IFT 608 / IFT 702
Planification en intelligence artificielle

Extraction automatique d'heuristiques à partir d'un modèle

Froduald Kabanza
Département d'informatique
Université de Sherbrooke

Contenu

- Graphe de planification
- Générer un plan à partir d'un graphe de planification
- Extraction automatique d'une heuristique à partir d'un graphe de planification

Modèle STRIPS ou PDDL

RAPPEL

Langage PDDL

- Un **problème** dans le langage PDDL est un tuple *P= (F, O, I, G)*
  - *F* est un ensemble de propositions *(variables booléenes)*
  - *O* est un ensemble d'opérateurs *(actions)*
  - $I \subseteq F$ est l'état initial *(conjonction de faits vrai)*
  - $G \subseteq F$ est le but *(conjonction de fait à rendre vrai)*

- Des **opérateurs PDDL** o $\in$ O représentés par:
  - La liste des préconditions *Pre(o)*) $\subseteq F$
  - La liste des effets positifs *Add(o)* $\subseteq F$
  - La liste des effets négatifs *Del(o)* $\subseteq F$
  - Optionnellement, la durée et le coût de l'action.

Transformation – Du langage au modèle

- Un **problème** $P = (F, O, I, G)$ détermine un **modèle d'états** $S(P)$ (avec des transitions entre les états) tel que:
  - Les états $s \in S$ sont des ensembles de propositions de F
  - Les états buts s sont tel que $G \subseteq s$
  - Les actions $a \in A(s)$ sont des opérateurs tel que $Pre(a) \subseteq s$
  - L'état successeur est s' = S- Del(a) + Add(a)
  - Par défaut les coûts $c(a,s)$ sont tous égaux à 1.
  - Par défaut les durées $d(a,s)$ sont tous égaux à 1.

- La **solution** (optimale) de P est la **solution** (optimale) de $S(P)$
- Plusieurs **aspects** de PDDL et ses extensions :
  - Patrons d'actions
  - Actions conditionnelles
  - Actions non-déterministes ou probabilistes

Comment résoudre un modèle PDDL

- Deux approches traditionnelles :
  - Exploration de l'espace d'états – que nous avons vu
  - Exploration de l'espace de plans – que nous ne verrons pas
- Deux rôles joués par le langage PDDL :
  - Expression concise du modèle des actions
  - Extraction automatique des heuristiques

Extraction d’heuristiques à partir d’un modèl

IDÉE DE BASE

Extraction d'heuristique à partir d'un modèle

- Une heuristique peut être **expliquée** comme étant une fonction de coût **optimale** pour un problème **relaxé** (simplifiée)
  (Minsky 19961; Pearl 1983; Kahneman, 2011).

- Une façon de relaxer un problème de planification *P* est de supprimer les *delete-lists* des modèles d'actions de *P*.

- Le problème ainsi relaxé est noté P^+.

- Si $c^*(P^+)$ est le coût de la solution optimale de *P*, alors $c^*(P^+)$ est une heuristique admissible pour P.

  - Dans un état s, on peut utiliser $h(s) = c^*(P^+)(s)$ (le coût de la solution optimale en partant de l'état initial s).

- Ce genre d'idée est exploitée par PlanSyS (intégré avec ROS) et FF

Relaxation d’un problème de planification

GRAPHE DE PLANIFICATION

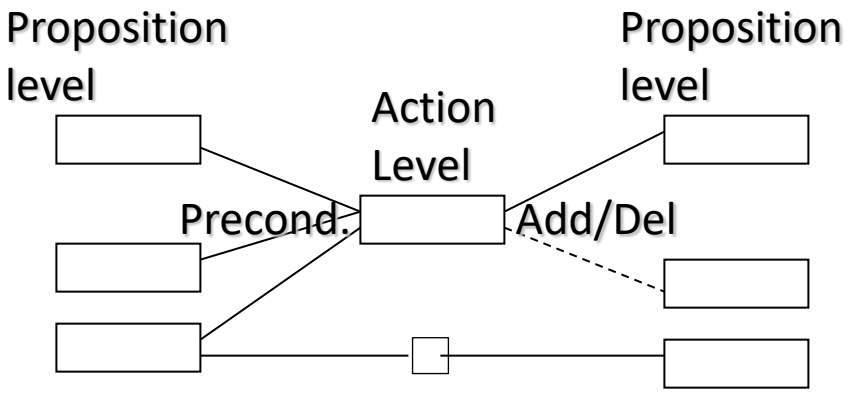
Graphe de planification - Idée

- Étant donné un problème P=(F, O, I, G), un algorithme de planification par exploration de l'espace d'états consiste à
  - Vérifier s'il existe une **séquence d'actions** permettant d'**atteindre** un état satisfaisant le but G à partir de l'état initial I.
- Un **graphe de planification** est une structure, représentant une **relaxation** du problème P et permettant d'**estimer efficacement** quel **ensemble de propositions est atteignable** par **quelles actions**.
- Efficacement parce que le graphe est de **taille polynomiale** par rapport à F et est construit en **temps polynomial** par rapport à F.
- Si on **relaxe le graphe de planification** à son tour (en **ignorant les delete-lists**), on obtient une **relaxation P+ du problème P permettant de calculer des heuristiques efficacement**.

Graphe de planification – Définition Formelle

- Étant donné un problème P=(F, O, I, G), le **graphe de planification pour P**, noté **P^+,** est un graphe constitué de:
  a) **Deux types de nœuds** (propositions, actions), organisés par **niveaux qui alternent**, et
  b) Un ensemble de paires de propositions ou d'actions mutuellement exclusives, appelé **mutex.**

- Le niveau proposition approxime les état atteignables par les actions du niveau précédent.
- Le niveau action approxime les actions permises (exécutables) dans les états approximés par le niveau précédent.
- Arcs:
  - » Préconditions
  - » Effets positifs (ligne continue) et négatives (ligne pointillée)

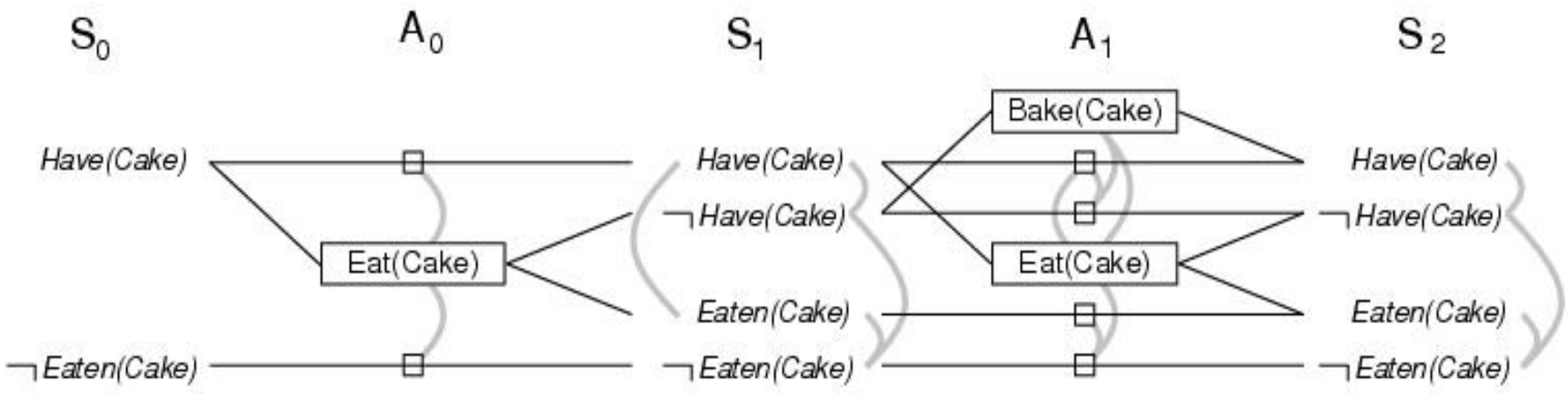


Algorithme de génération du graphe de planification

- Premier niveau : état initial
- Répétitivement, ajouter le niveau suivant:
  - Action : toutes les actions dont les préconditions sont satisfaites par le niveau proposition précédent.
  - Proposition (littéraux): Tous les effets des actions du niveau précédent et de l'action *no-op*.
- Construction de l'ensemble *mutex*: à venir ….

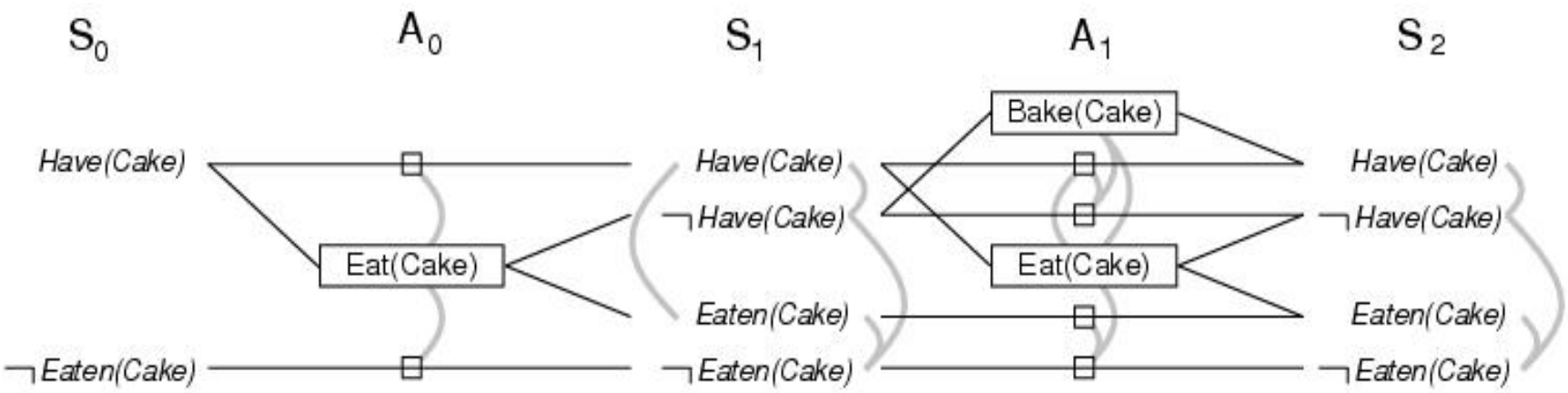
Exemple 1 [Russel and Norvig, 2009, Chapître 10]

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg Have(Cake)$
EFFECT: $Have(Cake)$



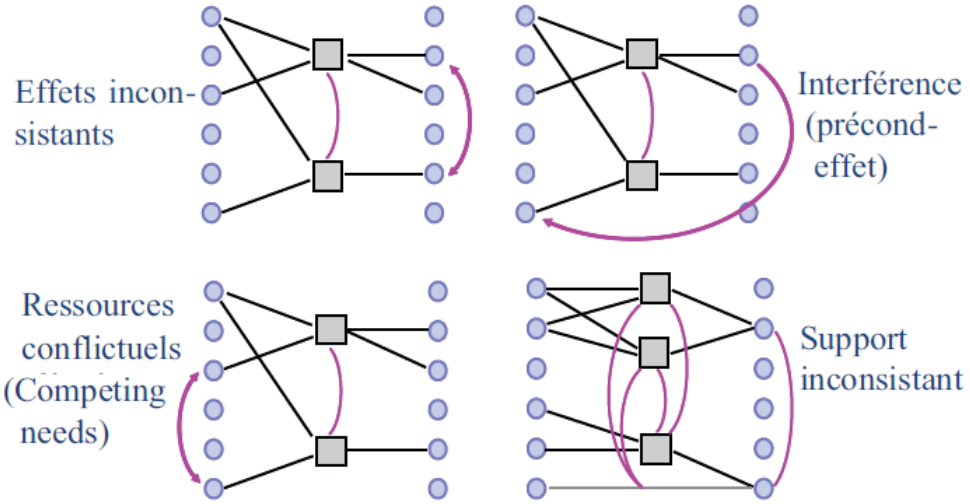
Construction de l'ensemble mutex

- Deux actions sont mutuellement exclusives (mutex) si:
  - Inconsistance *:* l'une nie l'effet d'une autre
  - Interférence*:* l'une supprime la précondition d'une autre
  - Ressources conflictuels : elles ont des préconditions mutex
- Deux propositions sont mutuellement exclusives (mutex) si:
  - L'une est la négation de l'autre ou
  - Support inconsistant: toutes les paires d'actions ayant ces propositions comme effets sont mutex.



Construction de l'ensemble mutex

- Deux actions sont mutuellement exclusives (mutex) si:
  - Effets inconsistants *:* l'une nie l'effet d'une autre
  - Interférence (précondition-effet) *:* l'une supprime la précondition d'une autre
  - Ressources conflictuels : elles ont des préconditions mutex
- Deux propositions sont mutuellement exclusives (mutex) si:
  - Inconsistance: l'une est la négation de l'autre ou
  - Support inconsistant: toutes les paires d'actions ayant ces propositions comme effets sont mutex.



Algorithme GRAPHPLAN: AIMA (IFT615) Page 383

function GRAPHPLAN(*problem*) **returns** solution or failure

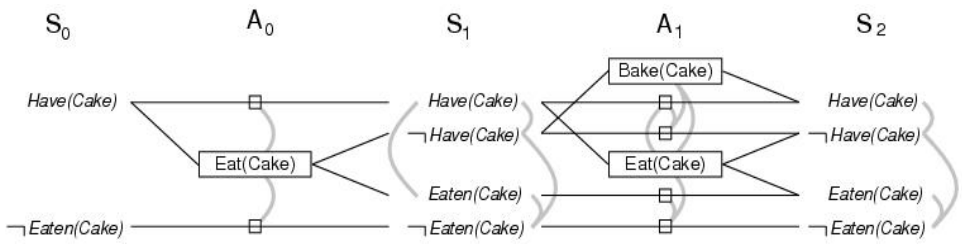
graph ← INITIAL-PLANNING-GRAPH(*problem*)
goals ← GOALS[*problem*]
loop do
  if *goals* all non-mutex in last level of *graph* **then do**
    solution ← EXTRACT-SOLUTION(*graph*, *goals*, LENGTH(*graph*))
    if *solution* ≠ *failure* **then return** *solution*
    else if NO-SOLUTION-POSSIBLE(*graph*) **then return** *failure*
  graph ← EXPAND-GRAPH(*graph*, *problem*)

Figure 11.13 The GRAPHPLAN algorithm. GRAPHPLAN alternates between a solution extraction step and a graph expansion step. EXTRACT-SOLUTION looks for whether a plan can be found, starting at the end and searching backwards. EXPAND-GRAPH adds the actions for the current level and the state literals for the next level.

Russel and Norvig. Artificial Intelligence : A Modern Approach, 2009, chapitre 10.

Algorithme Extract Solution

- Recherche à **rebours** (**backwards**)
  - **Niveau-par-niveau** pour mieux exploiter les mutex
  - **Pour chaque but** à l'itération *n*, **trouver une action** ayant ce but dans ses effets et **non mutex** avec une action déjà choisie
  - Les **préconditions** de ces actions deviennent les **(sous-)buts à l'étape n-1**
  - Trouver les **actions** ayant pour effets les (sous-)buts de l'étape n-1.
  - S'il n'y en a pas **Backtrack**
  - **Memoisation**



Exemple 2 [Russel and Norvig, 2009, Chapître 10]

$Init(At(Flat, Axle) \wedge At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
PRECOND: $At(Spare, Trunk)$
EFFECT: $\neg At(Spare, Trunk) \wedge At(Spare, Ground))$
$Action(Remove(Flat, Axle),$
PRECOND: $At(Flat, Axle)$
EFFECT: $\neg At(Flat, Axle) \wedge At(Flat, Ground))$
$Action(PutOn(Spare, Axle),$
PRECOND: $At(Spare, Ground) \wedge \neg At(Flat, Axle)$
EFFECT: $\neg At(Spare, Ground) \wedge At(Spare, Axle))$
$Action(LeaveOvernight,$
PRECOND:
EFFECT: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$
$\wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))$

Figure 11.7 The simple flat tire problem description.

Exemple 2 [Russel and Norvig, 2009, Chapître 10]

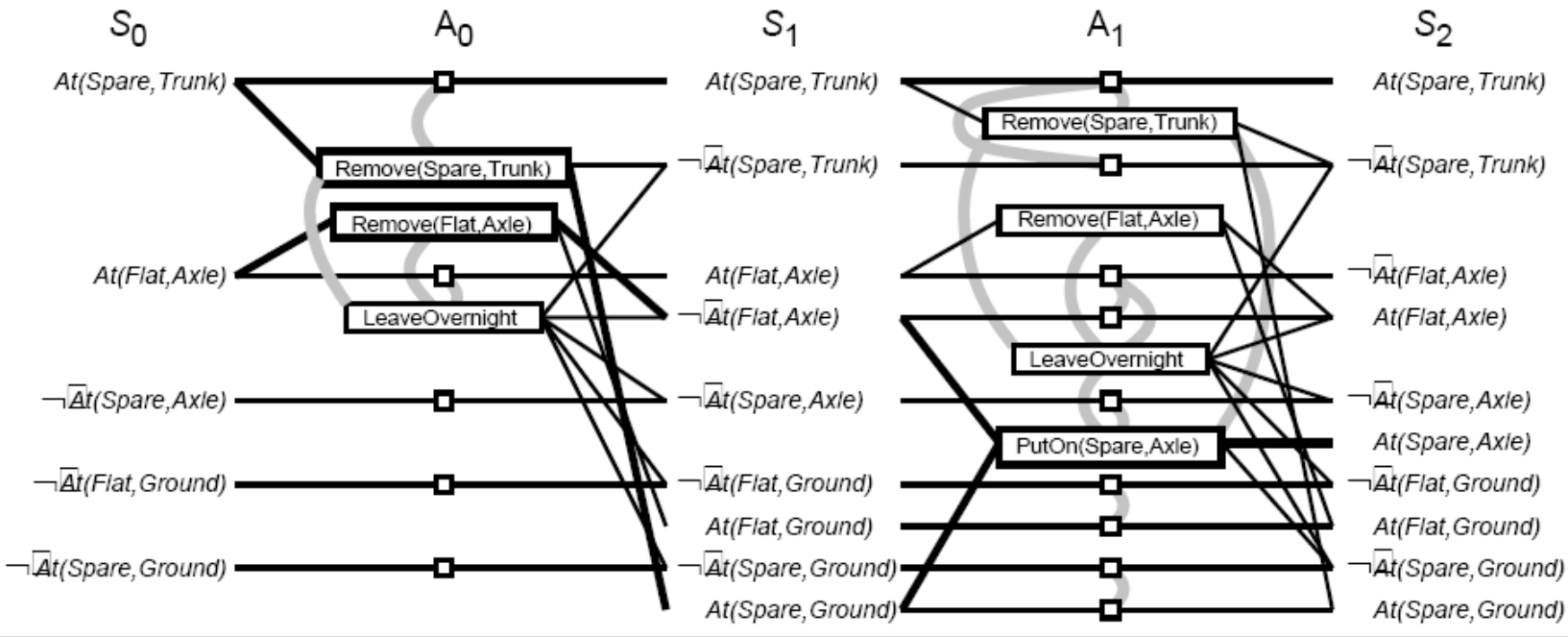


Figure 11.14 The planning graph for the spare tire problem after expansion to level S_2. Mutex links are shown as gray lines. Only some representative mutexes are shown, because the graph would be too cluttered if we showed them all. The solution is indicated by bold lines and outlines.

EXTRACTION D'HEURISTIQUES

Extraction d'heuristiques- Idée

- Pour extraire des heuristiques d'un problème *P=(F, O, I, G),* on relaxe le problème en **ignorant les delete-lists** dans les opérateurs *O*.

- On construit un **graphe de planification relaxé P^+** dans lequel on ignore les delete-lists

- **On n'a plus les mutex**, puisqu'il ne peut y avoir des conflits sans les delete-lists.

- L'estimation heuristique (***h***) du coût optimal pour arriver au but G à partir d'un état s est alors le premier niveau contenant toutes les propositions de G dans le graphe relaxé P^+ à partir de l'état S.

Formellement

Étant donné un *P=(F, O, I, G),* pour calculer h(s):

- Construit le **graphe de planification relaxé** P^+ avec les niveau P_0, A_0, A_1, …
  - $\mathbf{P_0 = \{p \in s\}}$
  - $\mathbf{A_i = \{a \in O \mid Pre(a) \subseteq P_i\}}$
  - $\mathbf{P_{i+1} = Pi\ U\ \{p \in Add(a) \mid a \in A_i\}}$
- P^+ **représente** implicitement l'heuristique $h_{max}(s) = \min i$ such that $G \subseteq Pi$
  - Approche utilisée par FF
- Il existe des heuristiques plus précises obtenues de P_+ de façon additive (plutôt que max) en faisant une recherche à rebours dans P^+.

Planification avec des balises, apprentissage d'heuristiques

AU-DELÀ DE L'EXTRACTION D'HEURISTIQUES

EHC, Actions Utiles, Balises

- Dans la formulation précédente de la planification par exploration de l'espace d'états, l'heuristique *h(s)* est utilisé comme une boite noire.

- Des planificateurs comme FF et LAMA vont plus loin.

- Elles exploitent la structures de l'heuristique et/ou du problème à résoudre:
  - Concept d'actions utiles
  - Concept de balises (propositions qui doivent être vraies dans **chaque** plan satisfaisant le but)
- Utilisent de nouveau algorithmes (mieux que A*)
  - Enforced Hill Climbing (EHC)
  - Multi-queue Best First Search
- Cela donne des planificateurs bien plus efficaces … jusqu'à un certain point.

Apprentissage d'heuristiques

- Référence 1:
  - Xu et al. Discriminative Learning of Beam-Search Heuristics for Planning. IJCAI 2007.
  - URL: http://www2.parc.com/isl/members/syoon/ijcai07-beam.pdf
- Référence 2: http://ipc.icaps-conference.org/ (Learning Track)
- À ne pas confondre avec l'apprentissage des modèles d'actions

Ce qu'il faut retenir

- L'idée de base, couverte ici, est que étant donné un problème P, on peut relaxer le problème pour définir une fonction heuristique pour une recherche dans un espace d'états.

- Le graphe de planification est un des concepts de base utilisé pour relaxer un problème.

  - Vous devez être capable d'expliquer comment une heuristique est définie à partir d'un graphe de planification relaxé en ignorant les *delete-lists*.

- Les approches récentes, non couvertes ici, vont au-delà en exploitant la structure du problème (exemple, les balises) et en utilisant de nouveaux algorithmes de recherche (exemple, *Enfonced Hill Climbing*).

Vous devriez être capable de …

- Définir ce qu'un graphe de planification
- Décrire le graphe de planification pour un problème donné
- Expliquer comment une plan est généré à partir d'un graphe de planification
- Expliquer comme est heuristique pour une recherche dans un espace d'états est défini à partir d'un graphe de planification relaxé en ignorant les *delete-lists*.