Résolution de Problèmes
&
Intelligence Artificielle

Bernard ESPINASSE
Professeur à l'Université d'Aix-Marseille

2004

- **Introduction, complexité des algorithmes, classes de problèmes, espaces de problèmes, ...**
- **Résolution dans des espaces d'états**
- **Résolution par décomposition de problèmes**

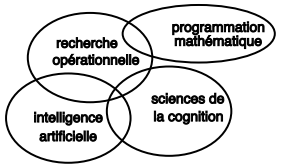
I - Introduction à la Résolution de Problèmes

- Activité humaine en général subordonnée à un objectif
- La réalisation de cet objectif pose problème

Activités courantes: on dispose d'une masse importante de connaissances de sens commun largement partagées

Activités + spécialisées: les connaissances pertinentes disponibles auprès d'un nb restreint de personnes (experts)

résolution de Pb = intersection de divers domaines



- **J.L. Laurière,** "Résolution de problèmes par l'homme et la machine", Eyrolles, 1986
- **A.Newell, H.A. Simon,** "Human Problem Solving", Prentice Hall, 1972
- Notes de cours de Paul Bourgine et Michel Joubert

Résolution de problèmes

Sciences cognitives:

- *psychologie: science du comportement humain*
  - découvrir relations entre stimulus et comportement
  - postule processus mentaux inobservables (boite noire)
- *psychologie cognitive:*
  - décrire les processus de raisonnement (ouvrir boite noire)
  - nbreuses expériences sur sujets résolvant des problèmes

Intelligence Artificielle:

- *1960: concept de recherche heuristique dans domaine des jeux (échecs)*
- *1970: problèmes plus difficiles: quantité importante de connaissances , résolution non par exploration combinatoire, mais par connaissances experts*

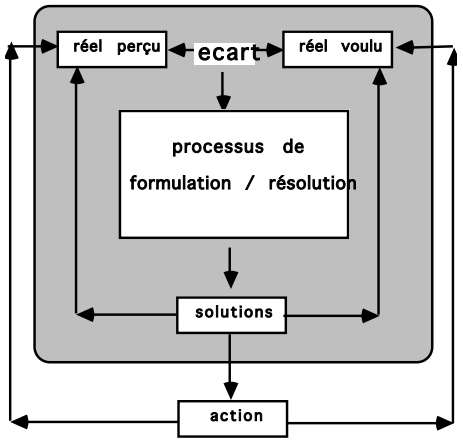
R.O. et prog. mathématique:

- *1960: démarche analogue à l'IA: découverte des heuristiques*

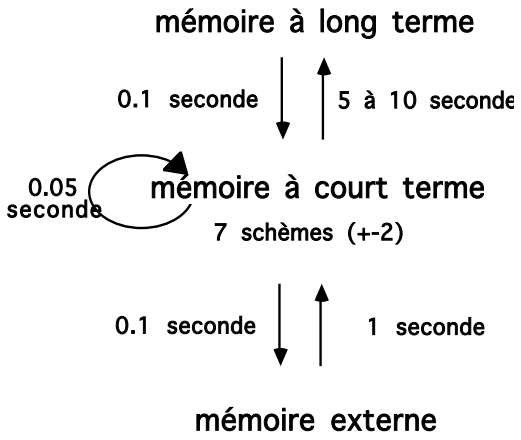
Nouvelle attitude:

Résolution de Pb = connaissances humaines + puissances de calcul de la machine

Comment surgit un problème



Limites de nos capacités cognitives



La complexité des problèmes

en maths :

- pb résolu <=> existence d'une preuve construite
- ne s'intéresse pas à qté de calcul nécessaire à l'algorithme proposé

en informatique :

- ordinateur = machine physique limitée
- limite taille mémoire
- limite vitesse d'exécution instructions

d'où:

- ***étude de la complexité des algorithmes,***
- ***classement des problèmes en fonction de leur complexité***

en général, si taille données alors pb. augmente :

-> temps et mémoire augmente rapidement = **explosion combinatoire**

-> d'où fournir à la machine des méthodes de **recherche heuristiques**

Complexité des algorithmes et des problèmes

Complexité d'un algorithme :

soit un algorithme A et sa structure de données associée de taille N

- **complexité de A = fonction de N = O(N)**

  O(N) = nb d'opérations nécessaires à son exécution dans le pire des cas (config de données moins favorable)

- **complexité (en temps) d'un problème:**

  = complexité du meilleur algorithme connu pour le résoudre

Classe de problèmes:

définie en fonction de la nature de O(N):

- problèmes à faible complexité, **classe P**
- problèmes à complexité intermédiaire, **classe NP**
- problèmes à très forte complexité, **classe E**

Classe P des problèmes polynomiaux

- **O(N) = polynôme en N**
- **Ensemble restreint de problèmes:**
  - trier un ensemble de n nombre: **O(n.log n)**
  - trouver un circuit eulérien dans un graphe de n arêtes: **O(n)**
  - rechercher un mot dans un texte de longueur n: **n.O(n)**
  - construire l'arbre recouvrant de coût mini (n arêtes) : **O(n.log n)**
  - plus court chemin dans un graphe n sommets, m arcs: **O(mxn)**
  - composantes connexes dans un graphe de n sommets: **$O(n^3)$**
  - fermeture transitive: dans un graphe de n sommets **$O(n^3)$**
  - couplage maximum dans un graphe de n sommets: **$O(n^{5/2})$**
  - flot maxi dans un graphe: **$O(n^3)$**
  - test de planéité d'un graphe: **O(n)**
  - programmation linéaire: **$O(n^3)$**

Classe E des problèmes exponentiels par nature

- **O(N) = exponentielle en N: O(N) = f (N)^N**
- **Exemple de problèmes:**
  - construire tous les sous-ensemble d'un ensemble: **(2^n)**
  - catastrophique ! : **($f(n)^{g(n)^n}$)**
  - peu d'algorithmes efficaces connus

Classe P et E:

- *machine de Turing déterministe (instructions lire, écrire, répéter, si...alors)*

Classe NP

- *machine de Turing non-déterministe (instruction choix en plus; travail dans toutes les directions offertes par ce choix)*

Classe NP de pbs Non déterministes Polynomiaux:

= pbs de complexité polynomiale sur machine de Turing non-déterministe

si bon choix à chaque étape => temps sur ordinateur séquentiel est polynomial

- **exemple de problèmes:**
  - faisabilité d'une expression logique,
  - coloration d'une carte géographique en 3 couleurs,
  - voyageur de commerce,
  - circuit hamiltonien,
  - partition d'un ensemble
  - ...

-> usage d'heuristique pour faire de bons choix

-> ce sont des problèmes qui relèvent de l'IA

Un exemple de NP pb

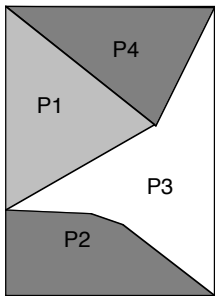
Coloration d'une carte géographique en 3 couleurs (c1, c2, c3), en faisant en sorte que 2 pays limitrophes (Pi et Pj) soient coloriés différemment :

algo. non-déterministe utilisé:

```
couleur (P1) <- c1 {colorier un pays ainsi que les autres adjacents}
REPETER pour i ∈ [1:n]
    SI P1 adjacent à Pi ALORS couleur (Pi) <-- c2
FIN i
    REPETER pour k ∈ [1:n]      TANT QU'il existe un
                                pays non colorié
        SI Pk n'est coloriable qu'avec la couleur cj
                    ALORS couleur (Pk) <-- cj
    FIN i
SI tous les pays sont coloriés ALORS FIN

SINON soit Pmin le pays de plus petit disposant
        de deux couleurs j1 et j2:

couleurs (Pmin) , <- CHOIX [j1, j2]
```



- usage d'une machine de turing non déterministe (avec l'instruction choix)
- libère le programmeur de la gestion des retours arrières (backtrack)

Espace de problème

= l'ensemble des états possibles des connaissances utiles à la résolution du problème

peut comporter :

- objets,
- relations,
- opérateurs de transformation d'états,
- diverses infos pour gérer le processus de recherche,...

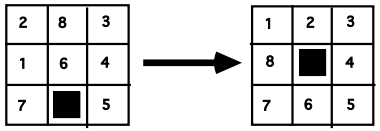
2 grands types d'espace de pb:

- **espace à transformation d'états:**
  - l'espace de pb est la donné d'un état initial, d'un état final et d'opérateurs permettant de passer d'un état à un autre
- **espace à réduction de problème:**
  - un problème peut parfois être décomposé en sous-problèmes isolés tels qu'une solution apportée à chacun d'eux donne une solution au problème initial

Espace à transformation d'états

exemple: le taquin à 9 cases

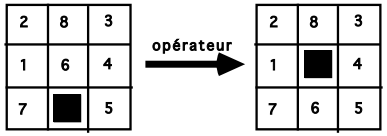
1) notion d'état



situation initiale — situation finale

2) notion d'opérateur:

déplacement de la case vide vers le haut, le base la droite ou la gauche

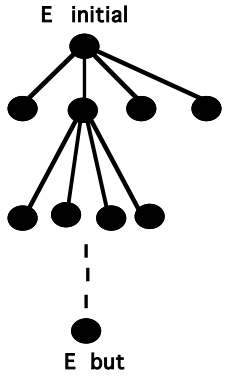


état 1 — état 2

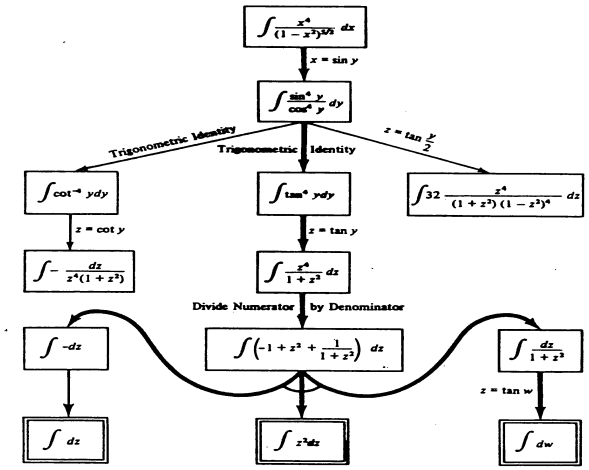
Espace à transformation d'états

exemple: le taquin à 9 cases

3) arbres "OU" (graphes):

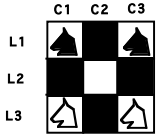


Espace à réduction de problème : exemple: le cas de l'intégration (arbres "ET/OU") :

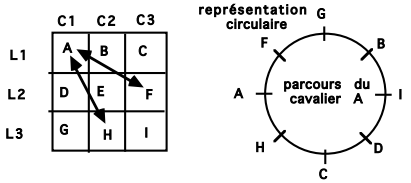


Savoir changer de représentations

Exemple: le cas des 4 cavaliers



première formulation : échanger la place de cavaliers noirs avec celle des blancs, en un minimum de coups



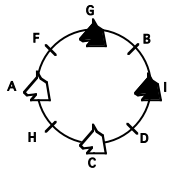
remarques :

- la case E n'apparaît pas dans cette représentation (pas partie du pb...)
- cette représentation peut être utilisée pour tous les autres cavaliers

Savoir changer de représentations

Cas des 4 cavaliers

nouvelle formulation du pb: échanger cavaliers noirs placés en A et C avec les cavaliers blancs placés en G et I



Solution triviale :

• il suffit de faire tourner l'ensemble d'un demi-tour pour amener C sur G, A sur I, G sur C et I sur A

Méthodes de résolution de problèmes

II - Méthodes de résolution dans des espaces d'états

- méthodes aveugles :
  - méthodes par saturation ou en largeur d'abord
  - méthodes en profondeur ou en profondeur d'abord
- méthodes heuristiques :
  - meilleur d'abord (best-first)
  - A*

III - Méthodes de résolution par décomposition de problèmes

- représentation par arbre ET/OU
- stratégies de résolution de problèmes décomposables
  - min-max
  - alpha-beta
  - ...

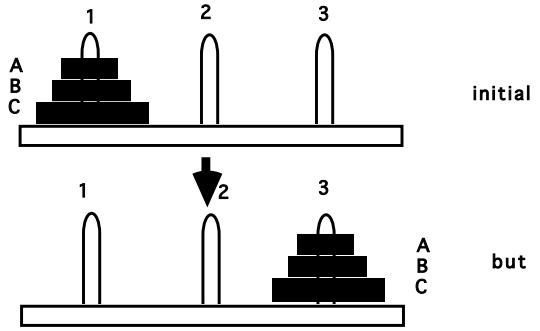
II - Résolution dans un espace d'états

PLAN

- **Représentation d'un problème dans un espace d'état:**
  - *état, opérateur, espace d'état,...*
- **Méthodes de résolution:**
  - ***méthodes aveugles:***
    - largeur d'abord (breadth-first)
    - profondeur d'abord (depth-first)
  - ***méthodes heuristiques:***
    - meilleur d'abord (best-first)
    - A*

Résolution dans un espace d'états

Cas des tours d'Hanoï



Enoncé :

• reconstituer en position 3, la tour initialement construite en position 1

• on ne peut utiliser les positions 1,2,3 qu'en déplaçant un disque à la fois, en le prenant au sommet d'une tour pour le placer sur une position vide ou au sommet d'une tour, sur un disque de taille supérieure

Cas des tours d'Hanoï

Choix d'une représentation

a) notion d'états:

- énumérer les positions respectives des disques A,B,C, sur les tours 1,2,3 :
  ex : soit l'état: A en 2, B en 3, A en 1 ; représenté par (2 3 1)
- dans cette représentation : état initial: (1 1 1) et état final: (3 3 3)

b) notion d'opérateurs:

- transforme un état en un autre état,

- représentation adoptée: opérateur 012 déplace le sommet de la tour 1 en position 2

- il existe 6 opérateurs au total:

012 déplace le sommet de la tour 1 en position 2
013 déplace le sommet de la tour 1 en position 3
021 déplace le sommet de la tour 2 en position 1
023 déplace le sommet de la tour 2 en position 3
031 déplace le sommet de la tour 3 en position 1
032 déplace le sommet de la tour 3 en position 2

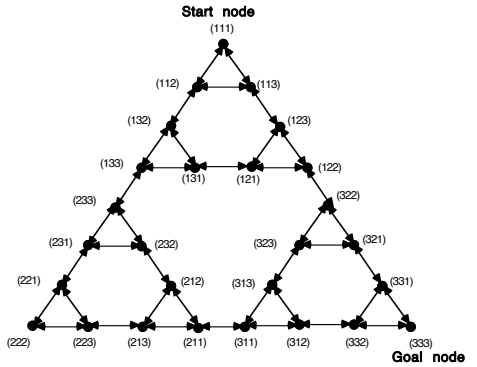
- exemple: si le jeu est dans l'état (2 3 2), seuls les opérateurs 021, 023, 031 sont applicables:
021 (232) = (231)
023 (232) = (233)
031 (232) = (211)

Cas des tours d'Hanoï

Choix d'une représentation

c) l'espace d'états :

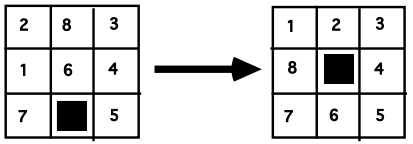
graphe en OU: (ici le graphe ne représente pas les opérateurs)



Cas du taquin

Choix d'une représentation

a) représentation des états: matrice 3x3



b) représentation des opérateurs:

- permuter la case vide avec:

- case du dessus

- case du dessous

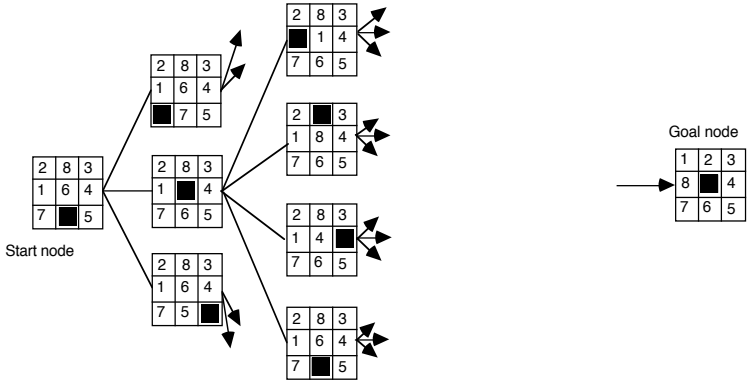
- case du droite

- case du gauche

Cas du taquin

Choix d'une représentation

c) arbre des états: (obtenu selon la méthode de résolution adoptée)



1 - Méthodes aveugles de résolution dans un espace d'états :

Méthode en largeur d'abord (par saturation)

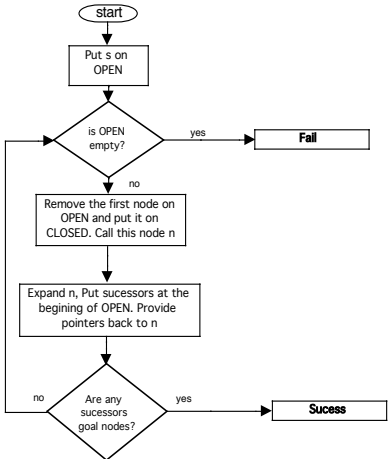
- **principe:**
  - le système étant dans un état donné, on engendre tous les états qui peuvent être obtenus à partir de cet état avec les opérateurs dont on dispose
  - il y a succès quand l'état engendré est l'état final recherché
  - sinon on mémorise l'ensemble des états engendrés et on réitère pour chacun d'eux, tour à tour, le même procédé

- **algorithme:**

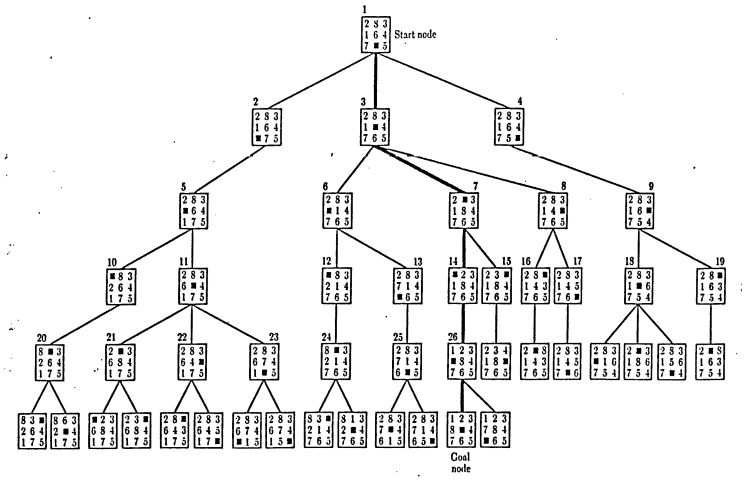
  on a besoin de 2 listes:

  - **OPEN** : représentant l'ensemble des états engendrés non encore traités
  - **CLOSED** : sert à mémoriser l'ensemble des noeuds déjà traités

Méthode en largeur d'abord (par saturation)



Méthode en largeur d'abord (par saturation): cas du taquin

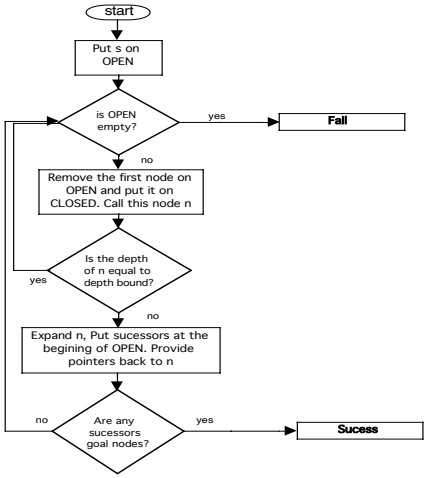


Méthode en profondeur d'abord (en profondeur)

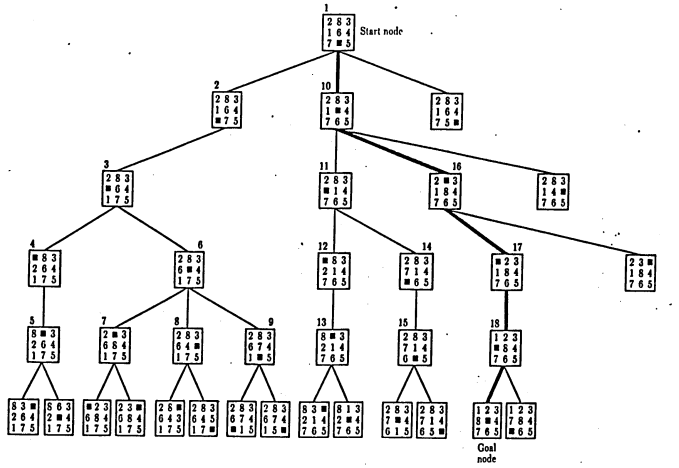
- **principe:**
  - le système étant dans un état donné, **engendrer** tous les états pouvant en être issus
  - il y a **succès** si l'un des descendant est l'état final recherché
  - sinon, **choisir** un de ces états, **mémoriser** les autres et **réitérer le procédé** avec l'état choisi
  - en cas **d'échec**, **recommencer** avec un des états mémorisés

- **algorithme:**
  - **on a besoin de 2 listes (cf précédemment) :**
    - **OPEN** : représentant l'ensemble des états engendrés non encore traités
    - **CLOSED** : sert à mémoriser l'ensemble des noeuds déjà traités
  - **nécessaire de fixer a priori un niveau de profondeur maxi** à ne pas dépasser en cas d'échec

Méthode en profondeur d'abord : algorithme



Méthode en profondeur d'abord : le taquin (profondeur = 5)



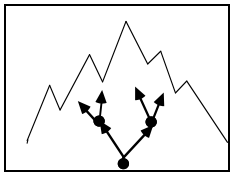
2 - Méthodes heuristiques de résolution dans un espace d'états

Méthode du meilleur d'abord (Best-First)

- **principe:**

On dispose d'une fonction d'évaluation heuristique f(n) de la situation au noeud n

- développer prioritairement le **noeud** n qui possède la **meilleure évaluation f(n) (Best-First)**



métaphore de l'équipe d'alpiniste

- **inconvénient :**
  - peut devenir très gourmand en place mémoire (explosion du nb de noeuds en mémoire)

  -> adopter en plus une **stratégie de retour arrière, consistant à revenir sur les choix antérieurs**

Méthode A*

- **principe:**

- **cas particulier** du Best-first
- n = noeud représentant un état, la fonction d'évaluation de n sera de la forme:

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

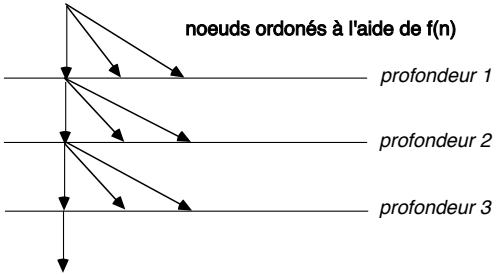
- on cherche un chemin de coût mini entre le **noeud source** et l'un des **noeuds but**:
  - **g(n)** est le coût du chemin déjà parcouru **(coût consenti)**
  - on suppose connue une fonction heuristique **h(n)**, estimant par défaut le coût optimal inconnu de n jusqu'à un noeud but **(coût restant)**

Théorème :

si le but est à distance finie, A* fournit le chemin de coût minimal

Méthode A*, cas du taquin

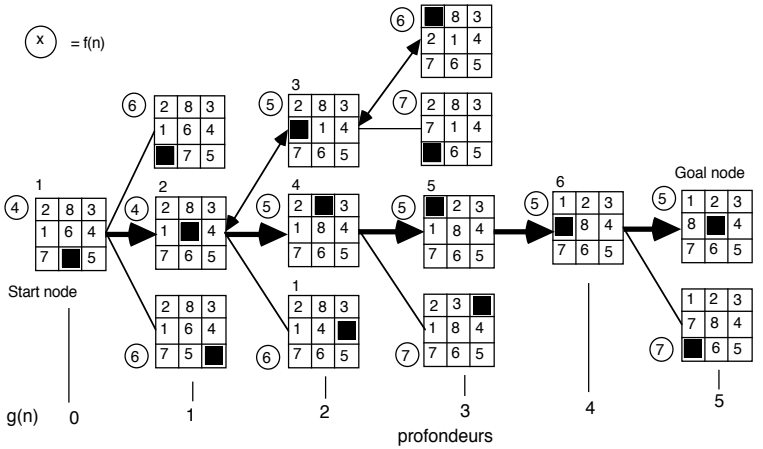
• n = noeud représentant un état,
• fonction d'évaluation de n : **f(n) = g(n) + h(n)**
avec :
  • **g(n)** = profondeur du noeud n depuis l'état initial
  • **h(n)** = nb de cases mal placées dans le noeud n par rapport au noeud final



• **recherche informée en profondeur d'abord avec retour arrière**
• **combiner éventuellement avec retour arrière informé (avec algo. BF)**

Méthode A*, cas du taquin

Exemple: pour l'état initial : f(initial) = 0+4=4



• **choix du noeud de plus petit score** : celui qui a le plus de chance de mener à la solution

Mesure de performance d'une méthode

= **estimations** mesurant l'**efficacité de la méthode** en comparant :
  • le chemin de résolution préconisé par la méthode avec
  • le chemin menant de l'état initial à l'état final

• **pénétration P** = nb noeuds L sur la branche menant de l'état initial à l'état final / Nombre total de noeuds engendrés

soit : **P = L / T**

plus P est proche de 1, plus efficace est la méthode

• **facteur de branchements effectifs B**
  • si T = nb total de noeuds engendrés avant obtention d'une solution, supposons qu'il existe un arbre, de profondeur L, dont chaque noeud a un nb constant de descendants :

$$B + B^2 + \dots B^L = T$$

  • B permet d'estimer le nb d'éventualités rencontrées chaque fois que les descendants d'un noeud ont été engendrés.

Plus B est proche de 1, plus efficace est la méthode

Mesures	Méthode par saturation	Méthode en profondeur avec heuristique
P	0,108	0,385
B	1,86	1,34

III - Résolution par décomposition de problèmes

Plan

• **Représentation de la décomposition d'un pb par arbre ET/OU:**
  • Représentation d'un pb
  • Notion d'opérateur de décomposition
  • Représentation graphique de la décomposition d'un pb
  • Méthode GBF, Méthode AO*, Méthode Min-Max, Méthode Alpha-Béta, …

Résolution par décomposition de problèmes

Cas des tours d'Hanoï

choix d'une représentation en arbre ET/OU

- **sous-problèmes:**
  - pour construire la tour en 3, il est nécessaire de placer le disque C en position 3, vide au préalable
  - pour pouvoir placer C en 3, A et B doivent être placés en position 2,
  - alors seulement, on pourra déplacer C en 3 et continuer à résoudre le pb
- **représentation du pb:**
  - état initial: (1 1 1) -> (3 3 3) état final
  - même représentation pour chaque sous pb
- **notion d'opérateur de décomposition:**
  - transforme une description de pb en un ensemble de descriptions de pb réduits

Résolution par décomposition de problèmes

Cas des tours d'Hanoï

l'ensemble des sous-pbs est :

s/pb 1:

(1 1 1) -> (1 2 2)

nécessite des décompositions pour être résolu

s/pb 2:

(1 2 2) -> (3 2 2)

résolu par l'opérateur 013,

d'où s/pb2 = s/pb *primitif*

s/pb 3:

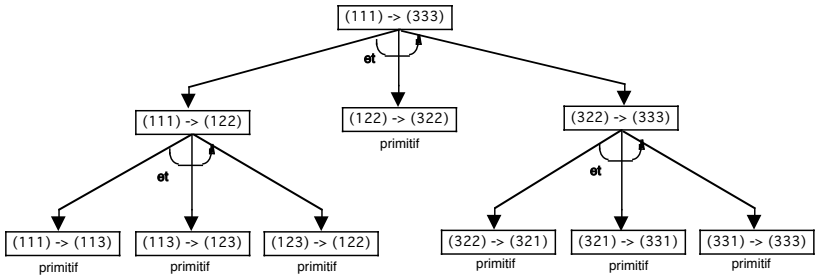
(3 2 2) -> (3 3 3)

nécessite des décompositions pour être résolu

Cas des tours d'Hanoï

représentation graphique de la décomposition:

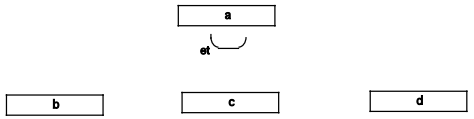
arbre en ET :



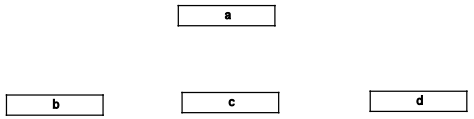
Arbres ET/OU

- on représente la décomposition d'un pb en s/pb obtenus par application d'opérateurs de réduction par des arbres ET/OU
- dans un arbre ET/OU, si un noeud possède des descendants, ils sont tous:
  - **soit ET d'où : noeud ET**
  - **soit OU d'où : noeud OU**

noeud ET : pour que le pb représenté par un noeud ET soit résolu, il faut que tous les pbs représentés par ses descendants le soient:



noeud OU : pour que le pb représenté par un noeud OU soit résolu, il suffit que l'un des pbs représentés par ses descendants le soient:



Exemple du système de production (système expert)

Ensemble de règles de production :

r1: **A -> E**
r2: **B -> D**
r3: **H -> F**
r4: **E, G -> C**
r5: **E, K -> B**
r6: **D, E, K -> C**
r7: **G, K, F -> A**

problème: les faits G, H, K étant donnés, peut-on produire C ?

a) représentation du pb:
- base de faits: BdF initiale: G, H, K
- **pb**: (G, H, K) dans BdF => C dans BdF ?

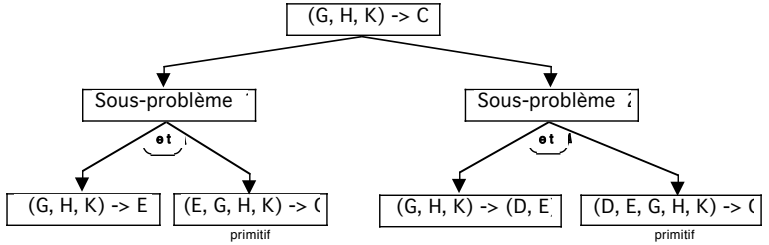
b) opérateurs de transitions:
= règles de production: si faits de gauche dans BdF -> faits de droite dans BdF

c) opérateurs de décomposition:
= conditions d'application des opér. de transition (corresponde à une utilisation des règles en chaînage arrière)

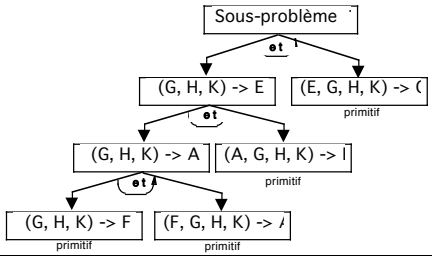
Exemple du système de production (système expert)

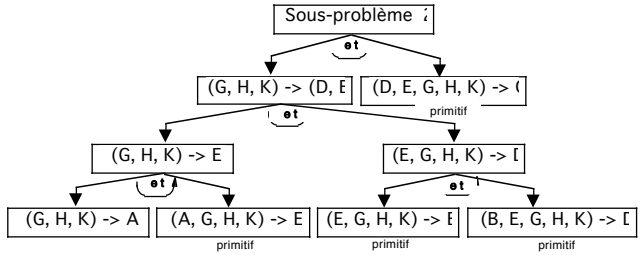
d) s/pbs:
- le pb initial : (G, H, K) dans BdF -> C dans BdF ?
- peut être résolu en résolvant le s/pb : (G, H, K) dans BdF -> (D,E) dans BdF ?
- puisque le s/pb suivant est primitif : (D,E, K) dans BdF -> C dans BdF ?

e) représentation graphique:



Exemple du système de production (suite)





Exemple du système de production

f) graphe des états: états de la base de faits du système expert