Intelligence Artificielle

Recherche dans un espace d'états

Maria Malek

Département Informatique

États, Actions & Résolution de problèmes

- Comment résoudre un problème ?

États, Actions & Résolution de problèmes

- Comment résoudre un problème ?
- Recherche des solutions ..

États, Actions & Résolution de problèmes

- Comment résoudre un problème ?
- Recherche des solutions ..
- *Recenser les états d'un système donné et trouver parmi ces états une ou plusieurs solutions*.

États, Actions & Résolution de problèmes

- Comment résoudre un problème ?
- Recherche des solutions ..
- *Recenser les états d'un système donné et trouver parmi ces états une ou plusieurs solutions*.
- Comment ?
  - Le passage d'un état à un autre se fait par l'application d'une action donnée.
  - Développement *d'un arbre de recherche* + *une stratégie de recherche*.

Stratégie de recherches & Modélisation

- Algorithme de recherche aveugle en profondeur ou en largeur (coûteux).

Stratégie de recherches & Modélisation

- Algorithme de recherche aveugle en profondeur ou en largeur (coûteux).
- **Objectif** : diminution du temps de recherche.

Stratégie de recherches & Modélisation

- Algorithme de recherche aveugle en profondeur ou en largeur (coûteux).
- **Objectif** : diminution du temps de recherche.
- *Intégration d'une heuristique : Algorithme A*.*

Stratégie de recherches & Modélisation

- Algorithme de recherche aveugle en profondeur ou en largeur (coûteux).
- **Objectif** : diminution du temps de recherche.
- *Intégration d'une heuristique : Algorithme A*.*
- Modélisation par un quadruplet (S, E_0, F, T) où :
  - S est l'ensemble de tous les états.
  - E_0 est l'état initial, $E_0 \in S$.
  - F est l'ensemble des états finaux $F \subset S$.
  - T est la fonction de transition : associe à chaque état E_i un ensemble de couples (A_{ij}, E_{ij}) tq A_j soit une action élémentaire permettant de passer de l'état E_i à l'états E_{ij}.

Résolution de problème

- Trouver une séquence $E_0, E_1, .., E_j, ..E_n$ tel que $\exists A_1, ..A_j, ..A_n$ tels que $(A_j, E_j) \in T(E_{j-1})$ et $E_n \in F$.

Résolution de problème

- Trouver une séquence $E_0, E_1, .., E_j, ..E_n$ tel que $\exists A_1, ..A_j, ..A_n$ tels que $(A_j, E_j) \in T(E_{j-1})$ et $E_n \in F$.
- Un arbre de recherche est construit ;
  - La racine de l'arbre : l'état initial.
  - Un état E_{ij} est fils d'un autre état E_i s'il existe une action qui permet d'obtenir E_{ij} à partir de E_i.
  - Si une des feuilles correspond à un état final, la solution est trouvée.

Résolution de problème

- Trouver une séquence $E_0, E_1, .., E_j, ..E_n$ tel que $\exists A_1, ..A_j, ..A_n$ tels que $(A_j, E_j) \in T(E_{j-1})$ et $E_n \in F$.
- Un arbre de recherche est construit ;
  - La racine de l'arbre : l'état initial.
  - Un état E_{ij} est fils d'un autre état E_i s'il existe une action qui permet d'obtenir E_{ij} à partir de E_i.
  - Si une des feuilles correspond à un état final, la solution est trouvée.
- Méthode :
  - Recherche aveugle en profondeur,
  - Recherche aveugle en largeur,
  - L'algorithme A*.

La recherche aveugle - En profondeur

- En profondeur : développement d'une branche entière avant de parcourir le reste de l'arbre :
  - La solution est trouvée : arrêt ou continuer à chercher les autres solutions (*backtracking*).
  - La solution n'est pas trouvée (état d'échec), poursuivre la recherche (*backtracking*).
  - Une branche infinie est à explorer : un test d'arrêt à une profondeur maximale sera appliqué.

La recherche aveugle - En profondeur

- En profondeur : développement d'une branche entière avant de parcourir le reste de l'arbre :
  - La solution est trouvée : arrêt ou continuer à chercher les autres solutions (*backtracking*).
  - La solution n'est pas trouvée (état d'échec), poursuivre la recherche (*backtracking*).
  - Une branche infinie est à explorer : un test d'arrêt à une profondeur maximale sera appliqué.
- Complexité liée à l'ordre d'exploration des branches.

La recherche aveugle - En largeur

- En largeur : Visiter les états en parcourant l'arbre niveau par niveau.

La recherche aveugle - En largeur

- En largeur : Visiter les états en parcourant l'arbre niveau par niveau.
- On utilise une file d'attente.

La recherche aveugle - En largeur

- En largeur : Visiter les états en parcourant l'arbre niveau par niveau.
- On utilise une file d'attente.
- *la recherche s'arrête quand on état final est trouvé ou bien une profondeur maximale est trouvée*.

La recherche aveugle - En largeur

- En largeur : Visiter les états en parcourant l'arbre niveau par niveau.
- On utilise une file d'attente.
- *la recherche s'arrête quand on état final est trouvé ou bien une profondeur maximale est trouvée*.
- Très cher en temps et espace,

La recherche aveugle - En largeur

- En largeur : Visiter les états en parcourant l'arbre niveau par niveau.
- On utilise une file d'attente.
- *la recherche s'arrête quand on état final est trouvé ou bien une profondeur maximale est trouvée*.
- Très cher en temps et espace,
- *mais* garantie de trouver la solution (si elle existe).

La recherche aveugle - En largeur

- En largeur : Visiter les états en parcourant l'arbre niveau par niveau.
- On utilise une file d'attente.
- *la recherche s'arrête quand on état final est trouvé ou bien une profondeur maximale est trouvée*.
- Très cher en temps et espace,
- *mais* garantie de trouver la solution (si elle existe).
- Pas de problème de branche infinie.

L'algorithme de recherche en profondeur

- FONCTION ExplorationProf ($E_i, DejaVu, N$): *Booleen*
  - res : *Booleen*
  - SI $E_i \in F$
    - $res \leftarrow VRAI$
  - SINON SI $N = 0$
    - $res \leftarrow FAUX$
  - SINON PourTout $(A_j, E_j) \in T(E_i)$ ET NON ($E_j \in DejaVu$)
    - SI $ExplorationProf(E_j, DejaVu \cup E_j, N - 1) = VRAI$
      - $Afficher A_j, E_j$
      - $res \leftarrow VRAI$
  - *Retourner* res

L'algorithme de recherche en largeur

- Fonction ExplorationLarg(E_0) : *Liste*
  - F : FILE
  - $F \leftarrow fileVide$, $L \leftarrow listeVide$
  - $Ajouter(F, E_0)$
  - TANTQUE NON vide(F)
    - $inserer(L, premier(F))$
    - SI NON $premier(F) \in F$
      - PourTout $(A_j, E_j) \in T(premier(F))$ $[ajouter(F, E_j)]$
      - $supprimer(F)$
    - SINON
      - $F \leftarrow fileVide$
    - *Retourner* L

Les différents problèmes de Recherche d'un

- Recherche d'une *solution quelconque* : algorithmes de recherche en profondeur d'abord

Les différents problèmes de Recherche d'une

- Recherche d'une *solution quelconque* : algorithmes de recherche en profondeur d'abord
- Recherche de *toutes les solutions* : construire l'arbre en largeur d'abord en introduisant une stratégie qui n'explore pas les branches ne menant pas à une bonne solution.

Les différents problèmes de Recherche d'un

- Recherche d'une *solution quelconque* : algorithmes de recherche en profondeur d'abord
- Recherche de *toutes les solutions* : construire l'arbre en largeur d'abord en introduisant une stratégie qui n'explore pas les branches ne menant pas à une bonne solution.
- Recherche de la *meilleure solution* selon un critère donné : un coût qui sera associé à l'ensemble des *actions* formalisant le problème.

Introduction dun coût

- Notations :
  - $k(E_i, E_j)$ Le coût de l'action la moins chère pour aller de E_i à E_j si elle existe.
  - $k^*(E_i, E_j)$ Le coût de la séquence d'actions la moins chère pour aller de E_i à E_j.
  - $g^*(E_i)$ $g^*(E_i) = k^*(E_0, E_i)$
  - $h^*(E_i)$ $h^*(E_i) = \min k^*(E_i, E_j)$ avec $E_j \in F$, autrement dit $h^*(E_i)$ représente le coût minimal pour atteindre l'objectif si ce chemin existe.
  - $f^*(E_i)$ $f^*(E_i) = g^*(E_i) + h^*(E_i)$ Autrement dit $f^*(E_i)$ représente le coût minimal d'une solution passant par E_i si elle existe.

Recherche Guidée : Définitions

- Une solution est *optimale* s’il n’existe pas aucune solution de coût strictement inférieur.

Recherche Guidée : Définitions

- Une solution est *optimale* s'il n'existe pas aucune solution de coût strictement inférieur.
- Une méthode est *admissible* si chaque solution optimale est trouvée en un temps fini.

Recherche Guidée : Définitions

- Une solution est *optimale* s'il n'existe pas aucune solution de coût strictement inférieur.
- Une méthode est *admissible* si chaque solution optimale est trouvée en un temps fini.
- Une *heuristique* est une mesure associée à un état donné qu'on notera $h(E_i)$.
  - $h(E_i)$ est *coïncidente* si $\forall E_j \in F, h(E_j) = 0$.
  - $h(E_i)$ est *presque parfaite* si $h(E_j) < h(E_i)$ où E_j est un état qui suit E_i.
  - $h(E_i)$ est *consistante* si $h(E_i) - h(E_j) \leq k^*(E_i, E_j)$.
  - $h(E_i)$ est *monotone* si $\forall (E_i, E_{ij}), (A_{ij}, E_{ij}) \in T(E_i), h(E_i) - h(E_{ij}) \leq k(E_i, E_{ij})$.
  - $h(E_i)$ est minorante si $h(E_i) \leq h^*(E_i)$

Théorèmes sur les heuristiques - 1

- h est monotone ssi h est consistante
  - Démonstration :
    - *Supposons que h est consistante , dons pour toute paire d'états nous avons* $h(E_i) - h(E_j) \leq k^*(E_i, E_j)$ *et plus particulièrement si* $E_j \in Succ(E_i)$ *alors* $h(E_i) - h(E_j) \leq k^*(E_i, E_j)$ *et par définition nous avons* $k^*(E_i, E_j) \leq k(E_i, E_j)$*.*
    - *Maintenant supposons que la stratégie est monotone Soient* (E_0, E_n) *une paire d'états pour laquelle il y a un chemin optimal* $E_1, ..E_{n-1}, E_n$*, nous avons* $\forall i(h(E_{i-1}) - h(Ei) \leq k(E_{i-1}, Ei))$ *En sommant nous obtenons :* $h(E_0) - h(E_n) \leq \sum_{i=1}^{n} k(E_{i-1}, Ei)$ *et donc :* $h(E_0) - h(E_n) \leq k^*(E_0, E_n)$

Théorèmes sur les heuristiques - 2

- Si h est monotone et coïncidente, alors h est minorante.
  - Démonstration :
    - *Par hypothèse de monotonie et sur un chemin optimal, on obtient que* $h(E_0) - h(E_n) \leq k^*(E_0, E_n)$ *et puisque le chemin est optimal alors* $h(E_0) - h(E_n) \leq h^*(E_0)$ *et puisque l'heuristique est coïncidente alors* $h(E_0) \leq h^*(E_0)$

L'algorithme A* : Principe

- Utilisation d'une heuristique pour pendant l'exploration de l'arbre.

L'algorithme A* : Principe

- Utilisation d'une heuristique pour pendant l'exploration de l'arbre.
- Deux files sont utilisées pour stocker les états visités et à visiter : *Inactif* & *Actif*
  - Parmi tous les successeurs d'un état, sont ajoutés à actifs *Actif* :
    - les états non visités,
    - les états déjà visités ayant un coût actuel moins élevé.
  - Une mesure $f(e) = g(e) + h(e)$ est associée à chaque état.
  - La file *Actif* est triée par ordre de f croissant.

L'algorithme A*

- Procedure A*()
  - e,e' : ETAT, Actif, Inactif : FILE,
  - $Actif \leftarrow [e_0]$, $Inactif \leftarrow []$
  - $g(e_0) \leftarrow 0$, $e \leftarrow e_0$
  - TANTQUE *non* $fileVide(Actif)$ *ET non* $e \in F$
    - $supprimer(Actif)$,$inserer(Inactif, e)$
    - PourTout$e' \in Succ(e)$, SI *non* ($e' \in Actif$ *ET* $e' \in Inactif$) *OU* $g(e') > g(e) + k(e, e')$
      - $g(e') \leftarrow g(e) + k(e, e')$, $f(e') \leftarrow g(e') + h(e')$
      - $pere(e') \leftarrow e$
      - $ajouterTrier(Actif, e')$
    - SI *non*($fileVide(Actif)$)
      - $e \leftarrow premier(Actif)$

Complément : Théorèmes & Définition

- Si tout chemin de longueur infinie a un coût infini, si sur un chemin optimal, la valeur de l'heuristique est bornée et si chaque état a un nombre fini de successeurs alors l'algorithme A* termine.

Complément : Théorèmes & Définition

- Si tout chemin de longueur infinie a un coût infini, si sur un chemin optimal, la valeur de l'heuristique est bornée et si chaque état a un nombre fini de successeurs alors l'algorithme A* termine.
- Si les conditions de terminaisons sont réalisées et si l'heuristique est minorante alors l'algorithme A* est admissible.

Complément : Théorèmes & Définition

- Si tout chemin de longueur infinie a un coût infini, si sur un chemin optimal, la valeur de l'heuristique est bornée et si chaque état a un nombre fini de successeurs alors l'algorithme A* termine.
- Si les conditions de terminaisons sont réalisées et si l'heuristique est minorante alors l'algorithme A* est admissible.
- L'heuristique h2 est *plus informée* que h1 si toutes les deux sont minorantes et si $h2(e) > h1(e)$, $\forall e \in S$.