Rescue Drone Task: PDDL Planner

[2]

Oriol Miró
Víctor Carballo
Júlia Amenós

October 2024

Planning and Approximate Reasoning (MIA-MESIIA)
Practical Exercise 1: PAR

Contents

1 Introduction to the problem

In this project we have to develop a PDDL program that plans the route for a rescue drone. The map consists of a grid-like board in which there is people distributed. These people must be carried by the drone to a special cell known as the *safe-zone*. The drone can move to adjacent cells, as these are not occupied by an obstacle. The safe-zone has limited capacity, which limits our ability to drop people into it.

The goal of this project is to write a PDDL domain that models the previous scenario, and provide a set of actions taken by the drone to bring every person to the safe-zone. This plan must be efficient in the number of actions taken by the drone. Some actions are suggested in the statement:

- `move(d1,d2)`: the drone moves from location `d1` to location `d2`.
- `pickup(p,l)`: the drone picks up person `p` at location `l`.
- `dropoff()`: the drone drops the person that it's carrying into the safe-zone.

However, no specific details on the preconditions or effects are provided.

1.1 Assumptions

The problem statement left some constraints open to interpretation. After discussion with the professor, we assume the following:

1. There is only one drone.
2. There is only one safe-zone.
3. The drone can only carry one person at a time.
4. We take adjacency as the cardinal directions (no diagonals).
5. The grid *can* be rectangular ($n \times m$ with $n \neq m$).
6. The capacity of the safe-zone is not bound by the size of the grid (despite the problem statement suggesting so, as per the professor).
7. There exists an action to free up space in the safe-zone.
8. All actions have the same cost.

Later on we will challenge Assumption 1 (Section 4.6), and study the impact of our decision.

2 Problem Analysis

After reviewing the problem, in this section we focus on analyzing the different alternatives that PDDL enables for modeling a solution, with an emphasis on the motivations for the choices made. We also provide an estimate of the search space of the problem.

2.1 Search Space

The search space represents all possible configurations (states) that the system can be in. In the context of our problem, this includes every combination of:

- Drone States
- Person states
- Safe-zone occupancy levels

Let N_p be the number of people and N_l be the total number of *accessible* locations, that is locations with no obstacles nor blocked off by them (notice $N_l \leq N \times M$, with N and M being the number of rows and columns in the grid respectively). Then, the drone can be at any moment at one of N_l locations, and either be carrying any of the N_p people or not carrying anyone). The total number of drone states (D_s) is therefore:

$$D_s = N_l \times (N_p + 1) \tag{1}$$

Moving on to the permutations of people, each person can either be at one of $N_l - 1$ locations (as they can not *be* in the safe-zone, that counts as being rescued), be rescued, or be carried by the drone, thus the number of configurations for all people (P_s) is:

$$\begin{aligned} P_s &\leq (\text{Person\_states})^{N_p} \\ P_s &\leq (N_l - 1 + 1 + 1)^{N_p} \\ P_s &\leq (N_l + 1)^{N_p} \end{aligned} \tag{2}$$

This provides only an upper bound, as we are not considering only that one person can be carried by the drone at a time; moreover, we are counting states twice, as we both account for the drone carrying anybody and anybody being carried by the drone.

Finally considering the safe-zone occupancy states (S_S), since the occupancy can range from 0 up to the safe-zone capacity (S_C), we have:

$$S_s = S_c + 1 \tag{3}$$

Combining all these elements, the total number of states in the search space is:

$$\text{Total\_states} \leq D_s \times P_s \times S_s \tag{4}$$

Substituting the values we have:

$$\begin{aligned} \text{Total\_states} &\leq (N_l \times (N_p + 1)) \times (N_l + 1)^{N_p} \times (S_c + 1) \\ &\leq N_l(N_p + 1)(N_l + 1)^{N_p}(S_c + 1) \end{aligned} \tag{5}$$

Notice that what we have computed is an *upper bound*, given the assumptions made in the process.

2.2 Objects

The different actors in the problem statement are represented via different types of objects in the domain. Namely, those elements are the drone, people to be rescued, obstacles and the safe-zone. The board itself is also an element that is represented through its cells, known here as the *location*.

However, not all the items mentioned before should be converted to objects. For example, there is just one drone which has two properties: its location and whether it's carrying a passenger. Thus, it can be fully modeled through predicates and no special object is needed. Something similar happens with the safe-zone, it can be fully defined with a predicate for its location and a function for its capacity and current occupation. Obstacles can be similarly modeled through predicates that encode their position. Consequently, the list of necessary objects is:

- **Person**: individuals stranded in various locations that need to be rescued.
- **Location**: grid cells representing the disaster area, including obstacles, safe-zone, and normal cells.

2.3 Predicates and Functions

In the previous section we discarded the need of several objects by relegating their properties to a series of predicates and functions. For example, to model the drone we need its current position and whether it's carrying a passenger. Thus, each one of these characteristics needs a specific predicate. Analogously, obstacles need a predicate for their position and so does the safe-zone, in addition to a function to capture the current as well as the maximum occupation. Also, even though people are represented as objects, they have properties too which need to be represented as predicates, namely their position, whether they have been rescued and if they are being carried by the drone. Finally, one of the restrictions mentioned in the statement is that the drone can only move to adjacent cells. The concept of adjacency can be modeled in several ways, but in this case we have decided to insert one predicate for each pair of adjacent cells. Taking all of this into consideration, the list of predicates results as follows:

- `drone-at(?l)`: indicates that the drone is at location `l`.
- `person-at(?p,?l)`: person `p` is at location `l`.
- `obstacle(?l)`: location `l` is blocked by an obstacle.
- `safe-zone(?l)`: location `l` is the designated safe-zone.
- `adjacent (?l1 ?l2)`: locations `l1` and `l2` are adjacent. The adjacency of two locations is symmetric, so it is only necessary to define one direction.
- `rescued(?p)`: person `p` has been rescued.
- `carrying(?p)`: the drone is carrying person `p`.
- `drone-is-carrying-somebody`: the drone is currently carrying someone (and therefore can not carry anyone else).

And the functions used are:

- `safe-zone-occupancy`: The current number of people in the safe-zone.
- `safe-zone-capacity`: The maximum number of people the safe-zone can hold.

2.4 Operators

Objects, predicates and functions describe the state of the board at any given point. However, we still need actions that enable us to move from one state to the other. Such actions are known in PDDL as the *operators*.

Most of the operators needed in the problem were already mentioned in the statement. Those are `Move`, `Pick-Up` and `Drop-Off`. Here we provide more insight into them. Also, there is another operator that has not been mentioned in the problem's statement: `empty_safe_zone`. When the number of people in the safe-zone reaches its limit, this operator is in charge of emptying the safe-zone and enable the drone to keep rescuing people. It is not stated the conditions in which the removal of people from the safe-zone takes place, so we won't assume any extra restrictions over it.

Thus the list of operators results in:

- **Move**
  - **Action**: `move(?from, ?to)`
  - **Description**: the drone moves from location `?from` to an adjacent location `?to`, as long as `?to` is not blocked by an obstacle.
- **Pick-Up**
  - **Action**: `pick-up(?p, ?l)`
  - **Description**: the drone picks up a person `?p` who is located at `?l`, as long as the drone is currently at that location and is not already carrying another person.
- **Drop-Off**
  - **Action**: `drop-off(?p, ?l)`
  - **Description**: the drone drops off a person `?p` at the safe-zone `?l`, as long as `?l` is the safe-zone, the drone is currently at `?l`, it is carrying `?p`, and the safe-zone has capacity to accept more people.
- **Empty safe-zone**
  - **Action**: `empty_safe_zone`
  - **Description**: the drone treats the people in the safe-zone, which effectively resets the safe-zone's occupancy to zero. This action can only be executed when the occupancy of the safe zone is equal to its capacity.

3 PDDL Implementation

The following is the PDDL domain file that defines the actions, predicates, and functions for the rescue drone problem as discussed above:

Listing 1: PDDL Domain Definition for Rescue Drone

```
(define (domain rescue_drone)
  (:requirements :adl :typing :fluents)
  (:types person location)
  (:predicates
      (drone-at ?l - location)
```

```
        (person-at ?p - person ?l - location)
        (obstacle ?l - location)
        (safe-zone ?l - location)
        (adjacent ?l1 - location ?l2 - location)
        (rescued ?p - person)
        (carrying ?p - person)
        (drone-is-carrying-somebody)
    )
    (:functions
        (safe-zone-occupancy)
        (safe-zone-capacity)
    )
    (:action move_drone
        :parameters (?o - location ?f - location)
        :precondition (and (or (adjacent ?o ?f) (adjacent ?f ?o))
                           (drone-at ?o)
                           (not (obstacle ?f)))
        :effect (and (not (drone-at ?o))
                     (drone-at ?f))
    )
    (:action pickup
        :parameters (?p - person ?l - location)
        :precondition (and (drone-at ?l)
                           (person-at ?p ?l)
                           (not (safe-zone ?l))
                           (not (drone-is-carrying-somebody)))
        :effect (and (drone-is-carrying-somebody)
                     (carrying ?p)
                     (not (person-at ?p ?l)))
    )
    (:action drop
        :parameters (?p - person ?l - location)
        :precondition (and (carrying ?p)
                           (drone-at ?l)
                           (safe-zone ?l)
                           (< (safe-zone-occupancy) (safe-zone-
                              capacity)))
        :effect (and (not (drone-is-carrying-somebody))
                     (not (carrying ?p))
                     (rescued ?p)
                     (increase (safe-zone-occupancy) 1))
    )
    (:action empty_safe_zone
```

```
      :parameters ()
      :precondition (= (safe-zone-occupancy) (safe-zone-
         capacity))
      :effect (assign (safe-zone-occupancy) 0)
  )
  )
```

It is worth noting that we have chosen the number of steps as the metric to minimize. We decided not to introduce a specific cost variable for optimization, as we assumed that all actions have equal weight. For example, while we considered minimizing the number of drone movements, this goal is inherently addressed by reducing the total number of steps.

4 Experiments

The code was tested using a variety of test cases, including different grid sizes, obstacle placements, and safe-zone capacities. These tests include those suggested in the assignment description (a specific 4×4 instance, and any 5×5 instance), a test challenging our model's ability to empty the safe-zone, a test presenting an impossible position, and a large test experimenting with a very complex situation. Moreover, we performed an extension where we test our planner when the drone has an increased capacity.

Metric-FF [1] was used in our experiments due to its capability to handle both logical and numerical constraints, as it extends the Fast-Forward (FF) planner to support numerical state variables. This allows the planner to take into account dynamic numerical conditions, such as the safe-zone capacity.

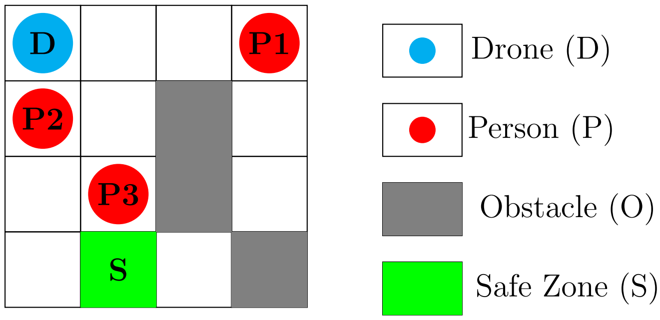
The problem definitions are automatically generated via a python script which takes as input a problem in ASCII format and generates the corresponding PDDL file. An example for a problem defined via ASCII, in this case the example board of case 1, is:

```
D__P
P_O_
_PO_
_S_O.
```

The script used can be found in annex B (5).

4.1 Test Case 1

The first test is given in the assignment and consist in a very basic problem with three people to rescue and some obstacles in a 4×4 grid with safe-zone capacity of four:



Results and Analysis

```

ff: parsing domain file
domain 'RESCUE_DRONE' defined
 ... done.
ff: parsing problem file
problem 'EX1' defined
 ... done.


no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

advancing to distance:   14
                         12
                         11
                         10
                          9
                          8
                          7
                          6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step     0: MOVE_DRONE L1 L2
         1: MOVE_DRONE L2 L3
         2: MOVE_DRONE L3 L4
         3: PICKUP P1 L4
```

```
        4: MOVE_DRONE L4 L3
        5: MOVE_DRONE L3 L2
        6: MOVE_DRONE L2 L6
        7: MOVE_DRONE L6 L10
        8: MOVE_DRONE L10 L14
        9: DROP P1 L14
       10: MOVE_DRONE L14 L10
       11: PICKUP P3 L10
       12: MOVE_DRONE L10 L14
       13: DROP P3 L14
       14: MOVE_DRONE L14 L10
       15: MOVE_DRONE L10 L6
       16: MOVE_DRONE L6 L5
       17: PICKUP P2 L5
       18: MOVE_DRONE L5 L6
       19: MOVE_DRONE L6 L10
       20: MOVE_DRONE L10 L14
       21: DROP P2 L14


time spent:    0.00 seconds instantiating 49 easy, 38 hard action templates
               0.00 seconds reachability analysis, yielding 24 facts and 37 actions
               0.00 seconds creating final representation with 24 relevant facts, 2
                   relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 53 states, to a max depth of 0
               0.00 seconds total time
```

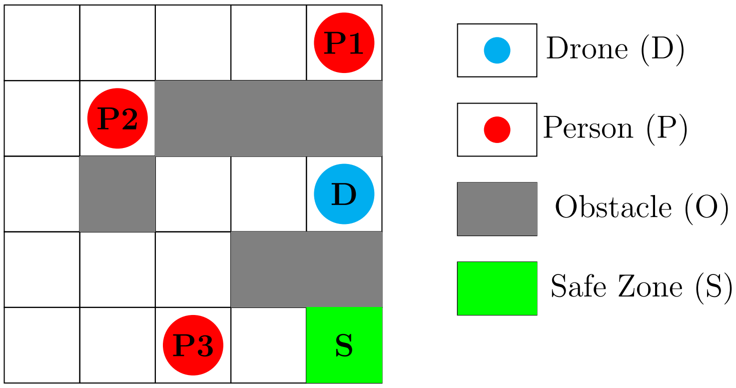
The solver correctly solves this puzzle. The steps taken are:

- **Rescue (P1):** The drone first goes to the upper row to pick up P1 in 4 moves, then surrounds the obstacle to leave it in the safe-zone in 6 steps.
- **Rescue (P3):** Then the drone moves a single square to rescue P3.
- **Rescue (P2):** Finally, the drone moves up and left to rescue P2 similarly to what it did for P1 and P3, thus concluding the mission.

As the capacity of the safe-zone is three, no removal of people from it is needed. We can also note that the planner finished in less than 0.01 seconds, with a total of 21 steps and 53 states expanded.

4.2 Test Case 2

Also as instructed in the assignment, we test a 5×5 grid configuration in which we added more obstacles in labyrinthine manner to increase complexity. The drone has to get out and navigate around a tunnel-like structure to transport the people to the safe-zone:



Results and Analysis

```

ff: parsing domain file
domain 'RESCUE_DRONE' defined
 ... done.
ff: parsing problem file
problem 'EX2' defined
 ... done.


no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

advancing to distance:   22
                         21
                         20
                         19
                         18
                         17
                         16
                         15
                         14
                         13
                         11
                         10
                          9
                          8
                          7
                          6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: MOVE_DRONE L15 L14
```

```
        1: MOVE_DRONE L14 L13
        2: MOVE_DRONE L13 L18
        3: MOVE_DRONE L18 L23
        4: PICKUP P3 L23
        5: MOVE_DRONE L23 L24
        6: MOVE_DRONE L24 L25
        7: DROP P3 L25
        8: MOVE_DRONE L25 L24
        9: MOVE_DRONE L24 L23
       10: MOVE_DRONE L23 L22
       11: MOVE_DRONE L22 L21
       12: MOVE_DRONE L21 L16
       13: MOVE_DRONE L16 L11
       14: MOVE_DRONE L11 L6
       15: MOVE_DRONE L6 L7
       16: MOVE_DRONE L7 L2
       17: MOVE_DRONE L2 L3
       18: MOVE_DRONE L3 L4
       19: MOVE_DRONE L4 L5
       20: PICKUP P1 L5
       21: MOVE_DRONE L5 L4
       22: MOVE_DRONE L4 L3
       23: MOVE_DRONE L3 L2
       24: MOVE_DRONE L2 L7
       25: MOVE_DRONE L7 L6
       26: MOVE_DRONE L6 L11
       27: MOVE_DRONE L11 L16
       28: MOVE_DRONE L16 L17
       29: MOVE_DRONE L17 L18
       30: MOVE_DRONE L18 L23
       31: MOVE_DRONE L23 L24
       32: MOVE_DRONE L24 L25
       33: DROP P1 L25
       34: MOVE_DRONE L25 L24
       35: MOVE_DRONE L24 L23
       36: MOVE_DRONE L23 L22
       37: MOVE_DRONE L22 L21
       38: MOVE_DRONE L21 L16
       39: MOVE_DRONE L16 L11
       40: MOVE_DRONE L11 L6
       41: MOVE_DRONE L6 L7
       42: PICKUP P2 L7
       43: MOVE_DRONE L7 L6
       44: MOVE_DRONE L6 L11
       45: MOVE_DRONE L11 L16
       46: MOVE_DRONE L16 L17
       47: MOVE_DRONE L17 L18
       48: MOVE_DRONE L18 L23
       49: MOVE_DRONE L23 L24
       50: MOVE_DRONE L24 L25
       51: DROP P2 L25


time spent:    0.00 seconds instantiating 76 easy, 58 hard action templates
               0.00 seconds reachability analysis, yielding 30 facts and 49 actions
               0.00 seconds creating final representation with 30 relevant facts, 2 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 178 states, to a max depth of 0
               0.00 seconds total time
```

Even though the scenario is more complex, the solution follows the same line as the one in the previous example:

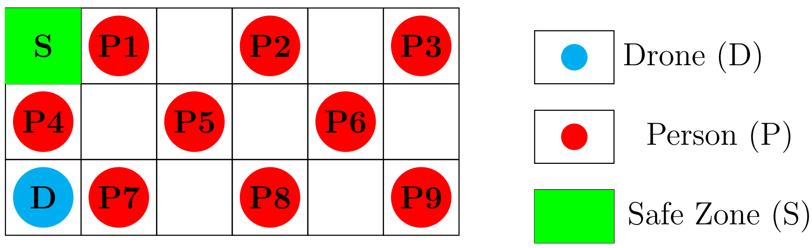
- **Rescue (P3):** The drone first goes to rescue the person closest to the safe-zone, and it does so in 8 moves.
- **Rescue (P1):** Interestingly, the drone now goes all the way around the obstacle to get to P1, ignoring P2.
- **Rescue (P2):** This time the drone rescues the last person remaining in the board.

This solution has more steps than the previous one due to the increased board size and complexity of the movements, but even with that the time that the solver takes to resolve the problem is neglectable, with 178 states explored and 51 steps in the solution.

4.3 Test Case 3

This third test case will be used as a way to test the functionality of the safe-zone capacity. It has a very high density of people to be rescued, with a safe-zone capacity of three. Thus, it will be necessary to execute the `EMPTY_SAFE_ZONE` function several times. Apart from that, it does not have obstacles in order to make the output as straightforward as possible, considering the bloat added by the large number of people.

The board used looks as:



Results and Analysis

```

ff: parsing domain file
domain 'RESCUE_DRONE' defined
 ... done.
ff: parsing problem file
problem 'EX3' defined
 ... done.


```

```
no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

advancing to distance:   34
                         32
                         31
                         30
                         29
                         28
                         27
                         26
                         25
                         24
                         23
                         22
                         21
                         20
                         19
                         18
                         17
                         15
                         14
                         13
                         12
                         11
                         10
                          9
                          8
                          7
                          6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: MOVE_DRONE L13 L7
        1: PICKUP P4 L7
        2: MOVE_DRONE L7 L1
        3: DROP P4 L1
        4: MOVE_DRONE L1 L2
        5: PICKUP P1 L2
        6: MOVE_DRONE L2 L1
        7: DROP P1 L1
        8: MOVE_DRONE L1 L2
        9: MOVE_DRONE L2 L3
       10: MOVE_DRONE L3 L4
       11: PICKUP P2 L4
       12: MOVE_DRONE L4 L3
       13: MOVE_DRONE L3 L2
       14: MOVE_DRONE L2 L1
       15: DROP P2 L1
       16: EMPTY_SAFE_ZONE
       17: MOVE_DRONE L1 L2
       18: MOVE_DRONE L2 L3
```

```
    19: MOVE_DRONE L3 L9
    20: PICKUP P5 L9
    21: MOVE_DRONE L9 L3
    22: MOVE_DRONE L3 L2
    23: MOVE_DRONE L2 L1
    24: DROP P5 L1
    25: MOVE_DRONE L1 L2
    26: MOVE_DRONE L2 L3
    27: MOVE_DRONE L3 L4
    28: MOVE_DRONE L4 L5
    29: MOVE_DRONE L5 L6
    30: PICKUP P3 L6
    31: MOVE_DRONE L6 L5
    32: MOVE_DRONE L5 L4
    33: MOVE_DRONE L4 L10
    34: MOVE_DRONE L10 L9
    35: MOVE_DRONE L9 L8
    36: MOVE_DRONE L8 L7
    37: MOVE_DRONE L7 L1
    38: DROP P3 L1
    39: MOVE_DRONE L1 L7
    40: MOVE_DRONE L7 L8
    41: MOVE_DRONE L8 L14
    42: PICKUP P7 L14
    43: MOVE_DRONE L14 L8
    44: MOVE_DRONE L8 L7
    45: MOVE_DRONE L7 L1
    46: DROP P7 L1
    47: MOVE_DRONE L1 L7
    48: EMPTY_SAFE_ZONE
    49: MOVE_DRONE L7 L8
    50: MOVE_DRONE L8 L9
    51: MOVE_DRONE L9 L10
    52: MOVE_DRONE L10 L16
    53: PICKUP P8 L16
    54: MOVE_DRONE L16 L10
    55: MOVE_DRONE L10 L9
    56: MOVE_DRONE L9 L8
    57: MOVE_DRONE L8 L7
    58: MOVE_DRONE L7 L1
    59: DROP P8 L1
    60: MOVE_DRONE L1 L7
    61: MOVE_DRONE L7 L8
    62: MOVE_DRONE L8 L9
    63: MOVE_DRONE L9 L10
    64: MOVE_DRONE L10 L11
    65: PICKUP P6 L11
    66: MOVE_DRONE L11 L10
    67: MOVE_DRONE L10 L9
    68: MOVE_DRONE L9 L8
    69: MOVE_DRONE L8 L7
    70: MOVE_DRONE L7 L1
    71: DROP P6 L1
    72: MOVE_DRONE L1 L2
    73: MOVE_DRONE L2 L3
    74: MOVE_DRONE L3 L4
    75: MOVE_DRONE L4 L5
    76: MOVE_DRONE L5 L11
    77: MOVE_DRONE L11 L17
    78: MOVE_DRONE L17 L18
    79: PICKUP P9 L18
    80: MOVE_DRONE L18 L12
```

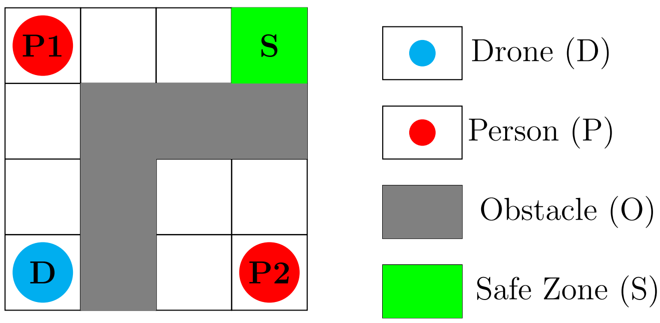
```
        81: MOVE_DRONE L12 L6
        82: MOVE_DRONE L6 L5
        83: MOVE_DRONE L5 L4
        84: MOVE_DRONE L4 L3
        85: MOVE_DRONE L3 L2
        86: MOVE_DRONE L2 L1
        87: DROP P9 L1


time spent:    0.00 seconds instantiating 163 easy, 54 hard action templates
               0.00 seconds reachability analysis, yielding 47 facts and 73 actions
               0.00 seconds creating final representation with 47 relevant facts, 2
                   relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 241 states, to a max depth of 0
               0.00 seconds total time
```

We will not describe how the rescue operation goes person by person. The most important aspect is that the EMPTY_SAFE_ZONE function gets called two times (recall that the capacity is three in this case) in steps 16 and 48, so the fluents work as expected. The execution time is again neglectable and the solver has expanded 241 states with a total of 87 steps in the solution.

4.4 Test Case 4

This experiment tests the planner's response to an unsolvable scenario, where obstacles block all paths to stranded individuals. The goal is to ensure the planner correctly identifies the situation as infeasible and terminates efficiently. The position can be visualised as:



Where the drone can not reach person P2.

Results and Analysis

```

ff: parsing domain file
domain 'RESCUE_DRONE' defined
 ... done.
```

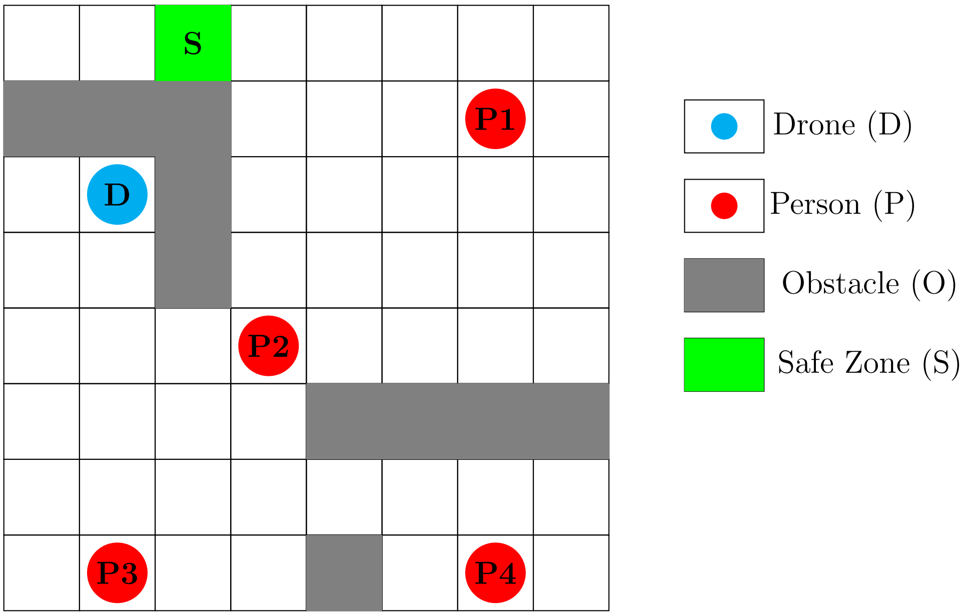
```
ff: parsing problem file
problem 'EX4' defined
 ... done.


ff: goal can be simplified to FALSE. No plan will solve it
```

The planner successfully identified the position is not solvable and returned **FALSE**. Unfortunately no extra information (such as how long it took to evaluate, or the nodes expanded until the realisation it was unsolvable) is provided, limiting the depth of the current analysis.

4.5 Test Case 5

This test involves a more complex 8x8 grid, where the safe-zone has a capacity of 6—more than enough to accommodate the 4 individuals needing rescue. While the safe-zone's capacity isn't a limiting factor, the increased grid size and numerous obstacles significantly expand the search space, testing the planner's ability to find an efficient solution. The position can be visualized as:



Results and Analysis

```

ff: parsing domain file
domain 'RESCUE_DRONE' defined
 ... done.
ff: parsing problem file
problem 'EX5' defined
```

```
 ... done.


no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

advancing to distance:   29
                         28
                         27
                         26
                         25
                         24
                         23
                         22
                         21
                         20
                         19
                         18
                         17
                         16
                         15
                         14
                         13
                         12
                         11
                         10
                          9
                          8
                          7
                          6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: MOVE_DRONE L18 L26
        1: MOVE_DRONE L26 L34
        2: MOVE_DRONE L34 L35
        3: MOVE_DRONE L35 L36
        4: MOVE_DRONE L36 L44
        5: MOVE_DRONE L44 L52
        6: MOVE_DRONE L52 L51
        7: MOVE_DRONE L51 L50
        8: MOVE_DRONE L50 L58
        9: PICKUP P3 L58
       10: MOVE_DRONE L58 L50
       11: MOVE_DRONE L50 L51
       12: MOVE_DRONE L51 L52
       13: MOVE_DRONE L52 L44
       14: MOVE_DRONE L44 L36
       15: MOVE_DRONE L36 L28
       16: MOVE_DRONE L28 L20
       17: MOVE_DRONE L20 L12
       18: MOVE_DRONE L12 L4
```

```
        19: MOVE_DRONE L4 L3
        20: DROP P3 L3
        21: MOVE_DRONE L3 L4
        22: MOVE_DRONE L4 L12
        23: MOVE_DRONE L12 L20
        24: MOVE_DRONE L20 L28
        25: MOVE_DRONE L28 L36
        26: MOVE_DRONE L36 L44
        27: MOVE_DRONE L44 L52
        28: MOVE_DRONE L52 L53
        29: MOVE_DRONE L53 L54
        30: MOVE_DRONE L54 L55
        31: MOVE_DRONE L55 L63
        32: PICKUP P4 L63
        33: MOVE_DRONE L63 L55
        34: MOVE_DRONE L55 L54
        35: MOVE_DRONE L54 L53
        36: MOVE_DRONE L53 L52
        37: MOVE_DRONE L52 L44
        38: MOVE_DRONE L44 L36
        39: MOVE_DRONE L36 L28
        40: MOVE_DRONE L28 L20
        41: MOVE_DRONE L20 L12
        42: MOVE_DRONE L12 L4
        43: MOVE_DRONE L4 L3
        44: DROP P4 L3
        45: MOVE_DRONE L3 L4
        46: MOVE_DRONE L4 L12
        47: MOVE_DRONE L12 L20
        48: MOVE_DRONE L20 L28
        49: MOVE_DRONE L28 L36
        50: PICKUP P2 L36
        51: MOVE_DRONE L36 L28
        52: MOVE_DRONE L28 L20
        53: MOVE_DRONE L20 L12
        54: MOVE_DRONE L12 L4
        55: MOVE_DRONE L4 L3
        56: DROP P2 L3
        57: MOVE_DRONE L3 L4
        58: MOVE_DRONE L4 L5
        59: MOVE_DRONE L5 L6
        60: MOVE_DRONE L6 L7
        61: MOVE_DRONE L7 L15
        62: PICKUP P1 L15
        63: MOVE_DRONE L15 L7
        64: MOVE_DRONE L7 L6
        65: MOVE_DRONE L6 L5
        66: MOVE_DRONE L5 L4
        67: MOVE_DRONE L4 L3
        68: DROP P1 L3


time spent:    0.00 seconds instantiating 257 easy, 187 hard action templates
               0.00 seconds reachability analysis, yielding 68 facts and 173 actions
               0.00 seconds creating final representation with 68 relevant facts, 2
                   relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.01 seconds searching, evaluating 285 states, to a max depth of 0
               0.01 seconds total time
```

The planner successfully solved the 8×8 grid, and despite the increased complexity did so swiftly. The key steps to reach the solution were as follows:

- **First Rescue (P3)**: The drone moved from L18 to L58 to pick up P3, avoiding obstacles efficiently. It then returned to the safe-zone (L3), completing the rescue in 20 steps (Steps 0–20).
- **Second Rescue (P4)**: After dropping off P3, the drone travelled to L63 to pick up P4. Due to the greater distance and obstacle layout, this cycle took 24 steps to return to L3 (Steps 21–44).
- **Third and Fourth Rescues (P2, P1)**: The drone rescued P2 from L36 and P1 from L15 in 12 steps each. Both individuals were successfully transported to the safe-zone at L3 (Steps 45–68).

Regarding complexity, the solution involved 69 steps, with the planner evaluating 285 states in total. Despite the increased grid complexity, the search remained efficient, completing in 0.01 seconds (altough such result is hardware-dependant)

4.6 Extension I

The emergency drone rescue company just received an unexpected increase of budget, and they want to upgrade their drone fleet; the new drones will not only be shinier and more futuristic-looking, but will also come with *increased capacity*, as an internal study revealed this was one of the most limiting factors in rescue missions. Therefore, in this extension we will challenge Assumption I (see 1.1), with a specific case of increased capacity (Test Case 6), and a study of how all previous tests would have fared had we had our new, shiny drones.

4.6.1 Modifications

The domain was modified in order to adapt it to the new conditions. We replaced the predicate `drone-is-carrying-somebody`, which was originally used to prevent the drone to pick up a person if it was already carrying one, with two new functions.

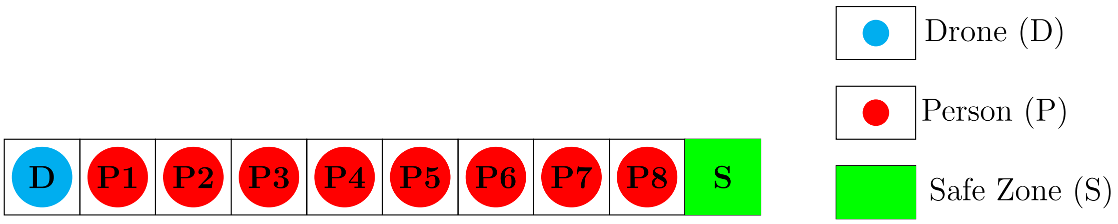
- `drone-occupancy`: The current number of people that the drone is carrying up.
- `drone-capacity`: The maximum number of people that the drone is capable of carrying.

4.6.2 Test Case 6

In this test we want to check how the drone's capacity comes into play. It consists of a 1x10 grid, where eight individuals, that need to be rescued, are in line. The drone starts at the leftmost location while the safe-zone is on the rightmost. It is interesting to see how capacity constraints are satisfied at all time.

The problem is defined with the following features:

- Drone occupancy: The drone can carry 3 people at a time.
- Safe-zone occupancy: The safe-zone can hold up to 7 people, while there are 8 individuals to rescue.



Results and Analysis

```
step    0: MOVE_DRONE L1 L2
        1: PICKUP P1 L2
        2: MOVE_DRONE L2 L3
        3: PICKUP P2 L3
        4: MOVE_DRONE L3 L4
        5: PICKUP P3 L4
        6: MOVE_DRONE L4 L5
        7: MOVE_DRONE L5 L6
        8: MOVE_DRONE L6 L7
        9: MOVE_DRONE L7 L8
       10: MOVE_DRONE L8 L9
       11: MOVE_DRONE L9 L10
       12: DROP P1 L10
       13: DROP P2 L10
       14: DROP P3 L10
       15: MOVE_DRONE L10 L9
       16: PICKUP P8 L9
       17: MOVE_DRONE L9 L8
       18: PICKUP P7 L8
       19: MOVE_DRONE L8 L7
       20: PICKUP P6 L7
       21: MOVE_DRONE L7 L8
       22: MOVE_DRONE L8 L9
       23: MOVE_DRONE L9 L10
       24: DROP P6 L10
       25: DROP P7 L10
       26: DROP P8 L10
       27: MOVE_DRONE L10 L9
       28: MOVE_DRONE L9 L8
       29: MOVE_DRONE L8 L7
       30: MOVE_DRONE L7 L6
```

```
       31: PICKUP P5 L6
       32: MOVE_DRONE L6 L5
       33: PICKUP P4 L5
       34: MOVE_DRONE L5 L6
       35: MOVE_DRONE L6 L7
       36: MOVE_DRONE L7 L8
       37: MOVE_DRONE L8 L9
       38: MOVE_DRONE L9 L10
       39: DROP P4 L10
       40: EMPTY_SAFE_ZONE
       41: DROP P5 L10


time spent:    0.00 seconds instantiating 81 easy, 18 hard action templates
               0.00 seconds reachability analysis, yielding 34 facts and 35 actions
               0.00 seconds creating final representation with 34 relevant facts, 4
                   relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 74 states, to a max depth of 0
               0.00 seconds total time
```

In the resulting planner, the drone picks up groups of three people twice and then the remaining person, ensuring it never exceeds the drone's carrying limit.

- **First rescue (P1, P2, P3):** The drone picks up the first three people who are closest to it and further away from the safe-zone. That way, it does not have to come back later, which helps optimizing the number of steps taken.
- **Second rescue (P6, P7, P8):** It can be seen that the drone never holds up more than its maximum capacity.
- **Third rescue (P4, P5):** The drone waits until for the safe-zone to be cleared before dropping the eighth person, hence respecting the capacity constraints.

Regarding complexity, the solution involved 41 steps and an evaluation of 74 states in total.

4.6.3 Test Case 7

We also wanted to perform a meta-analysis on the impact of using different drone capacities. For each of the previous maps (Tests 1, 2, 3, 5 and 6), the drone's capacity was varied from 1 to 10. We recorded the number of steps and the number of states explored in each execution.

Results and Analysis

Figure 1 shows that increasing drone capacity leads to a reduction in the number of steps required to complete the mission. This makes sense as lower capacities

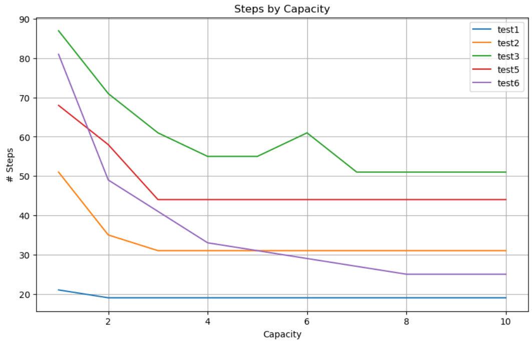


Figure 1: Steps by capacity.

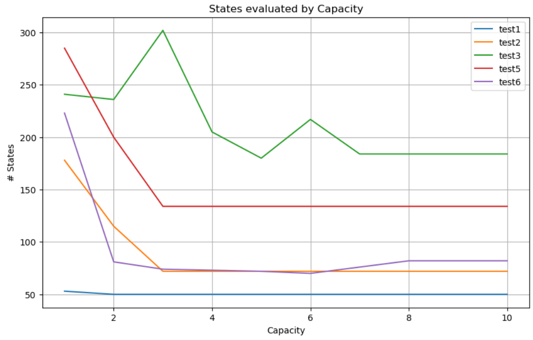


Figure 2: States evaluated by capacity.

involve having to do multiple trips to transport people; however, performance increase plateaus once the drone capacity equals the number of people.

On the other hand, Figure 2 shows how the number of states evaluated is typically reduced as drone capacity increases. As the drone can carry larger groups in each trip and reach the goal faster, it results in fewer intermediate decisions and the planner evaluates less states before finding a solution. Nevertheless, once the drone can transport enough people in a few trips, further increasing capacity does not simplify the problem much more. The extent of this improvement depends on the complexity of the problem, with more complex scenarios benefiting more from the increased capacity.

One could notice an anti-intuitive detail: there are “peaks” in the graph. This seems to suggest that, at times, performance *decays* as we increase drone capacity; nevertheless, we attribute this to the planner sometimes finding sub-optimal solutions. Most of these “peaks” come from Test Case 3 (see 4.3), which has a very complex configuration resulting in the planner being forced to make many intermediate decisions. Overall, the “peaks” do not impact performance much either.

5 Final Analysis and Conclusion

In this practice, we developed a PDDL program that models the actions of a rescue drone navigating a grid-like environment, tasked with transporting people to a safe-zone. The domain was designed to allow the drone to move, pick up individuals, and drop them off at a safe-zone. We conducted a series of experiments across varied grid configurations, obstacle placements, and safe-zone capacities. The experiments tested the efficiency of the planner and allowed us to explore the complexity of the problem as well as its search space. Additionally, we extended the problem by increasing the drone's carrying capacity and analysing its impact on performance.

A global analysis of the tests reveals how different factors - grid size, obstacle density, number of people, and safe-zone capacity - impact the complexity and efficiency of the drone's rescue mission. Smaller grids with fewer obstacles, like in Test Case 1, resulted in solutions with less nodes expanded, while more complex grids in Test Cases 2 and 5 increased the number of steps and states explored. With Test Case 3 and 6 we observed how the planner dealt with constraints like the safe-zone and drone capacity, and in contrast Test Case 4 showed that the planner could promptly identify unsolvable scenarios. Finally, on Test Case 7 we compared the solutions with different drone capacities and found that the heuristic of the planner does not always find the optimal solution for the problem.

During the design phase we also considered other alternatives to our assumptions, such as having multiple drones, multiple safe-zones, modelling time, etc., but decided against it as it would widen the scope of the work excessively, not permitting depth of analysis. Nevertheless, it could be interesting to explore these in future work.

References

[1] Joerg Hoffmann. The metric-ff planning system. https://fai.cs.uni-saarland.de/hoffmann/metric-ff.html, 2024. Top Performer in the Numeric Track of the 3rd International Planning Competition.

[2] OpenAI. This image was created with the assistance of dall·e 2. https://openai.com/dall-e-2, 2024. Accessed: 12-10-2024.

Annex A: Python Script for Generating PDDL Problem Files

Below are the pddl problem files for each one of the examples shown before.

A.1: Example 1

Listing 2: Problem definition for example 1

```
(define (problem ex1)
  (:domain rescue_drone)
  (:objects
    p1 p2 p3 - person
    l1 l2 l3 l4 l5 l6 l7 l8 l9 l10 l11 l12 l13 l14 l15 l16 -
       location
)
  (:init
    (adjacent l1 l5)
    (adjacent l1 l2)
    (adjacent l2 l6)
    (adjacent l2 l3)
    (adjacent l3 l7)
    (adjacent l3 l4)
    (adjacent l4 l8)
    (adjacent l5 l9)
    (adjacent l5 l6)
    (adjacent l6 l10)
    (adjacent l6 l7)
    (adjacent l7 l11)
    (adjacent l7 l8)
    (adjacent l8 l12)
    (adjacent l9 l13)
    (adjacent l9 l10)
    (adjacent l10 l14)
    (adjacent l10 l11)
    (adjacent l11 l15)
    (adjacent l11 l12)
    (adjacent l12 l16)
    (adjacent l13 l14)
    (adjacent l14 l15)
    (adjacent l15 l16)
    (drone-at l1)
```

```
    (safe-zone l14)
    (obstacle l7)
    (obstacle l11)
    (obstacle l16)
    (person-at p1 l4)
    (person-at p2 l5)
    (person-at p3 l10)
    (= (safe-zone-capacity) 3)
    (= (safe-zone-occupancy) 0)
  )
  (:goal (and
    (rescued p1)
    (rescued p2)
    (rescued p3)
))
)
```

A.2: Example 2

Listing 3: Problem definition for example 2

```
(define (problem ex2)
  (:domain rescue_drone)
  (:objects
    p1 p2 p3 - person
    l1 l2 l3 l4 l5 l6 l7 l8 l9 l10 l11 l12 l13 l14 l15 l16 l17
       l18 l19 l20 l21 l22 l23 l24 l25 - location
)
  (:init
    (adjacent l1 l6)
    (adjacent l1 l2)
    (adjacent l2 l7)
    (adjacent l2 l3)
    (adjacent l3 l8)
    (adjacent l3 l4)
    (adjacent l4 l9)
    (adjacent l4 l5)
    (adjacent l5 l10)
    (adjacent l6 l11)
    (adjacent l6 l7)
    (adjacent l7 l12)
    (adjacent l7 l8)
```

```
    (adjacent l8 l13)
    (adjacent l8 l9)
    (adjacent l9 l14)
    (adjacent l9 l10)
    (adjacent l10 l15)
    (adjacent l11 l16)
    (adjacent l11 l12)
    (adjacent l12 l17)
    (adjacent l12 l13)
    (adjacent l13 l18)
    (adjacent l13 l14)
    (adjacent l14 l19)
    (adjacent l14 l15)
    (adjacent l15 l20)
    (adjacent l16 l21)
    (adjacent l16 l17)
    (adjacent l17 l22)
    (adjacent l17 l18)
    (adjacent l18 l23)
    (adjacent l18 l19)
    (adjacent l19 l24)
    (adjacent l19 l20)
    (adjacent l20 l25)
    (adjacent l21 l22)
    (adjacent l22 l23)
    (adjacent l23 l24)
    (adjacent l24 l25)
    (drone-at l15)
    (safe-zone l25)
    (obstacle l8)
    (obstacle l9)
    (obstacle l10)
    (obstacle l12)
    (obstacle l19)
    (obstacle l20)
    (person-at p1 l5)
    (person-at p2 l7)
    (person-at p3 l23)
    (= (safe-zone-capacity) 3)
    (= (safe-zone-occupancy) 0)
  )
  (:goal (and
    (rescued p1)
```

```
    (rescued p2)
    (rescued p3)
))
)
```

A.3: Example 3

Listing 4: Problem definition for example 3

```
(define (problem ex3)
  (:domain rescue_drone)
  (:objects
    p1 p2 p3 p4 p5 p6 p7 p8 p9 - person
    l1 l2 l3 l4 l5 l6 l7 l8 l9 l10 l11 l12 l13 l14 l15 l16 l17
       l18 - location
)
  (:init
    (adjacent l1 l7)
    (adjacent l1 l2)
    (adjacent l2 l8)
    (adjacent l2 l3)
    (adjacent l3 l9)
    (adjacent l3 l4)
    (adjacent l4 l10)
    (adjacent l4 l5)
    (adjacent l5 l11)
    (adjacent l5 l6)
    (adjacent l6 l12)
    (adjacent l7 l13)
    (adjacent l7 l8)
    (adjacent l8 l14)
    (adjacent l8 l9)
    (adjacent l9 l15)
    (adjacent l9 l10)
    (adjacent l10 l16)
    (adjacent l10 l11)
    (adjacent l11 l17)
    (adjacent l11 l12)
    (adjacent l12 l18)
    (adjacent l13 l14)
    (adjacent l14 l15)
    (adjacent l15 l16)
```

```
    (adjacent l16 l17)
    (adjacent l17 l18)
    (drone-at l13)
    (safe-zone l1)
    (person-at p1 l2)
    (person-at p2 l4)
    (person-at p3 l6)
    (person-at p4 l7)
    (person-at p5 l9)
    (person-at p6 l11)
    (person-at p7 l14)
    (person-at p8 l16)
    (person-at p9 l18)
    (= (safe-zone-capacity) 3)
    (= (safe-zone-occupancy) 0)
  )
  (:goal (and
    (rescued p1)
    (rescued p2)
    (rescued p3)
    (rescued p4)
    (rescued p5)
    (rescued p6)
    (rescued p7)
    (rescued p8)
    (rescued p9)
))
)
```

A.4: Example 4

Listing 5: Problem definition for example 4

```
(define (problem ex4)
  (:domain rescue_drone)
  (:objects
    p1 p2 - person
    l1 l2 l3 l4 l5 l6 l7 l8 l9 l10 l11 l12 l13 l14 l15 l16 -
        location
)
  (:init
    (adjacent l1 l5)
```

```
    (adjacent l1 l2)
    (adjacent l2 l6)
    (adjacent l2 l3)
    (adjacent l3 l7)
    (adjacent l3 l4)
    (adjacent l4 l8)
    (adjacent l5 l9)
    (adjacent l5 l6)
    (adjacent l6 l10)
    (adjacent l6 l7)
    (adjacent l7 l11)
    (adjacent l7 l8)
    (adjacent l8 l12)
    (adjacent l9 l13)
    (adjacent l9 l10)
    (adjacent l10 l14)
    (adjacent l10 l11)
    (adjacent l11 l15)
    (adjacent l11 l12)
    (adjacent l12 l16)
    (adjacent l13 l14)
    (adjacent l14 l15)
    (adjacent l15 l16)
    (drone-at l13)
    (safe-zone l4)
    (obstacle l6)
    (obstacle l7)
    (obstacle l8)
    (obstacle l10)
    (obstacle l14)
    (person-at p1 l1)
    (person-at p2 l16)
    (= (safe-zone-capacity) 3)
    (= (safe-zone-occupancy) 0)
  )
  (:goal (and
    (rescued p1)
    (rescued p2)
))
)
```

A.5: Example 5

Listing 6: Problem definition for example 5

```
(define (problem ex5)
  (:domain rescue_drone)
  (:objects
    p1 p2 p3 p4 - person
    l1 l2 l3 l4 l5 l6 l7 l8 l9 l10 l11 l12 l13 l14 l15 l16 l17
       l18 l19 l20 l21 l22 l23 l24 l25 l26 l27 l28 l29 l30 l31
       l32 l33 l34 l35 l36 l37 l38 l39 l40 l41 l42 l43 l44 l45
       l46 l47 l48 l49 l50 l51 l52 l53 l54 l55 l56 l57 l58 l59
       l60 l61 l62 l63 l64 - location
)
  (:init
    (adjacent l1 l9)
    (adjacent l1 l2)
    (adjacent l2 l10)
    (adjacent l2 l3)
    (adjacent l3 l11)
    (adjacent l3 l4)
    (adjacent l4 l12)
    (adjacent l4 l5)
    (adjacent l5 l13)
    (adjacent l5 l6)
    (adjacent l6 l14)
    (adjacent l6 l7)
    (adjacent l7 l15)
    (adjacent l7 l8)
    (adjacent l8 l16)
    (adjacent l9 l17)
    (adjacent l9 l10)
    (adjacent l10 l18)
    (adjacent l10 l11)
    (adjacent l11 l19)
    (adjacent l11 l12)
    (adjacent l12 l20)
    (adjacent l12 l13)
    (adjacent l13 l21)
    (adjacent l13 l14)
    (adjacent l14 l22)
    (adjacent l14 l15)
    (adjacent l15 l23)
    (adjacent l15 l16)
```

```
    (adjacent l16 l24)
    (adjacent l17 l25)
    (adjacent l17 l18)
    (adjacent l18 l26)
    (adjacent l18 l19)
    (adjacent l19 l27)
    (adjacent l19 l20)
    (adjacent l20 l28)
    (adjacent l20 l21)
    (adjacent l21 l29)
    (adjacent l21 l22)
    (adjacent l22 l30)
    (adjacent l22 l23)
    (adjacent l23 l31)
    (adjacent l23 l24)
    (adjacent l24 l32)
    (adjacent l25 l33)
    (adjacent l25 l26)
    (adjacent l26 l34)
    (adjacent l26 l27)
    (adjacent l27 l35)
    (adjacent l27 l28)
    (adjacent l28 l36)
    (adjacent l28 l29)
    (adjacent l29 l37)
    (adjacent l29 l30)
    (adjacent l30 l38)
    (adjacent l30 l31)
    (adjacent l31 l39)
    (adjacent l31 l32)
    (adjacent l32 l40)
    (adjacent l33 l41)
    (adjacent l33 l34)
    (adjacent l34 l42)
    (adjacent l34 l35)
    (adjacent l35 l43)
    (adjacent l35 l36)
    (adjacent l36 l44)
    (adjacent l36 l37)
    (adjacent l37 l45)
    (adjacent l37 l38)
    (adjacent l38 l46)
    (adjacent l38 l39)
```

```
    (adjacent l39 l47)
    (adjacent l39 l40)
    (adjacent l40 l48)
    (adjacent l41 l49)
    (adjacent l41 l42)
    (adjacent l42 l50)
    (adjacent l42 l43)
    (adjacent l43 l51)
    (adjacent l43 l44)
    (adjacent l44 l52)
    (adjacent l44 l45)
    (adjacent l45 l53)
    (adjacent l45 l46)
    (adjacent l46 l54)
    (adjacent l46 l47)
    (adjacent l47 l55)
    (adjacent l47 l48)
    (adjacent l48 l56)
    (adjacent l49 l57)
    (adjacent l49 l50)
    (adjacent l50 l58)
    (adjacent l50 l51)
    (adjacent l51 l59)
    (adjacent l51 l52)
    (adjacent l52 l60)
    (adjacent l52 l53)
    (adjacent l53 l61)
    (adjacent l53 l54)
    (adjacent l54 l62)
    (adjacent l54 l55)
    (adjacent l55 l63)
    (adjacent l55 l56)
    (adjacent l56 l64)
    (adjacent l57 l58)
    (adjacent l58 l59)
    (adjacent l59 l60)
    (adjacent l60 l61)
    (adjacent l61 l62)
    (adjacent l62 l63)
    (adjacent l63 l64)
    (drone-at l18)
    (safe-zone l3)
    (obstacle l9)
```

```
    (obstacle l10)
    (obstacle l11)
    (obstacle l19)
    (obstacle l27)
    (obstacle l45)
    (obstacle l46)
    (obstacle l47)
    (obstacle l48)
    (obstacle l61)
    (person-at p1 l15)
    (person-at p2 l36)
    (person-at p3 l58)
    (person-at p4 l63)
    (= (safe-zone-capacity) 6)
    (= (safe-zone-occupancy) 0)
  )
  (:goal (and
    (rescued p1)
    (rescued p2)
    (rescued p3)
    (rescued p4)
))
)
```

Annex B: Python Script for Generating PDDL Problem Files

The following Python script is used to generate problem files for our PDDL domain, by reading a grid-based board from a file.

```python
import sys
import os


def getPred (indent_lev, name, *args):
  pred = "  "*indent_lev + f"({name}"
  for arg in args:
    pred += " " + arg
  return pred + ")\n"


def getBoardDims (board):
  dims = [len(l) for l in board.split("\n")]
```

```python
  assert(len(set(dims)) == 1)
  return (len(dims), dims[0])

def getBoardItems (board):
  # types of items are (D drone) (P person) (S safe zone) (O obstacle)
  posChar = lambda s, c: tuple(i for i,ch in enumerate(s) if ch == c)
  serialized_board = board.replace("\n", "")
  return {"D": posChar(serialized_board, "D"),
          "P": posChar(serialized_board, "P"),
          "S": posChar(serialized_board, "S"),
          "O": posChar(serialized_board, "O"),}

def genAdjacencyRestrictions (bdims, indent_lev):
  restr = ""
  for i in range(bdims[0]*bdims[1]):
    # Vertical adjacency
    if i//bdims[1] != bdims[0]-1:
      restr += getPred(indent_lev, "adjacent", f"l{i+1}", f"l{i+bdims[1]+1}")
    # Horizontal adjacency
    if i%bdims[1] != bdims[1]-1:
      restr += getPred(indent_lev, "adjacent", f"l{i+1}", f"l{i+2}")
  return restr

def genElementRestr (belems, indent_lev):
  restr = ""
  if len(belems["D"]) != 1:
    raise Exception(f"wrong number of drones: {len(belems["D"])} != 1")
  if len(belems["S"]) != 1:
    raise Exception(f"wrong number of safe zones: {len(belems["S"])} != 1")
  restr += getPred(indent_lev, "drone-at", f"l{belems["D"][0]+1}")
  restr += getPred(indent_lev, "safe-zone", f"l{belems["S"][0]+1}")
  for obs in belems["O"]:
    restr += getPred(indent_lev, "obstacle", f"l{obs+1}")
  for pnum, ploc in enumerate(belems["P"]):
    restr += getPred(indent_lev, "person-at", f"p{pnum+1}", f"l{ploc+1}")
  return restr

def getObjects (belems, bdims, indent_lev):
  people = " ".join([f"p{i+1}" for i in range(len(belems["P"]))])
  locations = " ".join([f"l{i+1}" for i in range(bdims[0]*bdims[1])])
  return " "*indent_lev + people + " - person\n" + " "*indent_lev + locations + " - location\n"

def getGoals (belems, indent_lev):
```

```
  return "".join([getPred(indent_lev, "rescued", f"p{i+1}") for i in range(len(belems["P"]))])



if len(sys.argv) != 4:
  raise Exception("this program expects input board path, safe zone capacity and t/f for verbose")
if sys.argv[3] not in ('t','f'):
  raise Exception("verbosity level not in (t,f)")
board_path = sys.argv[1]
safe_zone_capacity = int(sys.argv[2])
verbose = (sys.argv[3] == 't')
with open(sys.argv[1], "r") as f:
  board = f.read()

bdims = getBoardDims(board)
belems = getBoardItems(board)
objs = getObjects(belems, bdims, 2)
restr = genAdjacencyRestrictions(bdims, 2) + genElementRestr(belems, 2) +\
        f"    (= (safe-zone-capacity) {safe_zone_capacity})\n" + "    (= (safe-zone-occupancy) 0)\n"
goals = getGoals(belems, 2)

if verbose:
  print(f"Board ({bdims[0]}x{bdims[1]}):\n{board}\n")
  print(f"Elements: {belems}\n")
  print(f"Objects: {objs}\n")
  print(f"Restrictions:\n{restr}\n\n")

domain = fr"""(define (problem {os.path.basename(sys.argv[1]).split(".")[0]})
  (:domain rescue_drone)
  (:objects
{objs})
  (:init
{restr}  )
  (:goal (and
{goals}))
)"""

print(domain)
```

Listing 7: Python Script for Generating PDDL Problem Files