Algorithmique de la Planification

Cours 3 : Algorithmes et langages

Pierre REGNIER
IRIT - Université Paul Sabatier
http://www.irit.fr/~Pierre.Regnier

0. Plan de l'exposé

1. Définitions
2. Algorithmes essentiels
3. Langages de représentation
   - 3.1. STRIPS
   - 3.2. ADL
   - 3.3. PDDL

3.1. Définitions

- Un **état** E du monde de la planification est représenté par un ensemble fini de formules atomiques sans symbole de variable. Une formule atomique de base est aussi appelée un **fluent**.

- Un **opérateur** o est un modèle d'action. Il est représenté par son nom et un triplet <pr, ad, de> où pr, ad et de sont des ensembles finis de formules atomiques qui représentent ses préconditions, ajouts et retraits. Prec(o), Add(o), Del(o) dénotent respectivement les ensembles pr, ad, de de l'opérateur o. Une **action,** dénotée par a, est une instance de base d'un opérateur o (toutes les variables de o sont instanciées).

- L'action a est **applicable** sur un état E ssi Prec(a) $\subseteq$ E, l'état résultant est l'ensemble de fluents E' = E $\uparrow$ A = (E - Del(A)) $\cup$ Add(A).

3.1. Définitions

- Un **plan séquentiel** P est une séquence finie (éventuellement vide) d'actions notée $<a_1, a_2, \ldots, a_n>$.

- **L'application $\mathcal{A}$ d'un plan d'actions séquentiel** P sur un état E est définie par :

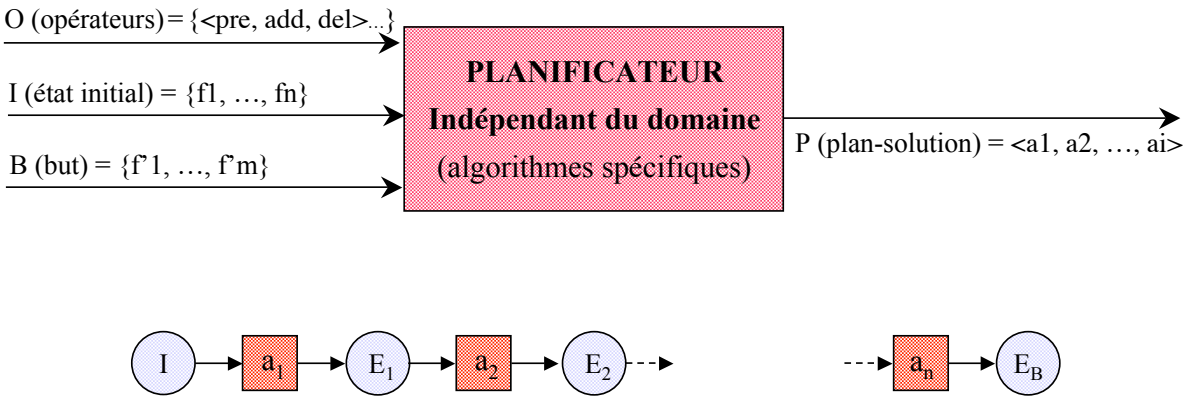
$$E\ \mathcal{A}\ P = \begin{cases} \text{Si } P = <> \text{ ou } E = \bot \text{ alors } E \\ \text{Sinon} \quad \begin{cases} \text{Si Prec(tête(P))} \subseteq E \\ \quad \text{Alors } (E \uparrow \text{tête(P)})\ \mathcal{A}\ \text{reste(P)} \\ \quad \text{Sinon } \bot. \end{cases} \end{cases}$$

- Un **problème de planification** est un triplet <O, I, B> où :
  - O dénote un ensemble fini d'opérateurs utilisables dans le domaine de la planification considéré (A dénote l'ensemble des actions produites par instanciation des opérateurs de O),
  - I est l'état initial du problème, il est représenté par un ensemble fini de fluents,
  - B est le but du problème, il est représenté par un ensemble fini de fluents.

© Régnier, IRIT 2006

3.1. Définitions

- Un **plan-solution P** au problème de planification <O, I, B> est une séquence d'actions $<a_1, a_2, \ldots, a_n>$ telle que $B \subseteq (I \mathcal{A} P)$: l'application successive de ces actions à I, donne un état résultant qui contienne B (appelé état-but).



3.1. Définitions

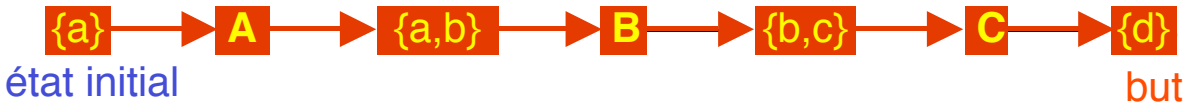
$A : a \rightarrow +b$

$B : a \rightarrow +c\ -a$

$C : b\ c \rightarrow +d$

$D : b \rightarrow +c\ +a\ -b$

$E : c \rightarrow +d\ -c$



état initial

but

Plan solution : <A, B, C>

3.1. Définitions

- Le problème général posé par la synthèse d'un plan-solution est très complexe car la planification implique :
  - la **sélection d'actions applicables ;**
  - le **choix** parmi elles**, d'actions pertinentes** pour se diriger vers le but et, par conséquent, le raisonnement sur leurs dépendances causales ;
  - un **raisonnement sur leurs interactions** pour obtenir un ordonnancement exécutable de ces actions.

- De nombreux algorithmes ont été mis au point pour planifier dans le cadre classique :
  - recherche dans les **espaces d'états ;**
  - recherche dans les **espaces de plans partiels** ;
  - **SATPLAN** (utilisation de techniques SAT) ;
  - **GRAPHPLAN** ;
  - **BLACKBOX** (GRAPHPLAN et techniques SAT) ;
  - **DPPLAN** (GRAPHPLAN et Davis et Putnam) ;
  - **GP-CSP** (GRAPHPLAN et techniques CSP)…

3.2. Algorithmes essentiels

HSP [Bonnet, Geffner, 1998]
VVPLAN [Régnier, Vidal, 1999]
HSP-R [Bonnet, Geffner, 1999]
ALTALT [Nguyen, Kambhampati, 2000]
FF [Hoffman, 2001]
YAHSP [Vidal, 2004]
DOWNWARD [Helmert, Richter, 2004]

Recherche dans les espaces d'états

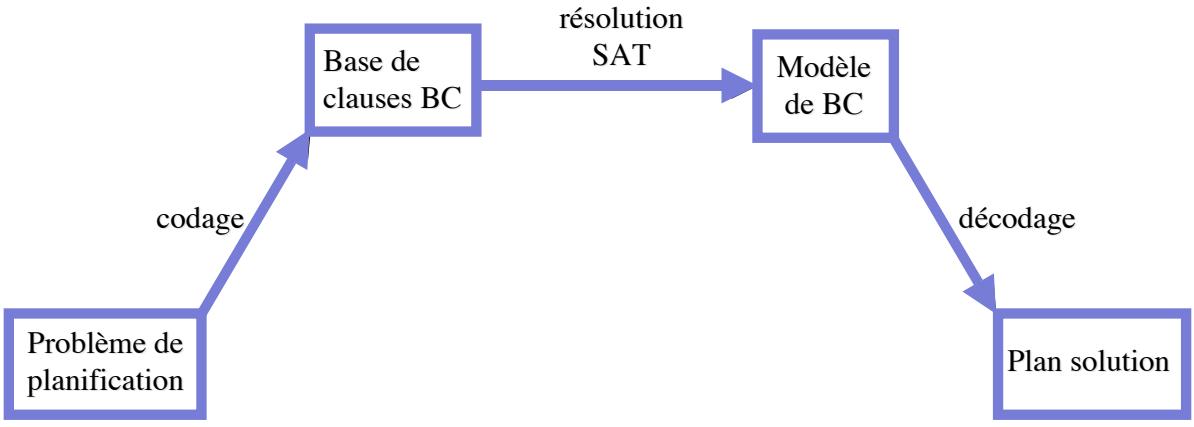
3.2. Algorithmes essentiels

TWEAK [Chapman, 1987]
SNLP [McAllester, Rosenblitt, 1991]
UCPOP [Penberthy, Weld, 1992]
REPOP [Nguyen, Kambhampati, 2001]
CPT [Vidal, Geffner, 2004]

Recherche dans les espaces de plans partiels

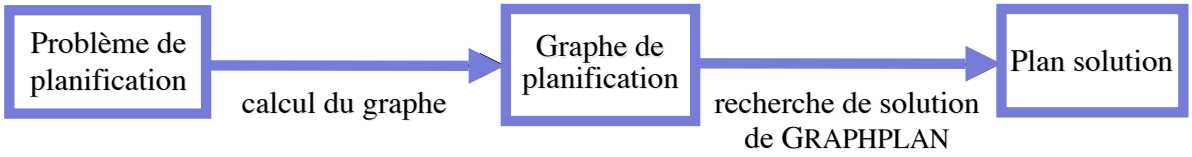
3.2. Algorithmes essentiels

SATPLAN [Kautz, Selman, 1992, 1996]
MEDIC [Ernst, Milstein, Weld, 1997]
TSP [Maris, Régnier, Vidal, 2002]



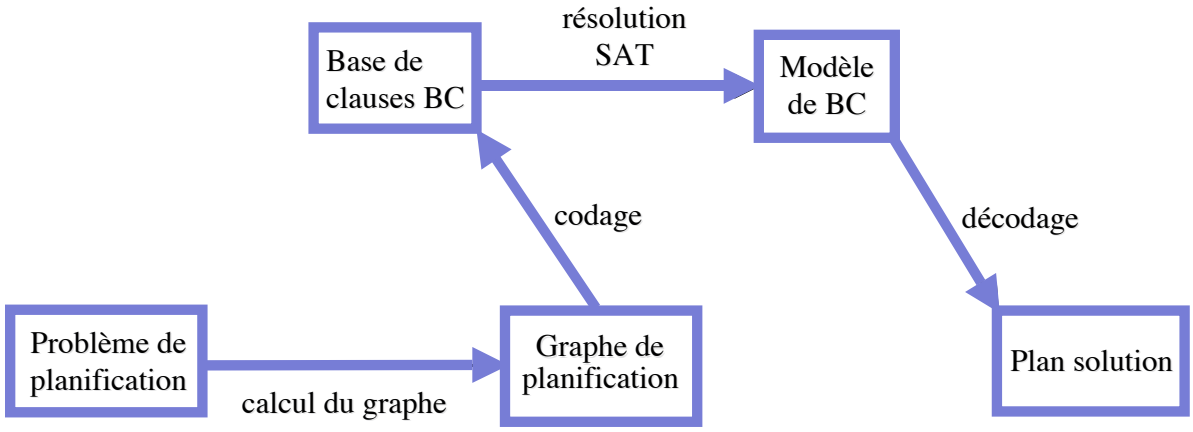
Compilation de plans : SATPLAN

GRAPHPLAN [Blum, Furst, 1995]
IPP [Koehler et col., 1997]
STAN [Long, Fox, 1999]
LCGP [Cayrol, Régnier, Vidal, 2000]



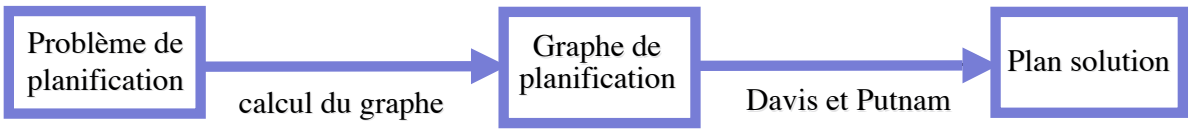
Compilation de plans : GRAPHPLAN

BLACKBOX [Kautz, Selman, 1999]
CSATPLAN [Baioletti, Marcugini, Milani, 1998]
TSP [Maris, Régnier, Vidal, 2002]



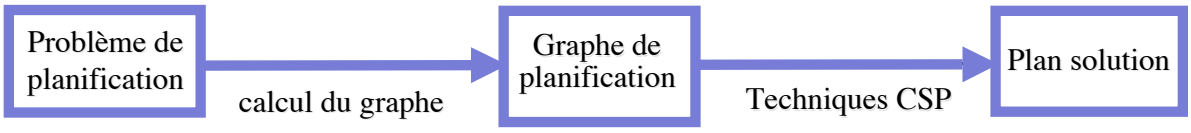
Compilation de plans : BLACKBOX

DPPLAN [Baioletti, Marcugini, Milani, 2000]
LCDPP [Vidal, 2001]



Compilation de plans : DPPLAN

GP-CSP [Do, Kambhampati, 2000]

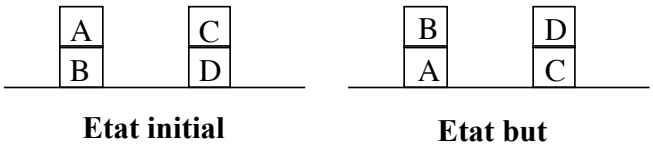


Compilation de plans : GP-CSP

3.3. Langages de représentation

- **Représentation logique des états et du but** (domaine des cubes sans pince) :

```
État init = {sur(A,B), surTable(B), sur(C,D), surTable(D),libre(A), libre(C)}
But = {sur(B, A), surTable(A), sur(D, C), surTable(C)}
```



Etat initial **Etat but**

- **Représentation STRIPS des opérateurs :** traitement du problème du décor
  Les faits non mentionnés en ajouts ou retraits restent inchangés après exécution.

```
Poser-sur-cube(?x, ?y, ?z) :
   Prec = {sur(?x, ?y), libre(?x), libre(?z)}
   Add = {sur(?x, ?z), libre(?y)}
   Del = {sur(?x, ?y), libre(?z)}
```

Poser-sur-table (?x)

3.3. Langages de représentation

- **Représentation logique des états et du but** (domaine du singe et des bananes) :

```
État init = {loc(singe,salon),loc(couteau,cuisine), loc(verre,cuisine),
             loc(bananes, garde-manger), loc(eau, garde-manger),
             pièce(salon), pièce(cuisine), pièce(garde-manger)}
But = {possède(singe, bananes), loc(singe, cuisine)}
```

- **Représentation STRIPS des opérateurs :**

```
Aller-à :  % le singe se déplace d'une pièce à l'autre %
   Nom(aller-à) = aller-à(?l1, ?l2)
   Prec(aller-à) = { loc(singe, ?l1), pièce(?l2), ¬=(?l1,?l2) }
   Add(aller-à) = { loc(singe, ?l2) }
   Del(aller-à) = { loc(singe, ?l1) }
```

Prendre-verre(?l) : % le singe prend le verre en ?l %

Prendre-couteau(?l) ; Prendre-bananes(?l) ; Prendre-eau(?l)

- **ADL (Action Description Language) [Pednault, 1989] :**

  Sous-ensemble de la logique du premier ordre. Un **opérateur** o est représenté par son nom et un doublet <pr, ef>. Ajouts et retraits sont regroupés dans une liste d'effets (ajouts : littéraux positifs, retraits : littéraux négatifs). Prec(o), Eff(o) dénotent respectivement les ensembles pr et ef de l'opérateur o. ADL permet l'utilisation de connecteurs logiques et de quantificateurs :

  – dans Pre(o) et Eff(o), ∧ représente une conjonction de formules ;

  – dans Eff(o), → dénote une implication, elle permet de représenter un effet conditionnel ;

  – dans Pre(o) et dans les antécédent des effets conditionnels, ∨ représente une disjonction de formules, il permet de représenter une précondition disjonctive ;

  – dans Pre(o) et Eff(o), les quantificateurs ∀ et ∃ représentent respectivement la quantification universelle et la quantification existentielle.

- **ADL, domaine des cubes**

```
Poser-sur :  % prendre cube ?x qui est sur cube ?y,le poser sur ?z %
     Nom(poser-sur) = poser-sur(?x, ?y, ?z)
     Pre(poser-sur) = sur(?x, ?y) ∧ libre(?x) ∧ libre(?z) ∧ ≠(?x, ?z) ∧
              ≠ (?y, ?z)
     Eff(poser-sur) = sur(?x, ?z) ∧ ¬sur(?x, ?y) ∧
             (≠(?y, Table) → libre(?y)) ∧ (≠(?z, Table) → ¬libre(?z))
```

- **ADL, domaine du singe et des bananes**

```
Aller-à :  % permet au singe de se déplacer d'une pièce à l'autre %
     Nom(aller-à) = aller-à(?l1, ?l2)
     Pre(aller-à) = loc(singe, ?l1) ∧ pièce(?l2) ∧ ¬=(?l1, ?l2)
     Eff(aller-à) = loc(singe, ?l2) ∧ ¬loc(singe, ?l1)

Prendre-ustensiles :  % le singe prend tous les ustensiles dans la pièce %
     Nom(prendre-ustensiles) = prendre-ustensiles(?l)
     Pre(prendre-ustensiles) = loc(singe, ?l) ∧ ∃ustensile x (loc(x, ?l))
     Eff(prendre-ustensiles) = ∀ustensile x (loc(x, ?l) → (possède(singe, x) ∧
             ¬loc(x, ?l)))
```

- **PDDL (Planning Domain Description Language) [Penberthy, Weld, 1992] :**

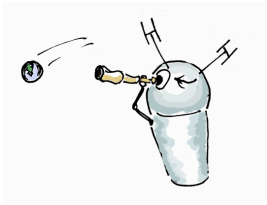
  PDDL+ (5 niveaux d'expressivité croissante : durées, effets dépendants du temps, ressources continues…) ; PDDL 2.1 [Fox, Long, 2002] regroupe les trois premiers niveaux. Actuellement, on en est à la version PDDL 3.

  – typage,
  – contraintes d'égalité,
  – effets conditionnels,
  – préconditions disjonctives,
  – quantification universelle,
  – mise à jour des variables d'état…

- **PDDL, domaine des cubes :**

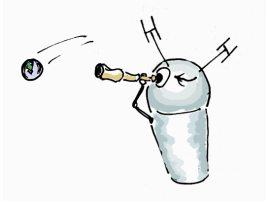
```
(define (domain blocksword)
        (: requirements :strips :equality :conditional-effects)
        (: predicates (on ?x ?y) (clear ?x))
        (: action puton
                : parameters (?x ?y ?z)
                : precondition
                        (and    (on ?x ?y)(clear ?x)(clear ?z)
                                (not (= ?y ?z)) (not (= ?x ?y))
                                (not (= ?x ?z)) (not (= ?x Table)))
                : effect
                        (and    (on ?x ?z) (not (on ?x ?y))
                                (when (not (eq ?y Table)) (clear ?y))
                                (when (not (eq ?z Table)) (not (clear ?z)))))
```

3.3. Langages de représentation



- **PDDL, domaine satellite :** observations par de multiples satellites équipés d'instruments différents
  - version **Strips** : relativement simple ;
  - version **numérique** : gestion de l'énergie (ressource consommable), capacité de stockage d'information limitée (plans pour acquérir toute l'info avec minimum d'énergie) ;
  - **temporelle simple** : utilisation simultanée possible de plusieurs satellites pour acquérir les informations recherchées (plans pour acquérir l'information au plus vite) ;
  - **temporelle** : temps critiques pour l'acquisition, temps de calibration différents…
  - **complexe** : version temporelle combinée avec la version numérique.

3.3. Langages de représentation

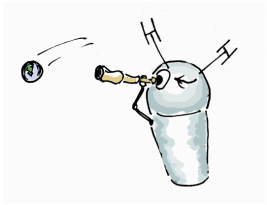


- **PDDL, domaine satellite :** observations par de multiples satellites équipés d'instruments différents

```
(define (domain satellite)
   (:requirements :strips :equality :typing)
   (:types satellite direction instrument mode)
   (:predicates
       (on-board ?i - instrument ?s - satellite)
       (supports ?i - instrument ?m - mode)
       (pointing ?s - satellite ?d - direction)
       (power-avail ?s - satellite)
       (power-on ?i - instrument)
       (calibrated ?i - instrument)
       (have-image ?d - direction ?m - mode)
       (calibration-target ?i - instrument ?d - direction))
```

© Régnier, IRIT 2006

3.3. Langages de représentation



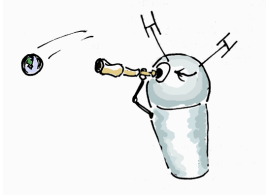
- **PDDL, domaine satellite :** observations par de multiples satellites équipés d'instruments différents

```
(:action turn-to
    :parameters    (?s - satellite
                    ?d-new - direction
                    ?d-prev - direction)
    :precondition (and (pointing ?s ?d-prev)
                       (not (= ?d-new ?d-prev)))
    :effect        (and (pointing ?s ?d-new)
                       (not (pointing ?s ?d-prev))))


(:action switch-on
    :parameters    (?i - instrument
                    ?s - satellite)
    :precondition (and (on-board ?i ?s)
                       (power-avail ?s))
    :effect        (and (power-on ?i)
                       (not (calibrated ?i))
                       (not (power-avail ?s))))…
```


UNIVERSITE PAUL SABATIER TOULOUSE III

3.3. Langages de représentation



- **PDDL, domaine satellite :** observations par de multiples satellites équipés d'instruments différents

```
(:durative-action turn_to
     :parameters (?s - satellite
                  ?d-new - direction
                  ?d-prev - direction)
     :duration   (= ?duration (slew-time ?d-prev ?d-new))
     :condition  (and (at start (pointing ?s ?d-prev))
                      (over all (not (= ?d-new ?d-prev))))
     :effect     (and (at end (pointing ?s ?d_new))
                      (at start (not (pointing ?s ?d-prev)))))

(:action turn_to
     :parameters    (?s - satellite
                     ?d-new - direction
                     ?d-prev - direction)
     :precondition (and (pointing ?s ?d-prev)
                        (not (= ?d_-new ?d-prev))
                        (>= (fuel ?s)(slew-time ?d-new ?d-prev)))
     :effect (and (pointing ?s ?d-new)
                  (not (pointing ?s ?d-prev))
                  (decrease (fuel ?s) (slew-time ?d-new ?d-prev))
                  (increase (fuel-used)(slew-time ?d-new ?d_prev))))
```

© Régnier, IRIT 2006