Automated (AI) Planning

Fatma Başak Aydemir

CmpE 480
Boğaziçi University
November 14, 2019

Table of contents

1. What is 'Planning'?
2. Generating and Solving Planning Problems
3. PDDL

What is 'Planning'?

What is 'Planning'?

What is 'Planning'?

Planning Problem

Given:

- A description of (possible) **initial state(s)**

Planning Problem

Given:

- A description of (possible) **initial state(s)**
- A description of **desired goals**

Planning Problem

Given:

- A description of (possible) **initial state(s)**
- A description of **desired goals**
- A description of a set of **possible actions**

Planning Problem

Given:

- A description of (possible) **initial state(s)**
- A description of **desired goals**
- A description of a set of **possible actions**

Generate:

Planning Problem

Given:

- A description of (possible) **initial state(s)**
- A description of **desired goals**
- A description of a set of **possible actions**

Generate:

- A **sequence of actions** that **leads to** one of **the goal states**.

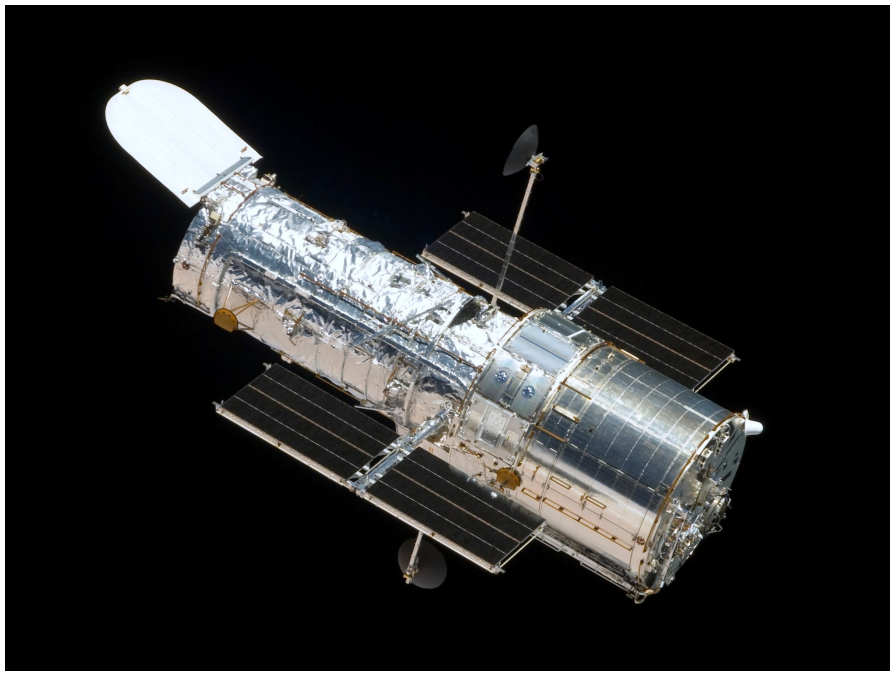
Planning Problem: Blocks

Where do we use planning?

- Space missions

Where do we use planning?

- Space missions
- Hubble telescope



Where do we use planning?

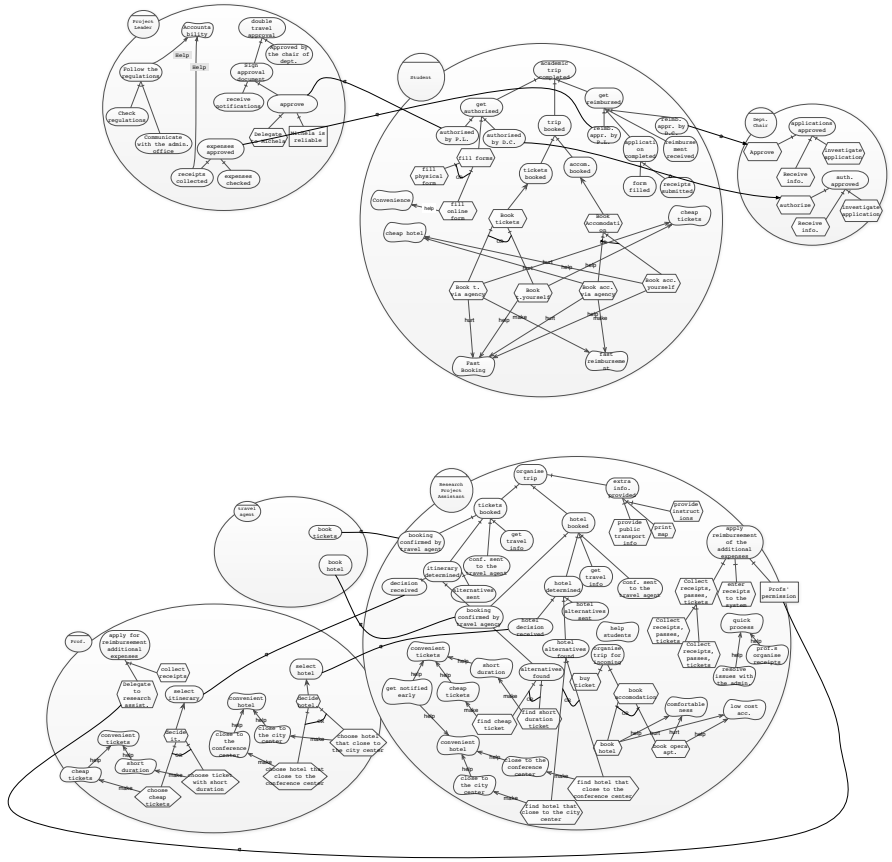
- Space missions
- Hubble telescope
- Logistics

Where do we use planning?

- Space missions
- Hubble telescope
- Logistics
- Games

Where do we use planning?

- Space missions
- Hubble telescope
- Logistics
- Games
- Requirements Engineering!



Why is planning so difficult?

- Uncertainty about effects of actions
- Uncertainty about environment
- Uncertain sensing and perception
- Agents own actions may have bad effects
- Time and resources are limited!
- Preferences

Let's simplify things a bit...

- Known initial state
- Deterministic actions
- Simple action representation
- Instantaneous actions
- No deadlines and sufficient resources
- Fully observable
- Single agent /No external events
- No concurrent actions

Categories of Planning Problems

1. Classical Planning
2. Planning under uncertainty
3. Multi-agent planning
4. Temporal planning/scheduling

Planning Problem

Given:

- A description of (possible) **initial state(s)**
- A description of **desired goals**
- A description of a set of **possible actions**

Generate:

- A **sequence of actions** that **leads to** one of **the goal states**.

Planning Concepts

- State
- Action
- Goal
- Plan

Example: The Blocks World

- Objects
  - Blocks: A, B, C
  - Table: $Table$
- States: Conjunctions of ground literals
  - $On(A, B)$, $On(C, Table)$, $Clear(B)$, $Holding(C)$
- Actions: Operator schemas with variables
  - $Pickup(x)$, $Putdown(x, y)$
- Domain Axioms:
  - 'At most one block on top of another'
  - 'Hand must be empty and block must be clear to pick up'

What is a state?

A complete description of the world.

- Instantiation of all state variables
- Truth assignment of all variables

Example

Tell me some state examples.. Yes, you!

What is an action?

Transition from one state to another

- Some actions may be applicable to some of the states.
- Have given effects on the state.
  - Deterministic
  - Non-deterministic
  - Probabilistic

STRIPS

- State is database of ground literals
- If literal is not in database, assumed to be false
- Effects of actions represented using add and delete lists (insert and remove literals from database)
- No explicit representation of time
- No logical inference rules

Something more expressive 1

- Conditional Effects

```
Pickup (b)
 Pre: Block(b), Handempty, Clear(b), On(b, x)
 Add: Holding(b) if (Block(x)) then Clear(x)
 Delete: Handempty, On(b, x)
```

- Quantified effects
- Disjunctive and negated preconditions

Something more expressive 2

- Inference operators
- Functional effects
- Disjunctive and negated preconditions

Something more expressive 3

- Disjunctive Effects
- Probabilistic effects
- External events, agents, concurrent events, etc

What is a goal?

A desired state

What is a plan?

- A sequence of instantiated actions.
- A set of instantiated actions
- A tree of instantiated actions
- A policy of instantiated actions

How to find a plan?

Solution Techniques

- Graph search
- Heuristic search
- Solve the transitioning matrix

Complexity

- Exponential number of actions
- Polynomial to number of states

Let's create a planner!

Let's create a planner!

Just kidding ☺
Let's learn how to use existing planners!

PDDL

Planning **D**omain **D**efinition **L**anguage : standard encoding language for classical planning

Objects	Things in the world
Predicates	Properties of the objects
Initial state	The state of the world we start in
Goal specification	Things we want to be true
Actions	Ways of changing the state of the world

How to put pieces together?

- A **domain file** for predicates and actions
- A **problem file** for objects,, initial states, and goal descriptions

How to put pieces together?

- A **domain file** for predicates and actions
- A **problem file** for objects,, initial states, and goal descriptions

Which one can we re-use?

Domain file

```
(define (domain <domain name>)
   <PDDL code for predicates>
   <PDDL code for actions>
)
```

The blocks world domain file

```
  ;; domain file: blocksworld-domain.pddl

(define (domain blocksworld)
  (:requirements :strips)

  (:predicates (clear ?x)
               (on-table ?x)
               (holding ?x)
               (on ?x ?y))

  (:action pickup
           :parameters (?ob)
           :precondition (and (clear ?ob) (on-table ?ob))
           :effect (and (holding ?ob) (not (clear ?ob))
                        (not (on-table ?ob))))
```

Problem file

```
(define (problem <problem name>)
   (:domain <domain name>)
   <PDDL code for objects>
   <PDDL code for initial state>
   <PDDL code for goal specification>
)
```

<problem name> is the string for this specific planning problem
<domain name> must match your domain file!

The blocks world problem file

```
;; problem file: blocksworld-prob1.pddl

(define (problem blocksworld-prob1)
  (:domain blocksworld)
  (:objects a b)
  (:init (on-table a) (on-table b) (clear a) (clear b))
  (:goal (and (on a b))))
```

Sample output

```
    Parsed Domain file blocksworld-domain.pddl successfully
    Parsed Problem file blocksworld-prob1.pddl successfully
    Performing search as in FF - first considering EHC with
only helpful actions
    2
    1
    (pickup a)
    (stack a b)
    Instantiation Time = 0.07sec
    Planning Time = 0.03sec
```

Running Example: Gripper

Gripper task with four balls: There is a robot that can move between two rooms and pick up or drop balls with either of his two arms. Initially, all balls and the robot are in the first room. We want the balls to be in the second room.

Objects: The two rooms, four balls and two robot arms.

Predicates: Is x a room? Is x a ball? Is ball x inside room y ? Is robot arm x empty? [...]

Initial state: All balls and the robot are in the first room. All robot arms are empty. [...]

Goal specification: All balls must be in the second room.

Actions: The robot can move between rooms, pick up a ball or drop a ball.

Objects

Objects

- Rooms: `rooma, roomb`
- Balls: `ball1, ball2, ball3, ball4`
- Robot arms: `left, right`

PDDL

```
(:objects rooma roomb
          ball1 ball2 ball3 ball4
          left right)
```

Predicates

Predicates

- ROOM(x)
- BALL(x)
- GRIPPER(x)
- at-robby(x)
- at-ball(x,y)
- free(x)
- carry()

PDDL

```
(:predicates (ROOM ?x) (BALL ?x)
             (GRIPPER ?x)
             (at-robby ?x)
             (at-ball ?x ?y)
             (free ?x) (carry ?x ?y))
```

Initial State

Initial State

ROOM(rooma) and ROOM(roomb) are true. BALL(ball1), ..., BALL(ball4) are true. GRIPPER(left), GRIPPER(right), free(left) and free(right) are true. at-robby(rooma) , at-ball(ball1, rooma), ..., at-ball(ball4, rooma) are true. Everything else is false.

Initial State

Initial State

ROOM(rooma) and ROOM(roomb) are true. BALL(ball1), ..., BALL(ball4) are true. GRIPPER(left), GRIPPER(right), free(left) and free(right) are true. at-robby(rooma) , at-ball(ball1, rooma), ..., at-ball(ball4, rooma) are true. Everything else is false.

PDDL

```
(:init (ROOM rooma) (ROOM roomb)
       (BALL ball1) (BALL ball2) (BALL ball3) (BALL ball4)
       (GRIPPER left) (GRIPPER right) (free left)
       (free right) (at-robby rooma)
       (at-ball ball1 rooma) (at-ball ball2 rooma)
       (at-ball ball3 rooma) (at-ball ball4 rooma))
```

Goal Specification

Goal Specification

at-ball(ball1, roomb), ..., at-ball(ball4, roomb) must be true.

PDDL

```
:goal (and (at-ball ball1 roomb)
           (at-ball ball2 roomb)
           (at-ball ball3 roomb)
           (at-ball ball4 roomb)))
```

Movement Action

Action

Description: The robot can move from x to y.

Precondition: ROOM(x) , ROOM(y) and at-robby(x) are true.

Effect: at-robby(y) becomes true. at-robby(x) becomes false. Everything else doesn't change.

PDDL

```
(:action move :parameters (?x ?y)
   :precondition (and (ROOM ?x) (ROOM ?y)
                      (at-robby ?x))
   :effect (and (at-robby ?y)
                (not (at-robby ?x))))
```

Pick-up Action

Action

Description: The robot can pick up x in y with z .The robot can move from x to y.

Precondition: ROOM(x) , ROOM(y) and at-robby(x) are true. BALL(x) , ROOM(y) , GRIPPER(z) , at-ball(x , y) , at-robby(y) and free(z) are true.

Effect: carry(z , x) becomes true. at-ball(x , y) and free(z) become false. Everything else doesn't change.

PDDL

```
(:action pick-up :parameters (?x ?y ?z)
   :precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z)
                      (at-ball ?x ?y) (at-robby ?y) (free ?z))
   :effect (and (carry ?z ?x) (not (at-ball ?x ?y))
                (not (free ?z))))
```

Drop Action

Action

Description: The robot can drop x in y from z

Precondition:

Effect:

PDDL

```
(:action drop :parameters (?x ?y ?z)
   :precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z)
                      (carry ?z ?x) (at-robby ?y))
   :effect (and (at-ball ?x ?y) (free ?z)
                (not (carry ?z ?x))))
```

Extras...1: Typing

```
(define (domain gripper-typed)
   (:requirements :typing)
   (:types room ball gripper)
   (:constants left right - gripper)
   (:predicates (at-robby ?r - room)
                (at ?b - ball ?r - room)
                (free ?g - gripper)
               (carry ?o - ball ?g - gripper))
   (:action move :parameters (?from ?to - room)
       :precondition (at-robby ?from)
       :effect (and (at-robby ?to)
                    (not (at-robby ?from))))
```

Extras...2: Type Hierarchy

```
(define (domain logistics-adl)
   (:requirements :adl :domain-axioms)
   (:types physobj - object
           obj vehicle - phssobj
           truck airplane - vehicle
           location city - object
           airport - location)
  (:predicates (at ?x - physobj ?l - location)
               (in ?x - obj ?t - vehicle)
               (in-city ?l - location ?c - city))
```

Extras...3: Conditional Effects

```
(:action drive-truck
 :parameters (?truck - truck ?loc-from ?loc-to - location
?city - city)
:precondition (and (at ?truck ?loc-from)
(in-city ?loc-from ?city)
(in-city ?loc-to ?city))
:effect (and (at ?truck ?loc-to)
(not (at ?truck ?loc-from))
(forall (?x - obj)
(when (and (in ?x ?truck))
(and (not (at ?x ?loc-from))
(at ?x ?loc-to))))))
```

Questions

?