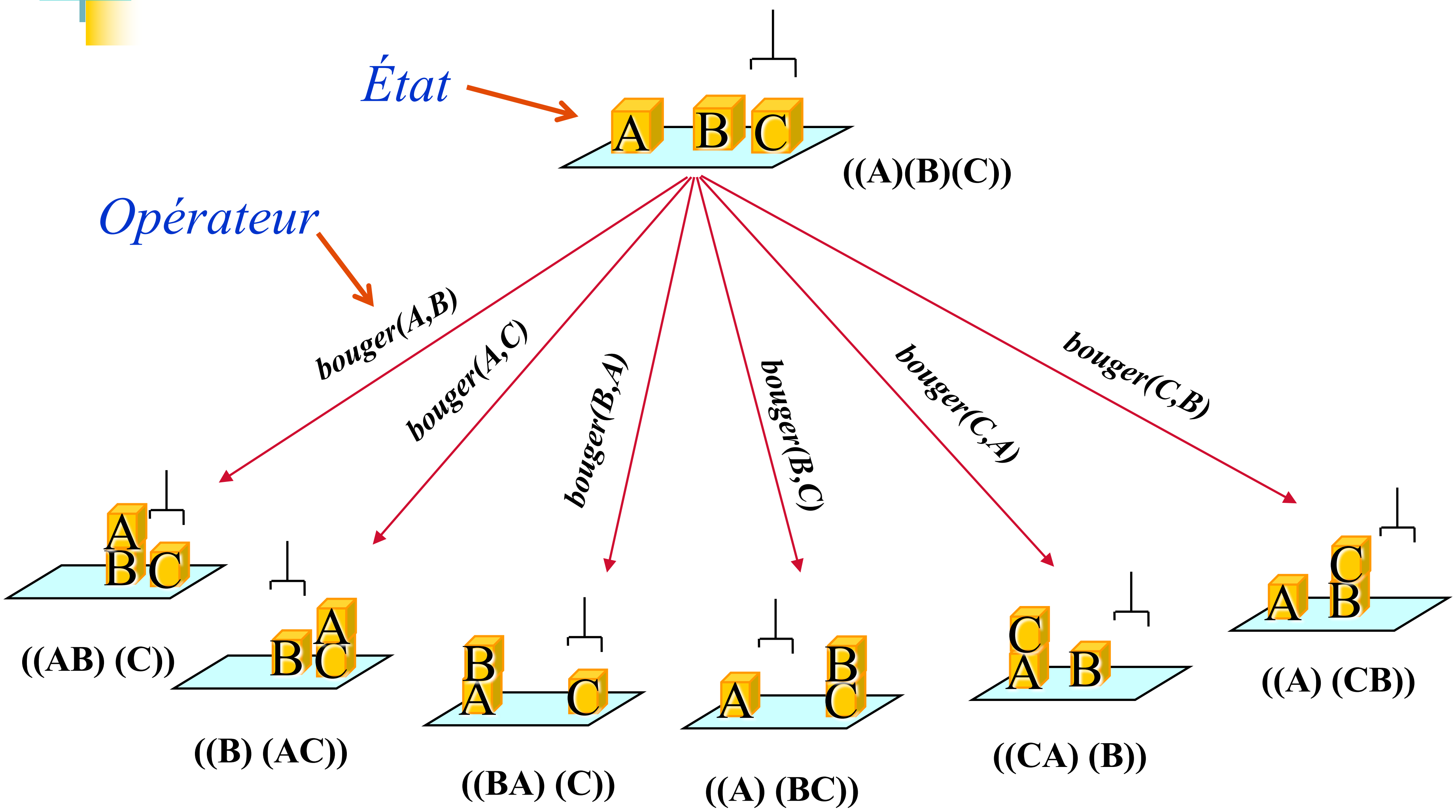
Démarche générale :

1. Trouver une bonne *représentation* du problème

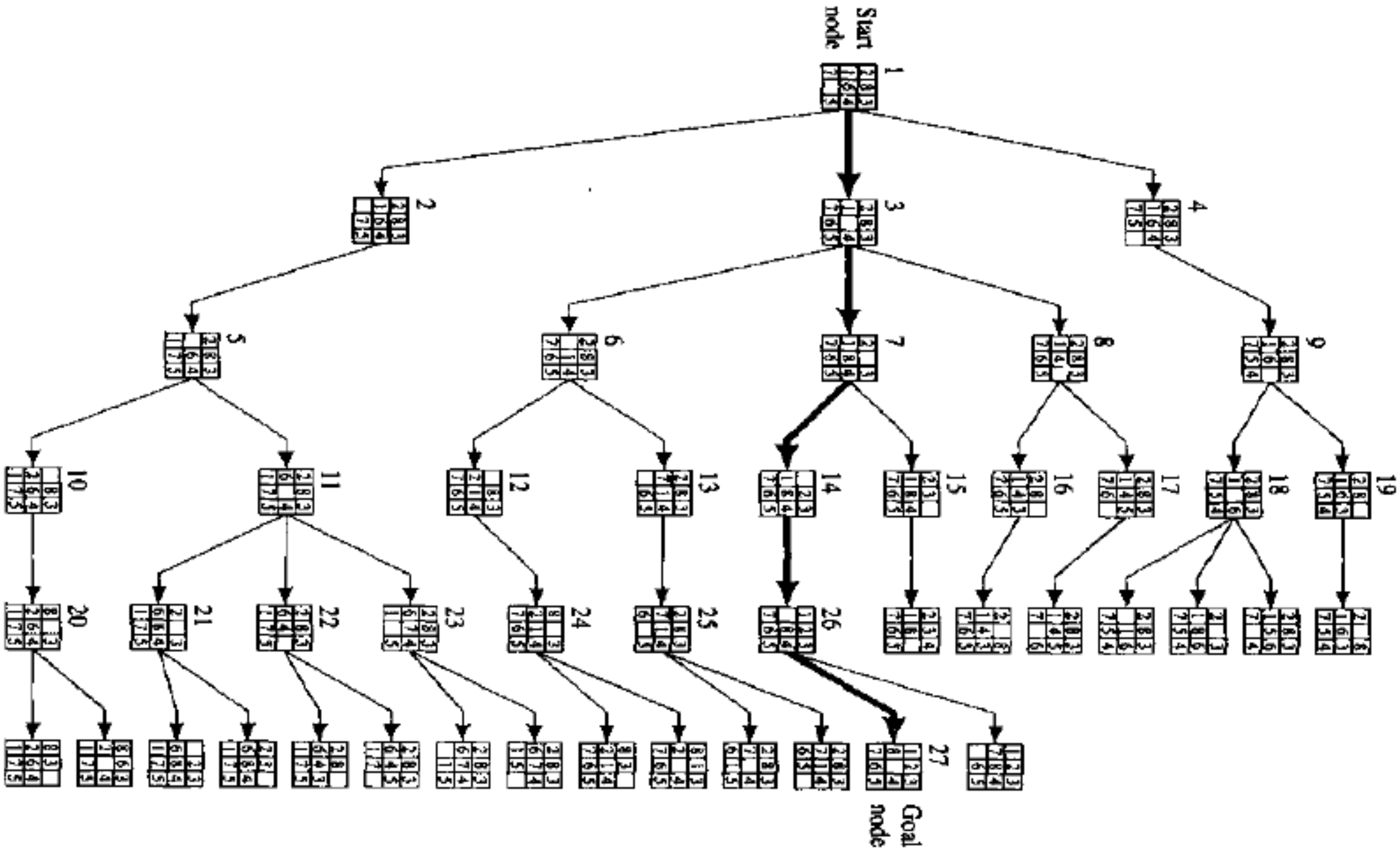
2. Trouver des *opérateurs* pour manipuler cette représentation

3. Effectuer un *contrôle de stratégie*

2. Notion de graphe de recherche (1)



2. Notion de graphe de recherche (1)



2. Notion de graphe de recherche (2)

On suppose que :

- *Les **actions** sont déterministes*

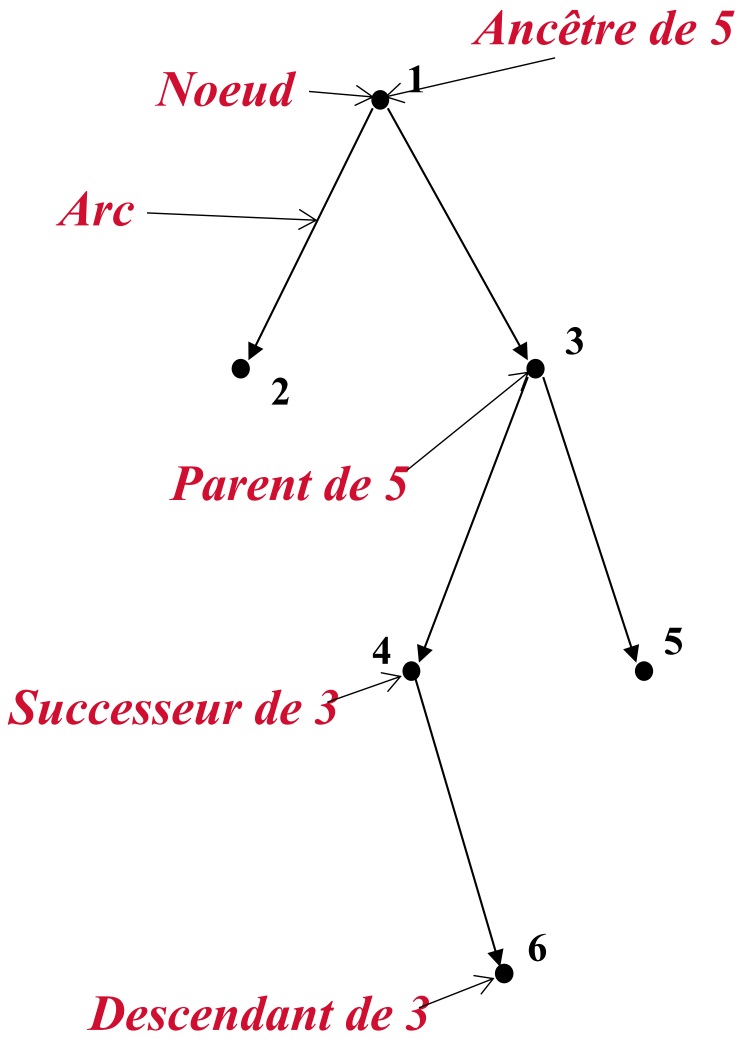
- *Les **actions** ont des effets « discrets » sur le monde*

2. Notion de graphe de recherche (2)

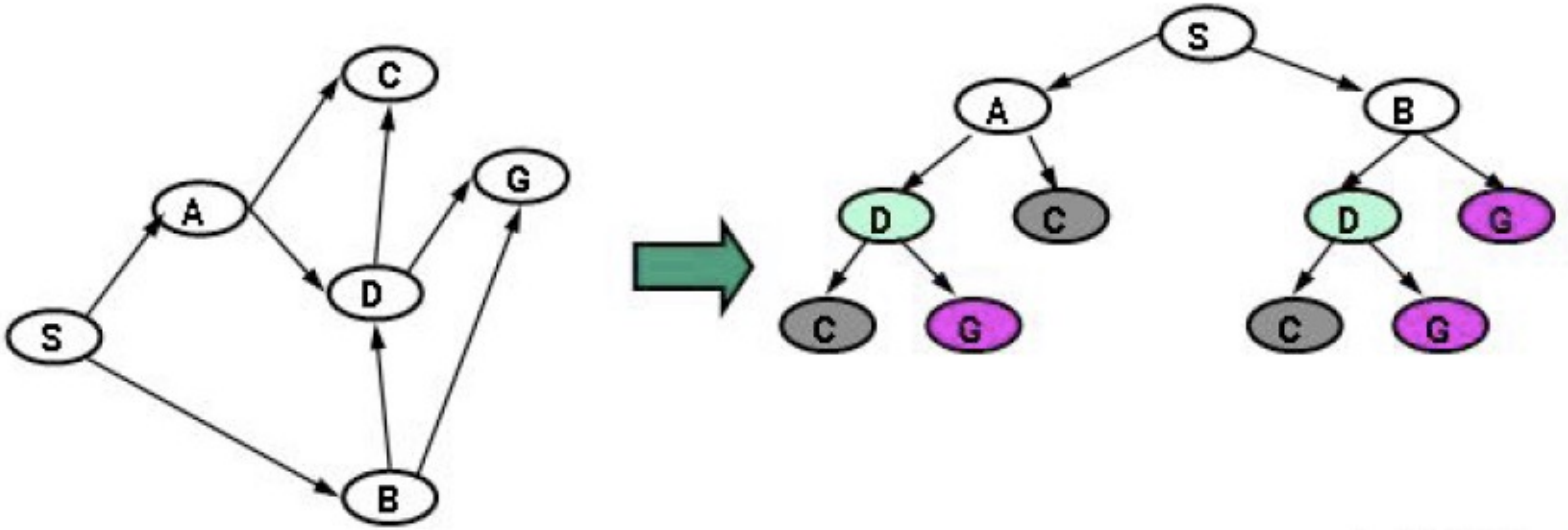
- *Graphe de résolution* ou *graphe d'états*

- *Graphe de recherche*
  - Nœud
  - Arc (et coût)
  - Parents / ancêtres / successeurs
  - Critère objectif
  - Développement d'un nœud

- *Stratégie de contrôle*
  - ***Fonction déterminant le choix du nœud à développer***
  - Recherche *aveugle* ou *non informée*
  - Recherche *heuristique* ou *informée*

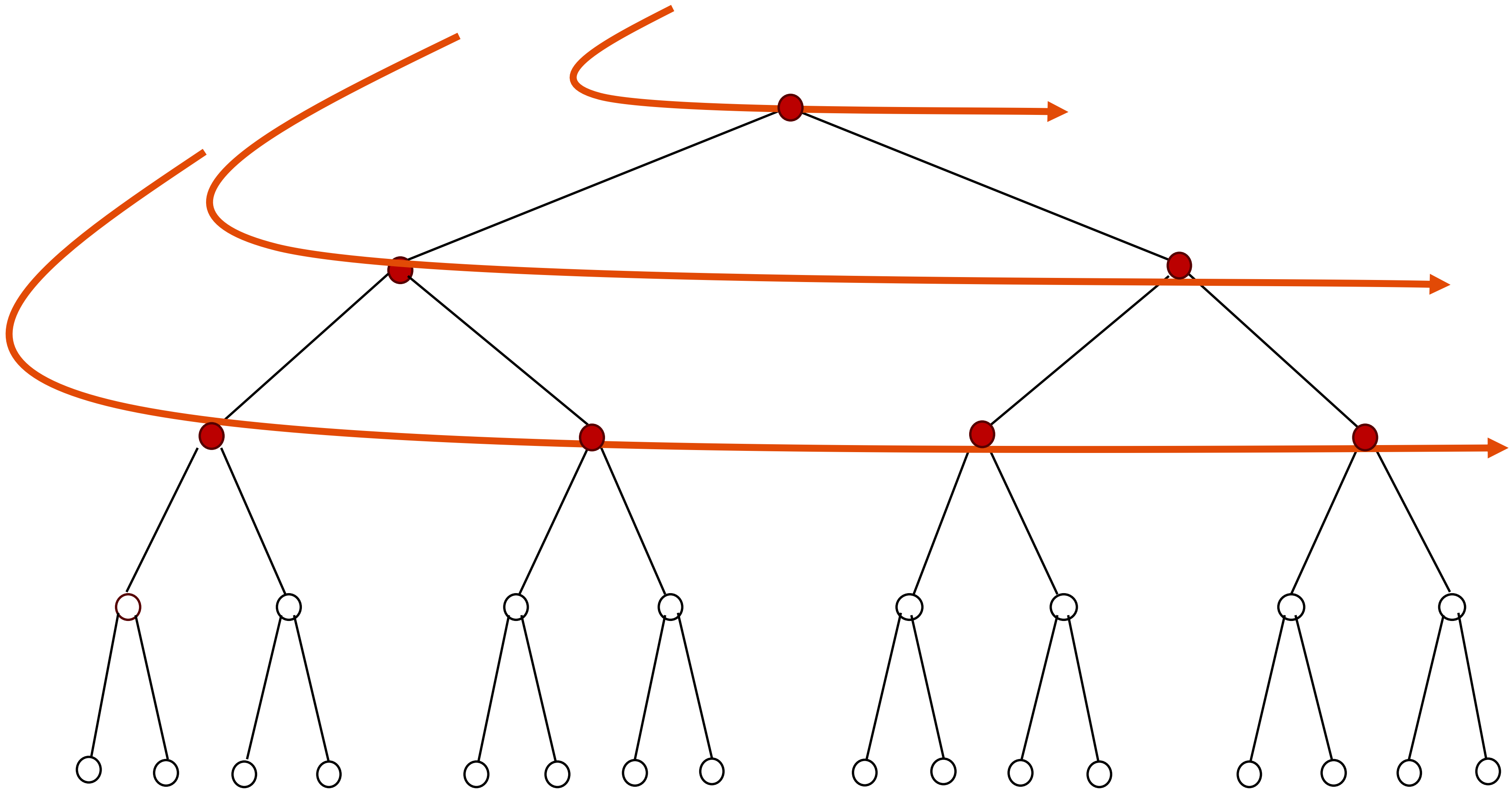


2. Graphe de recherche et arbre de recherche



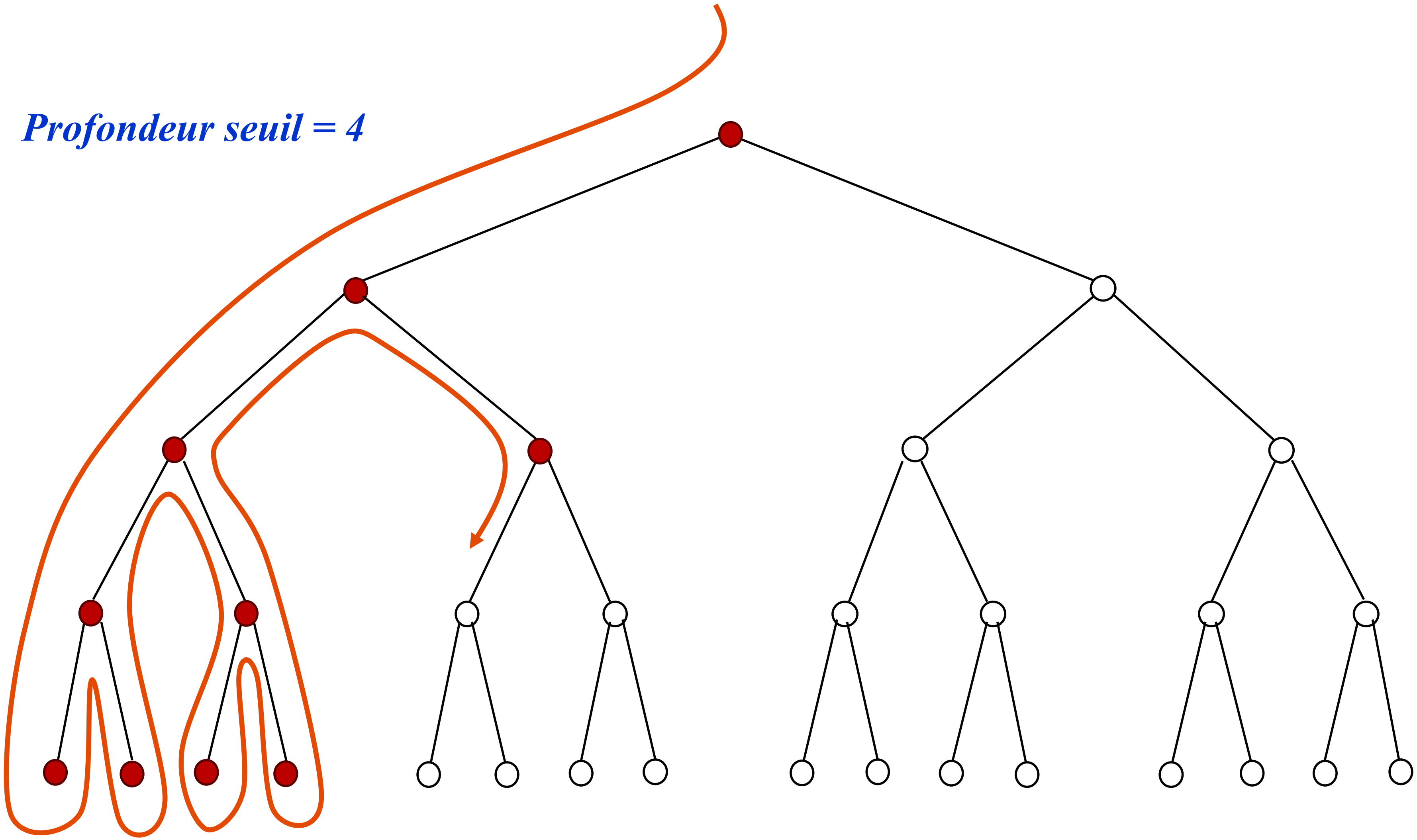
2. Recherche aveugle : en **largeur d'abord**

Stratégie systématique : niveau par niveau

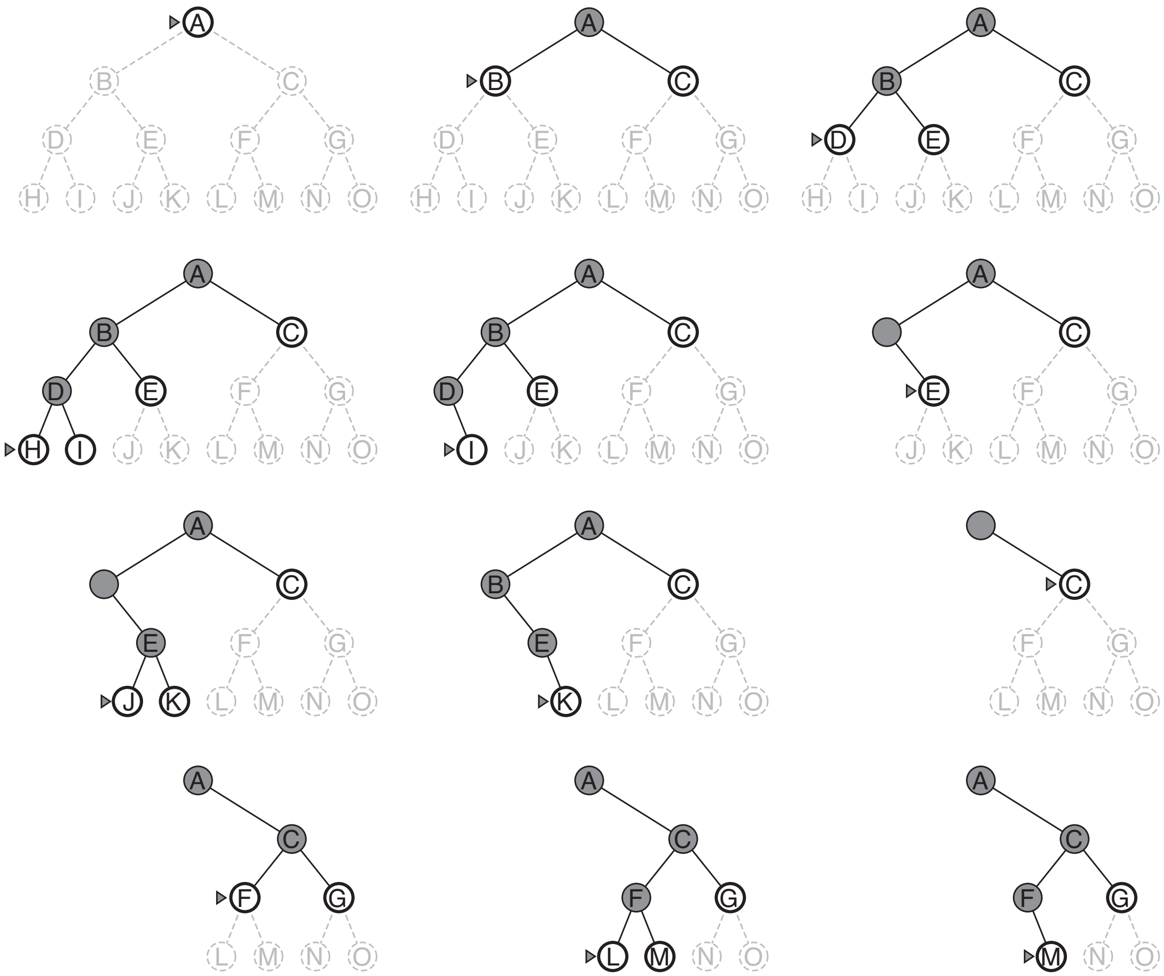


2. Recherche aveugle : en **profondeur d'abord**

Stratégie systématique : *avec retour arrière*



2. Recherche aveugle : en **profondeur d'abord**



Besoin en **mémoire** très limité

[Russell & Norvig « Intelligence Artificielle. Une approche moderne ». (4ème édition, 2021)], Fig 3.1.1

2. *Propriétés des stratégies de recherche*

- *Complétude*

  La stratégie parvient-elle nécessairement à une solution si il en existe une ?

- *Complexité en temps*

  Nombre de nœuds développés (ou évalués) durant la recherche

- *Complexité en espace*

  Nombre maximal de nœuds en mémoire lors de la recherche

- *Optimalité*

  La solution retournée est-elle optimale en coût ?

Facteurs :

- *b* **: facteur de branchement**
- *d* **: profondeur**
- *m* **: profondeur maximale de l' espace d' états**

2. **Largeur d'abord** : *propriétés*

- Complétude ?
  - Oui (si *b* est fini)
- Complexité en **temps** ?
  - Exponentiel en *d* : $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$
- Complexité en **espace** ?

$$O(b^d)$$

$$\sum_{i=0}^{d} b^i = \frac{b^{d+1} - 1}{b - 1}$$

- Optimalité ?
  - Oui (si coût de chaque arc = 1)

↙ *La **complexité en espace** est le gros **problème***

2. Profondeur d'abord : propriétés

- Complétude ?
  - Non : échoue si profondeur infinie sur une branche ou si boucles
- Complexité en **temps** ?
  - Exponentiel en *m* : $O(b^m)$ Dramatique si *m* >> *d*
  - Mais si les solutions sont denses dans le graphe, peut-être plus rapide que "largeur d'abord"
- Complexité en **espace** ?
  - $O(bm)$ c.a.d. espace mémoire **linéaire** !!
- Optimalité ?
  - Non

Space: the final frontier

- In many (most) search problems, **memory** is often the limiting factor
- Let's consider a searching tree with branching factor **8** and depth **10**.
  Suppose that a node requires 8 bytes of storage (very little).
- Breadth-first search may require up to

  $(2^3)^{10} \times 2^3 = 2^{33}$ bytes = 8,000 Mbytes = 8 Gbytes

- One strategy is to trade time for memory.
- For instance, we can emulate breadth-first search by repeated application of depth-first search, each up to a preset limit.

→ Progressive deepening search

Space: the final frontier

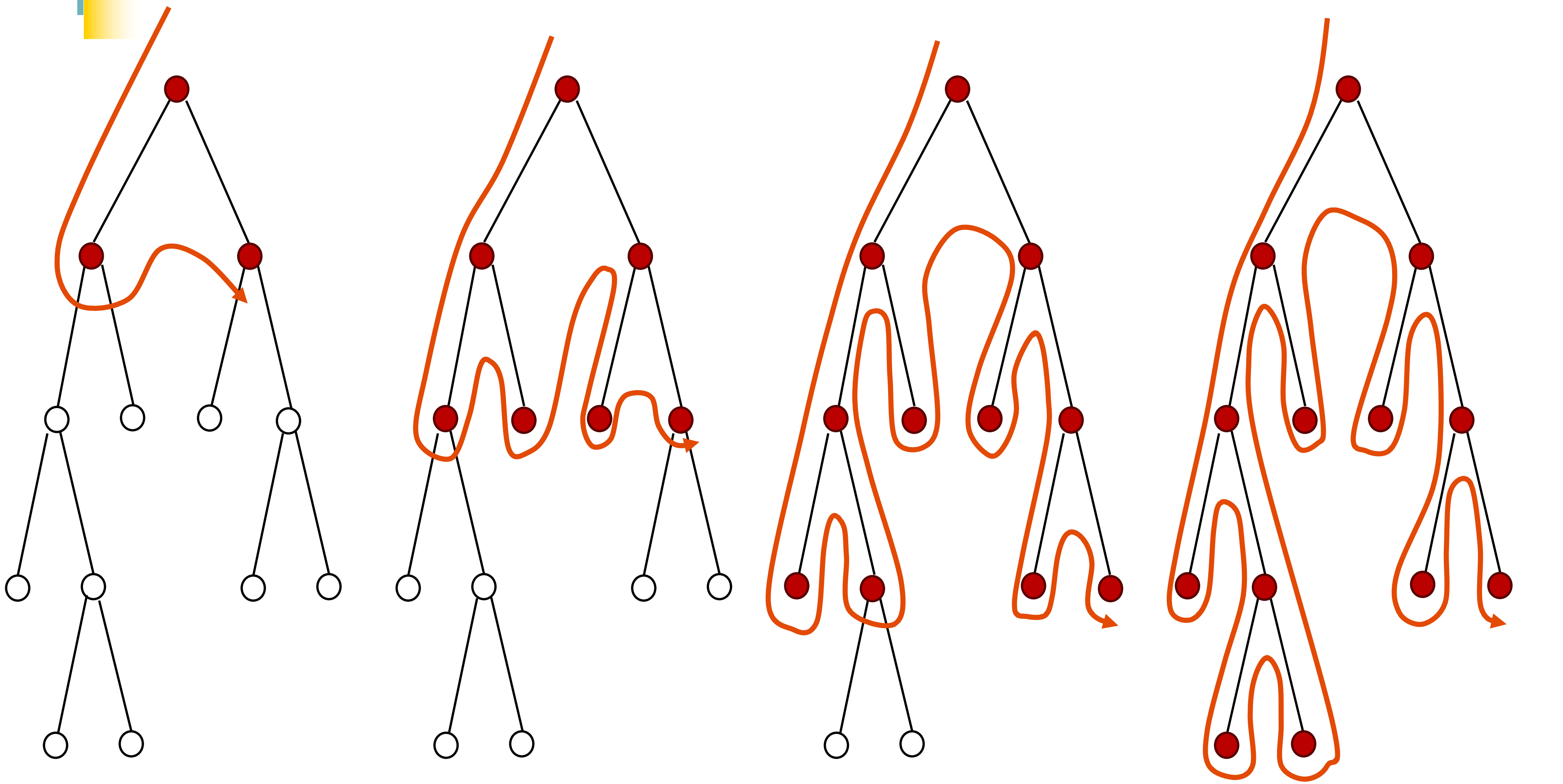
In many (most) search problems, **memory** is often the limiting factor

- Let's consider a searching tree with branching factor **10** and depth **10**.
  Suppose that a node requires 1k bytes of storage (e.g. position on a map).
- And a speed of 1O^6 nodes / s
- Breadth-first search will require less than 3 hours
  but need more than 10 Tbytes of memory!

  $(10)^{10} \times 10$
- One strategy is to trade time for memory.
- For instance, we can emulate breadth-first search by repeated application of depth-first search, each up to a preset limit.

→ Progressive deepening search

Profondeur itérative *(iterative deepening)*



Profondeur seuil = 1 ***Profondeur seuil = 2*** ***Profondeur seuil = 3*** ***Profondeur seuil = 4***

Richard Korf popularise et étudie cette technique en 1985.

Profondeur itérative : *propriétés*

- Complétude ?
  - Oui
- Complexité en temps ?

$$(d+1)\,b^0 \;+\; d\,b^1 \;+\; (d-1)\,b^2 \;+\ldots+\; b^d \;=\; O(b^d)$$

  - A peine plus que largeur d'abord
- Complexité en espace ?
  - $O(bd)$ cad espace mémoire **linéaire** !!
- Optimalité ?
  - Oui (si le coût des arcs = 1)

Méthode préférée quand grand espace de recherche et profondeur de la solution inconnue

- Calculs :

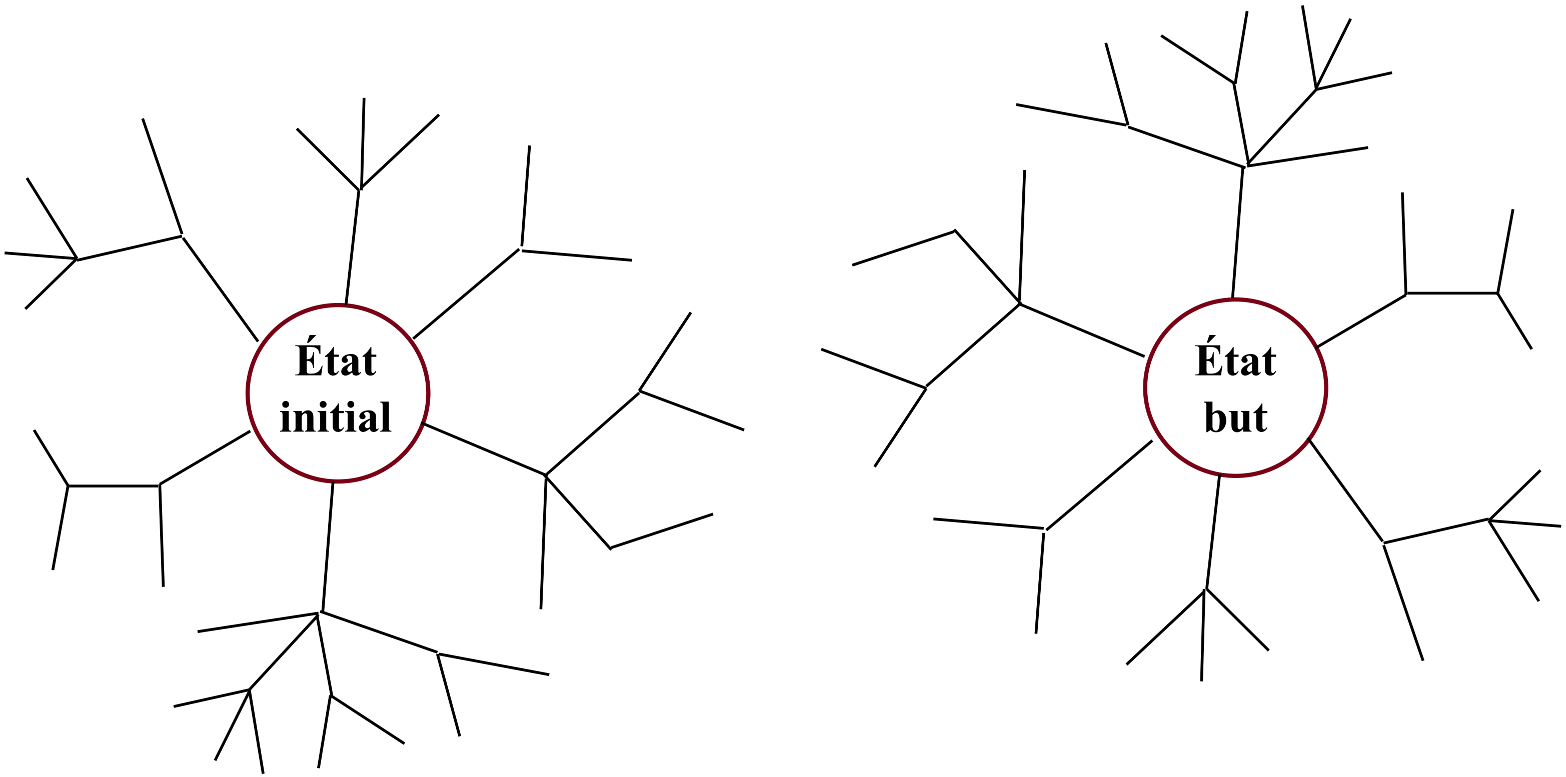
$$
\begin{aligned}
& b^k + 2\,b^{k-1} + 3\,b^{k-2} + \ldots + k\,b \\
&= b^k\left(1 + 2\,b^{-1} + 3\,b^{-2} + \ldots + k\,b^{1-k}\right) \\
&\leq b^k \left(\sum_{i=1}^{\infty} i\,b^{(1-i)}\right) \\
&= b^k \left(\sum_{i=1}^{\infty} i\,\frac{1}{b}^{(i-1)}\right) \\
&= b^k\,\frac{1}{(1-\frac{1}{b})^2}
\end{aligned}
$$

En **pratique** : si $b = 10$ et $k = 5$

N(IDS) = 50 + 400 + 3000 + 20000 + 100 000 = 123 450

N(BFS) = 10 + 100 + 1000 + 10000 + 100 000 = 111 110

2. Recherche bidirectionnelle



2. Coût uniforme

2. Les méthodes en **meilleur d'abord**

```
OUVERT ← {état initial} ;   FERME ← ∅   ;   Succès ←  Faux

Tant que OUVERT ≠ ∅ et Succès = Faux
   Etape 1 : Choix d'un noeud n dans OUVERT
   Si n est un état terminal alors Succès ← Vrai
      et retourner le chemin solution trouvé
   Sinon
      Etape 2 : développement de n
        Supprimer n de OUVERT
        L'ajouter à FERME
        Pour chaque successeur s de n
           Si s n'appartient ni à OUVERT ni à FERME
              ajouter s à OUVERT
              père(s)← n
           Sinon mise à jour éventuelle du père de s
Echec si OUVERT = ∅
```

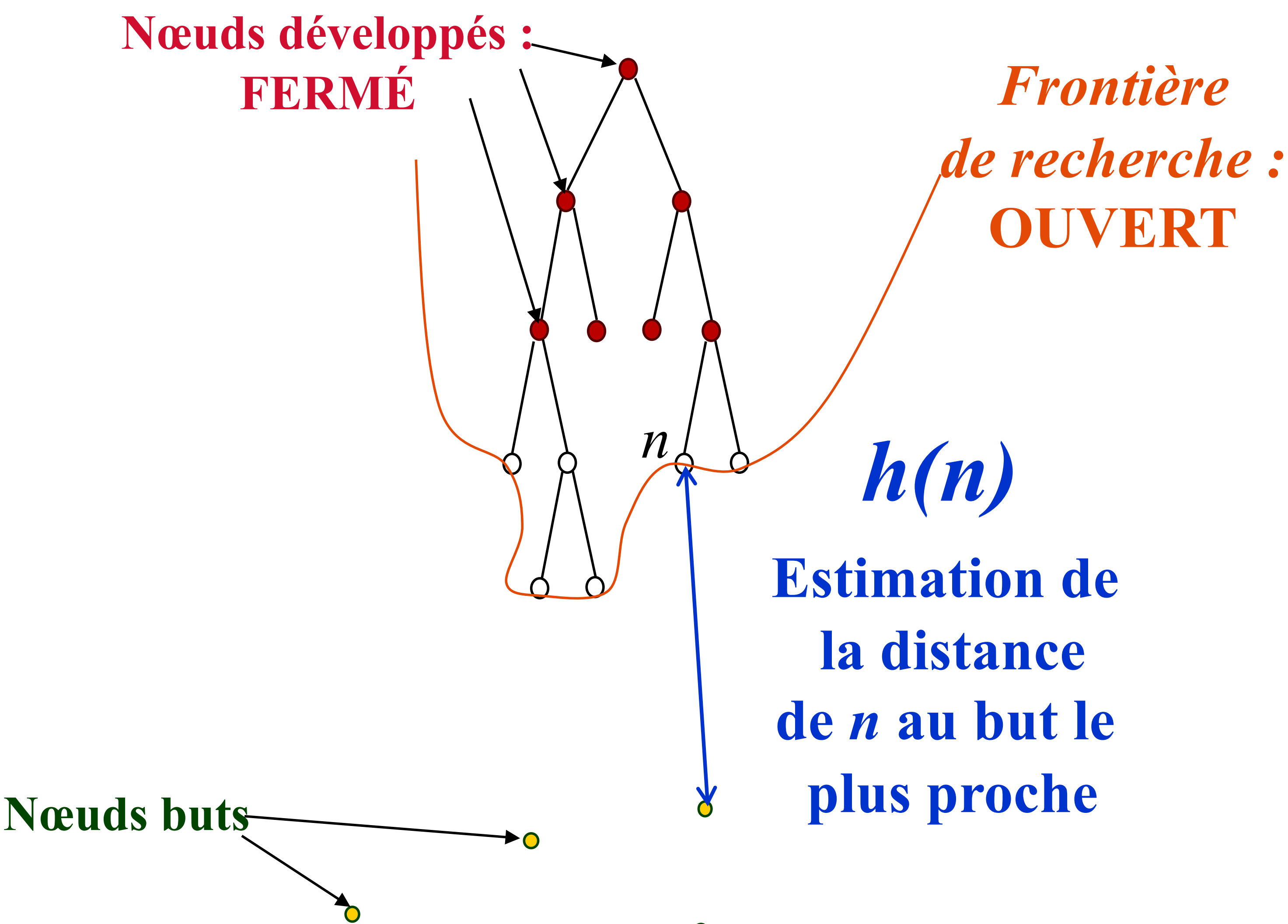
2. En meilleur d'abord : la **fonction d'évaluation**

- Le choix du nœud à développer dépend d'une fonction d'évaluation $f(n)$ estimant le mérite de n

- Les nœuds n sur la frontière de recherche sont ordonnés dans une liste OUVERT par ordre croissant

2. Les méthodes ***informées***

Disposent d'une information sur la *proximité au but*



2. La méthode A (A*)

Inventée en 1968
[Hart, Nilsson & Raphael]

Fonction d'évaluation :

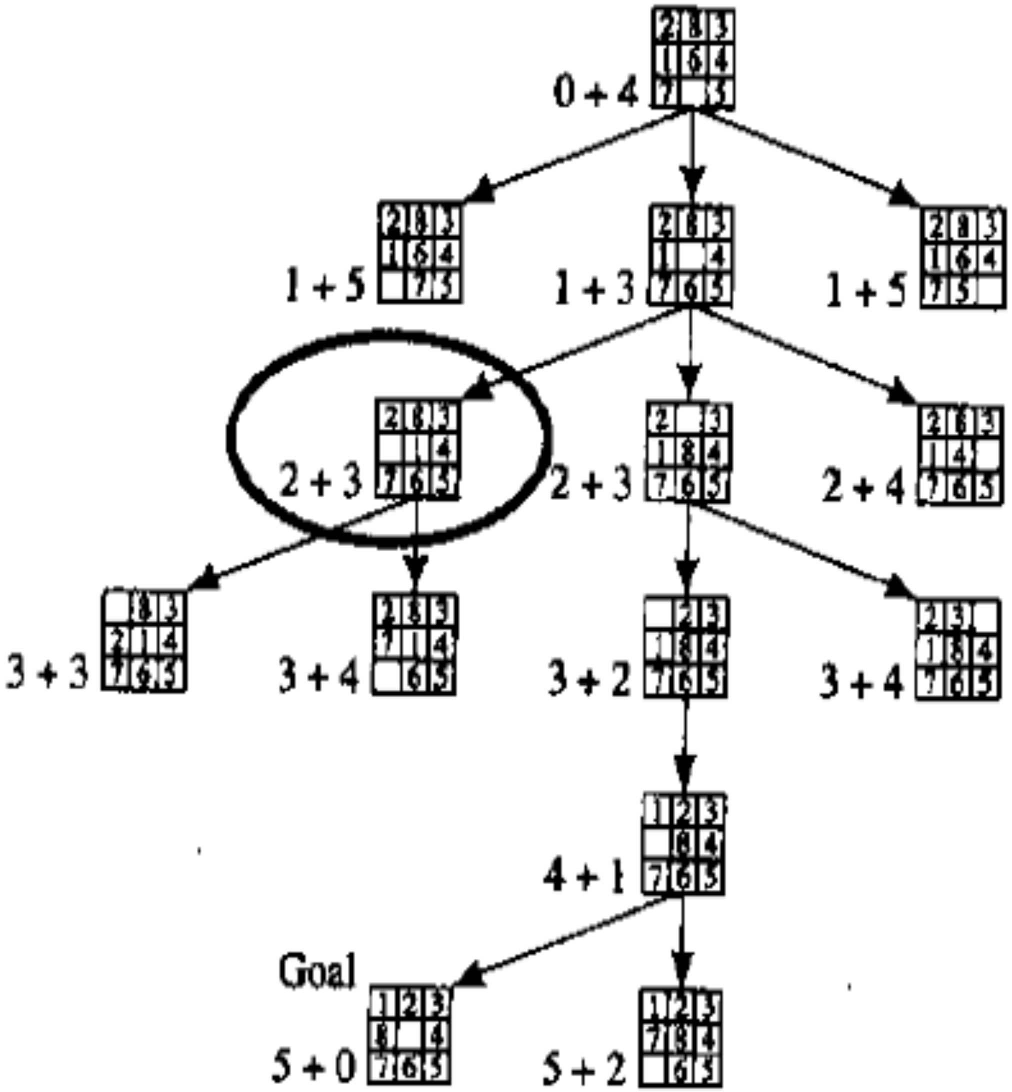
$$f(n) = g(n) + h(n)$$

Coût estimé d'un chemin de la racine à un objectif passant par *n*

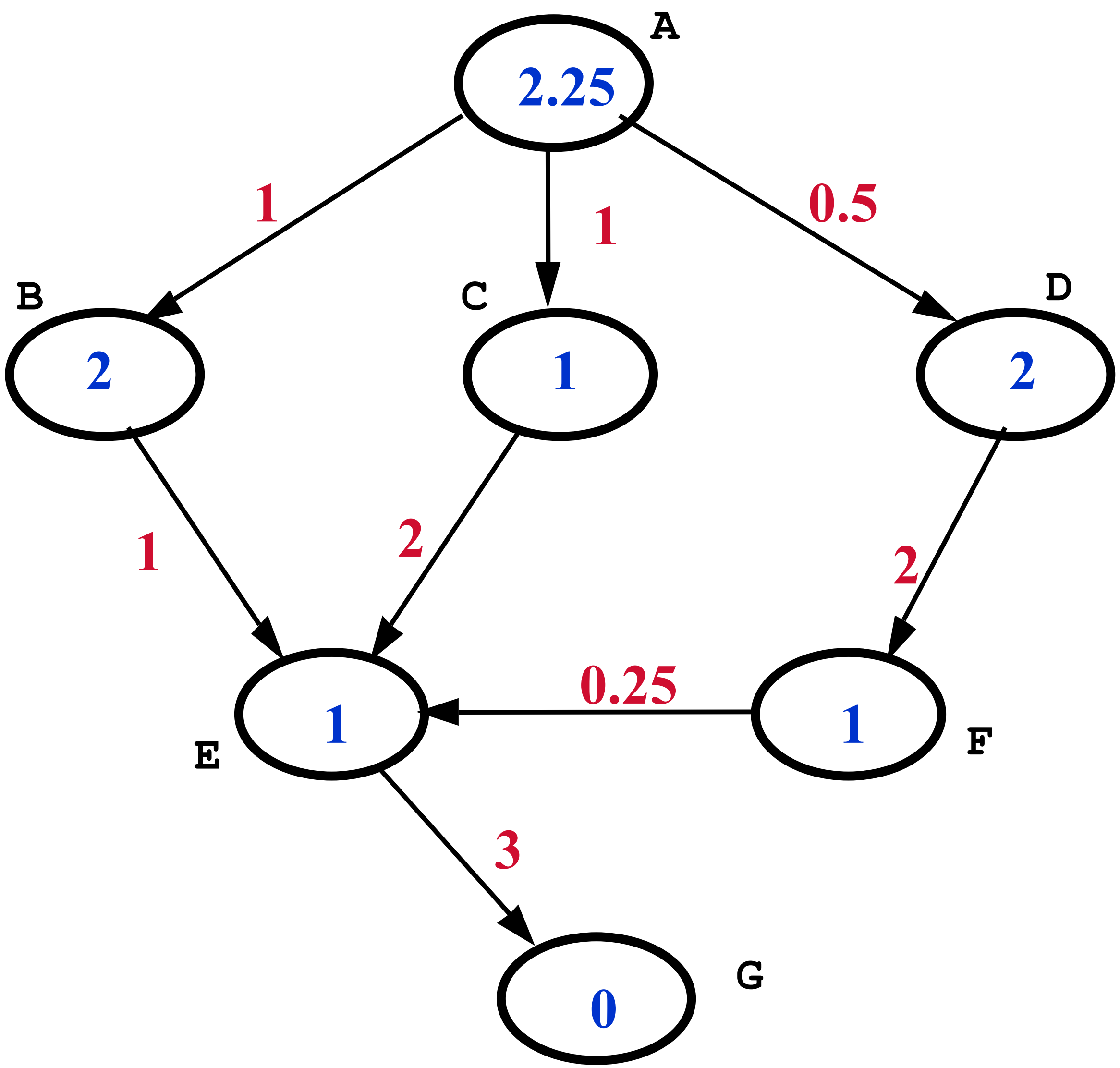
Coût du chemin le plus court connu actuellement pour aller de la racine au nœud *n* : <u>fonction dynamique</u>

Estimation du coût optimal pour atteindre un objectif à partir du nœud *n* : <u>fonction statique</u>

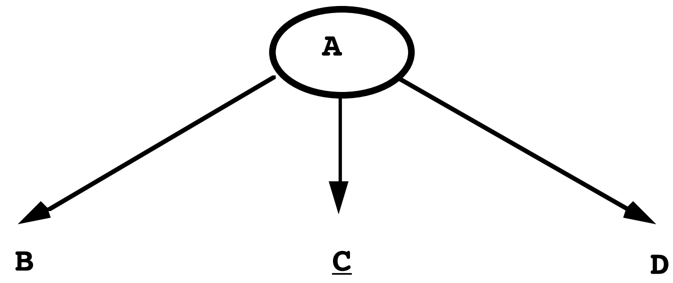
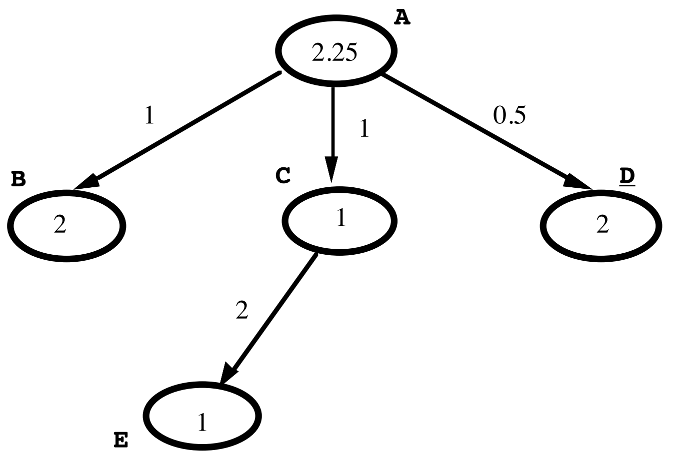
2. La méthode A (A*)



2. La méthode A (A*) : illustration (1/4)



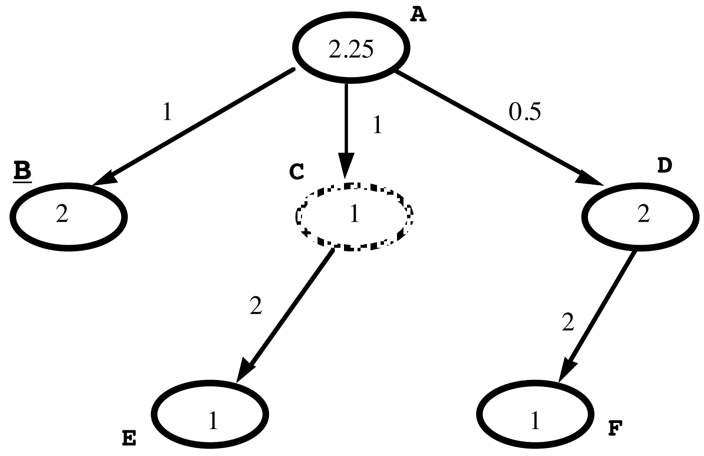
2. La méthode A (A*) : illustration (2/4)

FERMÉ / OUVERT	Développement de l'arborescence
FERMÉ : A(2.25) OUVERT : C(2), D(2.5), B(3)	
FERMÉ : C(2), A(2.25) OUVERT : D(2.5), B(3), E(4)	

2. La méthode A (A*) : illustration (3/4)

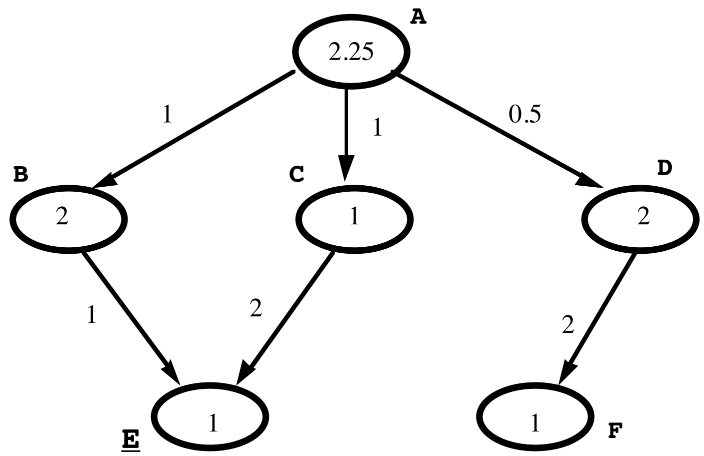
FERMÉ : C(2), A(2.25), D(2.5)

OUVERT : B(3), F(3.5), E(4)



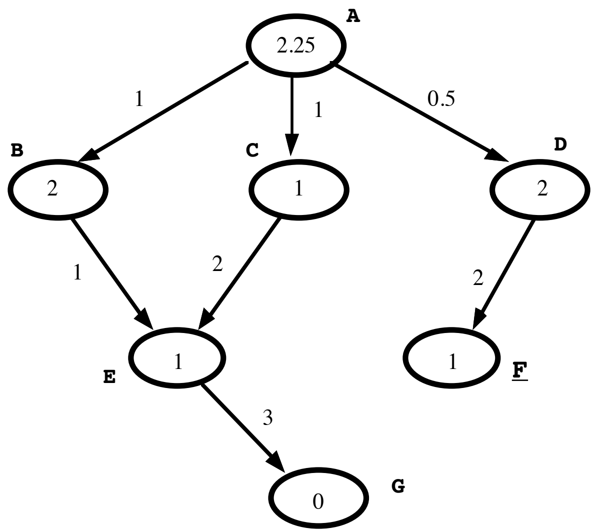
FERMÉ : C(2), A(2.25), D(2.5), B(3)

OUVERT : E(3) par B, F(3.5), E(4) par C est éliminé

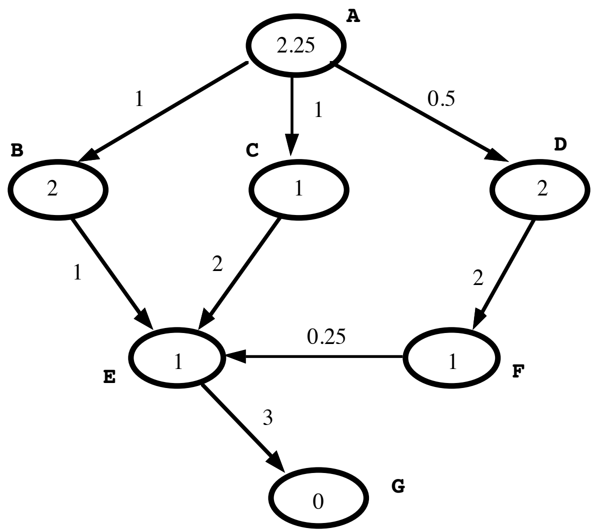


2. La méthode A (A*) : illustration (4/4)

FERMÉ : C(2), A(2.25), D(2.5), B(3),
E(3) par B
OUVERT : F(3.5), G(5)



FERMÉ : C(2), A(2.25), D(2.5), B(3),
E(3) par B, F(3.5)
OUVERT : E(3.75) par F, mais pas meilleur que E(3) par B de FERMÉ, donc pas de ré-actualisation de E ni donc de G(5). Et comme G(5) est le seul à rester dans OUVERT et que c'est le but, Succès par le chemin A-B-E-G



Soit la carte de la figure 10.2.1 représentant des villes de Roumanie et les distances entre villes à vol d'oiseau.

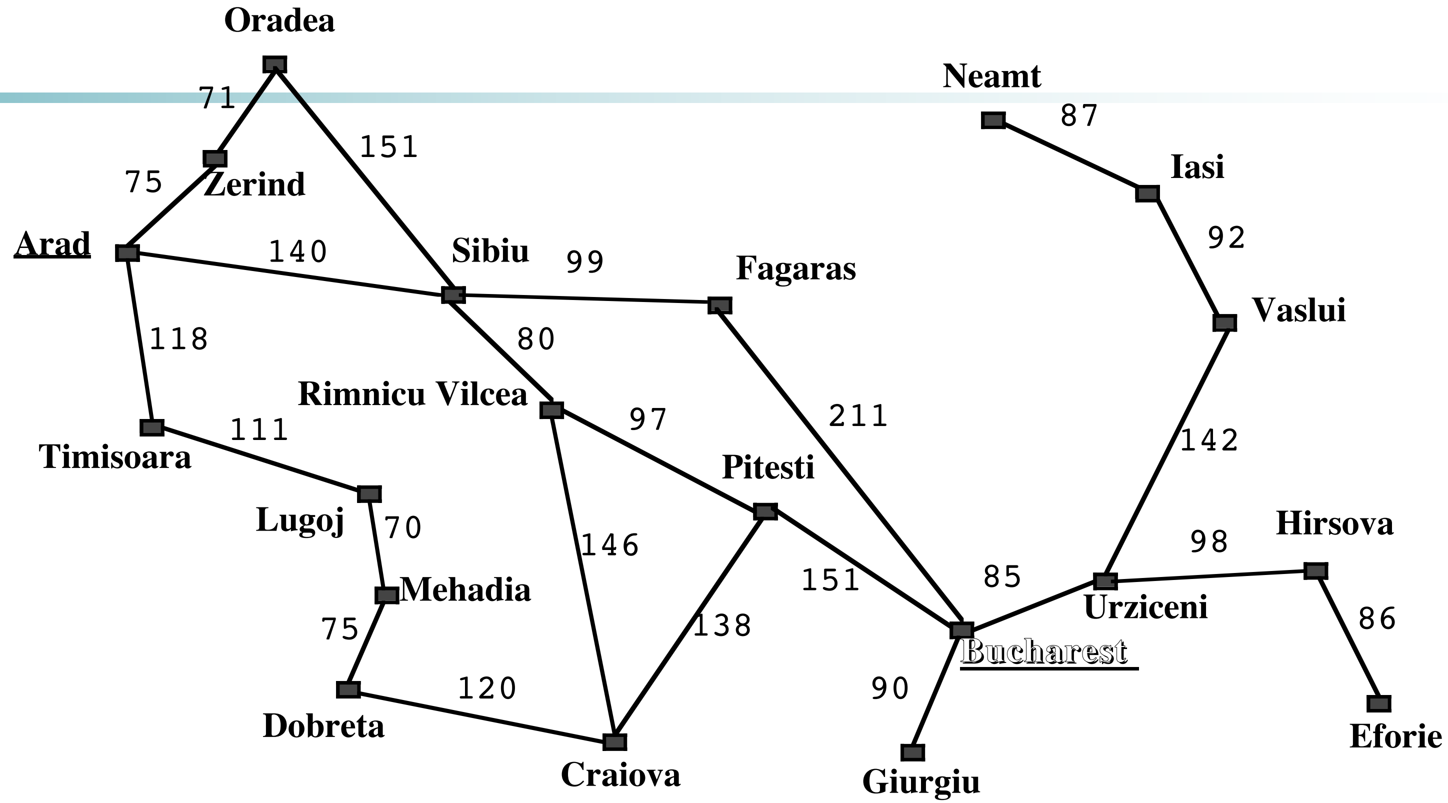


Figure 10.2.1 : Carte de la Roumanie avec les distances par la route entre grandes villes.

Arad	366	Fagaras	178	Mehadia	241	Sibiu	253
Bucharest	0	Giurgiu	77	Neamt	234	Timisoara	329
Craiova	160	Hirsova	151	Oradea	380	Urziceni	80
Dobreta	242	Iasi	226	Pitesti	148	Vaslui	199
Eforie	161	Lugoj	244	Rimnicu Vilcea	193	Zerind	374

Table 10.2.1 : Table des distances à vol d'oiseau à Bucharest.

2. L'optimalité de A*

Propriété (optimalité ou admissibilité de A*) :

Si la fonction d'estimation de la distance au but h *est systématiquement optimiste, l'algorithme A s'appelle A* et retourne nécessairement une solution optimale.*

Preuve :

2. L'*optimalité* de A*

Propriété (optimalité ou admissibilité de A*) :

> *Si la fonction d'estimation de la distance au but h est **systématiquement optimiste**, l'algorithme A s'appelle A* et retourne nécessairement une solution optimale.*

Preuve : **Supposons** qu'un chemin optimal soit de coût C^* mais que l'algorithme retourne un chemin de coût $C > C^*$

À tout instant, il doit exister un noeud *n* sur la frontière appartenant au chemin optimal

$f(n) > C^*$ (sinon il aurait été développé)

$f(n) = g(n) + h(n)$ (par définition)

$f(n) = g^*(n) + h(n)$ (car n est sur un chemin optimal)

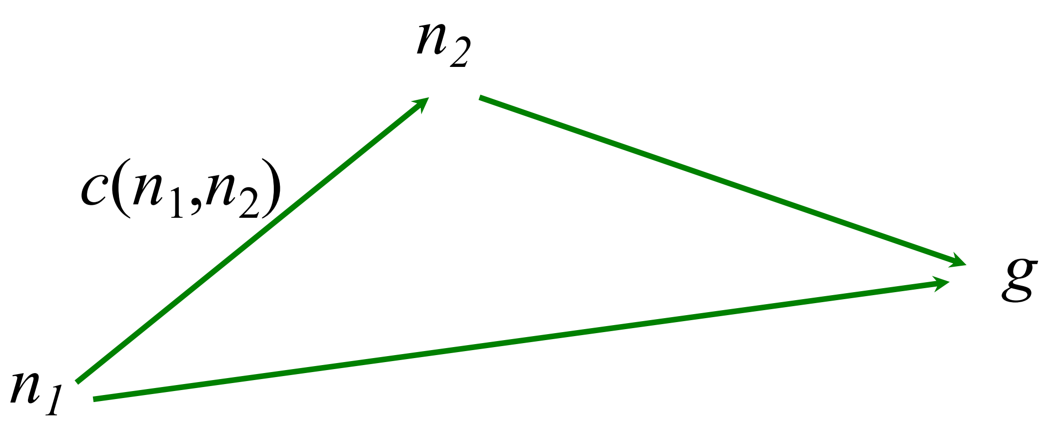
$f(n) \leq g^*(n) + h^*(n)$ (car h est admissible et donc $h(n) \leq h^*(n)$)

$f(n) \leq C^*$ (car par définition $C^* = g^*(n) + h^*(n)$) donc contradiction

2. La consistance

Propriété (*inégalité triangulaire*) :

$$\forall \; n_1, n_2, \quad h(n_1) \leq c(n_1, n_2) + h(n_2)$$



2. Algorithme plus ou moins informé

Définition :

Si deux versions A_1* et A_2* d'un algorithme A* ne diffèrent que par leur fonction d'estimation h avec n (non terminal) $h_1(n) < h_2(n)$, alors on dit que A_2* est mieux informé que A_1*.

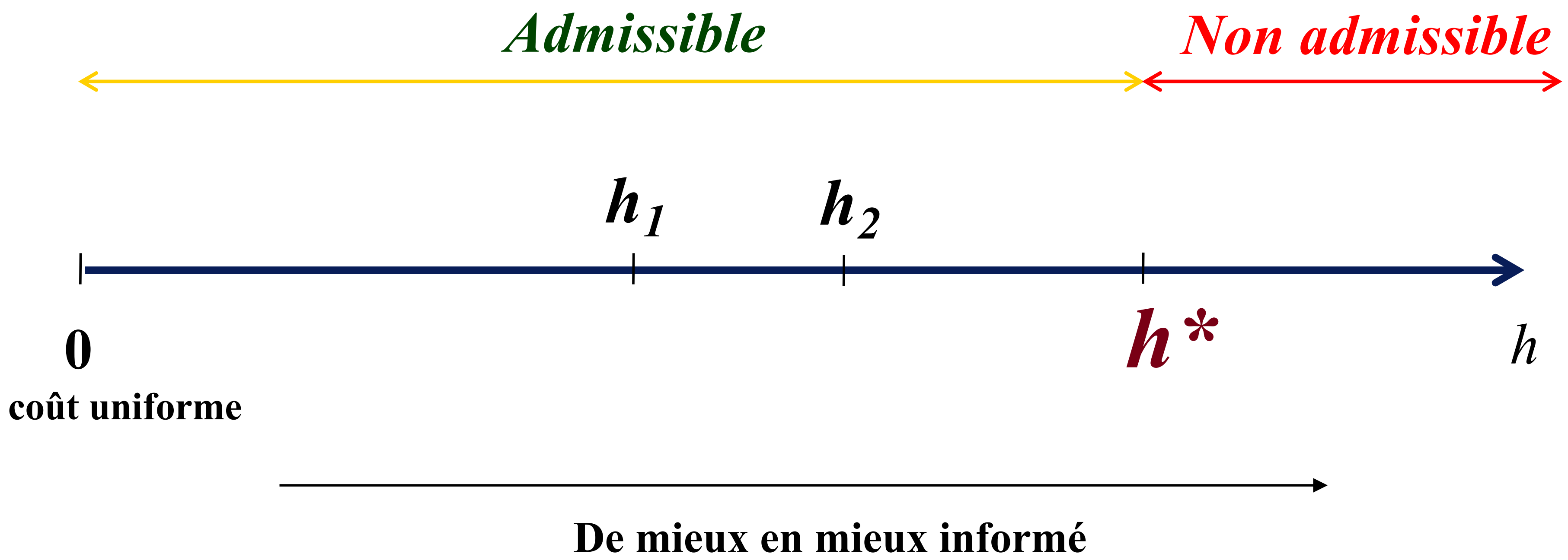
Théorème :

Si A_2* est mieux informé que A_1*, alors lorsque leurs recherches sont terminées, tous les nœuds développés par A_2* l'ont également été par A_1* (et peut-être d'autres).

⇙ A_2* est plus efficace que A_1*

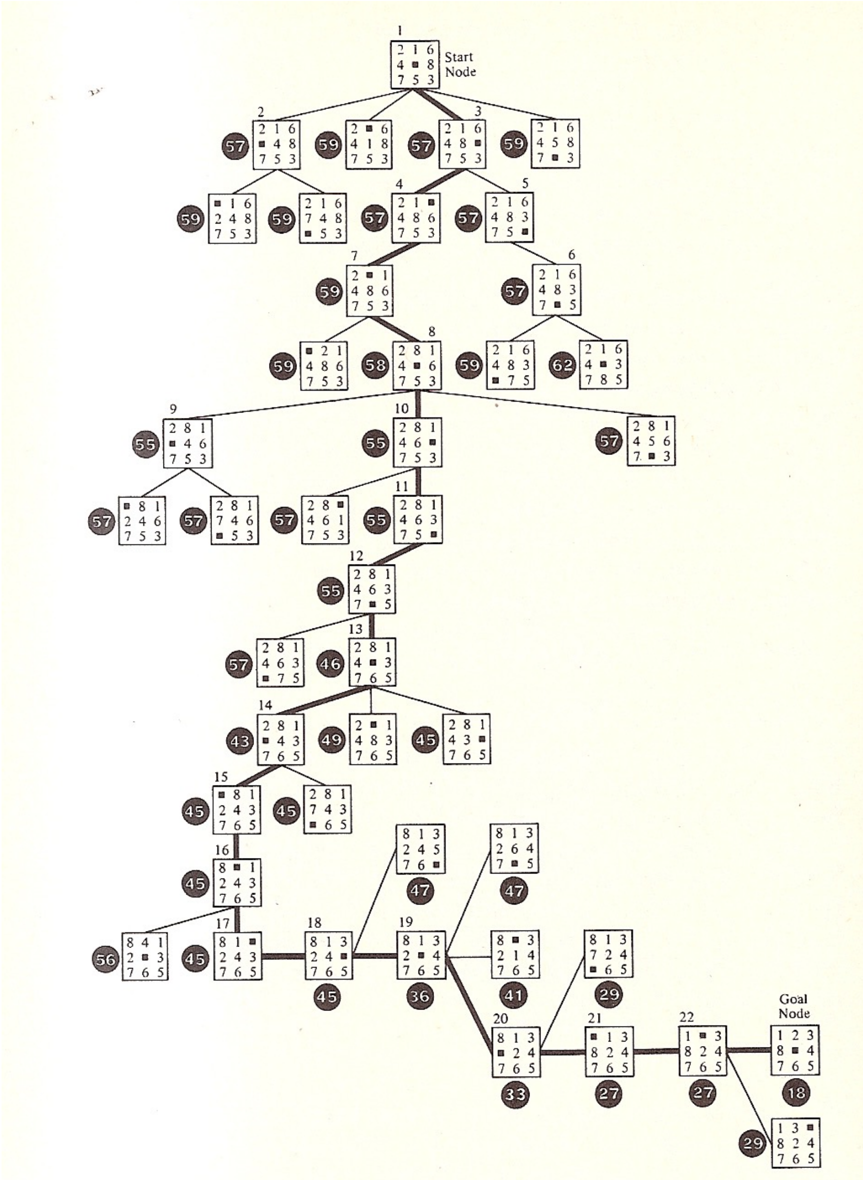
2. Algorithme plus ou moins informé

$$\forall n, \quad h_1(n) \leq h_2(n)$$



2. Algorithme plus ou moins informé : illustration

- Taquin



2. Algorithme plus ou moins informé : tableau comparatif

	Nombre moyen de nœuds développés			*Facteur de branchement effectif*		
Profondeur	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h2)$
2	10	6	6	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	364404	227	73	2,78	1,42	1,24
14	3473941	539	113	2,83	1,44	1,23
16		1301	211		1,45	1,25
18		3056	363		1,46	1,26
20		7276	676		1,47	1,27
22		18094	1219		1,48	1,28
24		39135	1641		1,48	1,26
Moyenne				2,8	1,43	1,27

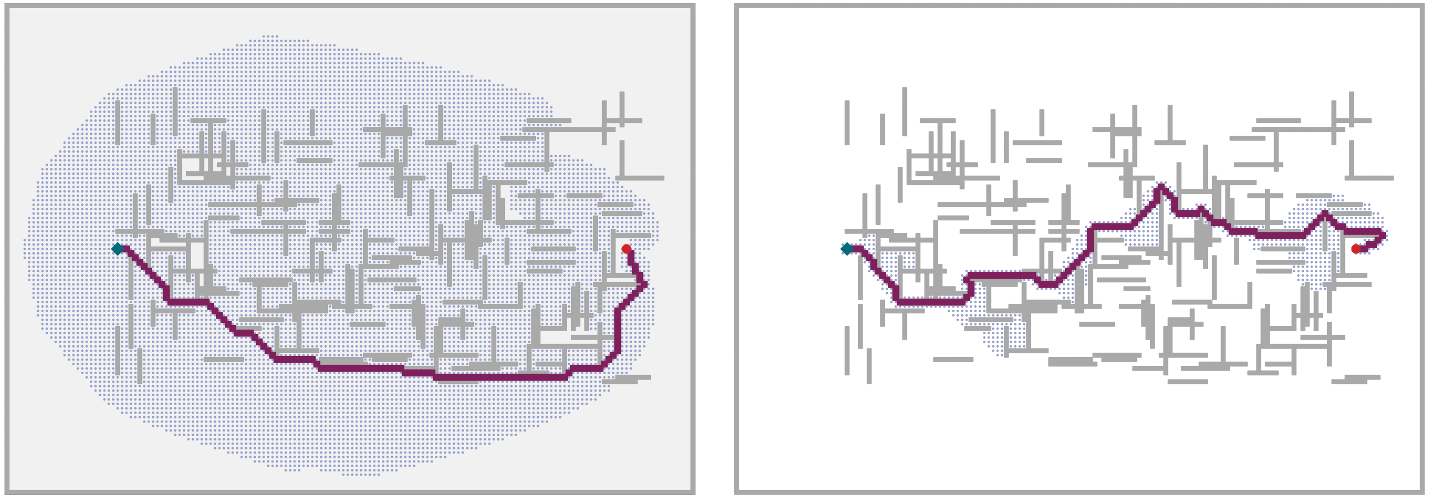
2. Algorithme plus ou moins informé : tableau comparatif

- Taquin 4x4 :
  - ~ 6 jours pour trouver une solution en moyenne sans heuristique
  - ~ 50s avec heuristique h_1
  - ~ 0.1s avec heuristique h_2

Korf, R. (2000) « *Recent progress in the design and analysis of admissible heuristic functions* », Proc. of SARA-2000, Springer-Verlag.

Algorithme A pondéré*

$$g(n) + W \times h(n) \quad (1 < W < \infty)$$



(a) (b)

Figure 3.21 Two searches on the same grid: (a) an A* search and (b) a weighted A* search with weight $W = 2$. The gray bars are obstacles, the purple line is the path from the green start to red goal, and the small dots are states that were reached by each search. On this particular problem, weighted A* explores 7 times fewer states and finds a path that is 5% more costly.

[Russell & Norvig « Artificial Intelligence. A Modern Approach». (4th edition, 2021)], Fig 3.21

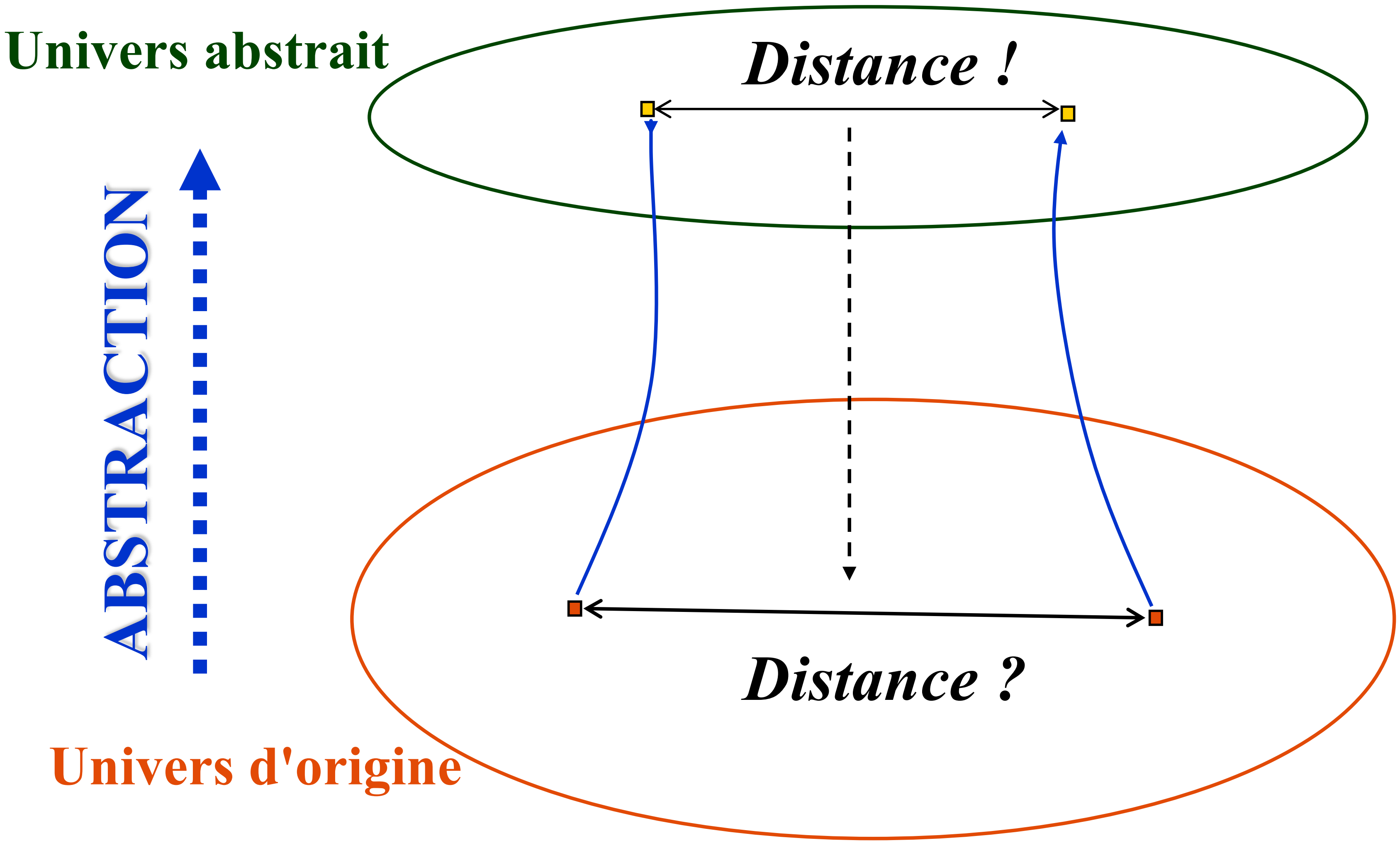
2. A* itératif (iterative-deepening A*: IDA*)

- [Nilsson, p.153]

2. *La découverte de fonction heuristique*

- Approches possibles
  - Par **abstraction**
  - Par essais et erreurs
  - Par apprentissage

Principe :


Univers abstrait
Distance !
ABSTRACTION
Distance ?
Univers d'origine

2. *La découverte de fonction heuristique par abstraction*

- Illustration : le jeu du taquin [Pearl, 1984]

  - **Soient les** *prédicats de description* :
    - `sur(X,Y) :` **la tuile X est sur la case Y**
    - `vide(Y) :` **il n'y a pas de tuile sur la case Y**
    - `adj(Y,Z) :` **les cases Y et Z sont adjacentes**

  - **Soit l'** *opérateur de déplacement* `bouger(X,Y,Z)` **défini par :**
    - Préconditions : `sur(X,Y) & vide(Z) & adj(Y,Z)`
    - Ajouts : `sur(X,Z) & vide(Y)`
    - Retraits : `sur(X,Y) & vide(Z)`

  - **Problème 1** : Trouver une séquence d'opérateurs `bouger(X,Y,Z)` instanciés pour aller de l'état initial à l'état final.

  - **Problème 2** : Estimer la distance au but, cad le nombre d'opérations nécessaires

2. La découverte de fonction heuristique par abstraction

Opérateur de déplacement `bouger(X,Y,Z)` **défini par :**

- Préconditions : `sur(X,Y)` & `vide(Z)` & `adj(Y,Z)`
- Ajouts : `sur(X,Z)` & `vide(Y)`
- Retraits : `sur(X,Y)` & `vide(Z)`

2. La découverte de fonction heuristique par abstraction

Opérateur de déplacement `bouger(X,Y,Z)` **défini par :**

Préconditions :	`sur(X,Y)`	&	`vide(Z)`	&	`adj(Y,Z)`
Ajouts :	`sur(X,Z)`	&	`vide(Y)`		
Retraits :	`sur(X,Y)`	&	`vide(Z)`		

- Abstraction par relaxation (suppresion) des préconditions
  - Suppression des préconditions `vide(Z)` & `adj(Y,Z)` :
    - Chaque carreau mal placé peut être directement mis sur la case objectif : $\boxed{h_1}$
  - Suppression de la précondition `vide(Z)` :
    - Chaque carreau mal placé peut être déplacé sur une case adjacente : $\boxed{h_2}$
  - Suppression de la précondition `adj(Y,Z)` :
    - Chaque carreau mal placé peut être directement mis sur la case vide : $\boxed{h_3}$

2. *La découverte de fonction heuristique par essais / erreurs*

- Principes :

  1. Sur un petit problème dont une solution optimale est connue : chercher une fonction h telle que $f(n) = g(n) + h(n)$

  2. Si plusieurs heuristiques admissibles h_i sont connues, prendre

$$h(n) \;=\; \arg\max_i \; h_i(n)$$

  3. Sélectionner des **attributs** de description de l'état qui semblent jouer un rôle dans l'estimation de la **distance** au but et les **combiner** dans une fonction d'évaluation (.... éventuellement par apprentissage)

2. La découverte de fonction heuristique : état de l'art

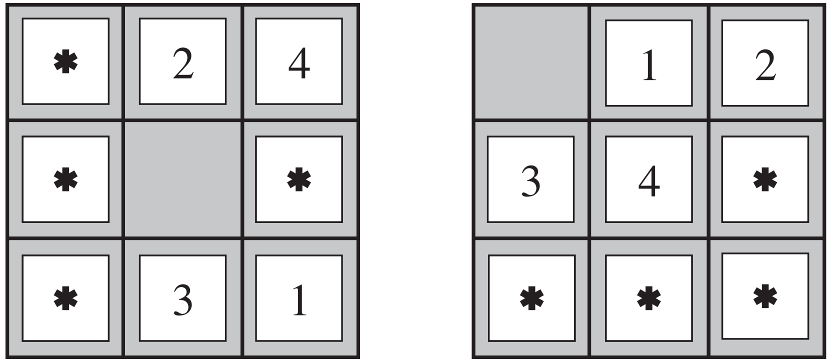
- EURISKO ? [Lenat, 1982] : recherche par transformation syntaxique dans l'espace des heuristiques

- Par abstraction
  - ABSOLVER [Priedetis, 1993]
    - Par la méthode "relaxed problem" : relâche les restrictions sur les opérateurs
    - Découverte de l'heuristique la plus efficace pour le taquin
    - Découverte d'heuristiques pour le Rubik's cube, ... (13 pbs)

- Par apprentissage
  - Apprentissage par renforcement (cf cours n° 4)
  - Apprentissage par recherche d'abstraction (Thèse J-D Zucker (1996))

Recherche avec utilisation de « *patterns* »

Un sous-problème



Start State Goal State

1ère idée :

- Principe

  1. Mémoriser la solution de **tous les sous-problèmes**, par exemple les sous-problèmes avec 4 tuiles -> génère une « **Pattern database** »
  2. Calculer une **heuristique admissible** pour chacun de ces sous-problèmes
  3. Prendre l'heuristique qui est **le max de toutes ces heuristiques**

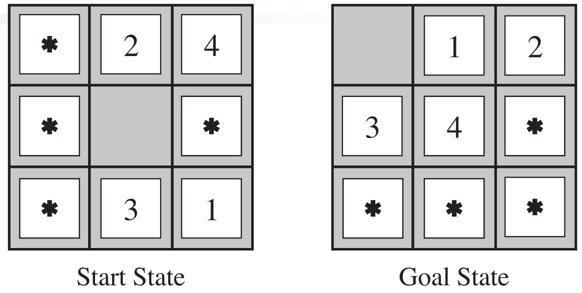
- E.g. le taquin à 15 tuiles est résolu en moyenne 1000 fois plus rapidement qu'avec l'heuristique de Manhattan

- Mais la taille d'une « pattern database » croît très très vite avec la taille du problème

Recherche avec utilisation de « patterns »

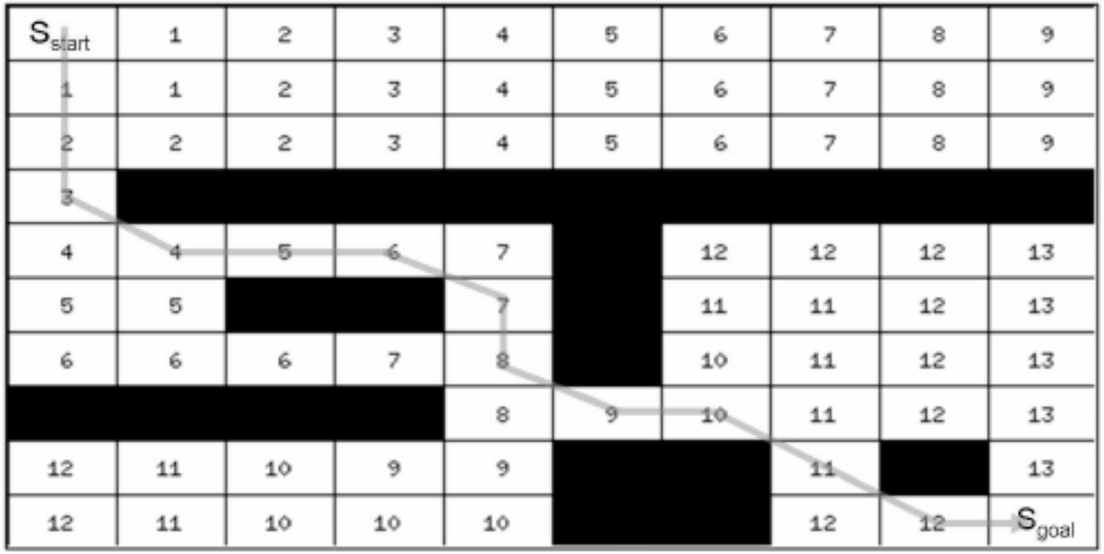
2ème idée :

Un sous-problème

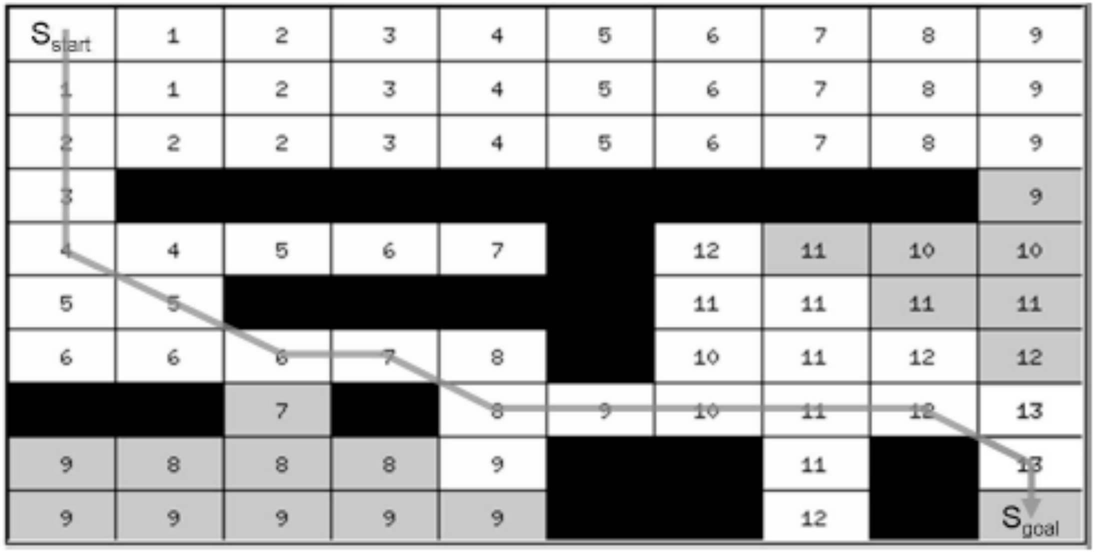


- Pourrait-on additionner les heuristiques au lieu d'en prendre le max ?
  - **Non**. Car les sous-problèmes sont généralement non indépendants
    - Il se pourrait que les mouvements pour résoudre le sous-problème avec les tuiles 1-2-3-4 soient communs avec la solution pour le sous-problème 5-6-7-8.
  - **Principe. Calculer des « disjoint pattern databases »**
    - **Pas simple**. Voir [Ariel Felner, Korf & Hanan « Additive Pattern database heuristics », JAIR, 22, 279-318, (2004)]
- E.g. le taquin à 15 tuiles est résolu en moyenne 10 000 fois plus rapidement qu'avec l'heuristique de Manhattan
- Le taquin à 24 tuiles est résolu 1 000 000 de fois plus rapidement.

LifeLong A*



Changed Eight-Connected Gridworld

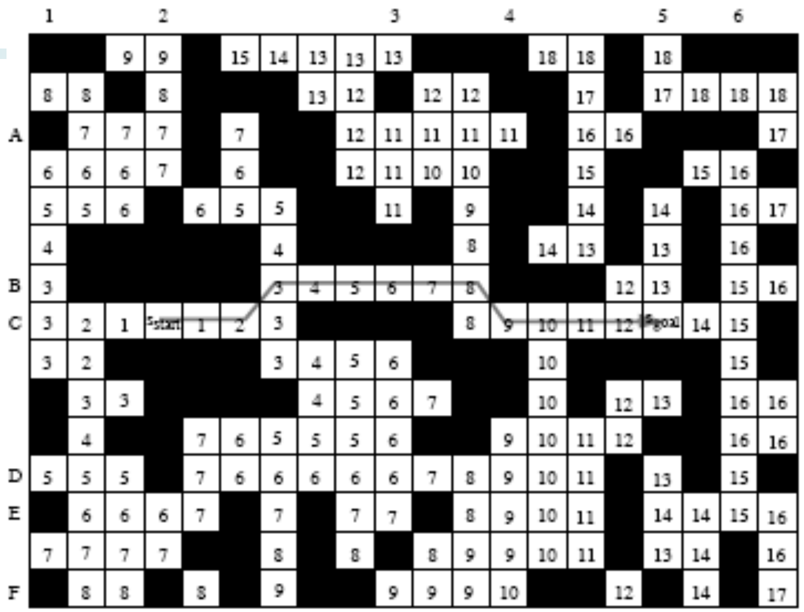


LifeLong A*

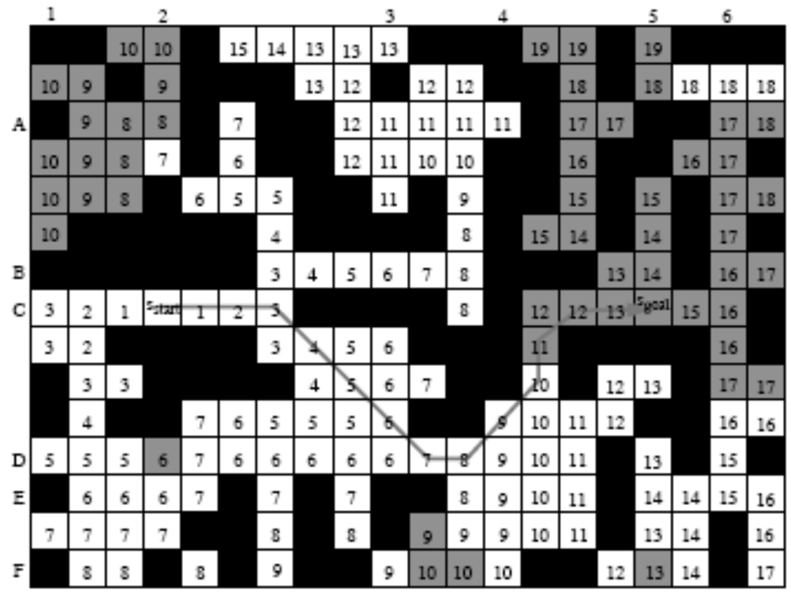
Comment réutiliser au mieux les connaissances issues des explorations antérieures du monde ?

[Koenig, Likhachev & Furcy, AIj (2003)]

[Fedon & Cornuéjols, 2008]



Changed Eight-Connected Gridworld



Extensions

Recherche de plus court chemin multi-agents

Problème : risque d'interférences « destructrices »

1. Algorithmes avec replanification
   - Recherche indépendante des chemins de chaque agent
   - Replanification si obstacle rencontré
     - Pb : si inter-blocages
     - Heuristique : ajout de « bruit »
       - *Position des agents*
       - *Heuristique de distance utilisée*

Recherche de plus court chemin multi-agents

Problème : risque d'interférences « destructrices »

2. Recherche coopérative
   - Recherche indépendante des chemins de chaque agent
     - Dans un **ordre prédéterminé**
     - Avec **table de réservation** remplie incrémentalement
   - Ne permet pas de résoudre certains blocages
   - Coûteux en temps
   - Heuristique :
     - *Changer l'ordre*
     - *Introduire de la replanification*

Recherche en temps réel

- Algorithmes

  - RTA*

  - LRTA* (Learning RTA*)

    - Modification de l'heuristique par apprentissage

Recherche avec cible mouvante

- Algorithmes

  - Moving target search

    - Une heuristique pour chaque position possible de la cible
      - Table d’heuristiques

Recherche hiérarchique de chemin

IA et jeux vidéo

Les graphes ET/OU

2. Les graphes ET/OU

- Motivation
- Illustrations
- Différences avec les algorithmes A
  - On ne maintient plus des chemins mais des sous-graphes
  - On a donc une nouvelle fonction d'évaluation
    - tenant compte du **mérite des noeuds frontières** (sous-problèmes)
    - mais aussi des **coûts de combinaison** des sous-problèmes

2. Algorithme AO* : illustration

- Structure chimique

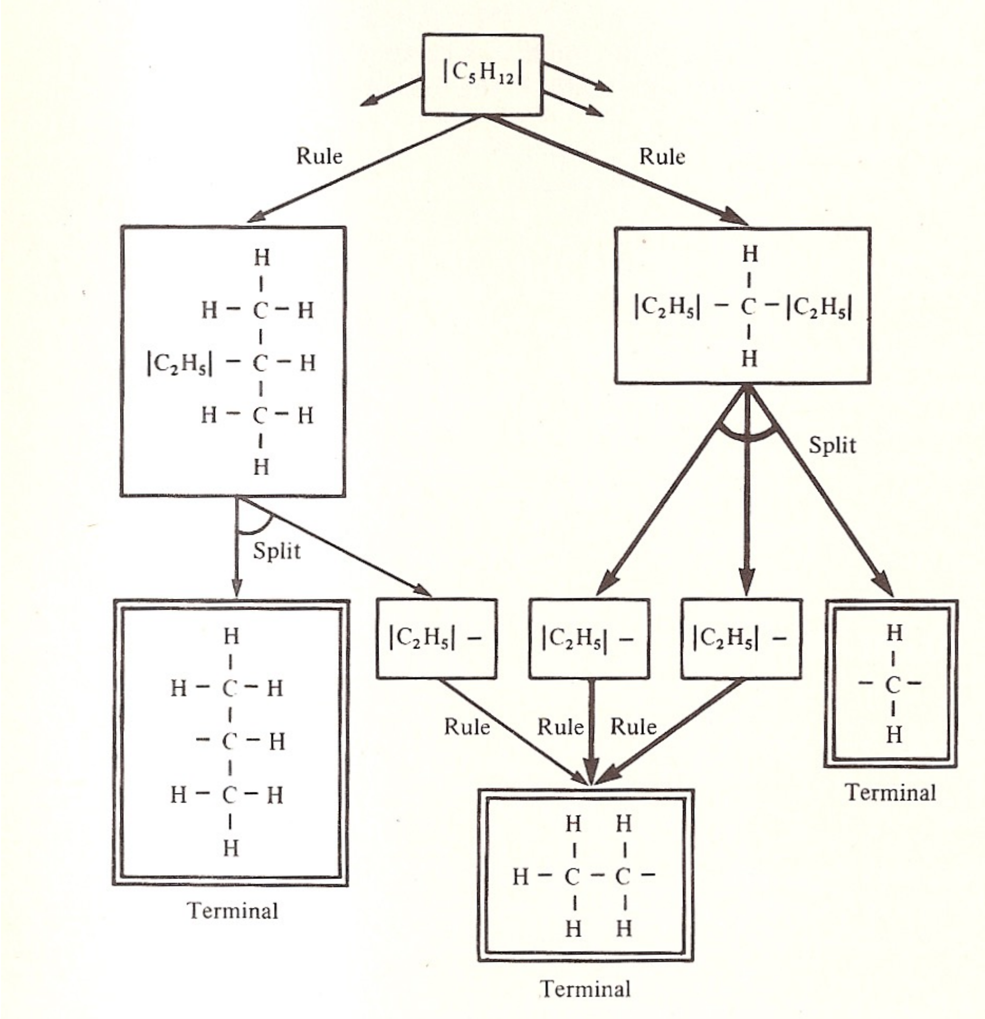


Fig. 1.11 An AND/OR tree for a chemical structure problem.

2. Algorithme AO* : illustration

- Calcul d' intégrale

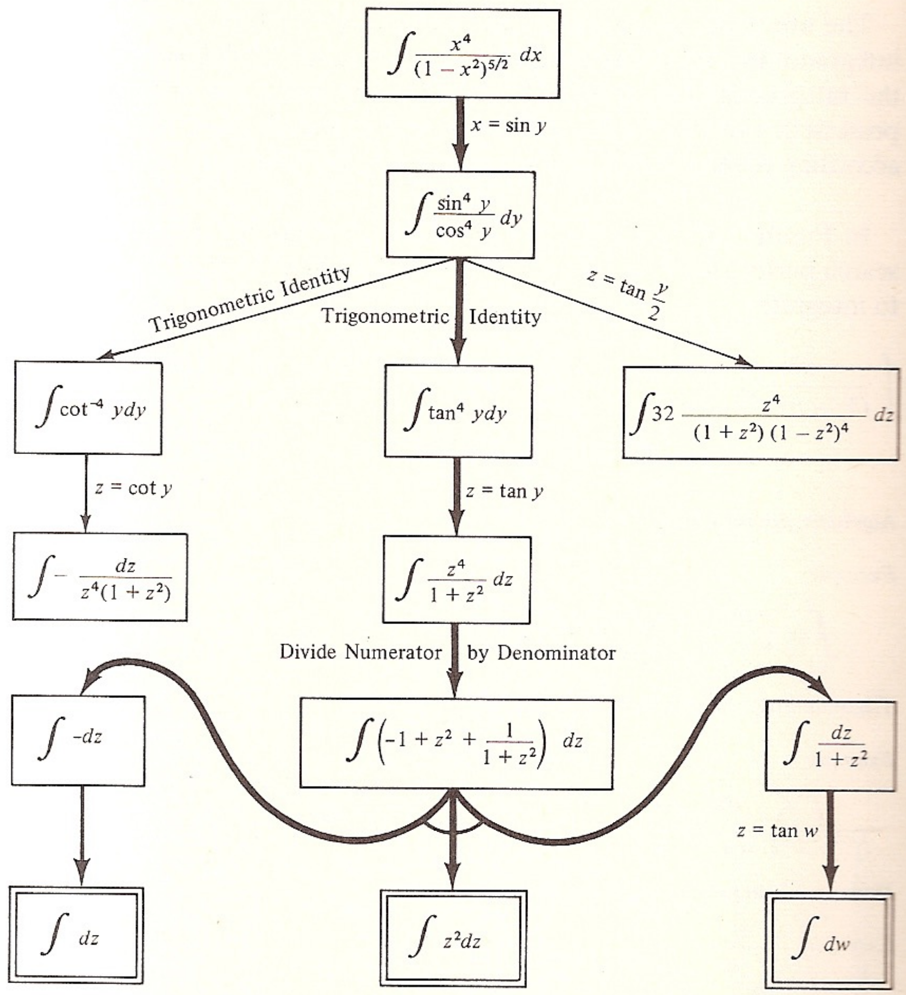


Fig. 1.13 An AND/OR tree for an integration problem.

2. Algorithme AO* : illustration

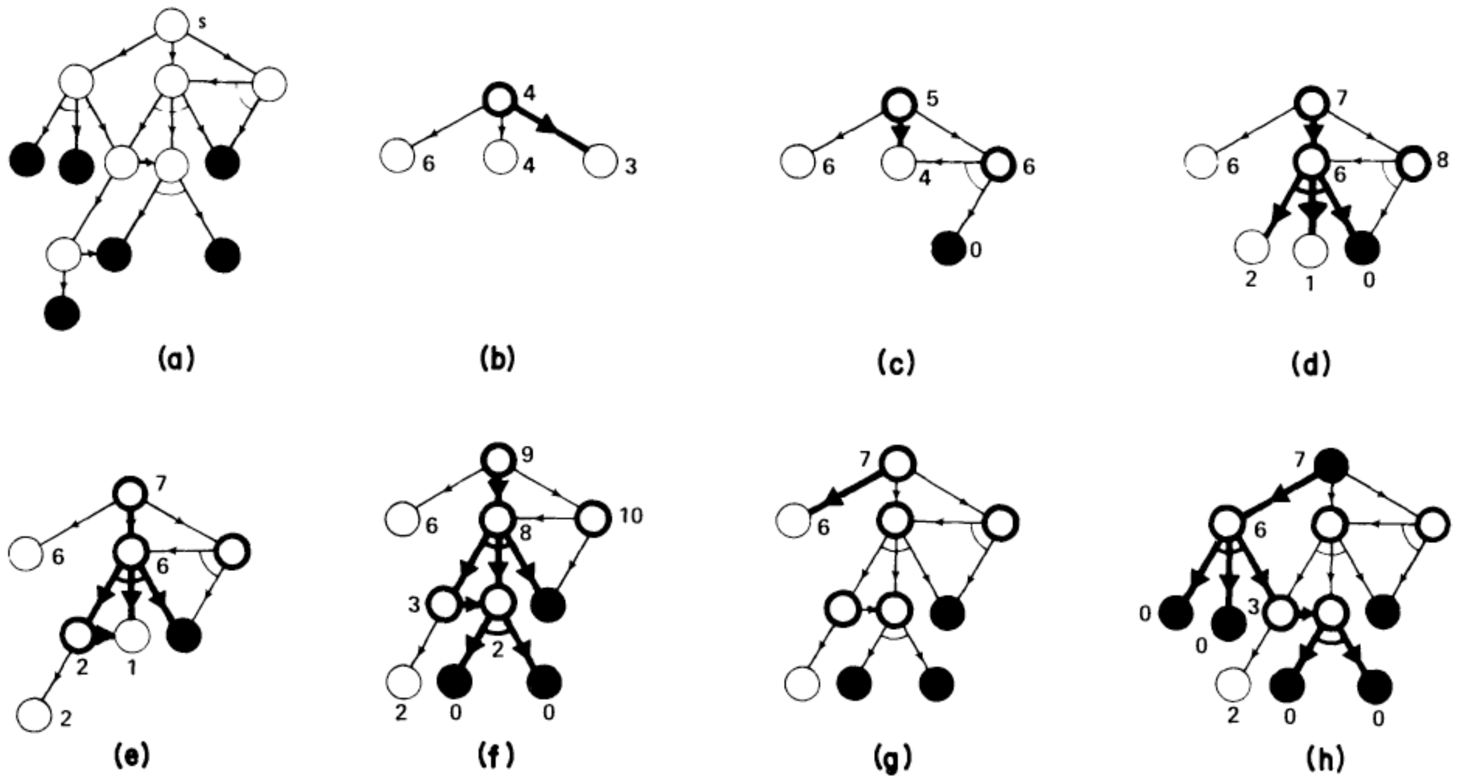


Figure 2.9

Successive steps in the execution of general-best-first (GBF) search on the implicit AND/OR graph of part (*a*). Solid circles represent solved nodes, heavy hollow circles nodes in CLOSED, and thin circles nodes in OPEN. The heavy lines stand, at each stage, for the current most promising solution base.

Sources documentaires

- Ouvrages / articles

  - R. Callan (2003) : *Artificial Intelligence*. Palgrave MacMillan.
  - T. Cazenave (2011) : *Intelligence Artificielle. Une approche ludique*. Ellipses.
  - I. Millington (2006) : *Artificial Intelligence for Games*. Morgan Kaufmann.
  - Nilsson N. (1998) : *Artificial Intelligence : A new synthesis*. Morgan Kaufmann.
  - J. Pearl (1984) : *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison Wesley.
  - Russel S. & Norvig P. (2009) : *Artificial Intelligence : A modern approach*. Prentice Hall, 2009 (3rd Ed.).

- Sites web
  - http://www.gameai.com/clagames.html