

# Planification

## GI313 - Optimisation

Charles Lesire-Cabaniols (ONERA / DCSD)  
charles.lesire@onera.fr

3A-SEM - 2010-2011

Introduction

Représentations

Planification dans l'espace d'état

Allez plus loin en planification

BE

## Introduction

Problèmes de planification

Modèles de planification

## Représentations

## Planification dans l'espace d'état

## Allez plus loin en planification

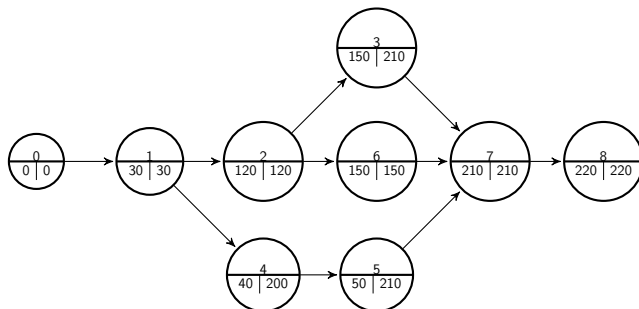
## BE

# Qu'est-ce que planifier ?

## ► Planning : Analyse, vérification d'un plan

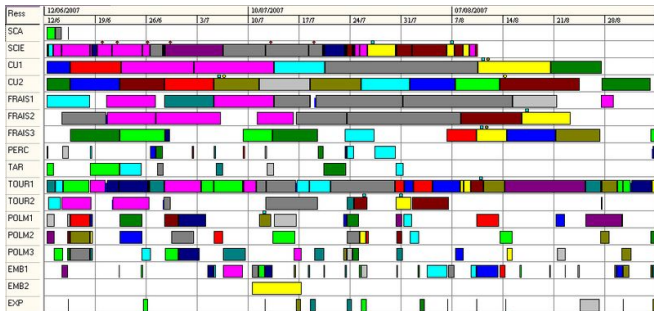
- on connaît les actions, leur organisation, leurs ressources
- on vérifie les contraintes, on obtient les chemins critiques

ex : élaboration interactive de plans, réseaux PERT...



# Qu'est-ce que planifier ?

- **Ordonnancement** : Organisation d'un plan
    - on connaît les actions à faire
    - on cherche leur organisation, les ressources à allouer
- ex : ordonnancement, gestion de ressources. . .



# Qu'est-ce que planifier ?

- ▶ **Planification** : Synthèse d'un plan
    - ▶ on connaît les buts à satisfaire, les actions possibles
    - ▶ on cherche leur les actions à faire pour atteindre ces buts, leur organisation, les ressources à allouer
- ex : gestion de l'activité d'un système autonome. . .

# Problèmes de planification

- ▶ Ingrédients :
  - ▶ Modèle de l'environnement
  - ▶ Modèles des actions possibles
  - ▶ Spécification des objectifs (buts, critères)
  - ▶ Données sensorielles sur l'état initial et l'état courant (si utilisation en ligne)
- ▶ Diverses formes selon :
  - ▶ Le type de tâches à planifier
  - ▶ La nature des modèles

## Modèle de la planification

Système Etat-Transition  $\Sigma = (S, A, E, \gamma)$

$S$  ensemble dénombrable d'états

$A$  ensemble fini de symboles d'actions

$E$  ensemble fini de symboles d'événements

$\gamma$  fonction de transition d'états

$$\gamma : S \times (A \cup E) \rightarrow 2^S$$

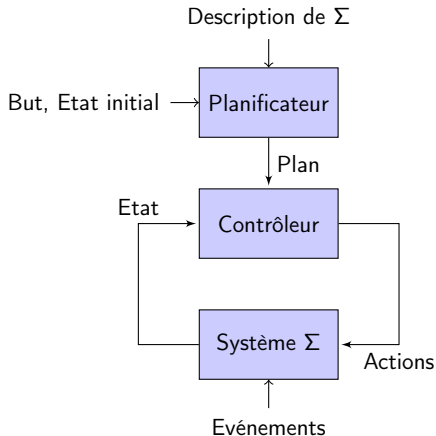
- ▶ si  $u \in E$ ,  $\gamma(s, u)$  transitions contingentes
- ▶ si  $u \in A$ ,  $\gamma(s, u)$  transitions contrôlées

## Problème de planification

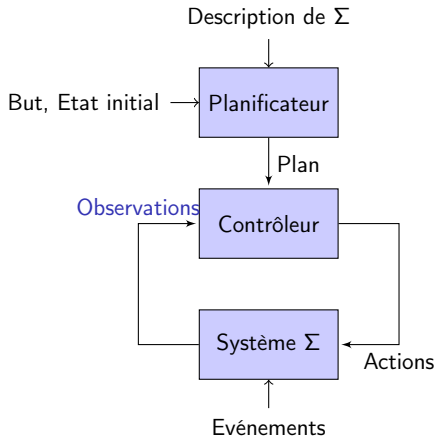
Quelles actions appliquer dans quels états en vue de réaliser des objectifs ?



# Modèle de la planification



# Modèle de la planification



# Modèle de la planification

- Plan :

$$\pi : S' \subset S \rightarrow A$$

- Observations :

- $O$  ensemble fini d'observations
- Fonction d'observation  $\eta : S \rightarrow O$

- Plan :

$$\pi : O' \subset O \rightarrow A$$

# Modèle de la planification

- Hypothèses classiques :
  1.  $\Sigma$  fini (monde fermé)
  2.  $\Sigma$  observable ( $\eta = I$ )
  3.  $\Sigma$  déterministe ( $|\gamma(s, u)| \leq 1$ )
  4.  $\Sigma$  statique
  5. Buts = états explicites
  6. Temps implicite
  7. Traitement hors-ligne

# Modèle de la planification

- ▶ Hypothèses classiques :
  1.  $\Sigma$  fini (monde fermé)
  2.  $\Sigma$  observable ( $\eta = I$ )
  3.  $\Sigma$  déterministe ( $|\gamma(s, u)| \leq 1$ )
  4.  $\Sigma$  statique
  5. Buts = états explicites
  6. Temps implicite
  7. Traitement hors-ligne
- ▶  $2 + 3 \Rightarrow$  contrôle en boucle ouverte

# Formes de planification

- ▶ Planification de mouvements
  - ▶ Trajectoires géométriques en 3D, lois de commande le long des trajectoires
- ▶ Planification de la perception
  - ▶ Quelle information est requise ?
  - ▶ Quand ? Où ? Comment ? Pour quoi ?
- ▶ Planification de tâches de manipulation
  - ▶ Primitives sensori-motrices, utilisant les forces, la vision...

# Formes de planification

- ▶ Planification de la communication
  - ▶ Interaction homme-robot
  - ▶ Coopération multi-robots
  - ▶ Quelles requêtes, comment, quels retours ?
- ▶ Planification de tâches générique
  - ▶ Modèles et algorithmes généraux à plusieurs types de problèmes

## Introduction

## Représentations

- Planification de tâches

- Représentation graphique

- Langage de représentation

## Planification dans l'espace d'état

## Allez plus loin en planification

## BE



# Planification de tâches

- ▶ Définition :
  - ▶ Synthèse d'une trajectoire abstraite dans un **espace de recherche**
  - ▶ Pour choisir et organiser des actions en **prédisant** leurs effets
  - ▶ En vue de **satisfaire un but** ou un critère
- ▶ Ingrédients :
  - ▶ Description des états du monde et des buts
  - ▶ Description des actions

# Planification de tâches

► Système Etat-Transition  $\Sigma = (S, A, \gamma)$

$S$  ensemble fini d'états

$A$  ensemble fini de symboles d'actions

$\gamma$  fonction de transition d'états

$$\gamma : S \times A \rightarrow A$$

# Représentation graphique

- ▶ Exemple des robots-dockers
  - ▶  $N$  sites
  - ▶  $K$  conteneurs à déplacer entre ces sites
  - ▶  $P$  piles réparties sur ces sites
  - ▶  $R$  robots pouvant acheminer ces conteneurs
- ▶ Actions :
  - ▶ **move** un robot  $r$  se déplace de  $l$  à  $l'$
  - ▶ **load** un robot  $r$  charge un conteneur  $k$  porté par un bras  $c$
  - ▶ **unload**
  - ▶ **take** un bras  $c$  saisit un conteneur  $k$  sur le sommet d'une pile  $p$
  - ▶ **put**

# Représentation graphique

- ▶ Complexité
  - ▶ Nombre d'états possibles :  $\mathcal{O}(n^r p^k k!)$
  - ▶  $n = 5, p = 15, r = 3, k = 100 \Rightarrow \sim 10^{277}$  états !!
- ▶ Impossible de construire  $\Sigma$  explicitement (et donc d'appliquer des techniques de recherche de chemin dans des graphes)

# Langage de représentation

- ▶ Langage de représentation des états et des actions
- ▶ Hypothèses classiques :
  - ▶ transitions instantannées, pas de durée, pas de parallélisme
  - ▶ monde statique, pas de transitions contingentes
  - ▶ connaissance complète
  - ▶ actions déterministes
- ▶ Formule : conjonction de littéraux
- ▶ Hypothèse du monde clos : ce qui n'est pas explicitement affirmé est faux

# Langage de représentation

```
(define (domain dock-worker-robot)
  (:requirements :strips :typing)
  (:types
    location ; several connected locations
    pile ; attached to location holds a pallet and a stack of containers
    robot ; holds at most 1 container, only 1 robot per location
    crane ; belongs to a location to pickup containres
    container
  )
  (:predicates
    (adjacent ?l1 ?l2 - location) ; l1 is adjacent to l2
    (attached ?p - pile ?l - location) ; p attached to l
    (belong ?c - crane ?l - location) ; c belongs to l
    (at ?r - robot ?l - location) ; r is at l
    (occupied ?l - location) ; there is a robot at l
    (loaded ?r - robot ?k - container) ; r loaded with container k
    (unloaded ?r - robot) ; r is empty
    (holding ?c - crane ?k - container) ; c is holding k
    (empty ?c - crane) ; c is empty
    (in ?k - container ?p - pile) ; k is within p
    (top ?k - container ?p - pile) ; k is on top of p
    (on ?k1 ?k2 - container) ; k1 is on k2
  )
)
```

# Langage de représentation

```
(:action move
:parameters (?r – robot ?from ?to – location)
:precondition (and (adjacent ?from ?to) (at ?r ?from) (not (occupied ?to)))
:effect (and (at ?r ?to) (not (occupied ?from)) (occupied ?to)
(not (at ?r ?from))))

(: action load
:parameters (?c – crane ?k – container ?r – robot)
:vars (?l – location)
:precondition (and (at ?r ?l) (belong ?c ?l) (holding ?c ?k) (unloaded ?r))
:effect (and (loaded ?r ?k) (not (unloaded ?r)) (empty ?c)
(not (holding ?c ?k))))

(: action unload
:parameters (?c – crane ?k – container ?r – robot)
:vars (?l – location)
:precondition (and (at ?r ?l) (belong ?c ?l) (loaded ?r ?k) (empty ?c))
:effect (and (unloaded ?r) (holding ?c ?k) (not (loaded ?r ?k))
(not (empty ?c))))
```

# Langage de représentation

```
(: action take
:parameters (?c – crane ?k – container ?p – pile)
:vars (?l – location ?else container)
:precondition (and (belong ?c ?l) (attached ?p ?l) (empty ?c)
  (in ?k ?p) (top ?k ?p) (on ?k ?else))
:effect (and (holding ?c ?k) (top ?else ?p) (not (in ?k ?p))
  (not (top ?k ?p)) (not (on ?k ?else)) (not (empty ?c))))

(: action put
:parameters (?c – crane ?k – container ?p – pile)
:vars (?l – location ?else container)
:precondition (and (belong ?c ?l) (attached ?p ?l)
  (holding ?c ?k) (top ?else ?p))
:effect (and (in ?k ?p) (top ?k ?p) (on ?k ?else)
  (not (top ?else ?p)) (not (holding ?c ?k)) (empty ?c))))
```



# Langage de représentation

```
(define (problem dwrbp1)
  (:domain dock-worker-robot)
  (:objects
    r - robot
    l1 l2 - location
    c1 c2 - crane
    p1 q1 p2 q2 - pile
    a b c d e f pallet - container)
  (:init
    (adjacent l1 l2) (adjacent l2 l1)
    (attached p1 l1) (attached q1 l1) (attached p2 l2) (attached q2 l2)
    (belong c1 l1) (belong c2 l2)
    (in a p1) (in b p1) (in c p1)
    (in d q1) (in e q1) (in f q1)
    (on a pallet) (on b a) (on c b)
    (on d pallet) (on e d) (on f e)
    (top c p1) (top f q1) (top pallet p2) (top pallet q2)
    (at r l1) (unloaded r) (occupied l1)
    (empty c1) (empty c2))
  (:goal
    (and (in a p2) (in b p2) (in c p2)
          (in d q2) (in e q2) (in f q2))))
```

# Langage de représentation

- ▶ forall ( ?x - type ) : boucle sur les éléments d'un type
- ▶ when cond effect : applique l'effet lorsque la condition est vraie

## Introduction

## Représentations

## Planification dans l'espace d'état

- Espace d'état

- Recherche en avant

- Heuristiques

- Recherche en arrière

## Allez plus loin en planification

## BE

# Transitions

- Calcul progressif :  $\text{result}(a, s)$

$$s \models \text{precond}(a) \Rightarrow s' = (s - \text{effets}^-(a)) \cup \text{effets}^+(a)$$

- Calcul inverse :  $\text{regress}(\gamma, a)$

$$\gamma \cap \text{effets}^-(a) = \emptyset \Rightarrow \text{regress} = \text{precond}(a) \cup (\gamma - \text{effets}^+(a))$$

# Forward

*Forward*( $S, S_g, path$ )

if  $s \models S_g$  then

    return path

else

    applicables  $\leftarrow \{a \in A / s \models precondition(a)\}$

    if applicables =  $\emptyset$  then

        return FAIL

    else

        Choose  $a \in$  applicables

        return *Forward*(*result*( $a, s$ ),  $S_g, path.a$ )

    end if

end if

# Forward

- ▶  $Forward(s_0, S_g, \emptyset)$
- ▶ Algorithme simple
- ▶ Algorithme complet
- ▶ Méthode **Choose** permet de guider la recherche :
  - ▶ en largeur (Breath-First Search)
  - ▶ en profondeur (Depth-First Search)
  - ▶ en prenant en compte le coût des actions (Best-First Search)
  - ▶ guidée par une heuristique

$A^*$ 

**Require :**  $G = (S, E)$  le graphe implicite,  $s_0$  état initial,  $S_g$  but

$\forall s \in S, g(s) \leftarrow \infty, p(s) \leftarrow s \quad c(s_0) \leftarrow 0, \mathcal{O} \leftarrow \{s_0\}$

**while**  $\mathcal{O} \neq \emptyset$  **do**

$x \leftarrow \operatorname{argmax}_{\operatorname{argmin} g(i)+h(i)} g(i)$

**if**  $x \models S_g$  **then**

**return** SUCCESS

**end if**

$\mathcal{O} \leftarrow \mathcal{O} / \{x\}$

**for all**  $y \in S / (x, y) \in E$  **do**

**if**  $g(y) > g(x) + k(x, y)$  **then**

$g(y) \leftarrow g(x) + k(x, y)$

$p(y) \leftarrow x$

$\mathcal{O} \leftarrow \mathcal{O} \cup \{y\}$

**end if**

**end for**

**end while**

$A^*$ 

- ▶ Si  $G$  est fini, l'algorithme **termine**
- ▶ Si  $h$  est minorante, l'algorithme est **complet** et **optimal**

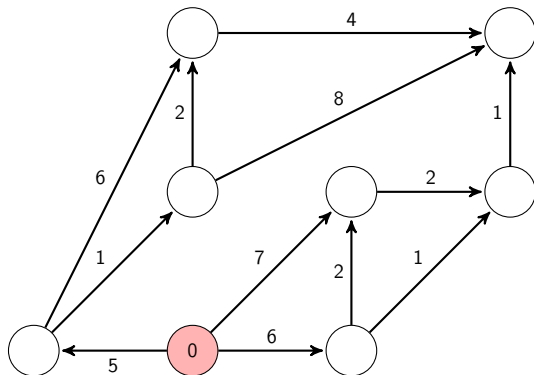
$$h \text{ minorante} \Leftrightarrow \forall s \in S, h(s) \leq g(s)$$

- ▶ Complexité :

- ▶  $h$  minorante :  $\mathcal{O}(N^2)$
- ▶  $h$  monotone :  $\mathcal{O}(N)$

$$h \text{ monotone} \Leftrightarrow \forall (u, v) \in E, h(u) - h(v) \leq k(u, v)$$



$A^*$ 

# Heuristiques

- ▶ **Relaxation** : on simplifie le problème pour estimer les coûts
- ▶ Compromis entre temps de calcul et qualité de l'heuristique
- ▶ Voyageur de commerce :
  - ▶  $h_1 = d(r, s)$  : distance à la ville suivante
  - ▶  $h_2 = d(r, s) + \sum_i \min_x d(x, s_i)$  : on minimise les distances aux villes restantes
  - ▶  $h_3 = d(r, s) + \text{coût d'un arbre de recouvrement minimal}$

# Heuristiques

## Relaxation

- ▶ Ne prendre en compte que les *effets*<sup>+</sup>
  - ▶ admissible, mais difficile à calculer
- ▶ Supprimer l'indépendance des sous-buts

# Heuristiques

## Relaxation

- Coût pour réaliser  $p$  à partir de  $s$  :

$$h(p, s) = \begin{cases} 0 & \text{si } s \models p \\ \min_a (1 + h(\text{precond}(a), s)) & \text{sinon} \end{cases}$$

- Heuristiques possibles :
  - Somme des coûts des littéraux

$$h(s) = \sum_{p \in S_g} h(p, s)$$

- Litéral le plus coûteux

$$h(s) = \max_{p \in S_g} h(p, s)$$

# Backward

*Backward*( $s_0, \gamma, path$ )

if  $s_0 \models \gamma$  then

    return path

else

    Choose  $g \in \gamma$

    relevant  $\leftarrow \{a \in A \mid effects(a) \models g\}$

    if relevant =  $\emptyset$  then

        return FAIL

    else

        Choose  $a \in$  relevant

        return *Forward*( $s_0, regress(\gamma, a), a.path$ )

    end if

end if

## Introduction

## Représentations

## Planification dans l'espace d'état

## Allez plus loin en planification

- Planification dans l'espace des plans

- Planification disjonctive

- Planification hiérarchique

- Représentation du temps

BE

## Planification dans l'espace des plans

- ▶ L'espace de recherche n'est plus l'espace d'état mais l'espace des plans
- ▶ L'algorithme passe d'un plan à un autre en essayant de l'améliorer
- ▶ Au départ, seulement l'état initial et les buts sont dans le plan
- ▶ Le principe est de résoudre les défauts du plan en insérant des actions
- ▶ Moindre engagement (pas d'action inutile)
- ▶ PSP, POP

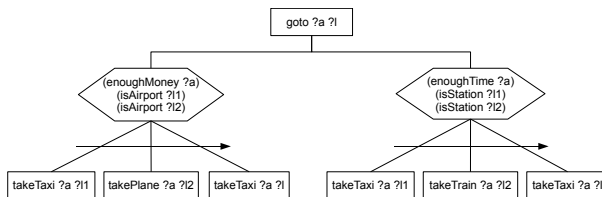
# Planification disjonctive

- ▶ Disjonction pas gérable par les méthodes précédentes
  - ▶ traitent un état/un but après l'autre
- ▶ GraphPlan
  - ▶ Accessibilité des littéraux buts
  - ▶ Phase de **construction** : élaboration descendante d'un P-graphe
  - ▶ Phase d'**extraction** : recherche ascendant à partir des buts



# Planification hiérarchique

- Structuration hiérarchique des actions
- Fournit une heuristique pour la recherche
- PSP, HTN, UMCP



# Traitement du temps

- ▶ Le temps est une ressource particulière
  - ▶ écoulement indépendant de l'action
  - ▶ disponible pour tous (parallélisme)
- ▶ Le temps est structuré par une relation transitive et asymétrique
  - ▶ il est irréversible
  - ▶ il ordonne la causalité

# Représentation du temps

- ▶ Représentation géométrique
- ▶ Représentation logique
  - ▶ argument d'un prédicat, en spécifiant les changements d'état
  - ▶ prédicats spécifiques pour les relations temporelles (algèbre d'Allen)
  - ▶ logiques temporisés (avec des valeurs numériques)

# Planification temporelle

- ▶ Programmation par contrainte avec des contraintes temporelles
- ▶ Planification dans l'espace des plans, plans temporels
  - ▶ IxTeT, RAX, parcPLAN

Introduction

Représentations

Planification dans l'espace d'état

Allez plus loin en planification

BE

Mission

Sujet ARD

Sujet Planification

# Mission

- ▶ Mission de recherche et d'extinction d'incendie
- ▶ Zone de missions découpée en sous-zones
- ▶ Exploration des sous-zones à la recherche d'un feu
- ▶ Extinction d'un feu si trouvé
- ▶ Retour à la base si pas de feu ou plus de cartouche d'extinction

## Sujet Application Robotique Dronique

- ▶ Développer l'architecture embarquée pour réaliser cette mission
- ▶ Environnement de développement en C++ à base de composants
- ▶ Composant de navigation (existants)
- ▶ Composant de détection (blob rouge sur image / à développer)
- ▶ Composant de cartographie (à développer)
- ▶ Composant de supervision (à développer)

# Sujet de planification

- ▶ Quelles actions réaliser pour accomplir la mission ?
  - ▶ Planification des actions, qui seront intégrées dans le superviseur
  - ▶ Algorithme FF (Fast Forward)
- 
1. Modélisation le domaine de planification (prédicats, actions)
  2. Modéliser un premier problème (en faisant des hypothèses sur l'environnement)
  3. Générer un plan
  4. Proposer une façon de rendre ce plan robuste aux défauts des hypothèses posées