Planification en IA

I - Le problème de la planification

Il s'agit de trouver une séquence d'actions permettant d'atteindre un but.

Cet objectif entre dans la catégorie très générale de la résolution de problème.

Dans la mesure où la recherche d'une solution à un problème peut s'assimiler à la recherche d'un chemin dans un graphe, on peut appliquer les méthodes de recherche telle : la recherche en largeur, en profondeur, guidée par une heuristique.

Planification en IA

Le problème majeur de la planification est celui du très grand nombre d'actions invocables.

Si l'on dispose de 10 actions (engendrer le nombre 0, engendrer le nombre 1, … , engendrer le nombre 9) et que l'on veut former le n° ISBN d'un ouvrage (par exemple ISBN0137903952),

le planificateur va tenter d'appliquer chacune d'entre elles pour produire chacun des 10 nombres requis.

On suppose que le bon n° ISBN permet d'accéder à la métadonnée d'un document de la BD.

Planification en IA

Si certains mots clés sont présents dans cette méta-donnée on conserve ce document.

Le planificateur pourra ainsi être amené à construire 10^{10} plans différents.

Sans heuristique pour guider le planificateur, la construction d'un plan d'actions peut s'avérer trop couteuse.

Dans le cas de la construction d'une voiture, le planificateur ne doit pas construire le plan de montage en essayant d'assembler n'importe quoi avec n'importe quoi.

Planification en IA

Une difficulté est alors d'avoir une bonne heuristique pour guider le planificateur.

Dans ce cas la recherche de solution (recherche d'un chemin dans un graphe) peut utiliser une méthode de recherche telle que A*.

Or un assemblage « intelligent » serait de construire

le chassis,

le bloc moteur,

la carrosserie,

la transmission,

le train de roulement, … et d'assembler le tout.

Planification en IA

Une telle heuristique existe-t-elle ? Il n'est pas évident qu'avec de simples critères de coût (pour la méthode A*) on y arrive.

Ce que suggère l'assemblage intelligent d'une voiture est la possibilité de décomposer un problème en sous problèmes. Chaque sous-problème peut lui-même faire l'objet d'une planification.

Le planificateur

Un planificateur typique manipule 3 entrées codées dans un langage formel (ex: STRIPS) qui utilise des prédicats logiques

Planification en IA

Ces entrées sont :

– une description de l'état initial d'un monde,
– une description d'un but à atteindre et
– un ensemble d'actions possibles (appelés aussi opérateurs de changement d'état)

Chaque action spécifie :

- des préconditions qui doivent être présentes dans l'état actuel pour qu'elle puisse être appliquée et
- des postconditions (effets sur l'état actuel).

D'autres paramètres/conditions peuvent intervenir.

Planification en IA

La recherche d'un plan

Elle se fait soit :

- en "marche avant" dans un espace d'états (recherche progressive),
- en "marche arrière" dans un espace d'états (recherche régressive),
- en "marche avant dans un espace de plans (graphplan)
- par transformation en problème de satisfiabilité de propositions.

Les problèmes en planification résultent des simplifications faites sur les conditions, sur le temps (discret, continu), les actions (déterministes ou non), l'observabilité de l'état du monde (complète ou non), etc.

Planification en IA

La planification hiérarchique

Certains problèmes sont difficilement résolubles du fait de leur complexité.

On peut gagner en efficacité en effectuant une abstraction des détails et en changeant la granularité des opérateurs de planification.

On peut par exemple commencer à planifier à haut niveau puis descendre dans le détail au besoin (ex: ABSTRIPS).

Planification en IA

La planification non linéaire

La planification classique résout les sous-buts dans un ordre donné (on dit linéairement).

Cela amène parfois à détruire ce qui a déjà été construit (anomalie de Sussman).

Exemple: un individu pieds nus doit se retrouver dans l'état où
- il porte sa chaussure droite,
- sa chaussure gauche,
- sa chaussette droite et
- sa chaussette gauche.

S'il cherche à réaliser les buts dans cet ordre, il échouera.

Planification en IA

Pour résoudre ce type de problème, on peut passer à des plans partiellement ordonnés dans lesquels l'ordre entre les actions n'est fixé que lorsque c'est nécessaire (*engagement au plus tard* ou *least commitment planning*).

Dans l'exemple précédent :

- mettre la chaussure gauche doit se faire après avoir mis la chaussette gauche
- idem pour la droite.

Par contre l'exécution du plan pour la gauche est indépendante de l'exécution pour la droite. Le plan global est donc partiellement ordonné.

Les planificateurs capable de gérer ce type de problème sont dits à ordre partiel (POP, NOAH).

Planification en IA

La planification non déterministe

Si l'hypothèse du déterminisme est abandonnée

et

si un modèle probabiliste de l'incertitude est adopté

alors

le problème est celui de la génération de politique (ou stratégie) que ce soit pour

- un processus de décision markovien (MDP)

ou

- un processus de décision markovien partiellement observable (POMDP).

Planification en IA

II - L'agent capable de planifier

Il est évidemment cognitif.

Il gère une représentation explicite des connaissances (croyances) sur le monde basée sur la logique (classique ou non).

L’agent peut avoir une réflexion sur le monde et communiquer à son propos.

Les états but sont caractérisés par des formules.

L’agent peut avoir une réflexion sur ses buts et sur les effets des actions.

Sa capacité à raisonner s'appuie sur un système de déduction logique (schémas d'axiomes, règles d'inférence).

Planification en IA

La planification peut être vue comme un processus inférentiel.

Elle se formule bien en termes de situations :

Etat initial : I

Actions (ensemble d'axiomes : conditions $\rightarrow$ effets) : A

Etat final : B

Une preuve constructive de : $\mathrm{I} \wedge \mathrm{A} \vdash \mathrm{B}$

produit un plan d’actions qui amène l’état résultat.

Planification en IA

Etat initial I :

$$at(home, s_0) \wedge \neg have(lait, s_0)$$

Actions (axiomes d'effet) A :

$$\forall a, s \quad have(lait, result(a, s)) \leftrightarrow [a = acheter(lait) \wedge poss(acheter(lait), s)] \vee [have(lait, s) \wedge a \neq poser(lait)]$$

Etat final B (requête):

$$\exists s \quad at(home, s) \wedge have(lait, s)$$

On ajoute deux axiomes (prédicat result') de construction de plan :

$$\forall s \quad result'(nil, s) := s$$

$$\forall a, p, s \quad result'(a.p, s) := result'(p, result(a, s))$$

Planification en IA

```
at(home) true_in s0.

not_have(milk) true_in s0.

have(milk) true_in result(buy(milk),S) :-
    buy(milk) possible_in S.

possible(buy(_)) true_in _.

at(L) true_in result(goto(L),S) :-
    member(L,[home,grocery]),
    goto(L) possible_in S.

not_at(L) true_in S :-
    at(H) true_in S,
    H \= L.
```

```
goto(L) possible_in S :-
    not_at(L) true_in S.

buy(_) possible_in S :-
    at(grocery) true_in S,
    possible(buy(_)) true_in S.

resulter([ ]) from S give S.

resulter([A | P]) from S0 give SF :-
    resulter(P) from result(A,S0) give SF.
```

Planification en IA

?- resulter(P) from s0 give S, have(milk) true_in S.

P = [goto(grocery), buy(milk)],

S = result(buy(milk), result(goto(grocery), s0)) ;

P = [goto(grocery), goto(home), goto(grocery), buy(milk)],

S = result(buy(milk), result(goto(grocery), result(goto(home), result(goto(grocery), s0)))) ;

P = [goto(grocery), goto(home), goto(grocery), goto(home), goto(grocery), buy(milk)],

S = result(buy(milk), result(goto(grocery), result(goto(home), result(goto(grocery), result(goto(home), result(goto(grocery), s0)))))) .

Planification en IA

La planification peut donc être vue comme un processus déductif.

Mais, tel quel,

il est très inefficace: espace de recherche trop important pour un démonstrateur.

Il faut

- soit un système d'inférence spécialisé pour une représentation réduite de la connaissance,

car

ce qu'on obtient avec la logique des prédicats du 1^{er} ordre n'est pas décidable (seulement semi-décidable)

Si p est un plan, alors ε.p est un plan

Si p est un plan, alors $a.a^{-1}$.p est un plan

Planification en IA

- soit un algorithme de planification adéquat.

STRIPS (STanford Research Institute Problem Solver)

Etats = Conjonction de littéraux fermés et sans symboles fonctionnels, sauf constantes (logique complète avec hypothèse du monde fermé).

Buts finaux = Conjonction de littéraux fermés et sans symboles fonctionnels, sauf constantes).

Pas de variable d'état explicite comme dans le calcul des situations.

Etat actuel, on ne peut pas accéder à d'autres états.

Planification en IA

Les actions en 3 parties :

Nom d’action : nom de fonction (avec paramètres)

Préconditions : conjonction de littéraux positifs

Effets : conjonction de littéraux positifs et négatifs

ADD les littéraux positifs sont ajoutés (ensemble ADD)

DEL les littéraux négatifs sont supprimés (ensemble DEL)

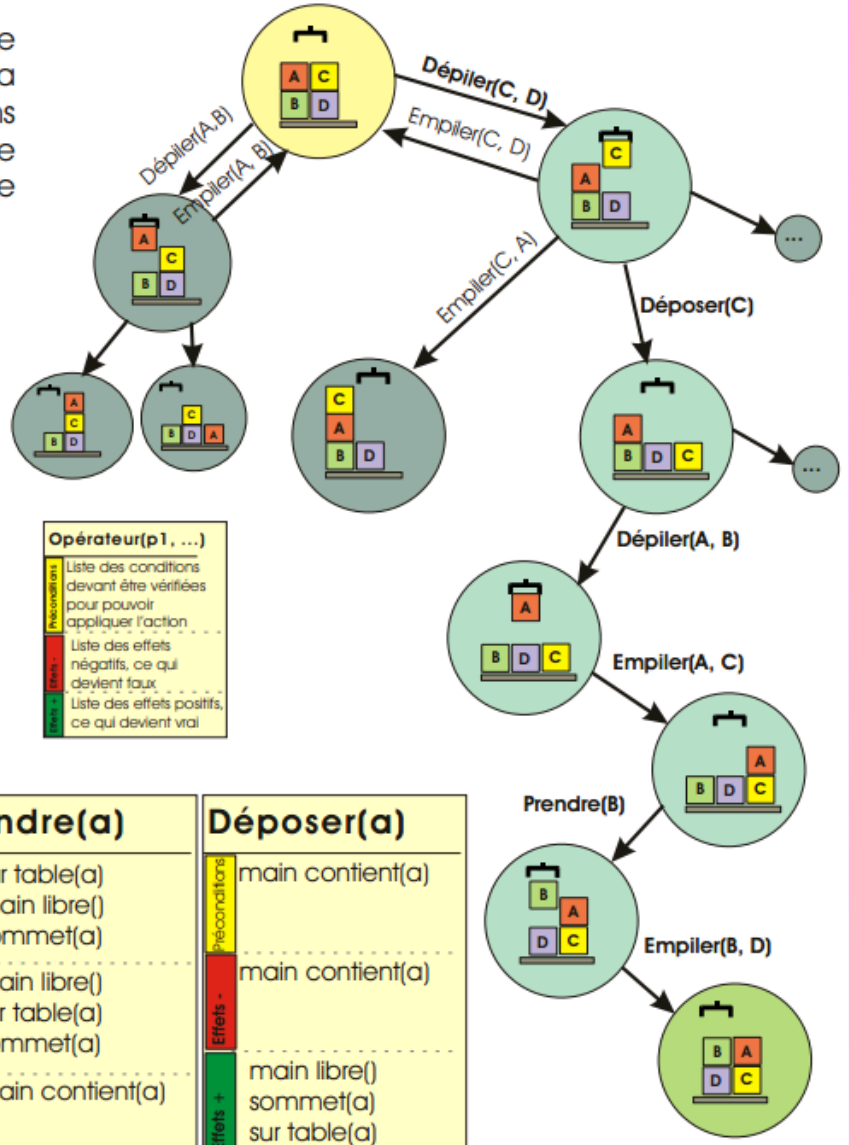
Le problème du "frame" est évacué.

(ie: difficulté d'exprimer en logique la dynamique d'une situation sans explicitement spécifier tout ce qui n'est pas affecté par les actions, comment limiter l'ensemble des informations décrivant une situation changeante afin qu'elle soit en accord avec les événements qui viennent de survenir).

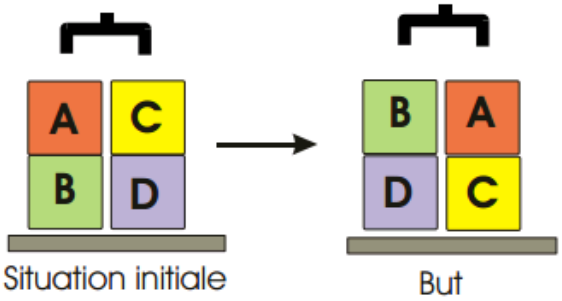
Planification en IA

Exemple : monde des blocs

Le « monde des blocs » est un exemple pédagogique souvent utilisé pour illustrer la planification en intelligence artificielle. Dans ce domaine, le planificateur doit trouver une séquence d'actions pour passer d'une configuration de blocs à une autre.



Problème



Situation initiale — But

Opérateur(p1, ...)

- Préconditions : Liste des conditions devant être vérifiées pour pouvoir appliquer l'action
- Effets - : Liste des effets négatifs, ce qui devient faux
- Effets + : Liste des effets positifs, ce qui devient vrai

Opérateurs

Dépiler(a, b)	Empiler(a, b)	Prendre(a)	Déposer(a)
Préconditions : sommet(a) sur(a, b) mainlibre()	Préconditions : sommet(b) main contient(a)	Préconditions : sur table(a) main libre() sommet(a)	Préconditions : main contient(a)
Effets - : main contient(a) sommet(a) sur(a, b)	Effets - : main contient(a) sommet(b)	Effets - : main libre() sur table(a) sommet(a)	Effets - : main contient(a)
Effets + : main contient(a) sommet(b)	Effets + : main libre() sommet(a) sur(a, b)	Effets + : main contient(a)	Effets + : main libre() sommet(a) sur table(a)

Planification en IA

Mécanisme de STRIPS

Un opérateur-action op peut être appliqué dans un état s (modèle de s), s'il y a une substitution σ de ses variables telle que toutes les préconditions sont vraies dans σ[s] :

$$\text{Precond}(\sigma[op]) \subseteq \sigma[s].$$

Dans l'état résultat, tous les littéraux positifs de Effect(σ[op]) qui étaient vrais dans s sont vrais et tous les littéraux négatifs de Effect(σ[op]) sont faux.

Définition

Si s est un état et op est une action, alors le résultat de

l'application de op à s sous la substitution σ est définie par

$$\text{resultat}(\sigma[op] , \sigma[s]) = \sigma[s] – \sigma[DELL] \cup \sigma[ADD]$$

où σ[DELL] (resp^t. σ[ADD]) est l'ensemble des littéraux négatifs (resp^t. positifs) de Effect(σ[op])

Planification en IA

Le problème du "frame" est résolu indirectement : les littéraux qui ne sont pas énoncés comme effets ne changent pas!

(principe de persistance)

La planification avec STRIPS devient un problème de recherche de parcours dans un graphe où les sommets sont les états et les chemins les plans solution.

Chaque action est une arête (arc) du graphe.

Action: aller(A,B)

Precond: $At(A) \wedge arc(A,B)$

Effect: $At(B) \wedge \neg At(A)$

$$\{At(A)\} \xrightarrow{aller(A,B)} \{At(B), \neg At(A)\}$$

Planification en IA

Recherche dans l’espace des états

A chaque état, on explore tous les chemins possibles définis par les opérateurs

Recherche par progression ou regression.

Recherche dans l’espace des plans

On commence par un plan partiel que l'on étendra successivement.

Planification en IA

Exemple "se chausser"

- Mettre les chaussettes
- Mettre les chaussures

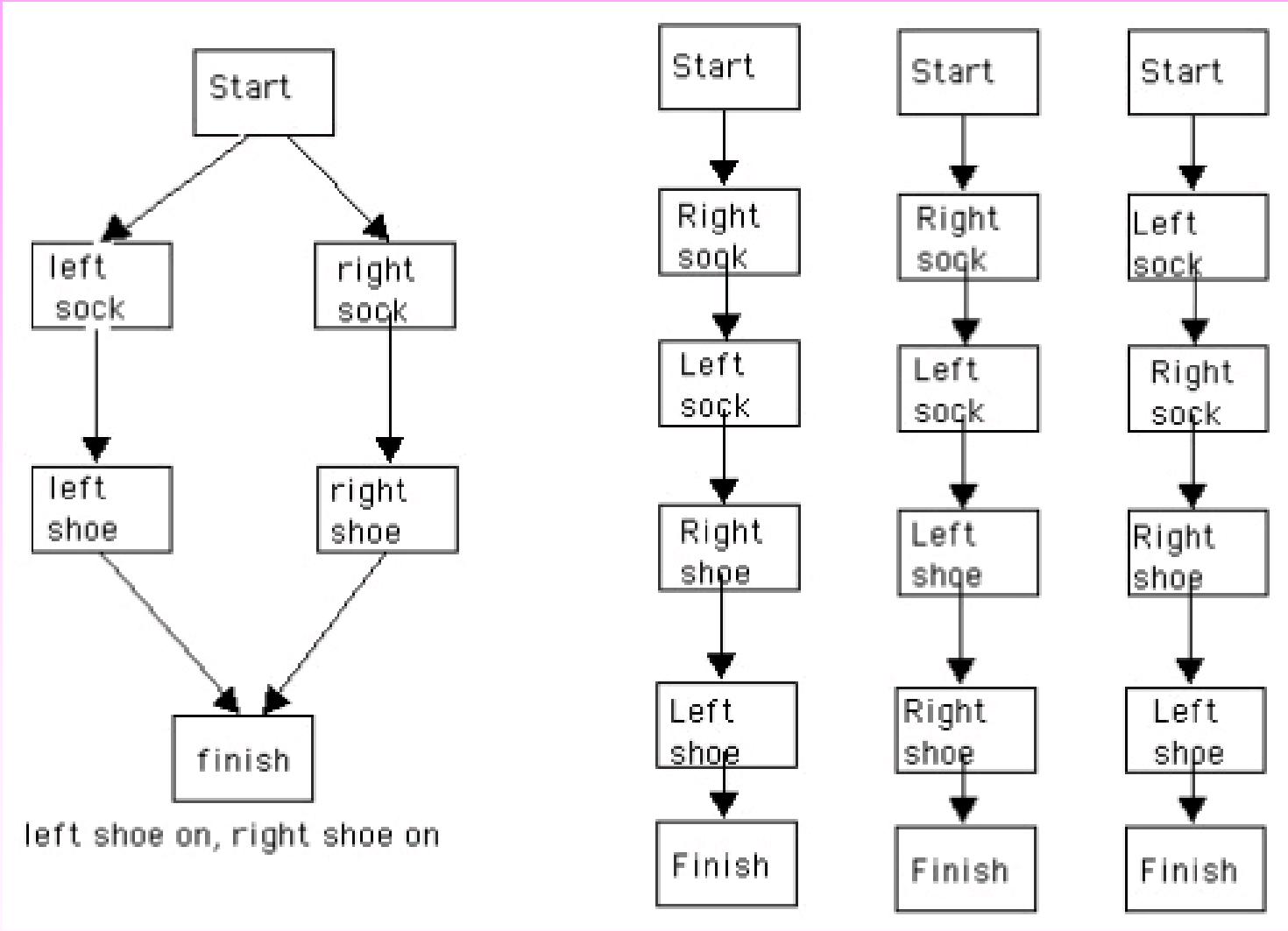
Les opérateurs d’affinement concrétisent le plan (plus de pas intermédiaires, plus de détail)

- Mettre chaussette gauche
- Mettre chaussette droite
- Mettre chaussure gauche
- Mettre chaussure droite

Les opérateurs modifient le plan.

Planification en IA

Les actions doivent elles être séquentielles ?



Planification en IA

Plans non-linéaires

Souvent il est impossible de fixer l'ordre trop tôt ou cela n'a pas de sens (mettre des chaussettes et des chaussures, s'habiller).

Les plans peuvent être non-linéaires ou partiellement ordonnés

Exemple:

right sock $\prec$ right shoe

et

left sock $\prec$ left shoe

Planification en IA

Représentation de plans non-linéaires

Définition

Etape = Opérateur STRIPS

Un plan consiste en

- un ensemble d'étapes partiellement ordonné (par $\prec$), où

  Si $\prec$ Sj ssi Si doit être exécuté avant Sj

- une substitution sur les variables s[x] = t , où x est une variable quelconque et t un terme (constante ou variable) qu'on lui substitue

- un ensemble de liens causaux : Si $\xrightarrow{c}$ Sj signifie

  "Si génère la précondition c pour Sj"

  (ce qui implique Si $\prec$ Sj)

Planification en IA

Complétude et Cohérence

La planification doit être complète et cohérente.

Plan complet

Chaque précondition d'une étape est valide :

$$\forall\ Sj \text{ si } c \in \text{Precond}(Sj) \text{ alors } \exists\ Si \quad Si \prec Sj \quad \text{et} \quad c \in \text{Effets}(Si)$$

et pour chaque linéarisation du plan :

$$\forall\ k \quad \text{si } Si \prec Sk \prec Sj \quad \text{alors} \quad \neg c \notin \text{Effets}(Sk)$$

Planification en IA

Plan cohérent

si Si $\prec$ Sj, alors Sj $\not\prec$ Si

et

si X = a, alors X $\neq$ b pour a, b constantes et X variable

(Unique Name Assumption UNA)

Un plan complet et cohérent est appelé solution du problème de planification.