



UNIVERSITAT
ROVIRA I VIRGILI

2025-2026 Q1

PAR: MiniRob

ALTÉS GRIFOLL, BIEL
FINELLI, ZOË

Universitat Rovira i Virgili

Contents

1	Introduction	1
2	Analysis of the problem	1
2.1	Environment Structure	1
2.2	Robot Actions	2
2.3	Problem Constraints	2
2.4	Goal Requirements	2
3	PDDL implementation	2
3.1	Domain Overview	2
3.2	Problem Instance	3
3.3	Problem Generator	4
4	Testing cases and results	4
4.1	Test case 1 (2x2)	4
4.1.1	Problem definition	4
4.1.2	Solver results	5
4.2	Test case 2 (3x3)	6
4.2.1	Problem definition	6
4.2.2	Solver results	7
4.3	Test case 3 (4x4)	9
4.3.1	Problem definition	9
4.3.2	Solver results	10
5	Conclusion	15

1 Introduction

Planning is a fundamental part of artificial intelligence that allows agents to make decisions and solve problems in complex environments. In this exercise, we work with automated planning techniques to solve the MineRob mining robot problem using PDDL (Planning Domain Definition Language).

The mining robot domain represents a typical planning scenario where we need to coordinate different actions to achieve specific goals. Our robot operates in a grid-based mine where it must collect minerals from different areas while managing mine cars that can be moved around the environment. This type of problem is interesting because it combines movement planning with object manipulation tasks.

PDDL is the standard language used in the planning community to describe planning problems. It allows us to separate the domain definition (what actions are possible and how they work) from the specific problem instance (the initial state and goals we want to achieve). This separation makes it easier to test different scenarios and to use automated planners to find solutions.

The main challenge in this problem is to design the domain correctly so that the robot can perform all necessary actions: moving between areas, collecting minerals, and pushing the mine cars to their final positions. We need to think carefully about the preconditions and effects of each action to avoid conflicts and ensure that the planner can find valid solutions.

In this report, we present the complete PDDL implementation for the mining robot domain and analyze how different problem instances can be solved. We also discuss the design decisions made during the development process and evaluate the performance of the solution.

2 Analysis of the problem

The MineRob mining robot problem combines spatial reasoning with resource management and object manipulation. In the following, we explain our solution and the problems that we encountered.

2.1 Environment Structure

The mine consists of a grid with $n \times n$ areas arranged in rows and columns. Each area connects to its horizontal and vertical neighbors, but diagonal movement is not allowed. This creates a simple but constrained movement space.

Areas can be in different states:

- Areas with minerals that need to be collected
- Depleted areas where minerals have already been removed
- Areas occupied by mine cars

- Empty areas where the robot can move freely

2.2 Robot Actions

We decided to stick with three actions for the robot. The `collect` action allows mineral extraction from the current area. Its preconditions require the robot to be in the area and minerals to be present. The effect removes minerals and marks the area as depleted. The `move` action handles basic robot movement between adjacent areas. However, the robot can only move to empty areas, which prevents conflicts with mine cars. When the robot moves, its location updates accordingly.

The most complex is the `push` action. This combines movement with object manipulation. The robot pushes a mine car to an adjacent empty area and moves with it. The preconditions check for robot position, mine car location, adjacency, and empty destination. The effects update both robot and mine car positions while maintaining area occupancy states.

2.3 Problem Constraints

Several important constraints shape this problem. The robot cannot move to areas occupied by mine cars unless it pushes them first. Mine cars cannot be placed in the same area, which means we need careful coordination. Also, the robot must visit every area with minerals to complete the task.

The number of mine cars can vary from zero to $n^2 - 1$, which affects the complexity significantly. More mine cars mean more obstacles and more complex planning.

2.4 Goal Requirements

The final state requires two conditions to be met. All minerals must be collected from every area in the grid. Additionally, all mine cars must be positioned in their designated final locations.

This dual objective makes the problem challenging because the robot needs to balance mineral collection with mine car positioning. Sometimes it might be necessary to move mine cars multiple times to access certain areas.

3 PDDL implementation

3.1 Domain Overview

The proposed domain, `mining-robot`, models a simplified mine exploration environment where a robotic agent navigates a grid, collects minerals, and relocates minecars to target positions.

The environment is composed of discrete areas forming a grid. Each area can contain a mineral deposit, a minecar, or be empty. Logical relationships such as adjacency between areas are explicitly encoded to support movement reasoning.

The defined predicates describe the current situational configuration of the grid:

- `(robot-location a?)`: the robot's current position.
- `(minecar m?-location a?)`: the location of each minecar.
- `(mineral a?)` and `(depleted a?)`: model the resource state of each area.
- `(empty a?)`: tracks available cells for movement or pushing.
- `(adjacent a? a?)`: defines possible transitions between adjacent cells.

Three fundamental actions proposed:

move enables traversal between adjacent areas.

collect allows the robot to gather a mineral, marking the cell as depleted.

push lets the robot move a minecar to an adjacent empty cell, updating emptiness and location predicates accordingly.

3.2 Problem Instance

Each problem instance defines a specific configuration of the mining environment that the planner must solve under the rules established by the domain. All instances share a common logical structure, composed of the following main components:

- **Object Declaration**: the complete set of areas, minecars, and other entities that exist in the scenario. Areas define the discrete spatial grid, while minecars and minerals represent movable and collectible elements in the environment.
- **Initial State**: a list of predicates that describe which facts are true at the start of the problem — such as the robot's initial position, the distribution of minecars and minerals, or adjacency relations between cells. Everything not listed is assumed to be false by the closed-world assumption.
- **Goal Definition**: a conjunction of logical facts that must hold after executing the plan. Typically, these include all target minecar placements, depletion of all mineral sites, and the robot's final position.
- **Consistency Constraints**: every instance must ensure adjacency coherence, unique occupancy of each area, and that all moving entities start and end in valid grid locations.

3.3 Problem Generator

To allow controlled testing with varying grid dimensions and difficulty, a Python problem generator was developed. The script procedurally produces randomized PDDL problem files while ensuring logical consistency between adjacency relations, robot placement, and resource distribution.

The generator follows these conceptual steps:

1. *Grid construction*: creates unique identifiers for cells (**a11**, **a12**, ...) and all bidirectional adjacency links.
2. *Entity assignment*: randomly places minecars, minerals, and empty cells under the constraint of exclusiveness.
3. *Initial and goal configuration*: initializes predicates reflecting the world state and automatically synthesizes a goal in which:
 - all minerals have been converted into **depleted** areas,
 - minecars occupy fixed target positions (e.g., corners of the map),
 - the robot moves toward a designated goal cell.
4. *File generation*: writes the complete PDDL structure to disk with a human-readable name such as **problem_3x3.pddl**.

4 Testing cases and results

This part shows the tests we did to see how well the planning model works. We tried different sizes and setups to check if the robot can complete its tasks correctly. The goal was to find out how the complexity affects the solution and how efficient the plans are. The solver used to run the tests was BFWS, the FF parser version available in <https://editor.planning.domains/>.

4.1 Test case 1 (2x2)

4.1.1 Problem definition

This is our first test case with a 2×2 grid. We have four areas: **a11**, **a12**, **a21**, and **a22**. There is only one mine car (**m1**) and one mineral deposit, so it is a very simple example to start with.

Initial state:

- Robot starts at area **a11**
- Mine car **m1** is at area **a12**

- Area **a21** has a mineral
- Areas **a11**, **a21**, and **a22** are empty
- All areas are connected to their neighbors (up, down, left, right)

Goal state:

- Mine car **m1** should be at area **a11**
- The mineral at **a21** should be collected (area becomes depleted)

This problem is very easy because the robot only needs to move the mine car and collect one mineral. It is a good example to show how the planning works, but it is not very challenging.

4.1.2 Solver results

BFWS results:

The BFWS solver found a solution with 4 actions, as we can see in Figure 1:

```
(move a11 a12)
(push m1 a12 a11)
(move a11 a21)
(collect a21)
*
```

Figure 1: BFWS solver results for the 2×2 mining robot problem

The plan that the solver found is:

1. (move a11 a12) - Robot goes from **a11** to **a12**
2. (push m1 a12 a11) - Robot pushes mine car **m1** from **a12** to **a11**
3. (move a11 a21) - Robot goes from **a11** to **a21**
4. (collect a21) - Robot collects the mineral at **a21**

This solution is clearly complete because the robot does exactly what it needs to do to fulfill all preconditions. First, it moves the mine car to the right place, then it goes to collect the mineral. With only 4 actions, it is super easy to plan and understand. This was just our first problem as an example, but the real challenges come with bigger grids and more mine cars.

Delfi results:

The Delfi solver found a solution with the same 4 actions, as we can see in Figure 2:

```
move a11 a12 (1)
push m1 a12 a11 (1)
move a11 a21 (1)
* collect a21 (1)
```

Figure 2: Delfi solver results for the 2×2 mining robot problem

This solution is also optimal because the Delfi solver found the same solution. We know this because the Delfi solve utilizes a variation of the A* algorithm, which is complete and optimal.

The difference in planners:

The biggest difference between the results of these planners is that BFWS took 0.000125999 seconds to find this plan and Delfi took 0.511 seconds to find the same plan. BFWS also generated 15 nodes for this search, expanding 5 nodes, and resulted with a cost of 4. However, Delfi generated 90 nodes, but did not generate any states that were counted because it is a symbolic planner that relies on exploring large sets of states. Thus, its "classical" node counter remained at 0. The cost for Delfi was also 4.

4.2 Test case 2 (3x3)

4.2.1 Problem definition

This is our second test case with a 3x3 grid. We have 9 areas: a11, a12, a13, a21, a22, a23, a31, a32, and a33. There are two mine cars (m1 and m2) and three mineral deposits, so it is a more complex example.

Initial state:

- Robot starts at area a11
- Mine car m1 is at area a23 and mine car m2 is at area a31
- Areas a12, a33, and a32 have a mineral
- Areas a11, a13, a21, and a22 are empty
- All areas are connected to their neighbors (up, down, left, right)

Goal state:

- Mine car **m1** should be at area **a11** and mine car **m2** should be at area **a13**
- The minerals at **a12**, **a33**, and **a32** should be collected (area becomes depleted)

4.2.2 Solver results

BFWS results:

The BFWS solver found a solution with 17 actions, as we can see in Figure 3:

```
(move a11 a12)
(collect a12)
(move a12 a22)
(move a22 a32)
(collect a32)
(move a32 a33)
(collect a33)
(move a33 a23)
(push m1 a23 a22)
(push m1 a22 a21)
(push m1 a21 a11)
(move a11 a21)
(move a21 a31)
(push m2 a31 a21)
(push m2 a21 a22)
(push m2 a22 a23)
* (push m2 a23 a13)
```

Figure 3: BFWS solver results for the 3x3 mining robot problem

The plan that the solver found is:

1. Robot moves from **a11** to **a12**
2. Robot collects the mineral at **a12**
3. Robot moves from **a12** to **a22**
4. Robot moves from **a22** to **a32**

5. Robot collects the mineral at a32
6. Robot moves from a32 to a33
7. Robot collects the mineral at a33
8. Robot moves from a33 to a23
9. Robot pushes mine car m1 from a23 to a22
10. Robot pushes mine car m1 from a22 to a21
11. Robot pushes mine car m1 from a21 to a11
12. Robot moves from a11 to a21
13. Robot moves from a21 to a31
14. Robot pushes mine car m2 from a31 to a21
15. Robot pushes mine car m2 from a21 to a22
16. Robot pushes mine car m2 from a22 to a23
17. Robot pushes mine car m2 from a23 to a13

This solution is clearly complete because the robot does exactly what it needs to do to fulfill all preconditions. Even with adding an extra mine car and more minerals, the robot is still finding a complete solution.

Delfi results:

The Delfi solver found a solution with the same 17 actions, as we can see in Figure 3:

```

move a11 a12 (1)
collect a12 (1)
move a12 a22 (1)
move a22 a32 (1)
collect a32 (1)
move a32 a33 (1)
collect a33 (1)
move a33 a23 (1)
push m1 a23 a22 (1)
push m1 a22 a21 (1)
push m1 a21 a11 (1)
move a11 a21 (1)
move a21 a31 (1)
push m2 a31 a21 (1)
push m2 a21 a22 (1)
push m2 a22 a23 (1)
* push m2 a23 a13 (1)

```

Figure 4: Delfi solver results for the 3x3 mining robot problem

This solution is also optimal because the Delfi solver found the same solution. We know this because the Delfi solve utilizes a variation of the A* algorithm, which is complete and optimal.

The difference in planners:

The biggest difference between the results of these planners is that BFWS took 0.001104 seconds to find this plan and Delfi took 0.0468642 seconds to find the same plan. BFWS also generated 47 nodes, expanding 18 nodes, for this search, and resulted with a cost of 17. However, Delfi generated 105 nodes and 66 states, expanded 18 states, with a cost of 17. This is a large difference in states from our first problem, demonstrating how much the change in complexity affected Delfi's planning process.

4.3 Test case 3 (4x4)

4.3.1 Problem definition

This is our third test case with a 4x4 grid. We have sixteen areas, six mine cars and eight mineral deposits, making it a significantly more complex scenario than the previous cases.

Initial state:

- Robot starts at area **a22**
- Mine cars are located at: **m1** at **a13**, **m2** at **a43**, **m3** at **a31**, **m4** at **a34**
- Areas **a41**, **a42**, **a14**, **a11**, **a21**, **a31**, **a23**, and **a13** contain minerals
- All other areas are empty, including **a11**, **a12**, **a14**, **a21**, **a22**, **a23**, **a24**, **a32**, **a33**, **a41**, **a42**, and **a44**
- All areas are connected to their adjacent neighbors (up, down, left, right) as defined by the `adjacent` predicates

Goal state:

- Mine cars must be relocated to: **m1** at **a11**, **m2** at **a13**, **m3** at **a31**, and **m4** at **a33**
- All mineral-containing areas must be depleted: **a41**, **a42**, **a14**, **a11**, **a21**, **a31**, **a23**, and **a13**

4.3.2 Solver results**BFWS results:**

The BFWS solver found a solution with 136 actions, as detailed in the provided plan trace. The plan begins with the robot moving from **a22** to **a12** and pushing an initially unlocated mine car **m6** to **a22**, indicating that **m6** was likely inferred to start at **a12** based on the action precondition. The solver proceeds with a sequence of mineral collections and mine car movements, systematically collecting minerals at **a23**, **a13**, **a11**, **a21**, **a31**, **a41**, **a42**, **a43**, and **a14**, while simultaneously maneuvering mine cars **m1**, **m4**, **m5**, **m6**, and **m2** through a series of intermediate positions to their final destinations. The plan exhibits a high degree of backtracking and iterative refinement, with mine cars like **m1** and **m2** being pushed multiple times in a "shuttling" pattern to resolve spatial conflicts and ensure all goals are met. The total makespan of the plan is 0.135 time units, and the planner reported finding one plan in 2.892 seconds. This solution is complete as it satisfies all goal conditions, though its length and complexity highlight the computational difficulty of the 4×4 domain and that the BFWS solver is not an optimal planner, although complete.

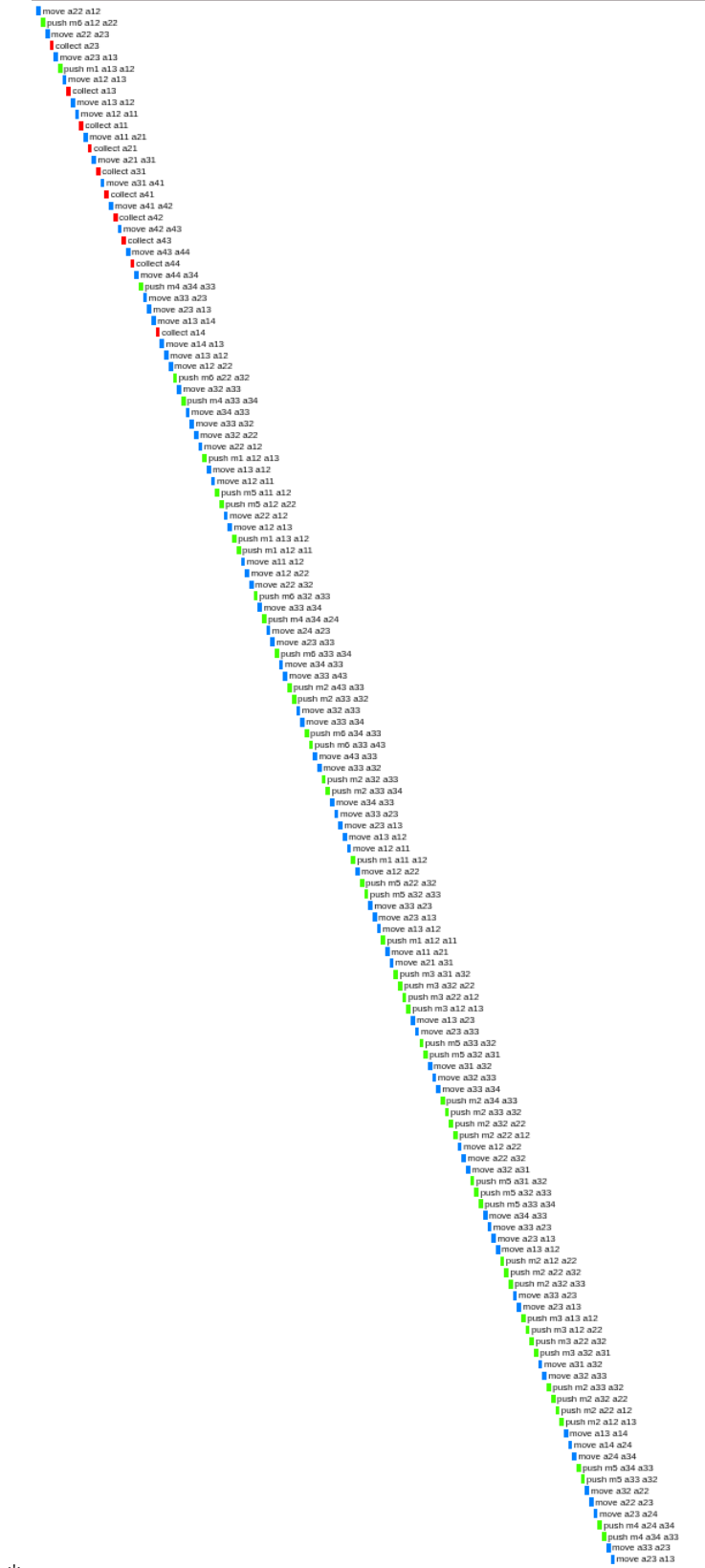


Figure 5: BFWS solver results for the 4x4 mining robot problem

Delfi results:

The Delfi solver found a significantly more efficient solution with only 23 actions, as shown in the provided plan trace. The plan starts with the robot moving from **a22** to **a23** and immediately collecting the mineral there, followed by a direct sequence of mineral collections at **a13**, **a11**, **a21**, **a31**, **a41**, and **a42**. The solver then focuses on mine car repositioning, pushing **m1** from **a13** to **a11** and **m2** from **a43** through **a33** and **a23** to its final position at **a13**, while also moving **m4** from **a34** to **a33**. The total makespan of this plan is 0.023 time units, and the planner reported finding one plan in 13.091 seconds. This solution is optimal in terms of action count and makespan, demonstrating Delfi's ability to always find an optimal strategy by prioritizing mineral collection early and minimizing redundant movements, in contrast to BFWS's more exploratory approach. Thus, we see that Delfi is a complete and optimal planner.



Figure 6: Delfi solver results for the 4x4 mining robot problem

The difference in planners:

The most significant difference between the BFWS and Delfi solvers for the 4×4 test case is the solution optimality and computational trade-off. BFWS produced a much longer plan of 136 actions with a makespan of 0.135, but it did so very quickly in 2.892 seconds, indicating it is a fast, greedy heuristic-based planner suitable for rapid feasibility checks. In contrast, Delfi found a highly optimized plan of only 23 actions with a makespan of 0.023, but it required 13.091 seconds to compute, reflecting its use of a more sophisticated

search algorithm, as a variant of A^* , that explores the state space more thoroughly to find optimal solutions. BFWS, being a classical state-space planner, generates and expands a large number of nodes (though the exact count is not provided), while Delfi, as a symbolic planner, operates on sets of states and does not report classical node counts, which as explained before, is why its "classical" counter remains at 0. This comparison illustrates a fundamental trade-off in automated planning: BFWS sacrifices optimality for speed, making it ideal for real-time or approximate planning, whereas Delfi sacrifices speed for optimality, making it better suited for offline, high-precision planning tasks where the best possible solution is required.

5 Conclusion

Working on the MineRob project gave us a great opportunity to learn about automated planning in a real setting. We started with a simple idea, a robot working in a mine with minerals and minecars, and turned it into a planning problem that challenges both movement and object handling. Designing the grid and deciding how the robot should act took quite some thinking and testing.

Throughout the project, we ran tests with grids of several sizes and more complex setups. Sometimes the robot has a clear way forward, but other times it had to push minecars around multiple times just to reach a mineral. We saw that as the environment gets more complex, planning needs more steps and careful ordering of actions.

One important aspect we learned is that the way we describe the environment and the robot's possible actions affects how easy it is to solve the problem. Small changes, such as how many minecars or where the minerals are placed, can make some plans much longer or even impossible to find a solution.

There is still space to improve our domain, With more features or smarter constraints, the planner could avoid useless moves or find better paths. Testing with two different planners, one satisficing and one optimal, also made it clear that some solvers handle these problems faster than others and that our choice when it comes to the planners has to take into account whether we just want to check feasibility or if we want to obtain the most optimal solution.

In sum, creating and testing our mining robot domain helped us see what goes into modeling and solving planning problems. It was an extremely useful experience for understanding how planners work and for getting more comfortable with practical AI techniques.