Unit 5: Planning

José Luis Ruiz Reina
Agustín Riscos

Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Inteligencia Artificial, 2012-13
Outline

Introduction
The PDDL formalism
Forward Search
Backward Search
Heuristics for planning
Partial Order Planning: POP
Planning in AI

- **Planning**: finding a sequence of actions that enable to reach a given goal when executed from a given initial state.
- **Plan**: sequence of actions that achieve the goal
- **Real-world applications:**
  - Robotics
  - Manufacturing by components assembly
  - Space missions

**Planning vs Scheduling**

- *Planning* focuses on finding a sequence of actions to achieve a goal and *Scheduling* emphasizes the efficient assignment of available resources to the actions
Planning problems in AI

• (Some) Issues to address:
  • Representation of the world and the actions transforming it
  • Algorithms for searching plans
  • Minimizing the resources consumed by the plan
  • Time when each action is executed
  • Monitorize the execution of the plan, including revisions in case of errors or contingencies

• For the sake of simplicity, along the unit we assume:
  • A finite number of states
  • Fully observable
  • Deterministic actions, precisely defined by their specification
  • Implicit time: actions are performed without duration
  • Planning is done a priori
  • Closed world hypothesis
Planning and search in states space

- Why search in states space is not enough:
  - Description of the real-world states is extremely complex
  - Huge variety of possible actions, most of them irrelevant for achieving the final goal
  - Actions only have effect on a small piece of the world (*frame problem*)
  - Heuristics *independent* of the domain are needed
  - An action can be judged necessary without having decided yet all the previous actions (*minimum commitment*)
  - It is often advisable to *decompose* into simpler subproblems

- Idea: use logic to represent states, actions and goals; and use algorithms handling this representation
Outline

Introduction

The PDDL formalism

Forward Search

Backward Search

Heuristics for planning

Partial Order Planning: POP
Example: the blocks world

- Classical example of planning domain.
- Elements involved:
  - A table.
  - A set of cube-shaped blocks.
  - A robot arm, able to hold one block at a time.
  - Each block can be either sitting on the table or piled on top of another block.
logic formalism: PDDL language

• A language to represent planning problems:
  • Constants: objects in the world (capitalized)
  • Variables that represent any object (lower case)
  • Predicate symbols (to express properties of objects)
  • Symbols to represent the actions

• Terminology:
  • Atoms: formulas of the form $P(o_1, \ldots, o_n)$, where $P$ is a predicate symbol and each $o_i$ is either a constant or a variable (no function symbols)
  • Literals: atoms or negation of atoms (symbol $-$ will be used as negation)
  • Closed atoms and literals: no variables on them

• States: conjunction of closed atoms
  • Closed world hypothesis: all atoms not explicitly stated is assumed to be false
Representation of states in the blocks world

- Description of a state:

```
C A
B
D
```

CLEAR(B), CLEAR(C), CLEAR(D), FREEARM(), ON(B,A), ONTHETABLE(C), ONTHETABLE(D), ONTHETABLE(A)

- Logic predicates used in this representation:

- CLEAR(x), block x is clear (nothing on top of it).
- FREEARM(), the arm is not holding any block.
- ONTHETABLE(x), block x is sitting on the table.
- ON(x,y), block x is sitting on top of y.
- HOLD(x), block x is being hold by the arm.
Representation of goals

- **Goals**: description of final states.
- **Goals** are represented as conjunction of literals (use of variables is allowed)
  - Variables appearing in goals are interpreted as existentially quantified
- **Satisfying (achieving) a goal**:
  - A state satisfies a goal if it is possible to replace the variables in the goal by constants in such a way that the state includes all positive literals of the *instantiated goal* and none of the negative
  - A state is a final state if it satisfies the goal
- **Important**: checking if a state satisfies a goal is merely a symbolic calculation
Example of goals in the blocks world

- Examples of goals:
  - \text{ON}(B,A), \text{ONTHETABLE}(A), \text{–ON}(C,B)
  - \text{ON}(x,A), \text{CLEAR}(x), \text{FREEARM}()
  - \text{ON}(x,A), \text{–ON}(C,x)
  - \text{ON}(x,A), \text{ON}(y,x)

The first three goals are satisfied by state 1 but not by state 2. The last one is satisfied by state 2 but not by state 1.
Description of actions schemes

• To deal with the *frame problem*, we only specify the *changes* produced by the action.

• An action is described by means of:
  
  • Its name and *all* the variables involved: $O(x_1, \ldots, x_n)$
  
  • Precondition: list of literals that need to be True in order for the action to be applicable.
  
  • Effects: list of literals indicating the changes to be produced when the action is executed.
Description of actions schemes

- In the effects list we distinguish:
  - Positive effects (or *addition* list): atoms that will start being True
  - Negative effects (or *erasing* list): atoms that will stop being True

- The use of variables implies that an action usually represents an *actions scheme*:
  - For each possible way of substituting variables by constants, we get a precise *action* (an *instance* of the scheme)
Example: actions in the blocks world

- Put a block on top of another:
  \[
  \text{PILE}(x, y) \\
  \text{Precond: } \neg\text{CLEAR}(y), \neg\text{HOLD}(x) \\
  \text{Effects: } -\text{CLEAR}(y), -\text{HOLD}(x),
  \text{FREEARM}(), \text{ON}(x, y), \text{CLEAR}(x)
  \]

- Lift a block which was on top of another:
  \[
  \text{UNPILE}(x, y) \\
  \text{Precond: } \text{ON}(x, y), \text{CLEAR}(x), \text{FREEARM}() \\
  \text{Effects: } -\text{ON}(x, y), -\text{CLEAR}(x), -\text{FREEARM}(),
  \text{HOLD}(x), \text{CLEAR}(y),
  \]

- Take a block from the table:
  \[
  \text{GRAB}(x) \\
  \text{Precond: } \text{CLEAR}(x), \text{ONTHETABLE}(x), \text{FREEARM}() \\
  \text{Effects: } -\text{CLEAR}(x), -\text{ONTHETABLE}(x), -\text{FREEARM}(),
  \text{HOLD}(x)
  \]

- Release a block on the table:
  \[
  \text{RELEASE}(x) \\
  \text{Precond: } \text{HOLD}(x) \\
  \text{Effects: } -\text{HOLD}(x),
  \text{ONTHETABLE}(x), \text{FREEARM}(), \text{CLEAR}(x)
  \]
Applicability of an action

- An action is *applicable* on a state if the latter satisfies its precondition
  - If there are variables in the precondition, then the applicability is defined *with respect to the substitution* $\theta$ used to satisfy the precondition
  - Sometimes, to simplify, the substitution will appear implicitly when referring to the action
  - For example, we write $\text{UNPILE}(A, B)$ to refer to the action $\text{UNPILE}(x, y)$ using the substitution $[x/A, y/B]$
  - On a given state, there could be several applicable actions coming from the same actions scheme
Result of applying an action

- The result of applying an applicable action (w.r.t. a substitution $\theta$) over a state $E$ is the state obtained by:
  - **Eliminate** from $E$ the atoms, instantiated by $\theta$, corresponding to the list of **negative** effects (if they were present)
  - **Add** to $E$ the atoms, instantiated by $\theta$, corresponding to the list of **positive** effects (if they were not present)
Example of application of an action (I)

* State before applying UNPILE(B,A):
E = \{CLEAR(B), CLEAR(C), CLEAR(D), FREEARM(),
      ON(B,A), ONTHETABLE(C), ONTHETABLE(D), ONTHETABLE(A)\}

* Preconditions of UNPILE(B,A):
Preconds = \{ON(B,A), CLEAR(B), FREEARM()\}

------- Conditions satisfied by the state -------
------- (thus action is applicable) -------

* Effects of UNPILE(B,A):
Effec− = \{ON(B,A), CLEAR(B), FREEARM()\}
Effec+ = \{HOLD(B), CLEAR(A)\}

* State after applying UNPILE(B,A):
E’ = (E − Effec−) U Effec+ =
\{CLEAR(C), CLEAR(D), ONTHETABLE(C),
  ONTHETABLE(D), ONTHETABLE(A), HOLD(B), CLEAR(A)\}
Example of application of an action (II)

* State before applying \text{RELEASE}(B):
  \( E = \{\text{CLEAR}(C), \text{CLEAR}(A), \text{CLEAR}(D), \text{ONTHETABLE}(C), \text{ONTHETABLE}(A), \text{ONTHETABLE}(D), \text{HOLD}(B)\} \)

* Preconditions of \text{RELEASE}(B):
  \( \text{Preconds} = \{\text{HOLD}(B)\} \)
  ------ Conditions satisfied by the state ------
  ------ (thus action is applicable) ------

* Effects of \text{RELEASE}(B):
  \( \text{Effec}^- = \{\text{HOLD}(B)\} \)
  \( \text{Effec}^+ = \{\text{ONTHETABLE}(B), \text{FREEARM}(), \text{CLEAR}(B)\} \)

* State after applying \text{RELEASE}(B):
  \( E' = (E - \text{Effec}^-) \cup \text{Effec}^+ = \{\text{CLEAR}(C), \text{CLEAR}(A), \text{CLEAR}(D), \text{ONTHETABLE}(C), \text{ONTHETABLE}(A), \text{ONTHETABLE}(D), \text{ONTHETABLE}(B), \text{FREEARM}(), \text{CLEAR}(B)\} \)
Plans and solutions

• Plan: (ordered) sequence of actions
  • First one applicable on the initial state and each one of the rest applicable on the result of the previous one

• Solution: plan that starting from the initial state obtains a state which satisfies the goal
Example: changing a flat tire

- **Language:**
  - Objects: SPARE-TIRE, FLAT-TIRE, AXLE, TRUNK, GROUND
  - Predicate: AT(−,−)

- **Initial state:**
  - AT(FLAT-TIRE, AXLE), AT(SPARSE-TIRE, TRUNK)

- **Final state:**
  - AT(SPARSE-TIRE, AXLE)
Actions for changing a flat tire

- **Take the spare tire out of the trunk:**
  
  \[
  \text{REMOVE} (\text{SPARE-\text{TIRE, TRUNK)})
  \]
  
  \[
  \text{Precond: AT} (\text{SPARE-\text{TIRE, TRUNK)})
  \]
  
  \[
  \text{Effects: AT} (\text{SPARE-\text{TIRE, GROUND}}), \quad \neg\text{AT} (\text{SPARE-\text{TIRE, TRUNK}})
  \]

- **Remove the flat tire from the axle:**
  
  \[
  \text{REMOVE} (\text{FLAT-\text{TIRE, AXLE}})
  \]
  
  \[
  \text{Precond: AT} (\text{FLAT-\text{TIRE, AXLE}})
  \]
  
  \[
  \text{Effects: } \neg\text{AT} (\text{FLAT-\text{TIRE, AXLE}}), \quad \text{AT} (\text{FLAT-\text{TIRE, GROUND}})
  \]

- **Put the spare tire on the axle:**
  
  \[
  \text{PUT-ON} (\text{SPARE-\text{TIRE, AXLE}})
  \]
  
  \[
  \text{Precond: } \neg\text{AT} (\text{FLAT-\text{TIRE, AXLE}}), \quad \text{AT} (\text{SPARE-\text{TIRE, GROUND}})
  \]
  
  \[
  \text{Effects: } \neg\text{AT} (\text{SPARE-\text{TIRE, GROUND}}), \quad \text{AT} (\text{SPARE-\text{TIRE, AXLE}})
  \]

- **Leave the car alone until next morning:**
  
  \[
  \text{LEAVEALONE} ()
  \]
  
  \[
  \text{Precond: } {}
  \]
  
  \[
  \text{Effects: } \neg\text{AT} (\text{SPARE-\text{TIRE, GROUND}}), \quad \neg\text{AT} (\text{SPARE-\text{TIRE, AXLE}}),
  \]
  
  \[
  \neg\text{AT} (\text{SPARE-\text{TIRE, TRUNK}}), \quad \neg\text{AT} (\text{FLAT-\text{TIRE, AXLE}}), \quad \neg\text{AT} (\text{FLAT-\text{TIRE, GROUND}})
  \]
Example: robot moving along a grid

- A robot is asked to move from an initial position to final position, being able to move along a grid
  - 8 possible moves: N, S, E, W, NW, NE, SW, SE
  - Some of the cells of the grid contain unavoidable obstacles
Representation of the problem of the robot in the grid

- **Language:**
  - Constants: numbers indicating horizontal and vertical coordinates
  - Predicates: ROBOT-AT(-,-) and FREE(-,-)

- **Initial state** (empty cells having no obstacles, and current position of the robot):
  
  FREE(1,1),...,FREE(6,2),FREE(11,2),...,FREE(12,12),
  ROBOT-AT(2,3).

- **Goal:** ROBOT-AT(10,11)

- **Actions** (the remaining seven are analogous to this one):

  MOVE-SW(x,y)
  Precond: ROBOT-AT(x,y), FREE(x+1,y-1)
  Effects: −ROBOT-AT(x,y), −FREE(x+1,y-1),
  ROBOT-AT(x+1,y-1), FREE(x,y)

- For this problem, a small extension of the semantics is required, since we need function symbols (+ and −), and we need the satisfiability test to sum or substract
Extensions to the PDDL formalism

• There exist planning systems that use a richer representation language

• For example:
  • Variables having types (this extension does not provide higher expressive power)
  • Use of function symbols
  • Use of the equal predicate
  • Evaluation of functions when applying actions
  • Disjunctions on the preconditions

• In general there exists a compromise between language expressiveness and simplicity of the algorithms handling the representation
Outline

Introduction

The PDDL formalism

Forward Search

Backward Search

Heuristics for planning

Partial Order Planning: POP
Search in states space

- A planning problem can be formalized in the framework of states space search:
  - States described by means of lists of closed atoms.
  - Actions as lists of preconditions and effects.
  - Function `is-final-state` described as a goal.

- The search for plans could be carried out by the search algorithms discussed in previous units: breadth-first, depth-first, best-first, A*,... 

- The use of the logic formalism allows to define domain-independent heuristics
  - For example, number of literals in the goal that still need to be satisfied by a state
Obtaining successors

- APILAR(x,y) [x=B, y=C]
- APILAR(x,y) [x=B, y=A]
- APILAR(x,y) [x=B, y=D]
- BAJAR(x) [x=B]
Obtaining successors

Example:

\[ S = \{ \text{CLEAR}(C), \text{CLEAR}(A), \text{CLEAR}(D), \text{ONTHETABLE}(C), \text{ONTHETABLE}(A), \text{ONTHETABLE}(D), \text{HOLD}(B) \} \]

Four possible applicable actions:
1), 2) and 3): \( A=\text{PILE}(x,y) \) with \( \text{THETA}=[x=B,y=C] \), \( \text{THETA}=[x=B,y=A] \) and \( \text{THETA}=[x=B,y=D] \), resp.
4): \( A=\text{RELEASE}(x) \) with substitution \( \text{THETA}=[x=B] \).

Successors:

\[ S_1 = \{ \text{CLEAR}(A), \text{CLEAR}(D), \text{ONTHETABLE}(C), \text{ONTHETABLE}(A), \text{ONTHETABLE}(D), \text{FREEARM}(), \text{CLEAR}(B), \text{ON}(B,C) \} \]

\[ S_2 = \{ \text{CLEAR}(C), \text{CLEAR}(D), \text{ONTHETABLE}(C), \text{ONTHETABLE}(A), \text{ONTHETABLE}(D), \text{FREEARM}(), \text{CLEAR}(B), \text{ON}(B,A) \} \]

\[ S_3 = \{ \text{CLEAR}(A), \text{CLEAR}(C), \text{ONTHETABLE}(C), \text{ONTHETABLE}(A), \text{ONTHETABLE}(D), \text{FREEARM}(), \text{CLEAR}(B), \text{ON}(B,D) \} \]

\[ S_4 = \{ \text{CLEAR}(A), \text{CLEAR}(C), \text{CLEAR}(D), \text{ONTHETABLE}(C), \text{ONTHETABLE}(A), \text{ONTHETABLE}(D), \text{FREEARM}(), \text{CLEAR}(B), \text{ONTHETABLE}(B) \} \]
(Forward) Depth-first search with heuristic

FUNCTION DEPTH-FIRST-SEARCH-H(INITIAL-STATE, GOAL, ACTIONS)
Return DFS-H-REC({}, {}, INITIAL-STATE, GOAL, ACTIONS)

FUNCTION DFS-H-REC(PLAN, VISITED, CURRENT, GOAL, ACTIONS)
1. If CURRENT satisfies GOAL, then return PLAN
2. Let APPLICABLE be the list of instantiated actions from ACTIONS, that are applicable to CURRENT and whose application yields a new state not present in VISITED
3. Let ORD-APPLICABLE be equal to HEURISTIC-SORT(APPLICABLE)
4. For each ACT in ORD-APPLICABLE
   4.1 Let S’ be the result of applying ACT to CURRENT
   4.2 Let RES equal to DFS-H-REC(PLAN·ACT, VISITED U {S’}, S’, GOAL, ACTIONS)
   4.3 If RES is not FAIL, return RES and halt
5. Return FAIL

• In this algorithm the function HEURISTIC-SORT remains to be defined
  • In general, it is defined by using an heuristic H over the states, that estimates the number of steps to reach the goal
Depth-first search with heuristic: properties

- The algorithm follows a Backtracking scheme, where the successors of the current state are heuristically sorted.

Properties:

- Sound, complete and always halts. Shortest solution is not guaranteed.
- Time complexity is exponential.
- Space complexity is linear wrt the maximum depth of the search tree (thus better option than best-first and $A^*$, unless we need the shortest solution and we have an admissible $H$).

- In practice, its efficiency depends on the quality of $H$.

- We will comment further on heuristics later on.
Outline

Introduction

The PDDL formalism

Forward Search

Backward Search

Heuristics for planning

Partial Order Planning: POP
Backward Search

- With a bad heuristic, forward search has the inconvenient of too much branching
  - Many applicable actions, but most of them being irrelevant for the goal
- Alternative approach: to proceed backwards, using the goal as a guide
  - Starts on the goal, actions are inversely applied and we try to reach the initial state
  - Nodes of the search tree: goals
- Key: only actions relevant for the goal are inversely applied
Actions relevant for the goal

Definition

- Let $G$ be a goal *without variables* (the case with variables will be addressed later).

An action $A$ is said to be *relevant* for $G$ if:

- At least one of the effects (positive or negative) of $A$ appears in $G$ with the same ”sign”
- None of the negative effects of $A$ appears in $G$ as positive
- None of the positive effects of $A$ appears in $G$ as negative

Intuitive idea:

- An action is relevant if it could have been the last action applied from a plan leading to the goal
Relevant actions: example

- Goal: \{\texttt{CLEAR(A)}, \texttt{CLEAR(B)}, \texttt{CLEAR(D)}, \texttt{ON(B,C)}\}

- Relevant actions: \texttt{PILE(B,C)}, \texttt{RELEASE(A)} y \texttt{RELEASE(D)}

- Example of action \textbf{not relevant} for the same goal: \texttt{PILE(A,D)} (although has \texttt{CLEAR(A)} as positive effect, it also has as negative effect \texttt{–CLEAR(D)}
Obtaining predecessors

If \( A \) is an action without variables relevant for a goal \( G \), then the \textit{predecessor goal} of \( G \) \textit{wrt} \( A \) is 
\[(G - \text{effects}(A)) \cup \text{precond}(A)\]

**Example:**

\[G = \{\text{CLEAR}(A), \text{CLEAR}(B), \text{CLEAR}(D), \text{ON}(B,C)\}\]

Predecessor with respect to the relevant action \( \text{PILE}(B,C) \)

\[G' = (G - \{\text{-CLEAR}(C), \text{-HOLD}(B), \text{FREEARM}(), \text{ON}(B,C), \text{CLEAR}(B)\})\]
\[\quad \cup \{\text{CLEAR}(C), \text{HOLD}(B)\}\]
\[= \{\text{CLEAR}(A), \text{CLEAR}(D), \text{CLEAR}(C), \text{HOLD}(B)\}\]

- Note that the logic formalism allows to easily calculate predecessors
Backward search with heuristic (no variables)

FUNCTION BACKWARD-SEARCH-H(INITIAL-STATE, GOAL, ACTIONS)
    Return BS-H-REC({}, {}, INITIAL-STATE, GOAL, ACTIONS)

FUNCTION BS-H-REC(PLAN, VISITED, INITIAL-STATE, G-CURRENT, ACTIONS)
1. If INITIAL-STATE satisfies G-CURRENT, return PLAN
2. Let RELEVANT be the list of instantiated actions from ACTIONS, that are relevant for G-CURRENT
   and such that the predecessor of G-CURRENT wrt the action is a goal that do not contain any of the VISITED
3. Let ORDERED-RELEVANT be equal to HEURISTIC-SORT(RELEVANT)
4. For each ACT in ORDERED-RELEVANT
   4.1 Let G' the predecessor goal of G-CURRENT wrt ACT
   4.2 Let RES be equal to BS-H-REC(ACT·PLAN, VISITED U {G'}, INITIAL-STATE, G', ACTIONS)
   4.4 If RES is not FAIL, return RES and halt
5. Return FAIL

• In this algorithm the function HEURISTIC-SORT remains to be defined
  • In general, it is defined by using an heuristic $H$ over the states, that estimates the number of steps to reach the goal
Backward search with heuristic: properties

- The algorithm performs a search in a space where the states are the goals
  - Includes an heuristic ordering of goals

Properties:

- Sound, complete and always halts.
  - Shortest solution is not guaranteed
- Theoretical complexity: same as forward search

- In practice, again due to high branching, its efficiency depends on the quality of the heuristic
  - We will comment further on heuristics later on
- Besides, in backward search we have a way of reducing branching, by minimum instantiation of actions
  - To this aim, we will use unification (similar to Prolog)
Complete vs partial instantiation

Example: Consider the following problem

- \( n + 2 \) blocks: \( A, A_1, A_2, \ldots, A_n, B \)
- Initial state:
  \[
  \{ \text{CLEAR}(A), \text{ON}(A, A_1), \text{ON}(A_1, A_2), \ldots, \text{ON}(A(n-1), A_n),
  \text{CLEAR}(B), \text{FREEARM}() \}
  \]
- Goal: \( \{ -\text{CLEAR}(B), \text{FREEARM}() \} \)
- The following \( n + 2 \) actions are relevant:
  \[
  \{ \text{PILE}(A, B), \text{PILE}(A_1, B), \ldots, \text{PILE}(A_n, B), \text{PILE}(B, B) \}
  \]

Idea to reduce branching
Leave the first argument of \texttt{PILE} unspecified
- Thus we have to accept goals with variables
- We will use the most general unifier
Most general unifier

- Unifier for two first-order logic terms \( s \) and \( t \):
  - Substitution \( \sigma \) (over the variables in \( s \) or in \( t \)) such that \( \sigma(s) = \sigma(t) \)

- Most general unifier (\( mgu \))
  - A unifier that is more general than any other unifier
  - Intuitively, \textit{more general} means that the variables are specified only the necessary minimum to make the two terms match

- Example: unification of \( P(x_1,A,x_2,x_3) \) and \( Q(x_4,x_5,B,x_5) \)
  - \([x_1/B, x_2/B, x_3/A, x_4/B, x_5/A]\) is a unifier, but not the most general one
  - \([x_1/x_4, x_2/B, x_3/A, x_5/A]\) is a \( mgu \)
Example of backward search with unification (I)

- Let us apply this idea to the previous problem
- Step 1:
  - The effect $\neg \text{CLEAR}(y_1)$ of the action $\text{PILE}(x_1,y_1)$ unifies with the literal $\neg \text{CLEAR}(B)$ of the goal (and it is relevant for $G$)
  - A mgu is the substitution $\sigma_1 = [y_1/B]$ (that is, $x_1$ is not instantiated)
  - We thus obtain a relevant action scheme: $\text{PILE}(x_1,B)$
  - New goal $G_1$:
    
    $$(G - \text{effects}(\text{PILE}(x_1,B))) \cup \text{precond}(\text{PILE}(x_1,B)) = \{\text{CLEAR}(B), \text{HOLD}(x_1)\}$$

- The initial state does not satisfy $G_1$, so we proceed
Example of backward search with unification (II)

- **Step 2:**
  - The effect $\text{HOLD}(x_2)$ of the action $\text{UNPILE}(x_2, y_2)$ unifies with the literal $\text{HOLD}(x_1)$ of the goal (and it is relevant for $G_1$)
  - A mgu is the substitution $\sigma_2 = [x_2/x_1]$
  - We thus obtain the relevant action scheme: $\text{UNPILE}(x_1, y_2)$
  - New goal $G_2$:
    \[
    (G_1 - \text{effects(UNPILE}(x_1, y_2)) \cup \text{precond(UNPILE}(x_1, y_2)) = \{\text{CLEAR}(B), \text{ON}(x_1, y_2), \text{CLEAR}(x_1), \text{FREEARM}()\}
    \]
  - The initial state satisfies $G_2$ (under the substitution $[x_1/A, y_2/A1]$)
  - By composing the substitutions we obtain the plan: $\text{UNPILE}(A, A1), \text{PILE}(A, B)$
  - Note: each time an action is considered, it is necessary to rename it using new variables (standardization)
Backward search with heuristic and unification

FUNCTION BACKWARD-SEARCH-H-U(INITIAL-STATE, GOAL, ACTIONS)
Return BS-H-U-REC({}, {}, INITIAL-STATE, GOAL, ACTIONS)

FUNCTION BS-H-U-REC(PLAN, VISITED, INITIAL-STATE, G-CURRENT, ACTIONS)
1. If INITIAL-STATE satisfies G-CURRENT, return PLAN
2. Let RELEVANT be the list of pairs (A,SIGMA) such that:
   * A is a standardized action from ACTIONS, relevant for STATE
   * SIGMA is a mgu for the effects of A that make the action be relevant and the corresponding literals from G-CURRENT
   * The predecessor of CURRENT wrt the action is a goal that does not have any subset of literals unifying any of the goals in VISITED
3. Let ORDERED-RELEVANT be equal to HEURISTIC-SORT(RELEVANT)
4. For each (A,SIGMA) in ORDERED-RELEVANT
   4.1 Let G’ the predecessor goal of G-CURRENT wrt (A,SIGMA); that is,
      G’=(SIGMA(G-CURRENT) - Effects(SIGMA(A)))
       U Preconds(SIGMA(A))
   4.2 Let RES be equal to
      BS-H-U-REC((A,SIGMA)·SIGMA(PLAN), VISITED U {G’}, INITIAL-STATE, G’, ACTIONS)
   4.3 If RES is not FAIL, return RES and halt
5. Return FAIL
Outline

Introduction

The PDDL formalism

Forward Search

Backward Search

Heuristics for planning

Partial Order Planning: POP
Heuristics for planning based on states space

- A fundamental component of the practical efficiency of the previous algorithms is the heuristic used for ordering states or goals
- This heuristic should estimate distance between states and goals (number of actions that we need to apply)
  - In forward search: for each state, distance to the goal
  - In backward search: for each goal, distance from the initial state
- If this estimation is below the actual minimum number of actions, we say that the heuristic is *admissible*
- Wanted: domain-independent heuristics
  - Based on the logic representation of states, goals and actions
Heuristics based on relaxing the problem

- Heuristics obtained by relaxing some of the restrictions of the problem and calculating the number of actions necessary for such relaxed problem
- Some ideas for relaxing a problem:
  - Ignore the negative preconditions and/or negative effects of the actions
  - Assume that each literal in a goal is achieved independently: estimate the number of actions necessary to achieve a goal as the sum of the number of steps necessary to achieve each of the literals of the goal
  - Assume that the number of actions necessary to achieve a goal is the maximum number of steps necessary to achieve one of its literals
  - Ignore all preconditions of the actions
  - Ignore certain predicates
The $\Delta_0$ heuristic

- Let us define the $\Delta_0$ heuristic based on the first and the second ideas just mentioned.
- Given a state $s$, an atom $p$ and a goal $g$ that only has positive literals without variables, we define recursively $\Delta_0(s, p)$ and $\Delta_0(s, g)$ as follows:
  - If $p$ appears in $s$, then $\Delta_0(s, p) = 0$
  - If $p$ does not appear in $s$ nor in the positive effects of any action, then $\Delta_0(s, p) = +\infty$
  - Otherwise, $\Delta_0(s, p) = \min_A \{ 1 + \sum_{q \in \text{precond}^+(A)} \Delta_0(s, q) | p \in \text{effects}^+(A) \}$
  - $\Delta_0(s, g) = \sum_{p \in g} \Delta_0(s, p)$
- Notation: $\text{precond}^+(A)$ and $\text{effects}^+(A)$ denote the positive literals of the precondition and of the effects of an action $A$, respectively.
The $\Delta_0$ heuristic (properties)

- Intuitively:
  - $\Delta_0(s, p)$ counts the least number of steps necessary starting from $s$ in order to satisfy $p$, assuming that the actions do not have any negative preconditions nor negative effects
  - $\Delta_0(s, g)$ is the sum of such estimations for each atom $p \in g$

- Note that, in general, the estimation made by $\Delta_0$ is not admissible
  - Anyway, it might perform quite well in practice
  - Besides, it is not intended to be used with $A^*$, but with depth-first search instead

- Given a $s$, the values $\Delta_0(s, p)$ for each atom $p$ can be calculated via an algorithm similar to Dijkstra’s shortest path algorithm
Using $\Delta_0$ heuristic in search (I)

In forward search:

- To each action $A$ applicable on the current state, we assign the heuristic value $\Delta_0(s, g^+)$, where $g^+$ is the set of the positive literals of the goal, and $s$ is the state resulting after applying the action $A$. 

![Diagram showing the heuristic value $\Delta_0(s, g^+)$ in forward search.](image)
Using $\Delta_0$ heuristic in search (II)

In backward search:

- To each action $A$ being relevant wrt the current goal, we assign the heuristic value $\Delta_0(s_0, g^+)$, where $s_0$ is the initial state, $g$ is the predecessor corresponding to the action $A$, and $g^+$ is the set of positive literals of $g$. 
Using $\Delta_0$ heuristic in search (III)

In the case of goals with variables:

- If $p$ is an atom with variables, $\Delta_0(s, p)$ is defined as the minimum of all $\Delta_0(s, \sigma(p))$, being $\sigma(p)$ instances without variables of $p$.
- If $g$ is a goal with variables, $\Delta_0(s, g)$ is the sum of $\Delta_0(s, p)$ for each positive literal $p$ of $g$. 
Outline

Introduction

The PDDL formalism

Forward Search

Backward Search

Heuristics for planning

Partial Order Planning: POP
Partially ordered plans

- New approach for searching plans, different from the states-space-based search:
  - Instead of searching through a space of states or goals, the search is conducted in a space of non-fully-specified plans (partially ordered plans)
- *Partially ordered plan*: a plan where only part of the precedences between actions are specified
Example: how to pass IA

• Language:
  • Objects: **IAING**, **ETSII**, **HOME**
  • Predicates: **AT(−)**, **STUDIED(−)**, **PASSED(−)**

• Initial state:
  **AT(HOME)**

• Goal:
  **AT(HOME)**, **PASSED(IAING)**

• Actions:
  **GO(x, y)**
  **STUDY(x)**
  **TAKE-EXAM-SUCCESS(x)**
  **P**: **AT(x)**
  **P**: **{}**
  **P**: **AT(ETSII), STUDIED(x)**
  **E**: **−AT(x), AT(Y)**
  **E**: **STUDIED(x)**
  **E**: **PASSED(x)**

• Actions of the type **GO(x, x)** will be discarded
Example of partial plan

INICIO

E−C−E(IA1)

IR(ETSII,CASA)

APROBADO(IA1)

FIN

EN(CASA)
Introduction

The PDDL formalism

Forward Search

Backward Search

Heuristics for planning

Partial Order Planning: POP

Partially ordered plans: graph components (I)

• Nodes: *Actions* (from the actions of the problem, having their preconditions and effects), that constitute the steps that should be performed when executing the plan

  • Two special actions: START (no preconditions and its effect is the initial state) and FINISH (no effects and its precondition is the final goal)

• Arcs: *restrictions of order* \( A \prec B \) between actions of the plan

(continues...)
Partially ordered plans: graph components (II)

- Another special type of arcs: *causal links*, $A \xrightarrow{p} B$, specifying that one of the preconditions of $B$ ($p$) is achieved by applying action $A$ (an ordering constraint is implicit)

- A set of *open preconditions*: those that do not have yet causal links that achieve them
Initial and final partial plans

- Initial partial plan: plan whose only actions are START and FINISH, with the restriction START ≺ FINISH, no causal links, and all preconditions of FINISH being open.
- Final partial plans: partial plans without conflicts between causal links, without cycles between ordering constraints and without open preconditions.
  - An action C has a conflict with (or threatens) a causal link A → B, if C has ¬p among its effects and, according to ordering constraints, C might be applied after A and before B.
Examples of partial plans

- Example of initial partial plan:

- Example of final partial plan:
Linearizations of final partial plans

- Linearization of a partial plan: line up all the actions in a row (sequence), without contradicting any of the ordering constraint deduced from the partial plan
- Key point: any linearization of a final partial plan constitutes a solution to the original problem
- In the example, two possibilities (both are solutions to the original problem):

```
ESTUDIAR(IA1) IR(CASA,ETSII) E−C−E(IA1) IR(ETSII,CASA)
IR(CASA,ETSII) ESTUDIAR(IA1) E−C−E(IA1) IR(ETSII,CASA)
```
Planification as search in the space of plans

• How to design an algorithm to find final partial plans?
  • Idea POP: start from the plan initial and apply transformations to the partial plans, refining them (fixing the flaws)
  • Basically, such transformations are either to achieve an open precondition or to neutralize a threat

• At each point there will be several refinement options, and not all of them lead to a final partial plan

• Problem: find the sequence of refinement that starting from the initial partial plan, yields a final partial plan (that is, with no cycles, no conflicts and no open preconditions)
  • It’s again a states space search!

• But now the states are partial plans and actions are refinement steps
POP Algorithm for the example of how to pass IA

- For the sake of a clear graphical representation, we will agree on the following conventions:
  - Some ordering constraints will not be drawn, in particular those associated with causal links
  - Preconditions and effects will not be displayed. In particular, open preconditions will not be drawn
  - When an action does not have any open precondition, it will appear as coloured box

- Step 1: Initial plan

  - Open preconditions: \texttt{PASSED(IAING), AT(HOME)}; let us achieve first \texttt{PASSED(IAING)}
  - Only one possibility: use a new action \texttt{TAKE-EXAM-SUCCESS(IAING)}
POP Algorithm for the example of how to pass IA

- Step 2:

- We achieve now the precondition \( \text{AT(HOME)} \) from FINISH
- Two possibilities: START or new action GO(ETSII,HOME)
- Consider first START, and try the second one in case of Fail
- When a precondition is achieved by adding a causal link from an action already in the plan, this refinement is called *simple establishment*
- If we need to include a new action in the plan, this refinement is called *step addition*
POP Algorithm for the example of how to pass IA

• Step 3:

  INICIO  →  E–C–E(IA1)  →  APROBADO(IA1)  →  FIN

  • We achieve now the precondition \texttt{AT(ETSII)} from 
    \texttt{TAKE-EXAM-SUCCESS(IAING)}
  • Only one possibility: use a new action \texttt{GO(HOME,ETSII)}
POP Algorithm for the example of how to pass IA

- Step 4:

- A threat shows up: from GO(HOME, ETSII) into the causal link between START and FINISH
- We try to fix the threat
  - forcing GO(HOME, ETSII) to be executed after FINISH (promotion), or else
  - forcing GO(HOME, ETSII) to be executed before START (degradation)
POP Algorithm for the example of how to pass IA

- But in both cases, a cycle would be created:
  - Promotion:
    - Degradation:
POP Algorithm for the example of how to pass IA

- Therefore, we get Fail and we must go back to the last branching point, which was in step 2
  - And choose the other alternative: new action $\text{GO(ETSII,HOME)}$ to achieve the precondition $\text{AT (HOME)}$ in $\text{FINISH}$

- Step 5:

- We select now the precondition $\text{STUDIED(IAING)}$ in $\text{TAKE-EXAM-SUCCESS(IAING)}$
  - The only possibility to achieve it is a new action $\text{STUDY(IAING)}$
POP Algorithm for the example of how to pass IA

• Step 6:

- We select now the precondition $\text{AT(ETSII)}$ in $\text{TAKE-EXAM-SUCCESS(IAING)}$
  - The only possibility to achieve it is a new action $\text{GO(HOME,ETSII)}$
POP Algorithm for the example of how to pass IA

• Step 7:

At this point there are threats in the plan: for example, from GO(ETSII,HOME) into the causal link between GO(HOME,ETSII) and TAKE-EXAM-SUCCESS(IAING)
• We try to fix the threat using degradation; if this doesn’t work, then we try promotion
POP Algorithm for the example of how to pass IA

- **Step 8 (degradation):**

  - New threat: from GO(HOME,ETSII) into the causal link between GO(ETSII,HOMEx) and FINISH
  - We try to fix the threat by promotion or degradation
POP Algorithm for the example of how to pass IA

- But in both cases, a cycle would be created:
  - Promotion:
  
  ![Diagram of the promotion process]

  - Degradation:
  
  ![Diagram of the degradation process]
Therefore, we get Fail

We must go back to the last branching point (step 7), and pick the other alternative: fix the threat from GO(ETSII,HOM) into the causal link between GO(HOME,ETSII) and TAKE-EXAM-SUCCESS(IAING), by promotion.

Step 9:
POP Algorithm for the example of how to pass IA

- We achieve now the precondition $\text{AT(ETSII)}$ from $\text{GO(ETSII,HOME)}$
  - Two possibilities: simple establishment with $\text{GO(HOME,ETSII)}$ or step addition with $\text{GO(HOME,ETSII)}$.
  - We select the first option, but will reconsider if necessary
- Step 10:
POP Algorithm for the example of how to pass IA

- We achieve now the precondition \texttt{AT} (\texttt{HOME}) from \texttt{GO} (\texttt{HOME}, \texttt{ETSII})
  - Three possibilities: simple establishment with \texttt{START}, simple establishment with \texttt{GO} (\texttt{ETSII}, \texttt{HOME}) or step addition with \texttt{GO} (\texttt{ETSII}, \texttt{HOME}).
  - We select the first option, but will reconsider if necessary
- Step 11 (final partial plan):

```
INICIO
IR(CASA, ETSII)
IR(ETSII, CASA)
EN(CASA)
EN(ETSII)
EN(ETSII)
EN(CASA)
APROBADO(IA1)
E−C−E(IA1)
ESTUDIADO(IA1)
```

```
IR(CASA, ETSII)
IR(ETSII, CASA)
EN(CASA)
EN(ETSII)
EN(ETSII)
EN(CASA)
```
POP Algorithm for the example of how to pass IA

- Finally, we perform linearization (two possibilities):

\[
\begin{align*}
\text{ESTUDIAR(IA1)} & \rightarrow \text{IR(CASA,ETSII)} & \rightarrow \text{E–C–E(IA1)} & \rightarrow \text{IR(ETSII,CASA)} \\
\text{IR(CASA,ETSII)} & \rightarrow \text{ESTUDIAR(IA1)} & \rightarrow \text{E–C–E(IA1)} & \rightarrow \text{IR(ETSII,CASA)}
\end{align*}
\]
As seen in the previous example, it is possible to implement a planning algorithm as a search

- More precisely, as a search of a sequence of refinement transformations that starting from the initial plan obtains a final partial plan

- A POP algorithm is just an algorithm that implements a search algorithm (e.g. depth-first) in the space of partial plans
POP Search tree for the example of how to pass IA
Transformations (refinements of partial plans)

- Achieving open preconditions: given an open precondition $p$ from an action $B$ of the plan, for each action $A$ having $p$ as effect, we can obtain a successor (refined) plan by applying some of the following steps (provided that the resulting plan has no cycles):
  - *simple establishment*: if action $A$ is already in the plan, add the restriction $A \prec B$ and the causal link $A \xrightarrow{p} B$
  - *Step addition*: add the new action $A$ to the plan, the restrictions $A \prec B$, START $\prec A$ and $A \prec$ FINISH, and the causal link $A \xrightarrow{p} B$ (it is even possible to add as new an action equal to an existing one, assuming that it represents a different step in the final linearized solution)
Fixing conflicts: given a threat from the action \( C \) into the causal link \( A \xrightarrow{p} B \), we can fix it and obtain a new successor plan by choosing one of the following options (provided that the new plan does not have any cycle):

- Add the constraint \( B \prec C \) \((promotion)\)
- Or add the constraint \( C \prec A \) \((degradation)\)
POP as a search process

- Search tree:
  - Nodes: partial plans
  - Branching due to election among actions that achieve open preconditions
  - Branching due to election among the two ways of fixing threats (promotion and degradation)

- Note: it is not necessary to consider branching due to the election of the next open precondition to achieve, neither for the election of which one of the current threats to fix. They will be dealt with in future steps, following a given order
  - Although such an order does make a difference wrt the efficiency of the search

- Depth-first search might get into an infinite branch
  - Therefore, we perform a bounded search (having a maximum number of actions in the plan)
Algorithm POP (recursive version)

FUNCTION POP(INITIAL-STATE, GOAL, ACTIONS, BOUND)
1. Return POP-REC(INITIAL-PLAN(INITIAL-STATE, GOAL), ACTIONS, BOUND)

FUNCTION POP-REC(PLAN, ACTIONS, BOUND)
1. If PLAN is final, return LINEARIZATION(PLAN) and halt.
2. If there exists a threat in PLAN from action C into A --p-->B, then
   2.1 Let PLANPR obtained from PLAN by promotion
   2.2 If PLANPR has any cycle, let RESULT equal to FAIL; otherwise:
      let RESULT equal to POP-REC(PLANPR, ACTIONS, BOUND).
   2.3 If RESULT is not FAIL, return RESULT and halt; otherwise:
      2.3.1 Let PLANDEG obtained from PLAN by degradation
      2.3.2 If PLANDEG has any cycle, let RESULT equal to FAIL; otherwise:
         let RESULT equal to POP-REC(PLANDEG, ACTIONS, BOUND).
      2.3.3 Return RESULT and halt.

(continues...)
Algorithm POP (recursive version, cont.)

3. Let P an open precondition in PLAN
4. For each action A in ACTIONS (either new or already in PLAN) having P as an effect, do:
   4.1 Let PLANEXT the result of achieving the precondition P in PLAN by a causal link from A (in the case of a new action, add the action to the PLAN)
   4.2 If PLANEXT has no cycles and its number of actions is lower or equal to BOUND, then:
      4.2.1 Let RESULT equal to POP-REC(PLANEXT, ACTIONS, BOUND)
      4.2.2 If RESULT is not FAIL, return RESULT and halt.
5. Return FAIL
POP: some considerations

- We have presented a simplified version of the algorithm. To be added:
  - Use of unification
  - Some variables might remain uninstantiated (principle of minimum-commitment)
  - Inequality restrictions
- The algorithm is sound and complete (provided that the bound is large enough)
- The presented pseudocode deals first with conflicts and then with open preconditions, but this is not mandatory
  - It is also not necessary to try always promotion and then degradation
POP: use of heuristics

- It is hard to estimate how “close” a partial plan is from a solution
- There are some elections in the algorithm that, if taken correctly, might produce an efficiency improvement:
  - At each step, which threat or which open precondition is refined?
  - When dealing with an open precondition, in which order do we try the actions that achieve it?
  - When fixing a conflict, in which order do we try promotion and degradation?
POP: some heuristics

Heuristic to select the next refinement step

• Select the threat or open precondition that has the least number of alternatives to fix it
  • Recall that threats always have two refinement alternatives
  • For open preconditions, as many alternatives as actions (new or existing) have it in their effects

Heuristics for the order of actions that achieve open precondition

• Sort them by the size of their preconditions lists (not very good)
• Use the planning graph
The Sussman anomaly

• Planning problem showing the lacks of the first planning algorithms (in the 70’s):

Estado inicial

C
A
B

Estado final

A
B
C

Initial state = \{ ONTHETABLE(A), ONTHETABLE(B), CLEAR(B), CLEAR(C), FREEARM(), ON(C,A) \}

Goal = \{ON(A,B), ON(B,C) \}
The Sussman anomaly

- The Sussman anomaly was a pioneer example proving that planning was not a trivial task.
- For example, a planner that tried to look first for a subplan solution for the first literal of the goal \( \text{ON}(A, B) \), and then tried to concatenate a solution for \( \text{ON}(B, C) \), will always Fail.
  - Since the second subplan would undo part of the achievements of the first one.
- This example showed the lacks of *not interleaved* planners (like the planner *STRIPS*).
The Sussman anomaly

• We will see how partial order planning solves the Sussman anomaly
  
  • Important note: to simplify the presentation, at each branching point we will select the right refinement transformations
  
  • That is, our “search” will never get fails, although it is unreal in practice
  
  • Also for the sake of simplicity, sometimes several steps will be applied together
The Sussman anomaly in POP

- The resulting partial plan is a solution: does not have any conflicts, nor cycles, nor open preconditions.
- Linearization of the plan:

  \[
  \text{DESAPILAR}(C,A) \rightarrow \text{BAJAR}(C) \rightarrow \text{AGARRAR}(B) \rightarrow \text{APILAR}(B,C) \rightarrow \text{AGARRAR}(A) \rightarrow \text{APILAR}(A,B)
  \]

- Note that the necessary steps to solve the initial two literals in the goals are interleaved.
Other approaches for planning

- SATPLAN, use of techniques based on propositional logic
- GRAPHPLAN
- Planning as PSR
- Planning with limited resources
- Planning with explicit time
- Planning by hierarchical decomposition
- Planning on environments with uncertainty
- Execution of the plan: watching and replanning
- Continuous planning
- Multiagent planning
Bibliography

  - Ch. 10: “Planning”.
  - Sec. 2.3 “Classical representation”.
  - Ch. 4 “State space planning”.
  - Ch. 5 “Plan space planning”.
  - Ch. 9 “Heuristics in planning”.