Emergency Services Logistics Problems

Davide Modolo (#229297)[1], Francesco Laiti(#232070)[2]

Abstract
The goal of this project was to solve five different scenarios involving injured/needy people and delivery robots using PDDL/HDDL planners. This report addresses and explains our implementation and the choices (assumptions) taken to do so. It explores "simple" planning, hierarchical tasks, durative actions and a PlanSys2 integration. The second scenario is built on the first, the third and the fourth are built on the second and the fifth is built on the fourth. The code is available on GitHub [1].

[1] *davide.modolo@studenti.unitn.it*
[2] *francesco.laiti@studenti.unitn.it*

Introduction

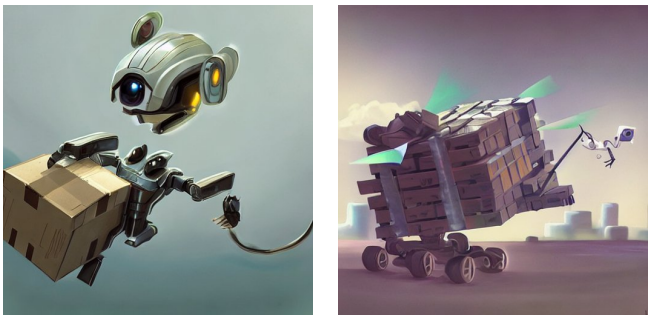
The main goals of this assignment are to model planning problems using PDDL/HDDL and use state-of-the-art planners to solve them.

The considered scenario is inspired by an emergency services logistics problem. The objective is to use robotic agents to deliver boxes containing emergency supplies to injured individuals in fixed positions.

The five tasks in brief are:

1. one robotic agent that can carry only one box has to deliver the supplies;
2. the robotic agent can now attach itself to a carrier that can load up to four boxes;
3. translate the problem 2. using hierarchical tasks;
4. translate the problem 2. using durative actions;
5. implement problem 4. for PlanSys2 planner using fake actions.

The list of planners we used is the following: **planning domains** (using their API [2]), **lama**, **optic**, **tfd** (the last three using *planutils* [3] library/interface/vm?), **PANDA** (downloaded from [4]) and **PlanSys2**. For PANDA, we needed Java version 8 [5] as recommended [6].



(a) 1^{st} task (one box per robot) **(b)** $2^{nd} \geq$ tasks (one carrier per robot with many boxes)

Figure 1. How we imagined the robots for the different scenarios, images by Stable Diffusion[7]

1. Tasks Formalization

We were tasked to address this problem in five different scenarios. The idea was to improve the implementation with more and more constraints. The second scenario is built on the first, the third and the fourth are built on the second and the fifth is built on the fourth.

When we will refer to **planner.domains**, we used their API functions inside a simple python script to get the results without copy-pasting domains and problem in the web UI.

The objective is to coordinate the robots to deliver boxes with emergency supplies to each injured person. The following assumptions are made:

1. Injured people are stationary and located at specific positions.
2. Boxes are initially placed at specific locations and can contain various contents like food, medicine, or tools.
3. Each person may or may not have a box with specific contents.
4. Robotic agents can fill, empty, pick up, move, and deliver boxes, with conditions on location and contents.
5. Multiple people can be at the same location, and box locations don't determine recipients.
6. Robotic agents can move directly between any location.
7. The goal is to accommodate future expansion with multiple robotic agents.

1.1 Task 1

For the first task, our initial state can be described with:

- all boxes are empty and at location **depot**
- all supplies are at location **depot**
- no injured people at location **depot**
- single robotic agent at location **depot**

A single robot can carry one single box at a time. The goal is to deliver each person the supplies they need. We imagined this type of robotic agent as shown in Figure 1a.

In our problem, both person1 and person2 needed food, and person2 needed also tools and medicine. Since people

don't care exactly which content they get, our goal includes the `or` condition for the food delivery. The `delivery` action, symbolizes the act of emptying a box to give the supply to the injured person.

This problem has been run using both **planner.domains** planner and **lama** planner. The first one returns just the first plan found, while the second planner can keep the search running, but in our case it returned just the first one because it was the optimal one.

Results will be analyzed in the "Results" section and the entire result will be appended at the end of the report.

1.2 Task 2

In addition to the first one, the second task changed how boxes are carried around. Now, instead of carrying one box at a time, a robotic agent can be attached to a carrier that can transport a maximum of 4 boxes.

Similarly to task 1, we imagined this type of robotic agent as shown in Figure 1b.

We created two different versions of this problem, the first one more compact but with just a single planner supporting it, using `:numeric-fluents` that allowed us to write the maximum number of boxes that can lay on a carrier as a variable; the second version is used as a baseline for problems three and four, and it also allowed us to use the **lama** planner (to search for plans that are better than the first found).

We also created a problem for the second type of domain having only three boxes and four necessary deliveries to show the ability and the possibility to use multiple times a single box with different supplies.

1.3 Task 3

As required, we kept the same actions as the solution proposed for Problem 2, with the same initial conditions and goal.

In this task, it was required to write the domain file using hierarchical actions (with tasks and methods).

We implemented the task `T_deliver_supply` in a recursive way, which resulted in a longer time required for the planner to find a plan, but it increased the readability of the code. Still, we kept the basic implementation (without the recursion) as a comment in the code.

We had to remove the `either` keyword from the predicate `location_at` since it was not supported; also we wrote two methods for the task `move_robot`, with and without the carrier attached.

1.4 Task 4

As required, we modified the domain in the second task adding a duration to the actions. We tried to be as consistent as possible; the full list of durations we gave to every action can be read in the section 'Predicates, Actions and more'.

Just like the second task, we also created a version with just three boxes and four deliveries to show that the domain is still consistent and the problem is solvable.

We also created a variation of the domain removing the `:negative-preconditions` to test with the **optics** planner.

1.5 Task 5

Writing the problem to make it work with PlanSys2 means creating some "fake actions" in C (or C++) that in real life should be mapped, for example, to the real movement of the robot. In our case, we printed in the console the movements as well as their progress.

To run this task, it's required to have two different terminals opened. One will run the `launch_terminal1.sh` file, the other one will run `launch_terminal2.sh`.

TODO: explain better and longer, with also the steps to run it (?)

2. Problem modelling

TODO: Write about:

- deliver means remove from carrier, remove supply without giving a location to the supply so it can't be taken again (like "the person consumes it"); and so on
- Evidence that planners can find solutions, should we add the outputs to this file or just "look at github for results"?
- ...

3. Predicates, Actions and more

Everything is commented on the GitHub repository, still, we will list here the predicates and the actions.

3.1 Task 1

When we have *predicate* and *not_predicate*, it is done to avoid situations in which the robot would pick up a box another robot has or similar.

Predicates

- located_at (either robotic_agent box supply person)
- robot_has_box
- robot_has_no_box
- box_with_supply
- box_is_empty
- delivered

Actions

- move_robot
- take_box
- drop_box
- fill_box
- empty_box
- deliver

3.2 Task 2

We only added the predicates and actions that we added to task 1. The version using `:numeric-fluents` differs from the standard one only by having the variable for the boxes and the required changes to the actions.

Predicates

- box_on_carrier
- box_loaded
- carrier_has_no_robot
- robot_has_no_carrier
- robot_carrier_attached
- carrier_has_no_boxes
- carrier_has_one_box
- carrier_has_two_boxes
- carrier_has_three_boxes
- carrier_has_four_boxes

Actions

- move_robot_with_carrier
- attach_carrier_to_robot
- detach_carrier_from_robot
- load_box_on_carrier
- unload_box_from_carrier

3.3 Task 3

In this sub-section, we will also list the tasks and the methods for the hierarchical implementation. Since the either keyword wasn't supported, we wrote the locations of every object as separate predicates. Any predicate or action already implemented in the previous problem will not be present in the following lists.

Predicates

- located_at_carrier
- located_at_robot
- located_at_box
- located_at_supply
- located_at_person

Primitive Tasks

- T_move_robot
- T_attach_carrier_to_robot
- T_detach_carrier_to_robot
- T_fill_box
- T_load_box_on_carrier
- T_unload_box_from_carrier
- T_deliver

Non-Primitive Tasks

- T_deliver_supply
- T_return_to_depot
- T_prepare_box

Primitive Methods

- M_move_robot
- M_move_robot_with_carrier
- M_attach_carrier_to_robot
- M_detach_carrier_to_robot
- M_fill_box
- M_load_box_on_carrier
- M_unload_box_from_carrier
- M_deliver

Non-Primitive Methods

- M_deliver_supply_with_already_carrier
- M_deliver_supply_but_first_attach_carrier
- M_deliver_supply_by_loading_supply_from_depot
- M_return_to_depot
- M_prepare_box

Actions no new actions.

3.4 Task 4

TODO: add notes from code

Since this problem required to give every action a duration, here we have the list of our choices and explanations.

We also added a predicate delivery_OR_refactored (with minimal duration since it's just a check); since the `OR` operator was not supported in the goal of **tfd** planner, we encoded a new predicate to say "it doesn't matter who takes which, but person1 and person2 will be getting food1 and food2".

Durations

- move_robot has duration 5
- move_robot_with_carrier has duration 7 (since carrying the weight should reduce the speed of the robotic agent)
- attach_carrier_to_robot & detach_carrier_from_robot have duration 2 (both)
- load_box_on_carrier & unload_box_from_carrier have duration 3 (since we thought that a box filled with a supply would be heavy)
- fill_box has duration 4 (since our idea is that the robot has to physically search and take the required supply when inside the depot)
- deliver has duration 3 (because it implies that the box is already off the carrier)

3.5 Task 5

TODO: what should we write here?

4. Results

4.1 Task 1

Both planners gave us almost the same plan as a result. The main difference is in actions 5 and 6, where the plan we got from `lama` moves the robot to the depot to take another box, while in the `planning.domains` result, the robot just delivers the supplies using only one box.

We can state that the resulting plans are optimal.

The objects for the domain and the problem can be seen in Figures 2 and 3.

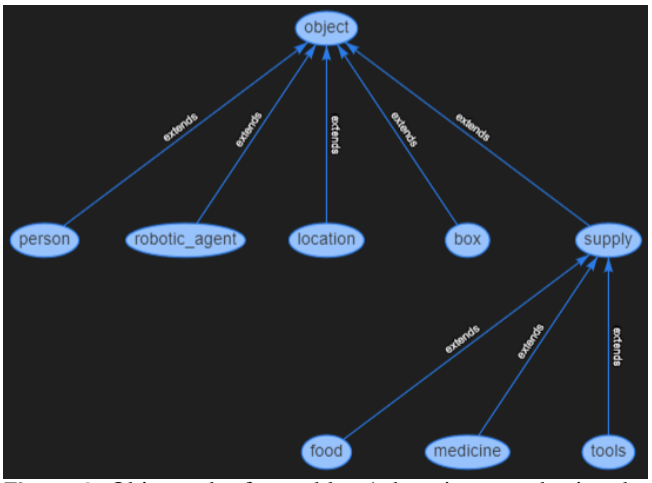


Figure 2. Objects plot for problem1 domain created using the PDDL add-on of VS Code

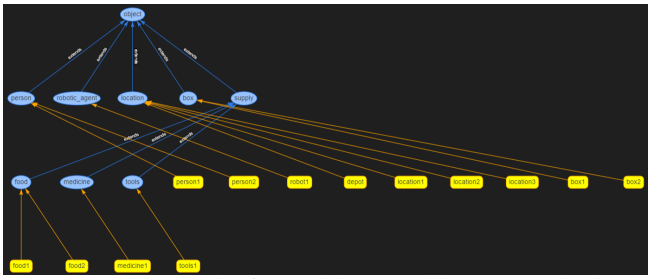


Figure 3. Objects plot for problem1 problem created using the PDDL add-on of VS Code

4.2 Task 2

As briefly explained before, we have two different implementations for this problem.

4.2.1 With `:numeric-fluents`

Unfortunately, it only gives one plan which is suboptimal, where the robotic agent moves one box at a time, even if it fills them all at once.

The objects for the domain and the problem can be seen in Figures 4 and 5.

4.2.2 Without `:numeric-fluents`

Since it was impossible to set up a variable for the maximum number of boxes a carrier can hold, we hard-coded four boxes.

Using **lama** in the *planutils* container, we first got 30 actions, then 28, then 27, then 24, then 21 and then finally down to 20 that we can see it's the most intuitive best approach: fill and load all boxes to deliver them to all three people.

w/ 3 boxes Having only three boxes available we can see that it forces the robotic agent to first deliver the three filled

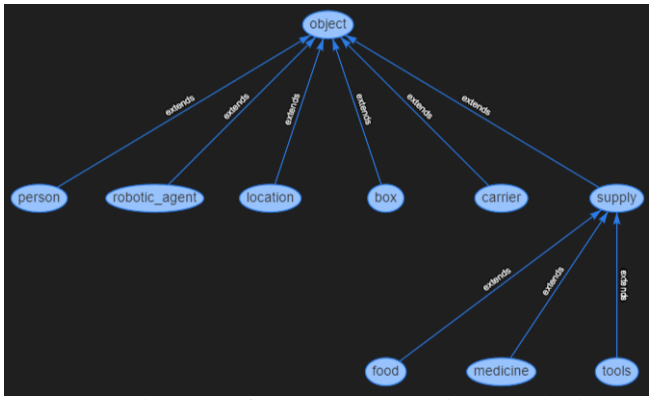


Figure 4. Objects plot for problem2 domain created using the PDDL add-on of VS Code

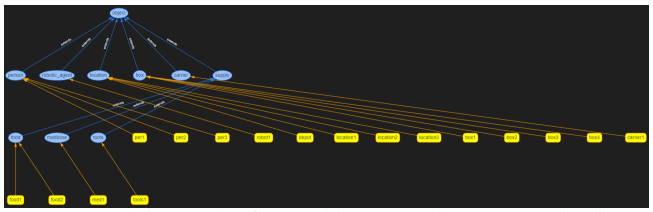


Figure 5. Objects plot for problem2 problem created using the PDDL add-on of VS Code

boxes and then take a "used" box back to the depot in order to fill it up with the last needed supply in order to deliver it.

4.3 Task 3

As an assumption, we decided to give each goal a "priority" (encoded as the ordering in the tasks). This was to "simulate" the idea of people with worse health conditions need higher priority (and some of them should be served before the others) and people who need tools have less priority than people that need food and/or medicine. As a result, the ordering is required.

Talking about the resulting plan, we can see that the robot works with only one box at a time, resulting in a sub-optimal solution. Still, since the planner returns only the first plan found, a different plan could require a different implementation.

We also tried to remove the ordering in the goal for test purposes, but unfortunately, we were not able to test this case properly. Even increasing the heap size of the JVM to 26GB (the maximum we had available) didn't help and after an hour and a half and almost 47 million nodes generated, PANDA was not able to find a plan. The command used to do so was `-Xmx26g` and the last line before stopping was: *nodes/sec: 8642 - generated nodes: 46767098 - fringe size: 17800537 - current modification depth: 33 - g(s)+h(s)= 51*

The same behaviour appeared without the "recursion" in the delivery method.

Since the `.hddl` extension is not recognized by the VS Code PDDL extension, we were not able to retrieve the plots as we did for the other tasks. Still, they should be identical

(or almost identical) to the one of Task 2 (Figures 4 and 5).

4.4 Task 4

TODO: write something more here.

Since this was an evolution of the Task 2, objects for the domain and the problem are the same as that task (visible in Figure 4 and 5).

4.4.1 With negative preconditions

Even in this task, the resulting plan is sub-optimal, for example, we can see that in the last three actions, the agent goes with the carrier to the depot to do nothing and then it goes to the last injured person to deliver the required supply.

w/ 3 boxes This adds just a single step to the final plan; showing again that the first plan is not optimal.

4.4.2 Without negative preconditions

We have one more step than the result using negative preconditions.

4.5 Task 5

TODO: write something more here.

A solution was successfully found for the extended problem using PlanSys2. The plan comprised a sequence of actions that were executed in parallel where feasible, and their execution was simulated step by step. The plan length was 25 steps.

The output indicated the timeline and details of each action's execution, including attaching carriers to robots, filling boxes, loading boxes onto carriers, moving robots with carriers, unloading boxes, and delivering items to various locations and individuals.

Throughout the execution of the plan, the system indicated progress for each action, such as attaching carriers, filling boxes, loading, moving, unloading, and delivering, with all actions reaching completion (100%).

5. Conclusions

To sum up, we wrote a PDDL implementation of the first task, we then expanded it adding the carrier so that the robot could carry more than one box at a time. The robot + carrier version has then been used as the baseline for the implementation of the last three implementations.

The *hierarchical* version (HDDL) uses the goals ordering to give a certain priority to every goal.

In the *durative actions* version we tried to give each action a consistent duration, justifying every value we gave.

For the *PlanSys2* implementation, we implemented some C++ functions (fake actions) that simulate the physical actions the robotic agent should do (with the progress printed as percentage).

Talking about results, some implementations gave us optimal plans after a few iterations, while we got sub-optimal plans in other implementations. This was mainly due to the fact that some planners stop the search for a goal when they find the first one, without the ability to look for a better one. In general, all the optimal plans we found follow the idea of preparing all the boxes first and then delivering in one shot (if there are enough boxes of course).

5.1 Future Works

It could be interesting to explore alternative and various approaches for future implementations, such as:

- use different durations in Task 4
- write different hierarchical tasks in Task 3
- test with more planners (mainly where we got only one plan)

References

[1] GitHub Repository of this Project. [Online]. Available: https://github.com/davidemodolo/Automated_Planning_Project

[2] planning.domains API. [Online]. Available: https://api.planning.domains/

[3] Planutils repo and Docker Image. [Online]. Available: https://github.com/AI-Planning/planutils

[4] PANDA Planner. [Online]. Available: https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.090/panda/PANDA.jar

[5] Docker Image for Java 8 'openjdk:8u342-jre'. [Online]. Available: https://hub.docker.com/_/openjdk/tags

[6] Java 8 recommendation. [Online]. Available: https://github.com/galvusdamor/panda3core

[7] Stable Diffusion online for Image Generation. [Online]. Available: https://stablediffusionweb.com/#demo

6. Planners results

The following pages contain the setups and the solutions of the planners.