Planning and Approximate Reasoning

Assignment 1

Planning: Mining Robot Task

19th October 2025

Omar MORALES

Introduction to the problem

The MineRob problem models the task of an autonomous mining robot operating in a structured environment represented as a grid. Each cell of the grid corresponds to a distinct area in a mine, where minerals may be present or depleted, and where one or more mine cars can be positioned. The robot's objective is to efficiently collect all minerals across the mine while correctly managing the position of the mine cars according to the final desired configuration.

In this environment, the robot can perform three main actions: **move** between adjacent areas, **collect** minerals from the area it currently occupies, and **push** mine cars into neighboring areas. The robot is capable of moving through areas occupied by a minecar, as it moves together with it during a push operation. However, when pushing a minecar, the destination area must be free to allow the car to be moved successfully. Movement between cells is constrained by adjacency relations (horizontal or vertical), ensuring realistic navigation within the mine grid.

Through these operations, the robot must plan a sequence of actions that transitions the initial state of the mine—where some areas contain minerals and mine cars—to a goal state in which all minerals have been extracted and the mine cars occupy predefined final locations.

This problem provides a simplified yet expressive scenario for testing classical planning techniques. It combines navigation, resource manipulation, and spatial reasoning within a single domain, requiring the planner to handle both movement coordination and dynamic environment changes caused by the robot's interactions with the mine cars. The domain and problem descriptions are formalized in PDDL (Planning Domain Definition Language), enabling the use of automated planners to generate optimal or near-optimal action plans for the MineRob agent.

Analysis of the problem

The MineRob problem represents a classical planning task defined in PDDL, where the goal is to control a mining robot in a discrete environment represented as a 3×3 grid (nine areas). The environment may include a variable number of mine cars. The robot can move, collect minerals, and push mine cars into adjacent cells. This section analyzes the structure of the domain and problem in terms of its main planning components: predicates, operators, objects, and search space.

Predicates

The domain defines six main predicates that describe the state of the environment:

- **(robot-location ?x):** indicates the current position of the robot in a given area.
- **(minecar-location ?m ?x):** represents the position of a specific mine car within the grid.
- **(minerals ?x):** denotes that the area contains minerals to be collected.
- **(depleted ?x):** indicates that an area no longer contains minerals.
- **(empty ?x):** specifies that an area has no mine car present.
- **(adjacent ?x1 ?x2):** defines adjacency relations between areas, allowing movement and pushing actions only between directly connected cells.

These predicates jointly capture both static relationships (such as adjacency) and dynamic properties (such as the positions of the robot and mine cars) that evolve as actions are executed.

Objects

The problem instance defines the following objects:

- **Areas:** `a1` to `a9`, representing the 3×3 grid cells.
- **Robot:** represented through the `robot-location` predicate.
- **Mine cars:** represented as `mine_car_*`, where `*` is replaced by an uppercase letter (e.g., `mine_car_A`, `mine_car_B`, `mine_car_C`) to indicate each car's unique identity. The domain allows for a variable number of mine cars, from zero (no mine cars in the mine) up to nine (one per area, excluding the robot's position).

Search Space *(updated for flexibility)*

The **search space** consists of all possible configurations of the robot, mine cars, and minerals within the grid, constrained by adjacency and occupancy rules.
In a 3×3 mine with k mine cars ($0 \leq k \leq 8$), the number of possible configurations increases exponentially with k, since each minecar adds an additional movable element to the state description.

The planner must search this space to find a valid sequence of actions that satisfies the goal conditions:

- All areas are depleted of minerals,
- All mine cars occupy their designated goal positions, and
- The robot is in its target final area.

This flexibility enables testing of different configurations and complexity levels, as the number of movable objects directly impacts the branching factor of the search space.

PDDL Implementation with comments

Domain

```
(define (domain minerob)
   (:requirements :strips :equality :adl)

 (:predicates
   (robot-location ?x)
   (minecar-location ?x1 ?x2)
   (minerals ?x)
   (depleted ?x)
   (empty ?x)
   (adjacent ?x1 ?x2)
 )

 ;; Action: Move
 (:action move
   :parameters (?from ?to)
   :precondition (and
     (robot-location ?from)
     ;; The robot can only move to adjacent areas.
     (adjacent ?from ?to)
```

```
    ;; The robot moves with the minecar when pushing, so it can pass through an area
occupied by the minecar.
   )
   :effect (and
     ;; The robot moves from ?from to ?to.
     (not (robot-location ?from))
     (robot-location ?to)
   )
 )

 ;; Action: Collect minerals
 (:action collect
   :parameters (?a)
   :precondition (and
     (robot-location ?a)
     (minerals ?a)
   )
   :effect (and

     (not (minerals ?a))
     (depleted ?a)
   )
 )

 ;; Action: Push minecar
 ;; Robot moves with the minecar to its new position.
 (:action push
   :parameters (?m ?from ?to)
   :precondition (and
     ;; Robot and minecar must be in the same position.
     (robot-location ?from)
     (minecar-location ?m ?from)
     (adjacent ?from ?to)
     (empty ?to)  ;; Minecar can only be pushed to empty areas.
   )
   :effect (and
     (empty ?from)
     (not (empty ?to))
     ;; Robot follows the minecar
     (not (minecar-location ?m ?from))
     (minecar-location ?m ?to)
     (not (robot-location ?from))
     (robot-location ?to)
   )
 )
)
```

Initial Problem

```
(define (problem minerob-example-1)
 (:domain minerob)

 (:objects
   a1 a2 a3 a4 a5 a6 a7 a8 a9 robot mine_car_A mine_car_B mine_car_C
```

```
)

(:init
  ;; Adjacencies (horizontal and vertical)
  (adjacent a1 a2) (adjacent a2 a1)
  (adjacent a1 a4) (adjacent a4 a1)
  (adjacent a2 a3) (adjacent a3 a2)
  (adjacent a2 a5) (adjacent a5 a2)
  (adjacent a3 a6) (adjacent a6 a3)
  (adjacent a4 a5) (adjacent a5 a4)
  (adjacent a4 a7) (adjacent a7 a4)
  (adjacent a5 a6) (adjacent a6 a5)
  (adjacent a5 a8) (adjacent a8 a5)
  (adjacent a6 a9) (adjacent a9 a6)
  (adjacent a7 a8) (adjacent a8 a7)
  (adjacent a8 a9) (adjacent a9 a8)

  ;; Initial positions
  (robot-location a7)
  (minecar-location mine_car_A a1)
  (minecar-location mine_car_B a6)
  (minecar-location mine_car_C a9)

  ;; This example does not have minerals
  (depleted a1) (depleted a2) (depleted a3) (depleted a4) (depleted a5)
  (depleted a6) (depleted a7) (depleted a8) (depleted a9)

  ;; empty areas (no minecar)
  (empty a2) (empty a3) (empty a4)
  (empty a5) (empty a7) (empty a8)
)

(:goal
  (and
    ;; All areas will be empty of minerals.
    (depleted a1) (depleted a2) (depleted a3)
    (depleted a4) (depleted a5) (depleted a6)
    (depleted a7) (depleted a8) (depleted a9)
    ;; Final position of the mine cars.
    (minecar-location mine_car_A a2)
    (minecar-location mine_car_B a3)
    (minecar-location mine_car_C a7)
     ;; Final position of the robot.
    (robot-location a4)
  )
)
)
```

Description of the selected planner

BFWS (Best-First Width Search) is a planning algorithm that combines elements of best-first search with the concept of novelty-based exploration. Unlike traditional heuristics that focus only on cost or distance to the goal, BFWS evaluates states based on how "novel" they are—measuring the appearance of new features not seen in previous states—while still incorporating heuristic guidance. This allows it to efficiently explore large search spaces with minimal node expansions, quickly finding low-cost solutions even in complex domains. In the initial problem, BFWS is particularly advantageous because it can find a solution plan with very low cost (11) extremely quickly, making it ideal when computational efficiency is critical or when the search space is large.

ENHSP (Expressive Numeric Heuristic Search Planner), on the other hand, is a forward-chaining heuristic planner designed to handle domains with numeric variables and complex action costs. It uses numeric heuristics to evaluate the desirability of each state, guiding the search toward cost-effective and feasible plans. ENHSP produces a fully detailed, executable action sequence, making it highly suitable for domains where understanding the exact steps is important. In the initial problem, it provides a clear 13-step plan with explicit actions for moving and pushing objects, ensuring that the plan can be directly implemented and verified.

Together, these planners offer complementary strengths: BFWS excels at rapidly finding low-cost solutions, while ENHSP provides clear, actionable plans with detailed step-by-step guidance. That's why I chose these two planners to run our test cases.

Testing cases and results

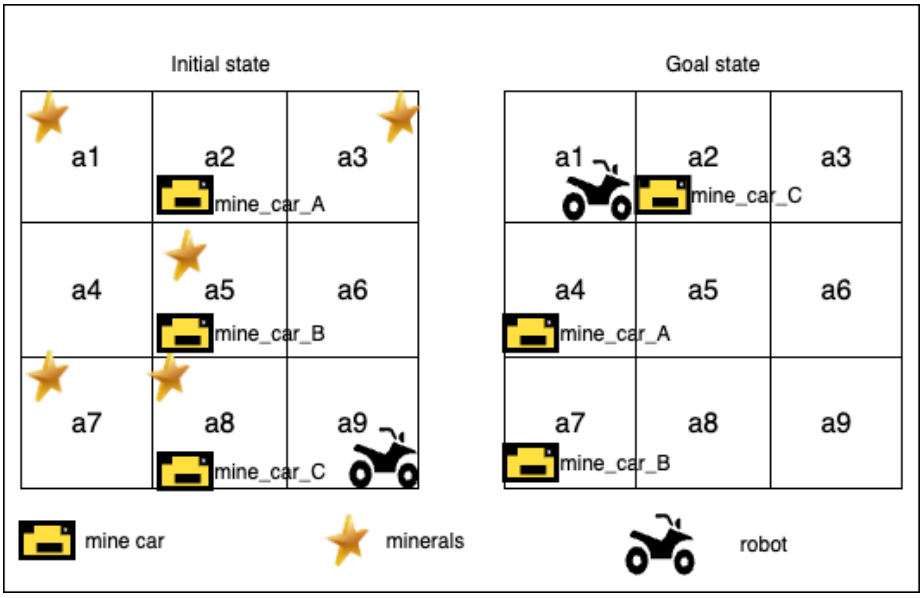
PDDL Editor web was used in this assignment.

Initial problem (problem1.pddl)

BFWS	ENHSP
--- OK. Match tree built with 96 nodes. PDDL problem description loaded: Domain: MINEROB Problem: MINEROB-EXAMPLE-1 #Actions: 96 #Fluents: 45 Goals found: 4 Goals_Edges found: 4 Starting search with 1-BFWS... --[4 / 0]-- --[3 / 0]-- --[3 / 1]-- --[3 / 2]-- --[3 / 4]-- --[2 / 0]-- --[2 / 1]-- --[2 / 2]-- --[2 / 3]-- --[2 / 5]-- --[1 / 0]-- --[1 / 2]-- --[1 / 3]-- --[1 / 5]-- --[1 / 7]-- --[1 / 8]-- --[0 / 0]-- --[0 / 1]-- --[0 / 3]-- Total time: 0.000286	domain parsed problem parsed grounding.. grounding time: 38 aibr preprocessing |f|:45 |x|:0 |a|:96 |p|:0 |e|:0 h1 setup time (msec): 11 g(n)= 1.0 h(n)=12.0 g(n)= 2.0 h(n)=11.0 g(n)= 3.0 h(n)=10.0 g(n)= 4.0 h(n)=8.0 g(n)= 5.0 h(n)=5.0 g(n)= 8.0 h(n)=3.0 g(n)= 11.0 h(n)=2.0 g(n)= 12.0 h(n)=1.0 problem solved found plan: 0.0: (move a7 a8) 1.0: (move a8 a9) 2.0: (push mine_car_c a9 a8) 3.0: (push mine_car_c a8 a7) 4.0: (move a7 a4) 5.0: (move a4 a1) 6.0: (push mine_car_a a1 a2) 7.0: (move a2 a5) 8.0: (move a5 a6)

Nodes generated during search: 64 Nodes expanded during search: 21 Plan found with cost: 11 Fast-BFS search completed in 0.000286 secs	9.0: (push mine_car_b a6 a3) 10.0: (move a3 a2) 11.0: (move a2 a1) 12.0: (move a1 a4) plan-length:13 metric (search):13.0 planning time (msec): 44 heuristic time (msec): 16 search time (msec): 39 expanded nodes:21 states evaluated:58 number of dead-ends detected:0 number of duplicates detected:26

Second problem (problem2.pddl)



BFWS	ENHSP
--- OK. Match tree built with 101 nodes. PDDL problem description loaded: Domain: MINEROB Problem: MINEROB-EXAMPLE-1 #Actions: 101 #Fluents: 55 Goals found: 9 Goals_Edges found: 9 Starting search with 1-BFWS... --[9 / 0]--	domain parsed problem parsed grounding.. grounding time: 39 aibr preprocessing |f|:55 |x|:0 |a|:101 |p|:0 |e|:0 h1 setup time (msec): 12 g(n)= 1.0 h(n)=35.0

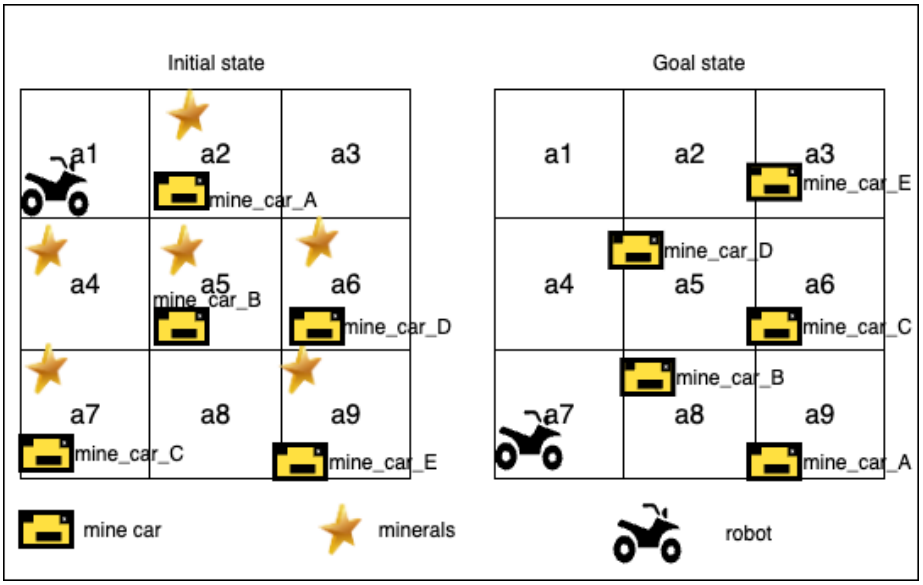
--[9 / 1]--
--[9 / 2]--
--[9 / 4]--
--[8 / 0]--
--[8 / 2]--
--[8 / 3]--
--[8 / 4]--
--[7 / 0]--
--[7 / 2]--
--[7 / 3]--
--[6 / 0]--
--[6 / 2]--
--[6 / 4]--
--[5 / 0]--
--[5 / 1]--
--[5 / 3]--
--[4 / 0]--
--[4 / 3]--
--[4 / 4]--
--[4 / 5]--
--[4 / 7]--
--[3 / 0]--
--[3 / 2]--
--[3 / 4]--
--[3 / 5]--
--[3 / 7]--
--[2 / 0]--
--[2 / 2]--
--[2 / 4]--
--[2 / 5]--
--[2 / 7]--
--[1 / 0]--
--[1 / 1]--
--[1 / 3]--
--[1 / 4]--
--[1 / 5]--
--[1 / 7]--
--[0 / 0]--
--[0 / 2]--
--[0 / 4]--
Total time: 0.000577
Nodes generated during search: 166
Nodes expanded during search: 54
Plan found with cost: 23
Fast-BFS search completed in 0.000577 secs

g(n)= 2.0 h(n)=27.0
g(n)= 3.0 h(n)=26.0
g(n)= 5.0 h(n)=25.0
g(n)= 8.0 h(n)=19.0
g(n)= 11.0 h(n)=17.0
g(n)= 12.0 h(n)=15.0
g(n)= 14.0 h(n)=13.0
g(n)= 16.0 h(n)=11.0
g(n)= 17.0 h(n)=10.0
g(n)= 20.0 h(n)=9.0
g(n)= 21.0 h(n)=8.0
g(n)= 22.0 h(n)=7.0
g(n)= 23.0 h(n)=6.0
g(n)= 23.0 h(n)=5.0
g(n)= 24.0 h(n)=3.0
g(n)= 27.0 h(n)=2.0
g(n)= 28.0 h(n)=1.0
problem solved

found plan:
0.0: (move a9 a8)
1.0: (move a8 a5)
2.0: (collect a5)
3.0: (push mine_car_b a5 a4)
4.0: (move a4 a1)
5.0: (collect a1)
6.0: (move a1 a2)
7.0: (push mine_car_a a2 a5)
8.0: (move a5 a4)
9.0: (push mine_car_b a4 a7)
10.0: (move a7 a8)
11.0: (move a8 a5)
12.0: (push mine_car_a a5 a4)
13.0: (move a4 a5)
14.0: (move a5 a8)
15.0: (push mine_car_c a8 a5)
16.0: (push mine_car_c a5 a2)
17.0: (move a2 a5)
18.0: (move a5 a8)
19.0: (collect a8)
20.0: (move a8 a7)
21.0: (collect a7)
22.0: (move a7 a4)
23.0: (move a4 a1)
24.0: (move a1 a2)
25.0: (move a2 a3)
26.0: (collect a3)
27.0: (move a3 a2)
28.0: (move a2 a1)

plan-length:29
metric (search):29.0
planning time (msec): 66
heuristic time (msec): 30
search time (msec): 61
expanded nodes:39
states evaluated:126
number of dead-ends detected:0
number of duplicates detected:44

Third problem (problem3.pddl)



BFWS	ENHSP
--- OK. Match tree built with 150 nodes. PDDL problem description loaded: Domain: MINEROB Problem: MINEROB-EXAMPLE-3 #Actions: 150 #Fluents: 75 Goals found: 12 Goals_Edges found: 12 Starting search with 1-BFWS... --[12 / 0]-- --[12 / 1]-- --[12 / 2]-- --[12 / 4]-- --[11 / 0]-- --[11 / 2]-- --[11 / 3]-- --[11 / 4]-- --[10 / 0]-- --[10 / 4]-- --[9 / 0]-- --[9 / 1]-- --[9 / 3]-- --[8 / 0]-- --[8 / 1]-- --[8 / 3]-- --[7 / 0]-- --[7 / 1]-- --[7 / 3]--	domain parsed problem parsed grounding.. grounding time: 41 aibr preprocessing |f|:75 |x|:0 |a|:150 |p|:0 |e|:0 h1 setup time (msec): 16 g(n)= 1.0 h(n)=57.0 g(n)= 2.0 h(n)=44.0 g(n)= 3.0 h(n)=43.0 g(n)= 6.0 h(n)=41.0 g(n)= 7.0 h(n)=35.0 g(n)= 10.0 h(n)=33.0 g(n)= 12.0 h(n)=31.0 g(n)= 15.0 h(n)=29.0 g(n)= 19.0 h(n)=27.0 g(n)= 20.0 h(n)=26.0 g(n)= 22.0 h(n)=20.0 g(n)= 23.0 h(n)=18.0 g(n)= 27.0 h(n)=16.0 g(n)= 28.0 h(n)=15.0 g(n)= 32.0 h(n)=12.0 g(n)= 44.0 h(n)=11.0 g(n)= 47.0 h(n)=10.0 g(n)= 55.0 h(n)=8.0 g(n)= 56.0 h(n)=7.0

--[6 / 0]--
--[6 / 2]--
--[6 / 3]--
--[6 / 4]--
--[5 / 0]--
--[5 / 3]--
--[4 / 0]--
--[4 / 2]--
--[4 / 3]--
--[4 / 4]--
--[3 / 0]--
--[3 / 2]--
--[3 / 3]--
--[3 / 4]--
--[3 / 5]--
--[3 / 6]--
--[2 / 0]--
--[2 / 4]--
--[2 / 5]--
--[2 / 6]--
--[2 / 8]--
--[2 / 9]--
--[2 / 10]--
--[2 / 11]--
--[2 / 12]--
--[2 / 13]--
--[2 / 14]--
--[2 / 15]--
--[1 / 0]--
--[1 / 1]--
--[1 / 2]--
--[1 / 4]--
--[1 / 5]--
--[1 / 6]--
--[1 / 7]--
--[1 / 9]--
--[1 / 10]--
--[0 / 0]--
--[0 / 1]--
Total time: 0.003156
Nodes generated during search: 693
Nodes expanded during search: 369
Plan found with cost: 46
Fast-BFS search completed in 0.003156 secs

g(n)= 57.0 h(n)=6.0
g(n)= 58.0 h(n)=3.0
g(n)= 60.0 h(n)=2.0
g(n)= 63.0 h(n)=1.0
problem solved

found plan:
0.0: (move a1 a4)
1.0: (move a4 a5)
2.0: (collect a5)
3.0: (push mine_car_b a5 a8)
4.0: (move a8 a5)
5.0: (move a5 a6)
6.0: (push mine_car_d a6 a5)
7.0: (move a5 a6)
8.0: (collect a6)
9.0: (move a6 a9)
10.0: (push mine_car_e a9 a6)
11.0: (move a6 a5)
12.0: (push mine_car_d a5 a4)
13.0: (collect a4)
14.0: (push mine_car_d a4 a5)
15.0: (move a5 a8)
16.0: (move a8 a9)
17.0: (collect a9)
18.0: (move a9 a6)
19.0: (push mine_car_e a6 a3)
20.0: (move a3 a2)
21.0: (collect a2)
22.0: (move a2 a5)
23.0: (push mine_car_d a5 a4)
24.0: (move a4 a7)
25.0: (collect a7)
26.0: (move a7 a4)
27.0: (push mine_car_d a4 a5)
28.0: (move a5 a8)
29.0: (move a8 a7)
30.0: (push mine_car_c a7 a4)
31.0: (move a4 a5)
32.0: (move a5 a8)
33.0: (push mine_car_b a8 a7)
34.0: (move a7 a4)
35.0: (move a4 a5)
36.0: (push mine_car_d a5 a8)
37.0: (move a8 a5)
38.0: (move a5 a4)
39.0: (push mine_car_c a4 a5)
40.0: (push mine_car_c a5 a6)
41.0: (move a6 a5)
42.0: (move a5 a8)
43.0: (push mine_car_d a8 a5)
44.0: (push mine_car_d a5 a4)
45.0: (move a4 a7)
46.0: (push mine_car_b a7 a8)
47.0: (move a8 a5)
48.0: (move a5 a2)
49.0: (push mine_car_a a2 a5)
50.0: (move a5 a8)
51.0: (push mine_car_b a8 a7)
52.0: (move a7 a4)
53.0: (move a4 a5)
54.0: (push mine_car_a a5 a8)
55.0: (push mine_car_a a8 a9)

	56.0: (move a9 a8) 57.0: (move a8 a7) 58.0: (push mine_car_b a7 a8) 59.0: (move a8 a7) 60.0: (move a7 a4) 61.0: (push mine_car_d a4 a5) 62.0: (move a5 a4) 63.0: (move a4 a7) plan-length:64 metric (search):64.0 planning time (msec): 119 heuristic time (msec): 76 search time (msec): 113 expanded nodes:138 states evaluated:406 number of dead-ends detected:0 number of duplicates detected:190

Analysis of the results

Initial problem

BFWS uses a novelty-based search strategy to explore states with minimal expansion while considering heuristic guidance. For this problem, BFWS generated 64 nodes and expanded 21 nodes, finding a plan with a cost of 11 in just 0.000286 seconds. This demonstrates its high efficiency in terms of search speed and minimal computational effort. The plan cost is low. The search complexity appears very manageable, as evidenced by the small number of nodes generated and expanded.

ENHSP, in contrast, employs numeric heuristics to guide forward-chaining search toward cost-effective plans. ENHSP produced a 13-step plan with an equivalent cost metric of 13, expanding 21 nodes and evaluating 58 states, taking 44 milliseconds for planning. Although the computational time is slightly higher than BFWS, ENHSP provides a detailed and fully executable plan, including precise sequences of moves and pushes, which is essential for implementation in the MineRob scenario. The planner also reports information about dead-ends and duplicate states, which can be valuable for understanding search dynamics.

Comparison:

- **Plan Cost:** BFWS produced a slightly lower-cost plan (11) compared to ENHSP (13), reflecting its focus on cost efficiency.

- **Plan Detail:** ENHSP provides explicit, actionable steps, whereas BFWS primarily reports cost and abstract search progress.

- **Nodes Generated/Expanded:** Both planners expanded the same number of states (21), but BFWS generated 64 states while ENHSP evaluated 58 states during its heuristic-guided search. This shows that BFWS explored slightly more states overall, while ENHSP focused on evaluating states in more detail.

- **Time Efficiency:** BFWS is significantly faster (0.286ms vs. 44ms), making it ideal for large search spaces or real-time scenarios.

Second problem

For this problem, BFWS generated 166 states and expanded 54 states, finding a plan with a cost of 23 in only 0.000577 seconds. Although the number of generated and expanded states increased compared to the first test case, BFWS still maintains a very low computational time, demonstrating its strength in quickly finding low-cost solutions even as the problem size grows.

In this case, ENHSP produced a **29-step plan** with a cost metric of 29, **expanded 39 states**, and **evaluated 126 states** in **66 milliseconds**. The increase in states evaluated compared to BFWS reflects the planner's more heuristic-guided exploration.

Comparison:

- **Plan cost:** BFWS produced a lower-cost plan (23 vs. 29 for ENHSP).

- **Nodes/states:** BFWS generated more states (166 vs. 126 evaluated by ENHSP) and expanded more states (54 vs. 39).

- **Time efficiency:** BFWS is significantly faster (0.577ms vs. 66ms), maintaining high efficiency even for a larger problem.

Third problem

For this problem, BFWS generated 693 states and expanded 369 states, finding a plan with a cost of 46 in 0.003156 seconds. The increase in states generated and expanded compared to previous test cases reflects the larger number of actions, fluents, and goals in this scenario. Despite the increased search space, BFWS maintains extremely fast planning time, demonstrating strong scalability for cost-focused search.

ENHSP produced a 64-step plan with a cost metric of 64, expanded 138 states, and evaluated 406 states in 119 milliseconds. The higher number of evaluated states shows the planner's thorough heuristic-guided exploration of the search space, which requires more computation than BFWS but successfully handles the larger problem instance.

Comparison:

- **Plan cost:** BFWS found a lower-cost plan (46 vs. 64 for ENHSP).
- **Nodes/states:** BFWS generated more states (693 vs. 406 evaluated by ENHSP) and expanded more states (369 vs. 138), indicating a broader exploration of the search space.
- **Time efficiency:** BFWS remains significantly faster (3.156ms vs. 119ms), showing its efficiency even for large, complex problems.

Conclusion

Analyzing the three problems highlights how BFWS and ENHSP perform as problem complexity increases. Across all test cases, BFWS consistently finds plans with lower cost (11, 23, 46) and extremely fast planning times, demonstrating its efficiency in rapidly exploring large search spaces. However, BFWS generates and expands more states than ENHSP in all cases (e.g., 64 vs. 58, 166 vs. 126, 693 vs. 406 generated states), which implies that its memory usage is higher, since more states need to be stored in the search tree during the search. This broad exploration strategy enables BFWS to maintain speed and cost efficiency but at the expense of higher memory consumption.

ENHSP, by contrast, performs a more heuristic-guided search, generating and evaluating fewer states and expanding fewer nodes (e.g., 21 vs. 21, 39 vs. 54, 138 vs. 369 expanded states) across the three problems. While this results in slightly higher planning times, ENHSP efficiently manages the search space and is able to scale to larger problem instances with more actions, fluents, and goals. Its selective evaluation strategy typically reduces memory usage compared to BFWS.

Overall, BFWS is most advantageous when speed and low plan cost are the main priorities, even if it requires more memory due to the larger number of states generated and expanded. ENHSP is more suitable when controlled memory usage and heuristic-guided exploration are important, and it remains effective even as problem complexity grows. Both planners successfully solve all three scenarios, demonstrating scalability, reliability, and distinct strengths depending on whether rapid search or heuristic-guided efficiency is preferred.