Systèmes Intelligents : Raisonnement et Reconnaissance

James L. Crowley

Deuxième Année ENSIMAG Deuxième Semestre 2006/2007

Séance 2 7 février 2007

Planification et Parcours de Graphe

L'Agent intelligent

Pour formaliser l'intelligence, Nilsson a proposé le concept d'un agent intelligent

L'agent a :

La capacité d'agir : Un corps "physique" capable de réaliser des actions.

Les buts. (Un but est un état recherché de l'environnement).
les buts peuvent êtres représentés au niveau symbole avec
La même technique que pour la connaissance,
mais les buts sont essentiels dans le niveau connaissance.

Des connaissances. La compétence.
La capacité de choisir les actions pour atteindre ses buts

Comportement : principe de rationalité

Les actions sont choisies pour atteindre les buts de l'agent.

Définition de l'intelligence :

la capacité de choisir les actions afin d'atteindre un objectif.

Ceci est formalisé par le problème de la planification.

Planification

Planification : La recherche d'une suite d'actions pour atteindre un but.

Pour étudier l'intelligence, on étudie la planification.

La planification concerne un "espace de problème" (Problem Space).

Un espace de problème (Problem Space) est défini par :

1) l'ensemble des états de l'univers, {U}, et
2) l'ensemble des opérateurs de changement d'état
(l'ensemble des actions, {A}.)

Un problème est :
un univers, {U},
un état initial, S
un ensemble de buts qui sont des états, {B}, et

La planification engendre une SEQUENCE d'actions qui transforme l'état S
en un des états b de {B}.

Les états de l'univers, {U} :

Chaque état est une description PARTIELLE de l'univers.

En général, un état est défini par un vecteur des propriétés.
Les propriétés peuvent êtres numériques ou booléens.
Les propriétés booléennes sont représentées par les prédicats

Exemples :
Robot mobile : position(x, y) ∧ heur(t)
Monde des blocs : sur table (A) ∧ Sur (A, B) ∧ MainVide
Pour simplifier la complexité du raisonnement,
on peut grouper les états pour former les états plus abstraits.

Monde des blocs

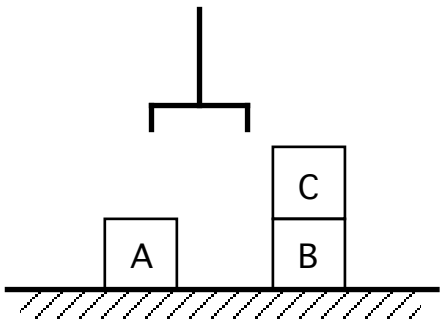
Le monde des blocs est un domaine TRES simple pour explorer les problèmes de la planification.

Il comprend un nombre finit de "blocs" qui ont un nombre fini d'états possibles.

Le monde des blocs est un univers composé de:
- un ensemble de blocs dans un état donné, et
- un agent qui effectue des actions primitives sur les blocs pour changer leur état.

Définition classique de monde des blocs
1) L'univers est composé d'un ensemble de blocs cubiques et d'une table.
2) Les blocs sont mobiles, la table est immobile.
3) L'agent est un bras-pince mobile.
4) Un bloc peut reposer sur la table, sur un autre bloc ou être dans la pince.
5) Il ne peut pas y avoir plus d'un bloc sur un autre bloc.
6) La table est suffisamment grande pour que tous les blocs puissent y prendre place
7) La pince ne peut déplacer qu'un bloc à la fois.

L'état de l'univers est décrit avec le calcul des prédicats en FBF sans variable.
Exemple :



Blocs: { A, B, C }
Prédicats:

Sur(A, B)	S(A, B)	Bloc A est sur bloc B.
SurTable(A)	ST(A)	Bloc A est sur la Table.
Tenu(A)	T(A)	Bloc A est dans la pince.
Libre(A)	L(A)	Il n'existe pas de bloc sur A , ou ¬∃x (Sur(x,A)) ou ∀x (¬Sur(x,A))
MainVide	MV	La main est vide, ou ¬∃x (T(x))

L'état de l'univers est exprimé en conjonction de prédicats :

MV ^ ST(A) ^ ST(B) ^ S(C, B) ^ L(C) ^ L(A)

Les actions:

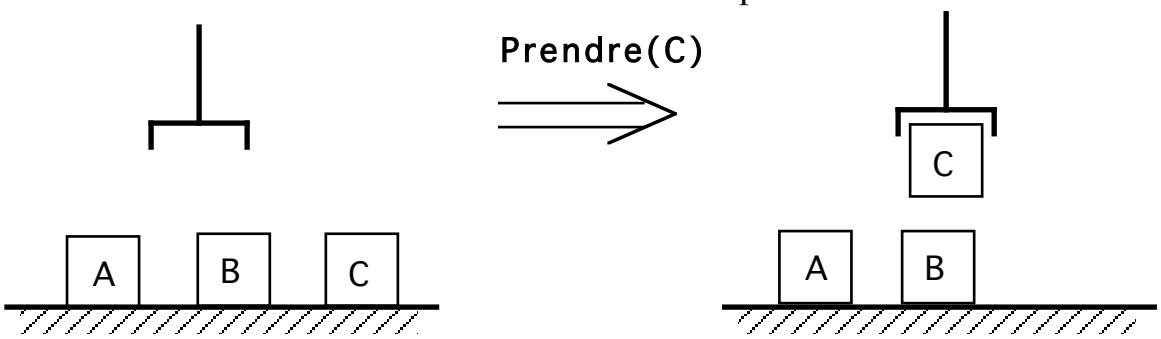
Les actions sont les opérations qui peuvent être appliquées sur les éléments (blocs) pour les changer d'état. Elles sont primitives.

Un formalisme pour les actions a été développé pour STRIPS (Stanford Research Institute Problem Solver) (1971)

Principe : liste explicite de tout changement d'état

Action: Nom(variables)

Précondition	doit être vrai avant l'action
Retraits	rendus faux par l'action
Postcondition	rendue vraie après l'action

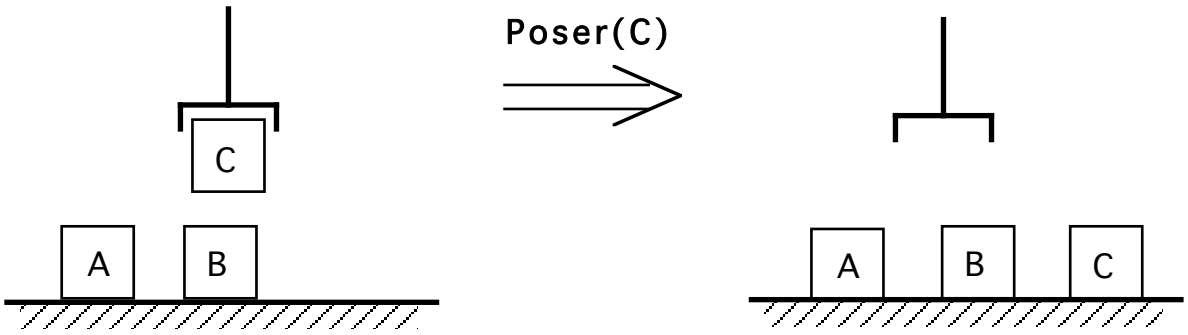


Prendre(x)

Pre: MV ^ L(x) ^ ST(x)

Ret: MV ^ L(x) ^ ST(x)

Aj: T(x)

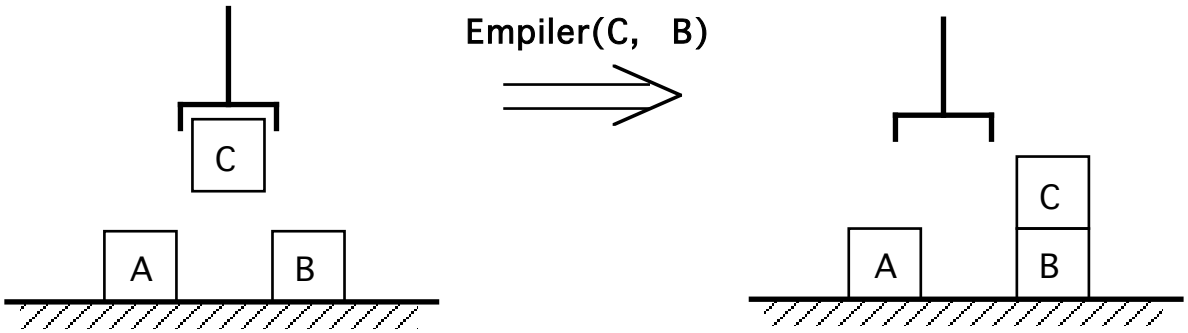


Poser (x)

Pre: T(x)

Ret: T(x)

Aj: L(x) ^ ST(x) ^ MV

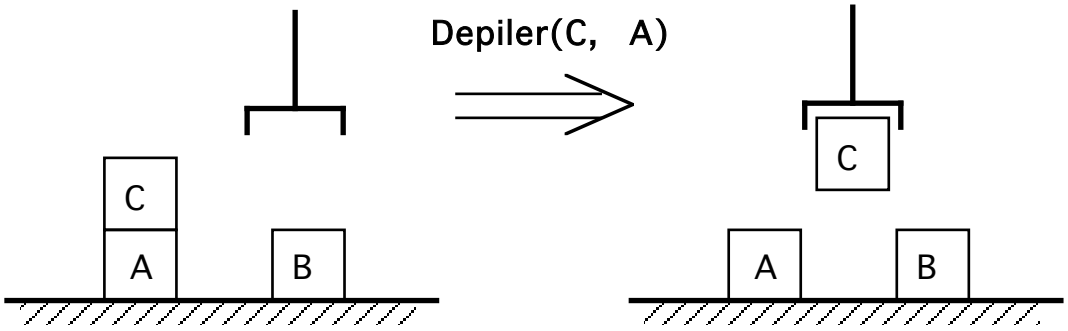


Empiler (x, y)

Pre: T(x) ^ L(y)

Ret: T(x) ^ L(y)

Aj: L(x) ^ S(x, y) ^ MV



Depiler (x, y)

Pre: L(x) ^ S(x, y) ^ MV

Ret: L(x) ^ S(x, y) ^ MV

Aj: T(x) ^ L(y)

Question: Pourquoi Poser(x) n'est pas Empiler(x, table)

Rep: Si on exécute Empiler(x, table), Libre(Table) n'est plus

vrai.

Planification comme Parcours de Graphes :

Un problème est :
un univers, {U},
un état initial, S
un ensemble de buts qui sont des états, {B}, et

La planification engendre une SEQUENCE d'actions qui transforme l'état S
en un des états b de {B}.

Le paradigme pour la planification est :

"Generate and Test"

1) A partir d'un état engendrer l'ensemble des actions possibles.
2) Tester ces actions pour trouver ce qui nous amener vers un état solution.
3) Aller à cet état et boucler.

La planification exige une recherche (d'une solution) => (parcours de graphe)

Les techniques de recherche d'une séquence d'actions sont
1) parcours de graphe en profondeur
2) parcours de graphe par niveau.
3) parcours heuristique
4) parcours hiérarchique

Les trois premières sont des variations de l'algorithme GRAPHSEARCH (Nilsson).

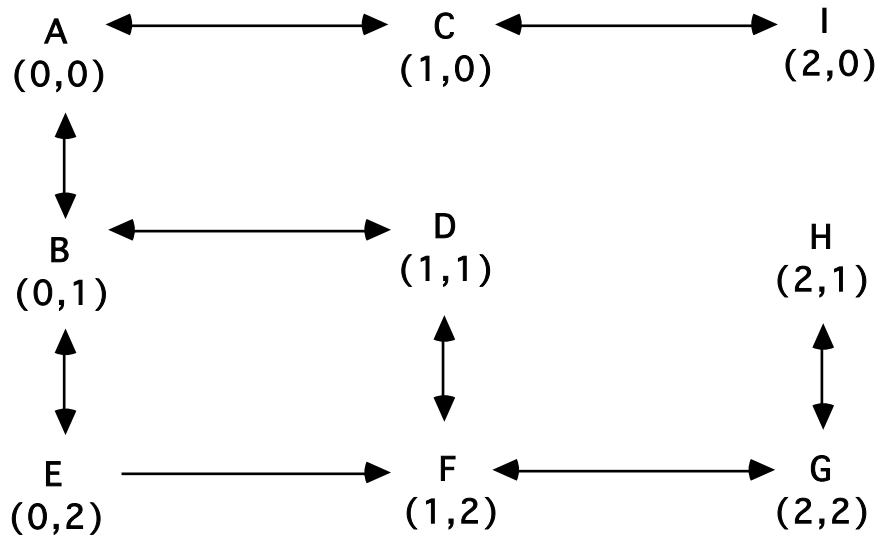
La complexité de calcul de la planification dépend de la connaissance
qui peut être appliqué au problème.

Le graphe d'états

En l'absence de connaissance, l'agent peut planifier par exploration de tout l'espace d'états, en cherchant un état de l'ensemble des buts.

Univers classique : planification du chemin pour un robot mobile

La connaissance sur la navigation est représentée par un réseau d'emplacements



Chaque emplacement a un nom, une localisation, et une liste des emplacements voisins accessibles par un déplacement rectiligne.

Si le robot est en A et si je dis "allez à H", il engendre un arbre de recherche.

Il y a trois techniques pour élaborer cet arbre :
par niveau, en profondeur, et par recherche heuristique.

Terminologie et notation :

1) Un arbre de recherche est composé de "noeuds" et d'"arcs".
2) Les arcs sont orientés. ex : n_i --> n_j
3) Dans (2) : n_j est le successeur (ou fils) de n_i,
n_i est le prédécesseur (ou père) de n_j.

Sortes de Parcours de Graphe

Par niveau : (Breadth First, Exhaustive)
Pour chaque noeud :

1) créer la liste des successeurs,
2) éliminer tous les prédécesseurs,
3) mettre la liste en fin d'une file (FIFO).
4) choisir n <- tête de file. Si n ≠ BUT alors aller à 1.

En profondeur : (Depth First, Exhaustive)
Pour chaque noeud :

1) créer la liste des successeurs,
2) éliminer tous les prédécesseurs,
3) empiler la liste sur une pile (LIFO).
4) choisir n <- sommet de pile. Si n ≠ BUT alors aller à 1.

Parcours de graphe heuristique :

Heuristique : une connaissance qui sert à guider.

Une fonction heuristique d'évaluation est une forme de connaissance d'un domaine
Il s'agit d'une fonction d'estimation du coût entre un état et le but le plus proche.

Parcours exhaustif :
- FIFO - parcours par niveau
- LIFO - parcours en profondeur

Heuristique :
- Algorithme A : fonction d'évaluation sans restriction.
- Algorithme A* : restriction sur la fonction d'évaluation.

On peut assurer que le chemin est optimal si f(n) satisfait les conditions d'optimalités.

L'algorithme GRAPHSEARCH (Nilsson)

Symboles :

T : arbre de recherche
G : ensemble des buts
d : noeud de départ
M : liste des successeurs
Open : ensemble des noeuds "ouverts"
(liste prête à être exploré)
Closed : ensemble des noeuds "fermés"
(liste déjà exploré)
n, e : un noeud

Algorithme :

1) Créer T, Open et Closed, (tous égaux à l'ensemble vide).
2) Mettre d dans Open et comme racine de T.
3) Boucle : si OPEN est vide, alors Exit avec échec.
4) Extraire n de Open. Open <- Open - n, Closed <- Closed + n

5) Si n est un élément de G alors : succès !!
Construire une pile avec les noeuds de T constituant le chemin entre n et s.
Exit en rendant la pile.
Fin si

6) Expansion de n : créer M (les successeurs de n)

7) Pour chaque noeud e de M,
si e est dans Closed alors M <- M - e.
sinon
a) optionnel : calculer le coût estimé f(e) pour e.
b) ajouter e dans Open (LIFO, FIFO, Trier)
c) ajouter e à T comme successeur de n.

8) Aller à l'étape 3

Optimalité de GRAPHSEARCH

Notation :

S : point de départ
B : but
$k(n_i, n_j)$: coût minimal entre n_i et n_j.
$g^*(n) = k(S, n)$: le coût réel minimal entre S et n.
$h^*(n) = k(n, B)$: Le coût réel minimal entre n et B.

$f^*(n) = g^*(n) + h^*(n)$ Le coût du chemin optimal entre S et B passant par n.

Remarque : il n'est pas toujours possible de connaître $h^*(n)$ ou $g^*(n)$; par conséquent il faut les estimer.

$h(n)$: coût estimé entre n et B.
$g(n)$: coût estimé entre S et n.
$f(n) = g(n) + h(n)$

NB: $g(n) = g^*(n)$ et si le coût satisfait une condition de monotonicité. c.a.d. si pour tout n_j successeur de ni

$$h(n_i) - h(n_j) \leq k(n_i, n_j)$$

(presque toujours vrai si $h(n) \leq h^*(n)$)

Mais il faut estimer h(n) (plus difficile)

1) sans condition : algorithme A.
2) h = 0, g(n) = d (profondeur) : recherche par niveau
3) Si $f(n) \leq f^*(n)$ alors le chemin sera optimal : (algorithme A*).

Remarque:

1) Il faut avoir une fonction de coût $k(n_i, n_j)$, qui exprime la connaissance de l'univers sous forme numérique.
2) Il faut avoir un moyen d'estimer le coût tel que

$$h(n) \leq h^*(n).$$

Exemple :
Pour un robot mobile : pour ni, nj voisins, soit $k(n_i, n_j) = \text{Dist}(n_i, n_j)$

alors en général: $k(n_i, n_j) = \Sigma$ distance traversée dans le graphe
Dans ce cas : $g(n) = g^*(n)$
et

$$h(n) = \text{Dist}(n, B) \leq h^*(n) = k(n_i, n_j)$$

Le chemin sera optimal. Mais comment faire sinon?

Complexité de calcul de GRAPHSEARCH

La complexité de l'élaboration du graphe d'états dépend de :

b : facteur de branchement. Le nombre moyen de nouveaux états qui peut être engendré à partir d'un état.

d : profondeur minimum entre l'état de départ et un but.

Parcours par niveau :
la solution sera trouvée avec un temps de calcul de l'ordre de $O(b^d)$ et un espace-mémoire de l'ordre de $O(b^d)$.

Parcours en profondeur :

temps de calcul de l'ordre de $O(b^d)$ avec des besoins de mémoire de l'ordre de O(d). MAIS, il faut donner une profondeur maximale de "retour arrière" sinon le parcours ne se terminera pas !

La profondeur maximale est rarement connue ! Si une valeur d est donnée et la solution est à $d' > d$, elle ne sera pas trouvée.

Parcours heuristique : le heuristique permet de reduire le facteur de branchement.

Si b est le facteur sans connaissance, h() permet de réduire le branchement à un facteur $c \leq b$. La complexité est alors :

Coût en calcul en $O(c^d)$, coût en mémoire en $O(c^d)$

Ceci permet la recherche dans un espace d'états plus grand.
Mais il y a toujours une croissance exponentielle.

La solution : application de connaissances symboliques

Complexités des Algorithmes de Parcours de Graphe :
1) En Profondeur : $O(b^d)$
2) Heuristique : $O(c^d)$ $c \leq b$
3) Hiérarchique : $O(d \ln b)$!!

La solution : application de connaissances symboliques en formes de sous-buts

L'utilsation de sous-buts

Les applications des sous-buts peuvent fortement réduire la complexité d'un algorithme de planification.

soit :

d : le nombre minimum des actions entre l'état de départ, i et un but b.

d_{is} : le nombre minimum des actions entre l'état de départ, i et un sous-but s.

d_{sb} : le nombre minimum des actions entre le sous-but s et le but, b.

Un sous-but divise le problème de profondeur d en deux.
La complexité du parcours vient :

$$O(b^{d_{is}} + b^{d_{sb}}) = O(b^{\max\{d_{is}, d_{sb}\}})$$

Si $\max\{d_{is}, d_{sb}\} < d$, alors la complexité sera réduite.

Si S est dans le chemin optimal, alors,

$$d = b^{d_{is}} + b^{d_{sb}}$$

et le sous-but coupera la complexité en deux !!

En général, il y a un ensemble des sous-buts :

$$S = \{ S_0, S_1, S_2, ..., S_N\}$$

S_N = BUT, S_0 = etat initial.

le coût de la recherche est $O(b^{\max(Dist(S_{n-1}, S_n)_)}$

Exemple : pour traverser un ruisseau, la difficulté dépend sur l'enjambée la plus grande.

Conclusion

L'algorithme optimal de résolution du problème est une HIERARCHIE des espaces des abstractions.

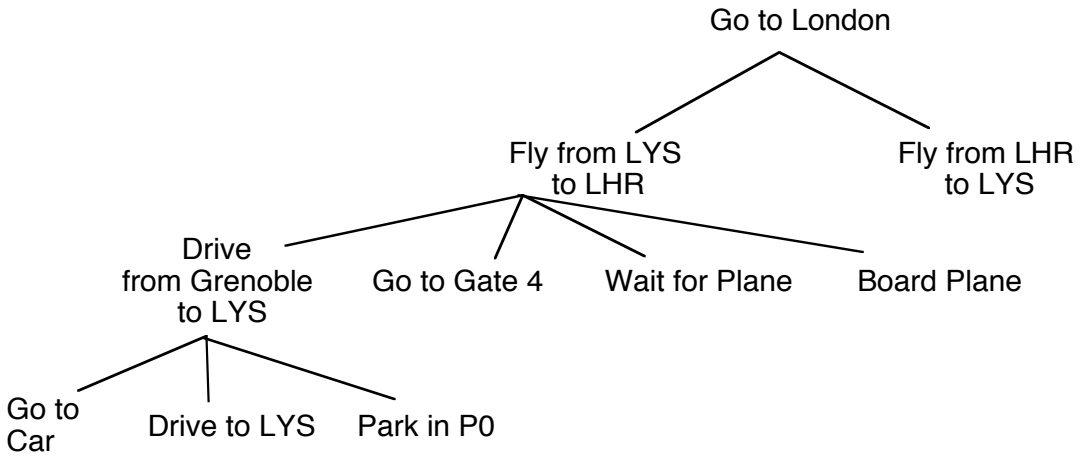
Chaque abstraction sert à grouper un ensemble d'états dans un méta-état.

Les méta-états sont les connaissances particulières de chaque domaine.

L'abstraction s'applique à toute forme de raisonnement.

Exercice

Un "script" est une structure hiérarchique des actions. Les actions peuvent être elles aussi des scripts.



Faire une analyse hiérarchique pour la tâche «aller de Bt. E de l'ENSIMAG au musée d'Orsay à Paris». Identifier les niveaux de la hiérarchie. Pour chaque niveau, faire une liste des actions possibles. Lister quelques coûts possibles. Existe-il des heuristiques optimales pour ces coûts?