AI Planning Lab
Probabilistic Planning with PPDDL and Safe-Planner

Halim Djerroud

revision 1.0

Introduction: Probabilistic Planning

Probabilistic Planning deals with uncertainty in action effects. Unlike deterministic planning where actions have predictable outcomes, probabilistic actions may fail or have multiple possible results.

Key concepts:

- **Non-deterministic effects**: Actions can have multiple outcomes
- **Policy**: A function mapping states to actions (not just a linear plan)
- **Strong cyclic policy**: Guarantees goal achievement despite failures
- **PPDDL**: Probabilistic Planning Domain Definition Language

In this lab, you'll use **Safe-Planner**, which compiles non-deterministic PPDDL domains into classical planning problems.

Installation

```
# Clone Safe-Planner
git clone https://github.com/mokhtarivahid/safe-planner.git
cd safe-planner

# Test installation
./sp --help

# Install Graphviz for visualization
sudo apt-get install graphviz xdot
```

Basic PPDDL Structure

```
(define (domain simple-navigation)
  (:requirements :strips :typing :non-deterministic)

  (:types location)

  (:predicates
    (at ?l - location)
    (connected ?from ?to - location))

  (:action move
    :parameters (?from ?to - location)
    :precondition (and (at ?from) (connected ?from ?to))
    :effect (and
      (not (at ?from))
      (oneof
        (at ?to)       ; success: reach destination
        (at ?from)))) ; failure: stay in place
)
```

Key difference: `oneof` creates non-determinism. The planner generates a **policy** that handles both outcomes.

Exercise 1: Warehouse Delivery Robot (45 min)

Scenario

A mobile robot delivers packages in a warehouse. The robot can move between zones, pick packages, and deliver them. However:

- **Movement may fail** (slippery floor, obstacles)
- **Package pickup may fail** (heavy objects, bad grasp)
- **Delivery validation may fail** (sensor error)

Warehouse layout:

```
[Storage] <--> [Corridor] <--> [Office]
                    |
                    v
                [Kitchen]
```

Task 1.1: Simple Navigation Domain (15 min)

Create `warehouse-domain.ppddl`:

```
(define (domain warehouse-navigation)
  (:requirements :strips :typing :non-deterministic)

  (:types location)

  (:predicates
    (robot-at ?l - location)
    (connected ?from ?to - location))

  ;; TODO: Define move action
  ;; Movement can succeed or fail (robot stays in place)
  (:action move
    :parameters (?from ?to - location)
    :precondition (and
      (robot-at ?from)
      (connected ?from ?to))
    :effect (and
      (not (robot-at ?from))
      (oneof
        (robot-at ?to)       ; success
        (robot-at ?from)))) ; failure: stays in place
)
```

Create `warehouse-problem-1.ppddl`:

```
(define (problem navigate-to-kitchen)
  (:domain warehouse-navigation)

  (:objects
    storage corridor office kitchen - location)

  (:init
    (robot-at storage)
    (connected storage corridor)
    (connected corridor storage)
    (connected corridor office)
    (connected office corridor)
    (connected corridor kitchen)
    (connected kitchen corridor))

  (:goal (robot-at kitchen))
)
```

Test:

```
./sp -d warehouse-domain.ppddl -p warehouse-problem-1.ppddl -c ff
```

Question 1.1a: How many actions are in the generated plan?
Question 1.1b: Draw the policy graph. What happens if the first `move` fails?

Task 1.2: Add Package Handling (20 min)

Extend the domain with package manipulation:

```
(define (domain warehouse-delivery)
  (:requirements :strips :typing :non-deterministic)

  (:types location package)

  (:predicates
    (robot-at ?l - location)
    (package-at ?p - package ?l - location)
    (holding ?p - package)
    (delivered ?p - package)
    (connected ?from ?to - location)
    (empty-hand))

  (:action move
    :parameters (?from ?to - location)
    :precondition (and (robot-at ?from) (connected ?from ?to))
    :effect (and
      (not (robot-at ?from))
      (oneof (robot-at ?to) (robot-at ?from))))

  ;; TODO: Define pick action
  ;; Can succeed or fail (package too heavy, bad grip)
  (:action pick
    :parameters (?p - package ?l - location)
    :precondition (and
      (robot-at ?l)
      (package-at ?p ?l)
      (empty-hand))
    :effect (oneof
      (and (holding ?p)
           (not (package-at ?p ?l))
           (not (empty-hand)))  ; success
      (and))                    ; failure: nothing changes

  ;; TODO: Define drop action (always succeeds)
  (:action drop
    :parameters (?p - package ?l - location)
    :precondition (and (robot-at ?l) (holding ?p))
    :effect (and
      (package-at ?p ?l)
      (empty-hand)
      (not (holding ?p))))

  ;; TODO: Define validate-delivery action
  ;; Validates delivery but can fail (sensor error)
  (:action validate-delivery
    :parameters (?p - package ?l - location)
    :precondition (and
      (robot-at ?l)
      (package-at ?p ?l))
    :effect (oneof
      (delivered ?p)    ; success: package marked as delivered
      (and))            ; failure: not validated yet
)
```

Create `warehouse-problem-2.ppddl`:

```
(define (problem deliver-one-package)
  (:domain warehouse-delivery)

  (:objects
    storage corridor office kitchen - location
    box1 - package)

  (:init
    (robot-at storage)
    (package-at box1 storage)
    (empty-hand)
    (connected storage corridor)
    (connected corridor storage)
    (connected corridor office)
    (connected office corridor)
    (connected corridor kitchen)
    (connected kitchen corridor))

  (:goal (delivered box1))
)
```

Test:

```
./sp -d warehouse-domain.ppddl -p warehouse-problem-2.ppddl -c ff -v 2
```

Question 1.2a: What is the minimum number of actions needed if everything succeeds?
Question 1.2b: What is the maximum number of action attempts before success is guaranteed?
Question 1.2c: Identify all the retry loops in the policy.

Task 1.3: Visualize the Policy (10 min)

Generate and view the policy graph:

```
# Run Safe-Planner (generates .dot file)
./sp -d warehouse-domain.ppddl -p warehouse-problem-2.ppddl -c ff

# Convert to PNG
dot -Tpng policy.dot -o policy.png

# View interactively
xdot policy.dot
```

Task: Annotate the policy graph with:

- State nodes (what action is chosen)
- Success edges (green)
- Failure edges (red)
- Retry loops (circles)

Exercise 2: Multi-Package Delivery (30 min)

Scenario

The robot must now deliver **two packages** to different locations:

- `box1` → `office`
- `box2` → `kitchen`

Task 2.1: Define the Problem (10 min)

Create `warehouse-problem-3.ppddl`:

```
(define (problem deliver-two-packages)
  (:domain warehouse-delivery)

  (:objects
    storage corridor office kitchen - location
    box1 box2 - package)

  (:init
    (robot-at storage)
    (package-at box1 storage)
    (package-at box2 storage)
    (empty-hand)
    ;; TODO: Add connections
    )

  (:goal (and
    (delivered box1)
    (delivered box2)))
)
```

Question 2.1: Can the robot carry two packages at once? Why or why not?

Task 2.2: Run and Analyze (10 min)

```
./sp -d warehouse-domain.ppddl -p warehouse-problem-3.ppddl -c ff
```

Question 2.2a: How many actions are in the main plan path?
Question 2.2b: Which package is delivered first? Why?
Question 2.2c: Count the number of states in the policy.

Task 2.3: Add Robot Breakdown (10 min)

Extend the domain with a `broken` predicate:

```
(:predicates
  ;; ... existing predicates ...
  (broken)
  (has-tools))

;; Movement can now cause breakdown
(:action move
  :parameters (?from ?to - location)
  :precondition (and
    (robot-at ?from)
    (connected ?from ?to)
    (not (broken)))
  :effect (and
    (not (robot-at ?from))
    (oneof
      (robot-at ?to)              ; success
      (robot-at ?from)            ; failure: stays
      (and (robot-at ?from)       ; breakdown!
           (broken)))))

;; Add repair action
(:action repair
  :parameters ()
  :precondition (and (broken) (has-tools))
  :effect (not (broken)))
```

Question 2.3: How does adding breakdown change the policy size?

Exercise 3: Theoretical Analysis (15 min)

Question 3.1: Probability Calculations

Assume:

- $P(\text{move succeeds}) = 0.8$
- $P(\text{pick succeeds}) = 0.7$
- $P(\text{validate succeeds}) = 0.9$

a) What is the probability of delivering one package successfully on the first attempt (no retries)?
b) What is the expected number of `move` attempts to go from `storage` to `kitchen` (2 moves)?
c) Calculate the expected total number of actions to deliver one package.
Hint: Expected attempts for action with success probability p: $E = \frac{1}{p}$

Question 3.2: Comparison with Classical Planning

Aspect	Classical (PDDL)	Probabilistic (PPDDL)
Output		
Handling failures		
Optimality metric		
Execution		

Question 3.3: Policy Properties

For the policy generated in Exercise 1:
a) Is it **complete**? (defined for all reachable states)
b) Is it **strong cyclic**? (guarantees reaching the goal)
c) Is it **optimal**? (minimizes expected cost)

Bonus: Extensions (+10 points)

Bonus 1: Safe Alternative Actions

Add a `careful-move` action that never fails but takes more time:

```
(:action careful-move
  :parameters (?from ?to - location)
  :precondition (and (robot-at ?from) (connected ?from ?to))
  :effect (and
    (not (robot-at ?from))
    (robot-at ?to)))  ; always succeeds
```

How does this change the generated policy?

Bonus 2: Obstacle Zones

Add a predicate `(has-obstacle ?l)` for dangerous zones. The robot should:

- Use `careful-move` in obstacle zones
- Use regular `move` in clear zones

Bonus 3: Compare Planners

Run the same problem with different planners:

```
./sp -d warehouse-domain.ppddl -p warehouse-problem-2.ppddl -c ff
./sp -d warehouse-domain.ppddl -p warehouse-problem-2.ppddl -c fd
```

Compare:

- Planning time
- Number of states in policy
- Structure of the policy

Grading Rubric

Component	**Points**	**Time**
Exercise 1: Basic Delivery		**45 min**
Task 1.1: Navigation domain	15	15 min
Task 1.2: Package handling	20	20 min
Task 1.3: Visualization	10	10 min
Exercise 2: Multi-Package		**30 min**
Task 2.1: Problem definition	5	10 min
Task 2.2: Analysis	10	10 min
Task 2.3: Robot breakdown	10	10 min
Exercise 3: Theory		**15 min**
Question 3.1: Probabilities	15	5 min
Question 3.2: Comparison	5	5 min
Question 3.3: Properties	10	5 min
Total	**100**	**90 min**
Bonus: Extensions	+10	

Submission

Submit a ZIP file containing:

- All `.ppddl` files (domains and problems)
- `report.pdf` with:

  - Answers to all questions
  - Screenshots of generated policies
  - Analysis and discussion

- policy.png files for visualization

Filename: lastname_firstname_ppddl_lab.zip

Tips and Common Errors

Syntax Tips

- Always declare :requirements :strips :typing :non-deterministic
- Use oneof for non-deterministic effects (not probabilistic)
- Each object must have a type
- Connect locations bidirectionally for movement

Common Errors

- **"No plan found"**: Check preconditions are satisfiable
- **"Syntax error"**: Verify parentheses balance
- **Empty policy**: Ensure goal is reachable with given actions
- **Infinite loops**: Check for dead-end states

Debugging

```
# Verbose mode shows detailed compilation steps
./sp -d domain.ppddl -p problem.ppddl -c ff -v 2

# View intermediate compiled domains
ls compiled_*

# Check if FF can solve individual compiled domains
ff -o compiled_domain_0.pddl -f compiled_problem_0.pddl
```