EFRAI 2010/2011
L3

Aide à la Décision

Recherche dans un espace d'états

Partie 2

Intelligence Artificielle
Artificial Intelligence

Recherche
dans un espace d'états

(partie 2)

Propriétés des algorithmes

Analyse comparative

Critères d'évaluation des algorithmes

- Complétude
  - La méthode garantit de trouver une solution si elle existe.
- Optimalité
  - Si plusieurs solutions existent, la méthode garantit de trouver la « meilleure » d'entre-elles.
  - Mais … :
    qu'est-ce que « la meilleure » ?

- « Meilleure » ?
  - Meilleure = suite d'actions de coût minimum pour atteindre un état solution
    - Nécessite un notion de coût cumulé calculé C*
    - g *coût uniforme*
    - g + h *A**
  - Meilleure = la solution qui se rapproche le plus d'un objectif
    ?...

Critères d'évaluation des algorithmes

- Complexité en temps
  - Combien de nœuds (d'états) faut-il produire / analyser pour trouver la solution ?

    ordre de grandeur)
- Complexité en espace
  - Combien de nœuds faut-il conserver en mémoire pour trouver une solution ?

    (ordre de grandeur)

- Paramètres pour complexité en temps et en espace:
  - b = facteur de branchement
    nombre (maximum) de successeurs d'état
  - d = profondeur dans l'arbre de recherche
    du nœud représentant l'état solution retenu
  - m = profondeur maximale de l'arbre de recherche

Principe d'analyse

- Complétude
  Optimalité

  Vérifier si / quand / où l'algorithme s'arrête
- Complexité en temps
  Complexité en espace

  Calculer le volume :

  - nombre d'états traités (i.e. nombre d'itérations)
  - nombre d'états sauvegardée en mémoire

  jusqu'à arrêt de la recherche

  dans le cas le plus défavorable

Exemple de calcul :

Recherche en profondeur d'abord

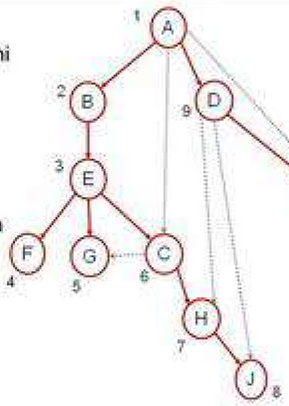
- Complétude :

  NON si espace d'états infini

  OUI si espace d'états fini
- Optimalité :

  NON car s'arrête sur le 1er état solution trouvé selon sa place dans l'arbre

  Éventuellement « très profond » dans l'arbre



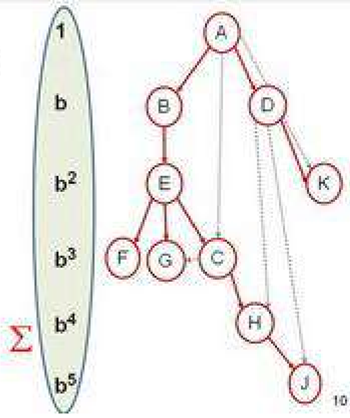
Recherche en profondeur d'abord

- Complexité en temps

  Dans le pire des cas, tous les états sont traités

  $= 1 + b + b^2 + \ldots + b^m$

  $= \mathcal{O}(b^m)$



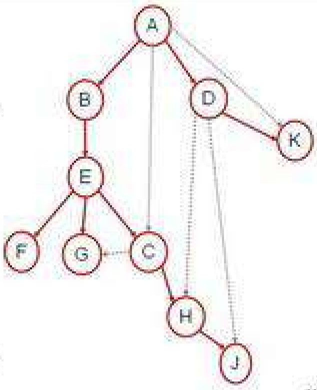
Recherche en profondeur d'abord

- Complexité en espace
  - *Théoriquement* : on peut oublier les états des sous-arbres entièrement analysés

    = $\mathcal{O}$ (b*m)

    ou $\mathcal{O}$ (m)

    - Mais risque de revenir sur un sous-arbre « oublié » et de le développer à nouveau



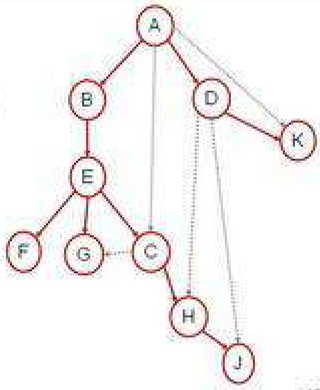
(C) Hervé Barbot, 2005-2011 11

Recherche en profondeur d'abord

- Complexité en espace
  - Si on gère les ensembles « ouverts » et « fermés » :

    = $\mathcal{O}$ (b^m)

    - Plus coûteux en mémoire, mais moins risqué en temps d'exécution si les branches d'analyse se rejoingnent…



(C) Hervé Barbot, 2005-2011 12

Les autres cas font l'objet de travail en TD.

Recherche dans un espace d'états

Search in a states space

Rappels, conclusion,

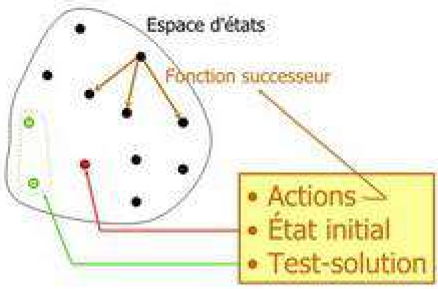
et ouverture

(C) Hervé Barbot, 2005-2011 35

Algorithme de recherche ?

- Une approche des plus puissantes de l'IA
- Mécanisme général basé sur la modélisation de l'environnement de recherche :
  - L'environnement, le domaine d'exploration est représenté par un ensemble ou « espace » d' « états »
  - Les transformations apportées à l'environnement sont simulées par des changement d'états
    - Différentes transformations possibles à partir d'un état sont des alternatives représentées dans l'espace d'états
  - L'objectif est de trouver une suite d'actions (de transformations) permettant d'amener un environnement de son état initial à un état « acceptable »

(C) Hervé Barbot, 2005-2011 36



(C) Hervé Barbot, 2005-2011 37

- Enoncé principal :
  - Etant donné une représentation de l'environnement sous forme d' « espace d'états »,
  - Etant donnée une situation initiale de départ de la recherche,
  - Quelle est la [meilleure] suite d'actions pouvant mener à une situaton acceptable ?
  - La réponse à ce problème se fait par une analyse d'alternatives dans un ordre dépendant d'une « stratégie de recherche » particulière.

(C) Hervé Barbot, 2005-2011 38

- Contraintes essentielles :
  - L'ensemble des états possibles n'est a priori pas connu.
  - Il peut être très grand.
  - Dans certains cas, il peut être infini.

  Donc :

  Les techniques de « théorie des graphes » classiques avec représentation explicite et exhaustive sont rejetées.

- Espace « très volumineux », voire « infini »
  - Nécessité de trouver un juste milieux entre
    - **ne pas chercher indéfiniment,**
    - trouver une solution aussi « acceptable » que possible.

  Problème de complétude !

- Espace « très volumineux », voire « infini »
  - Nécessité de trouver un juste milieux entre
    - ne pas chercher indéfiniment,
    - **trouver une solution aussi « acceptable » que possible.**

      Donc s'arrêter dès qu'une solution est trouvée ?...

  Problème d'optimalité !

- Hypothèses de base :
  - Environnement statique
  - Environnement discrétisable
  - Environnement observable
  - Actions déterministes

- « 1 état initial » ou
  « n états initiaux » ?
- « 1 état solution » ou
  « n états acceptables » ?
- « solution = 1 état isolé » ou
  « solution = un chemin » ?
- « trouver 1 solution » ou
  « trouver toutes les solutions » ?

- Paramètres à avoir en tête avant utilisation :
  - Nombre d'états dans l' « espace d'états »
    - Complexité en espace nécessaire à la recherche
  - Distribution des états solutions dans l'espace d'états
    - Une stratégie de recherche peut être meilleure qu'une autre
  - Espace mémoire nécessaire pour représenter un état
    - « x complexité en espace », cela peut faire beaucoup
  - Temps de recherche des alternatives (calcul des successeurs)
    - « x complexité en temps », cela peut aussi faire lourd...

Prncipe général : exploration d'un espace d'états jusqu'à trouver une solution

```
function tree_search ( problem, strategy)
returns solution or failure

set search_tree = initial_state ( problem )

Loop do :

  ¬ ∃ candidate_for_expansion ( search_tree )
     ⇒ return failure

  set leaf_node = choose_node ( search_tree, strategy )

  is_solution(leaf_node,problem)
     ⇒ return path from initial state to leaf_node

  ¬ is_solution(leaf_node,problem)
     ⇒ set search_tree = expand ( search_tree, leaf_node )
```

(C) Hervé Barbot, 2005-2011 46

- Stratégie 1 :
  recherche aléatoire
- Stratégies 2 et 3 :
  largeur d'abord
  profondeur d'abord
- Stratégies 4 et 5 :
  approfondissement itératif
  profondeur limitée

Ce sont des stratégies

- *« informatique »*
  - *Qui ne dépendent que d'un choix de mise en œuvre de l'espace d'états*
- *« aveugles »*
  - *Qui connaissent les « états » analysés mais ne peuvent rien en déduire pour accélérer la recherche*

(C) Hervé Barbot, 2005-2011 47

- Stratégie 6
  coût uniforme

Remarque :

- *Stratégie 2 :*
  largeur d'abord
  retourne la « meilleure » solution si calculée en terme de nombre d'actions

- Première approche vers la notion de « meilleure solution »
  - Si on considère que « meilleure » veut dire « à moindre coût cumulé » des différentes actions
- Limite : coûts négatifs
  - Non optimalité de la recherche

(C) Hervé Barbot, 2005-2011 49

- Stratégie 8 : recherche bi-directionnelle
  - En largeur d'abord depuis l'état initial
    +
    en largeur d'abord depuis l'état solution
  - Arrêt lorsque les deux espaces d'états se rejoignent
  - Connaissance de la fonction « action inverse » !
  - Problème technique : identification d'un état commun aux 2 espaces de recherche
    - Conservation d'au moins tous les états d'au moins un espace

(C) Hervé Barbot, 2005-2011 50

Etats multiples

- Problème crucial résolu par la gestion des ensembles 'ouverts' et 'fermés'
  - Avec l'impact sur la complexité en espace…

(C) Hervé Barbot, 2005-2011 52

Méthodes heuristiques

- Objectif : optimiser la recherche

  « rendre le processus plus efficace »

- Fonction heuristique :

  $h:\ \text{état} \rightarrow \text{valeur}$

  Estimation de l' « intérêt » qu'il y a à explorer un état plutôt qu'un autre

- Principe fondamental : poursuivre l'exploration à partir de l'état ayant la meilleure estimation heuristique

(C) Hervé Barbot, 2005-2011 53

- Propriété essentielle :
h (état solution) = 0
- Attention :
l'optimisation de la recherche dépend entièrement de la qualité de la fonction heuristique !
   - Fonction heuristique « presque parfaite » = respecte l'ordre des états que donnerait une fonction calculée, malheureusement inconnue

(C) Hervé Barbot, 2005-2011 54

« Best first »

- Combine largeur / profondeur
- 1ère vision « basique »

```
function best_first ( problem )
returns solution or failure

set search_space = initial_state ( problem )
Loop do :
  ¬ ∃ candidate_for_expansion ( search_space )
     ⇒ return failure
  set node = choose_lower_h ( search_space )
  is_solution (node, problem )
     ⇒ return path from initial state to node
  ¬ is_solution ( node, problem )
     ⇒ set search_space = successors ( node )
```

NON !

(C) Hervé Barbot, 2005-2011 56

- Best-first :
   - Tous les états non analysés restent candidats à analyse et expansion
      - Ensemble 'ouverts'
   - A chaque itération, on choisi celui qui est le plus « prometteur »
      - i.e. celui qui a la plus faible valeur heuristique

(C) Hervé Barbot, 2005-2011 57

- Greedy-search
   - La plus simple des stratégies « best-first »
   - Choix = valeur 'h' minimale parmi les états candidats à expansion
   - Trouve rapidement une solution si la fonction heuristique est performante
      - i.e. donne une bonne vision de la « distance restante »
   - Oublie les « coûts » effectifs des actions, donc ne trouve pas nécessairement la « meilleure » solution
      - Si « la meilleure » = le plus faible coût cumulé

(C) Hervé Barbot, 2005-2011 59

- A*

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

   - Combine coût cumulé (calculé) et coût restant (estimé)
   - Choix = valeur 'f' minimale parmi les états candidats à expansion
i.e. le chemin le plus prometteur de bout en bout (de l'état initial à un état solution)
   - « Meilleur » = « plus faible coût global » si la fonction heuristique est convenablement choisie...

(C) Hervé Barbot, 2005-2011 60

Variantes de A*

- Problèmes souvent très complexes
- Espace de recherche très grand
- Recherche heuristique pure devient peu efficace
- Encombrement mémoire

(C) Hervé Barbot, 2005-2011 61

- IDA*

  Iterative-deepening A*
  - Recherche à mémoire limitée

  - Reprend l'idée de profondeur limitée
  - Itération =
    développement en profondeur sur la limite f(e)
  - Limite basée sur les valeurs f(e)
    - Itération 'i' : on développe jusqu'à la plus petite valeur f(e) de tous les états qui avaient une valeur plus grande que la valeur limite au tour précédent

(C) Hervé Barbot, 2005-2011 62

- RBFS

  Recursive best-first search
  - Recherche à mémoire limitée
- MA*

  Memory-bounded A*
  - Recherche à mémoire limitée

(C) Hervé Barbot, 2005-2011 63

- SMA*

  Simplified MA*
  - Gestion mémoire spécifique
    - Élimination si nécessaire (lorsque l'encombrement mémoire est jugé trop important) des états les moins « intéressants »
      i.e. à valeur f(e) élevée
    - Retenir au nœud « ancêtre » restant dans l'espace d'états la valeur f la plus petite parmi celles de ses descendants supprimés

(C) Hervé Barbot, 2005-2011 64

Algorithmes d'amélioration itérative

- Si le chemin importe peu, et que seule la solution compte
- Le but est soit :
  - de trouver une solution optimale,
  - de trouver une configuration satisfaisant des contraintes
- On part d'une « configuration » complète et on essaie d'améliorer la « qualité » de la solution
- Méthodes :
  - Escalade (hill-climbing)
  - Recuit simulé (Simulated annealing)
  - Recherche locale en faisceau (local beam)
  - Algorithmes génétiques

(C) Hervé Barbot, 2005-2011 65

- Escalade (hill-climbing)
  - A chaque itération, on choisi le successeur ayant la « meilleure évaluation »
    si celle-ci est meilleure que l'évaluation de l'état courant
  - Arrêt si aucun des successeurs n'est choisi
  - Pas de sauvegarde de l'arbre de recherche
    Pas de regard en avant
  - Si pas de progrès : redémarrage à un autre point de départ
  - Avec suffisamment de redémarrage, la solution optimale sera éventuellement trouvée

(C) Hervé Barbot, 2005-2011 66

- Recuit simulé
  - Idée : permettre de mauvais déplacements dans le but d'échapper à des maximum locaux (un des problèmes du hill-climbing) :
    - Sélectionner un déplacement aléatoire s'il améliore la situation
      sinon lui affecter une probabilité < 1
    - Un paramètre (qui tend vers 0 avec le temps) permet de déterminer la probabilité d'un mauvais mouvement

(C) Hervé Barbot, 2005-2011 67

- Recherche local en faisceau
  - Commence avec k états aléatoires
  - A chaque itératon, on génère les successeurs des k états :
    - But atteint : stop
    - But non atteint : on recommence avec les k meilleurs états générés

- Algorithmes génétiques
  - « Evolution naturelle » :
    - Un fort a plus de chance de survivre
    - Deux individus forts donnent (généralement) des enfants forts
    - Si l'environnement évolue suffisamment lentement, les individus s'adaptent
    - Occasionnellement, des mutations (bénéfiques ou non…) interviennent
  - Gestion d'une « population d'hypothèses »
    - Population initiale
    - Reproduction et mutation engendrent la génération suivante
    - A chaque génération, application d'une fonction d'utilité pour garder les hypothèses qui ont le plus de chance de se reproduire (et de mener à une solution acceptable…)

Utilisation d'un algo de recherche dans un espace d'états si…

- Espace de recherche de petite taille :
  - Pas d'autre technique, ou
  - Pas « rentable » de développer quelque chose de plus efficace
- Espace d'états de grande taille :
  - Pas d'autre technique, et
  - Il existe de « bonnes » heuristiques