The "Health Care Scenario" Assignment Report

Automated Planning Theory and Practice — Academic Year 2024-2025

Alessandro Lorenzi (mat. 247177)
University of Trento, Italy
alessandro.lorenzi-1@studenti.unitn.it

1 INTRODUCTION

This report is prepared as part of the assignment for the course *Automated Planning Theory and Practice* (AY 2024-2025) [9]. The objective of the work is to model and solve planning problems in a healthcare scenario using PDDL (Planning Domain Definition Language) and HDDL (Hierarchical Domain Definition Language), as well as to integrate a temporal planning model within a robotic framework leveraging the `PlanSys2` infrastructure in ROS2. The assignment focuses on developing and testing planning models with increasing complexity:

(1) Modeling a base PDDL domain for task execution.
(2) Extending the base PDDL domain with more complex tasks.
(3) Structuring tasks using Hierarchical Task Networks (HTN).
(4) Introducing temporal planning with durative actions and concurrency constraints.
(5) Executing the temporal model in `PlanSys2`.

The deliverables include the PDDL/HDDL domain and problem files, along with the corresponding PlanSys2 implementation. The report documents the design choices, planner selection, and execution details, as well as an analysis of the results obtained, including a discussion on scalability and solver configurations.

The structure of this document is as follows: Section 2 describes the problems and the assumptions in detail. Section 3 outlines the modeling approach. Section 4 discusses the design choices, and Section 5 presents the results and analysis. Finally, Section 6 provides the conclusions.

2 PROBLEM DESCRIPTION

The presented scenario addresses the complex challenge of coordinating robotic agents to deliver medical supplies and accompany patients within a healthcare facility. The domain needs, in fact, to model the activities of both robots that (with or without carriers) have to manage boxes and contents, and also those dedicated to assist and travel with patients between the different locations.

2.1 Scenario Understanding

The key components and assumptions of this environment include the following:

(1) **Physical Layout**: The environment consists of multiple locations, including the entrance, central warehouse, corridors, and specialized areas, which shape the movement of robots and other objects within the scene. Specific connections between locations form the "road map" of available paths.
(2) **Medical Units**: Each unit is positioned at a specific location within the facility and may require particular medical supplies. Additionally, medical units can exist in different states concerning both supplies (needing or possessing contents) and patients.
(3) **Robotic Agents**: There are two distinct types of robots, each designated for a specific task: one for delivering supplies and the other for transporting patients. These robots cannot perform tasks outside their assigned roles. Both types of robots can move throughout the facility, either empty or carrying a patient or boxes.
(4) **Boxes**: Each box serves as a container for transporting a single supply. If a box is empty, a robot can fill it with a supply, making it ready for delivery. Boxes can be filled, loaded, emptied, and reused.
(5) **Supplies**: Various medical supplies (e.g., scalpels, aspirin, etc.) are typically stored in the central warehouse at the beginning.
(6) **Patients**: Each patient may need to reach a specific medical unit and is typically located at the entrance initially, requiring robotic assistance for transportation.
(7) **Supply Delivery**: To deliver a specific supply to a medical unit in need, an available robot must first reach the location where both the supply and an empty box are present. The robot fills the box, loads it onto itself, and begins moving, updating the locations of both the robot and the box. Upon reaching the destination, the robot unloads the box, making itself available for another task. The supply is then successfully delivered, eliminating the unit's need for it. A key assumption is that **unloaded empty boxes can be reused**, allowing the robot to repeatedly perform the cycle of filling, transporting, unloading, and refilling, even with a limited number of boxes. As detailed below, initially, robots can carry only one box at a time. However, from the second domain onward, carriers are introduced, enabling multiple loadings.
(8) **Patient Accompaniment**: Similar to supply delivery, when a patient needs to be transported to a specific medical unit, an available robot must first reach the patient's location and pick it up. As the robot moves, both its location and the patient's position are updated. The robot then transports and delivers the patient to the designated unit, completing the task and ensuring that the patient no longer requires transport. The domain assumption is that each robot can accompany only one patient at a time.

Starting from the second task, new assumptions are required to implement alternative way of moving boxes:

(1) **Carriers**: A newly introduced object type that enables robots to handle multiple supply deliveries efficiently. Carriers are locatable and assigned to specific robots, meaning each robot retains its designated carrier throughout the entire problem lifecycle.
(2) **Carrier Capacity**: Each carrier has a predefined maximum capacity, which is problem-specific and determines the number of boxes a robot can load, transport, and unload at a time. A key assumption is that **each carrier can hold up to three boxes**, as detailed in Section 4.3. This capacity constraint influences all carrier-related operations, ensuring that the system remains efficient and manageable even with multiple deliveries occurring simultaneously. The domain structure is designed for easy scalability, supporting larger capacities while maintaining explicit state management.
(3) **Supply Delivery with Carriers**: The introduction of carriers significantly enhances the efficiency of robotic agents by allowing them to load multiple boxes at once, optimizing the planning process and reducing the number of operations required. By minimizing redundant trips, this approach decreases the total execution time and improves resource utilization. Carrier operations are **position-specific to ensure accurate state tracking**, making it possible to determine both the number and identity of the boxes on each carrier at any given moment.
(4) **Movements with Carriers**: Since each robot is directly linked to its designated carrier, movement actions are inherently associated with both the robot and its carrier. This means that every movement affects not only the robot but also the carrier and any loaded boxes. Consequently, a robot cannot move independently without its assigned carrier, even when empty.

2.2 Initial State

The healthcare domain begins in a standardized configuration designed to optimize its logistical operations, reflecting a practical approach to healthcare resource management. The central warehouse acts as the primary logistics hub, housing all supplies and available boxes. Medical units are distributed throughout the facility in a logical arrangement that reflects real-world healthcare needs. Supply-handling robots are positioned at the warehouse, ready to begin distribution tasks, while patient-transport robots wait at facility entrances to assist incoming patients. When carrier systems are employed, they are integrated into this initial setup with clearly defined capacities and assignment. Despite this typical starting setting, the system has demonstrated its capability to handle various **alternative scenarios**, as reported in Sec. 5. For instance, supplies and boxes could be initially distributed across different locations, robots might start from different positions, as well as patients.

2.3 Goal

The goal state ensures that medical units have all necessary supplies for their specific functions and patient transportation must be completed correctly, with each patient reaching their designated care unit through the most appropriate route. Note that goal can include:

- medical units requiring one or more supplies and / or one or more patients;
- medical units having some supplies and / or patients;
- medical units not requiring any supplies and / or patients;
- restore some types to their initial state (e.g., box must be at warehouse, empty and unloaded at end, robots must return at warehouse, etc.).

3 DOMAIN MODELING PHILOSOPHY

Developing this healthcare logistics domain, a deliberate choice was made to embrace simplicity through **rigorous adherence to basic strips requirements** (`:strips`), consciously avoiding fancy PDDL operators (as "forall" or "exists"), conditional effects ("when"), or other features like numeric fluents.
This minimalist approach stems from several practical considerations. First, not all planners support these advanced features, and those that do might handle them differently. By restricting ourselves to basic STRIPS constructs - simple predicates, clear preconditions, and straightforward effects - we ensure **maximum compatibility across planning systems**. This becomes particularly valuable when transitioning between different planning paradigms, from classical PDDL to HTN and temporal planning, as well for the introduction of action durations and the execution in `PlanSys2`.
The decision to avoid universal and existential quantifiers, while potentially making some predicates more verbose, actually simplifies the domain's logical structure. Rather than relying on complex quantified expressions that might hide subtle dependencies, each relationship is explicitly stated. This transparency makes the domain easier to debug, verify, and modify, as analyzed in Sec. 5.

4 DESIGN CHOICES

In this section, all design choices are presented and briefly discussed. First, the objects of interest are introduced, as they are common to all problems. Then, the actions for each task are quickly explained. Since most of them remain unchanged across various tasks, after the initial introduction, only new instructions or modifications are reported. For the sake of conciseness, predicates are not listed here but can be found in the domain files, where comments and self-explanatory names provide clarity. However, whenever new key predicates are introduced in a task, an explanation is provided in the corresponding section.

4.1 Object Types

The locatable objects, that are medical units, boxes, supplies, robots, carriers, and patients, can be positioned throughout the facility. Robots are further specialized into box-handling and patient-transport types to reflect their distinct roles. The location type represents the different areas where these entities can exist. Here are the defined types (note that carrier is introduced from problem 2):

```
medical_unit supply patient box
carrier robot - locatable
robot_box robot_patient - robot
location - object
```

4.2 Problem 1

The actions implemented for the first basic domain are the following:

- `fill_box`: if the robot, the box, and the supply are in the same location and the box is empty and unloaded, then the box has the supply and is not empty anymore;
- `load_box`: if the robot and the box are in the same location, the box in unloaded and the robot available; causing the box to be loaded in the robot and the robot to not be empty;
- `unload_box`: if the box is loaded in the robot; causing the box to be unloaded and the robot to be empty;
- `deliver_supply`: if robot, box and unit are in the same location, and the unit requiring the supply in the box; causing the unit to have the supply and not need it anymore, and the box to be empty;

- `pick_up_patient`: if the robot and the patient are in the same location, the patient in unloaded and the robot available; causing the patient to be loaded in the robot and the robot to not be empty;
- `drop_off_patient`: if the patient is loaded in the robot; causing the patient to be unloaded and the robot to be empty;
- `deliver_patient`: if robot and unit are in the same location, the patient is accompanied and require the unit; causing the patient to be in the unit and not need it anymore;

- `move_empty_robot_box` if locations are connected; causing the update of the robot location;
- `move_robot_with_box` if locations are connected; causing the update of both the box and the robot location;
- `move_empty_robot_patient` if locations are connected; causing the update of both the box and the robot location;
- `move_robot_with_patient` if locations are connected; causing the update of both the patient and the robot location;

The formulization of the goals in the problem files is based on the `unit_has_supply` and `patient_at_unit` predicates.

4.3 Problem 2

In the second and next domains, the carriers are introduced and new actions are implemented for the handling of multiple box operations. As introduced in 2.1, the assumption here is that each robot-box has a maximum capacity of 3 boxes and this requires specific operation sets for different load states:

- 3 distinct load operations (one per box position)
- 3 distinct unload operations (one per box position)
- 3 distinct movement operations based on load state

The domain structure supports easy extension for larger capacities: to increase capacity to N boxes, it is needed to simply add N load, N unload and N movement actions.

The new needed actions are so:

- `load_first_box`
- `load_second_box`
- `load_third_box`
- `unload_one_box`
- `unload_from_two`
- `unload_from_three`

- `move_empty_carrier`
- `move_carrier_one_box`
- `move_carrier_two_boxes`
- `move_carrier_three_boxes`

This domain requires the new `robot_has_carrier` predicate, as well as `has_capacity_one`, `has_capacity_two`, etc., to define the actual carrier capacity in the problem file. These allow to have carriers with different maximum capacity assigned to the different robot-box. The domain instead works with `has_one_box`, `has_two_boxes` and `has_three_boxes` predicates. The decision to **avoid numeric fluents** aligns with the design philosophy outlined in Section 3, prioritizing maximum compatibility across planners. Instead, a more verbose predicate-based representation is preferred, as it eliminates potential issues and unexpected behaviors that may arise from handling numerical values differently across planners.

All the actions and predicates regarding the patient are the same of first domain.

4.4 Problem 3

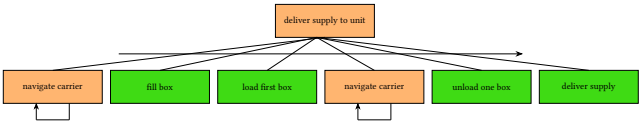
The third scenario is modeled using Hierarchical Task Networks (HTN). All actions and predicates remain the same as in Problem 2, but four new abstract tasks have been introduced, along with their corresponding methods.
In particular, the two main tasks defined and used in the goal formulation are:

- `deliver_supply_to_unit`
- `accompany_patient_to_unit`

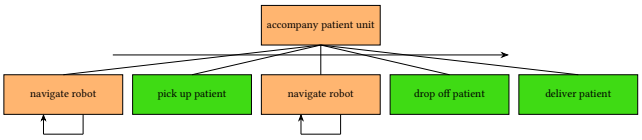
Additional tasks include those related to the movement of the robots, namely `navigate_carrier` and `navigate_robot_patient`. These tasks are handled using multiple methods, introducing recursion to allow consecutive steps until the destination is reached. More details, along with comments, are provided directly in the corresponding `.hddl` domain file.

Figure 1 is an illustration of the entire **decomposition of the two main tasks**. Note that the first task is extended to versions involving two and three boxes, but for simplicity, only the decomposition with a single box is displayed.

The goal is to incorporate knowledge into the tasks by refining the order of steps necessary to achieve the objective. Specifically, to deliver one or more supplies in boxes to the units, the robot must first move to the location of the supply (typically the central warehouse), fill the box with the supply, load the box into its carrier, navigate to the destination unit, and finally unload the box to deliver the supply. Similarly, for a robot to accompany a patient, it must first move to the patient's location, pick up the patient, navigate to the required unit, drop off and deliver the patient who no longer needs to be transported.



(a) Task: Deliver (one) Supply to Unit



(b) Task: Accompany Patient to Unit

Figure 1: Tasks for Problem 3. *The orange boxes represent the non-primite abstract tasks while green ones are the primitive (the actions), the straight lines illustrate the decomposition and the horizontal arrow indicates the sequence.*

4.5 Problem 4

This problem introduces the durative actions extension, where time and temporal constraints must be considered. The goal is to add a duration for each action from Problem 2 and integrate time constraints to prevent certain operations from being executed in parallel. The new domain adopts the `at start`, `over all`, and `at end` keywords before any predicate in the conditions and effects of the actions, as well as their durations.

The domain defines two fundamental predicates:

- `robot_box_not_busy`
- `robot_patient_not_busy`

These predicates are used to manage actions that can or cannot be executed in parallel, tracking the state of each robot. Specifically, **each agent can perform only one operation at a time**, and each action is considered **atomic**, meaning it must finish before another can begin. This prevents unrealistic scenarios, allowing multiple robots to work together in **parallel** without interfering with each other or exhibiting impossible behaviors.
The specific duration of each action is chosen to reflect a real-world scenario as closely as possible and are summarized in Table 1.

Action	Duration
Fill box	3 time units
Load box (1, 2 or 3)	2 time units
Unload box (1, 2 or 3)	2 time units
Deliver supply	1 time unit
Pick up patient	2 time units
Drop off patient	2 time units
Deliver patient	1 time unit
Move empty carrier	2 time units
Move carrier with box (1, 2 or 3)	3 time units
Move empty robot	2 time units
Move robot with patient	3 time units

Table 1: Duration of Actions

4.6 Problem 5

The final problem involves the execution of the Problem 4 scenario in PlanSys2 [6], a project implemented in ROS2 and inspired by ROSPlan [2], which is designed to integrate PDDL-based planning systems into robotics environments.

Following the tutorial [10], for each action the corresponding `Action Executor Client` class in C++ is developed, containing a "fake" implementation of the actions in order to build the matching performers within the node structure. The launcher includes a call to the exact same domain as in Problem 4, and it runs the executables that implement the PDDL operations. The injection of system knowledge is performed to ensure the same initial setting and goal, as discussed in Section 5.5, along with the execution details.

5 RESULTS

This section presents the results obtained and their analysis across the five different tasks. Each subsection outlines the problem setup, the planners utilized, the testing process and the corresponding outcomes, emphasizing the strengths and limitations of various approaches.

5.1 Problem 1

To comprehensively evaluate the domain, different problems of increasing complexity are tested. Both the report and the code project document these three selected problems, chosen for brevity while covering diverse settings:

- **`p1.pddl`**: The most complex scenario, following the traditional setup described in Section 2.2. It features a diversified roadmap, multiple supplies, and several patients. Given its size, solving it requires significant computational effort.
- **`p2_simple.pddl`**: A simpler scenario with two supply locations and two patients, where items are positioned in different locations. Only one robot of each type is available.
- **`p3_simple_multiple.pddl`**: Similar to `p2_simple.pddl` but with a total of four robots, allowing for an evaluation of multi-agent coordination.

These problems are solved using different planners and configurations to analyze their efficiency and suitability:

- `LAMA-first` [8] by `Planutils` [7] which quickly finds satisfying solutions within a reasonable time, but it does not guarantee optimality.
- `Fast Downward` [4] (A* + LM-cut) which guarantees optimality but is computationally expensive. It performs well on simple problems but struggles with time resources as complexity increases.
- `LAMA` [8] by `Planutils` [7] which extends `LAMA-first` by refining plans iteratively towards optimality. It eventually finds the optimal solution, but its runtime increases exponentially for larger problems.

The results confirm that the domain is well-structured for all tested problems. Since robots can only handle one box at a time in these settings, in the complex scenario (`p1.pddl`) the search space grows rapidly, making `LAMA-first` the only viable option for obtaining a solution in a reasonable timeframe. The simpler problems allow

for direct comparisons of planner performance, with both Fast Downward and LAMA producing optimal solutions efficiently.
Figure 2 in Appendix 6 is an example of planner output.

The folder contains the output plans of all the problems, p1.pddl was solved using LAMA-first, while the simplest two were solved using Fast Downward. Detailed instructions for running the experiments are included in the README file.

5.2 Problem 2

Similar to the previous problem, various settings and goals are tested to evaluate the domain's conformity, this time incorporating the new carrier type with distinct capacities to assess the robot's ability to handle multiple boxes simultaneously. The **p1.pddl** file retains the same roadmap and goals as the previous one, but introduces two types of box-robots (and carriers), one with a maximum capacity of 3 boxes and the other with a maximum capacity of 2 boxes. This modification is made to examine the robot's and carrier's ability of control. The patient setup remains identical to that of scenario 1. To manage the increased complexity, this problem is tested using LAMA-first for faster execution, and the output plan shows a near-optimal solution where both robots decide to handle 2 boxes at a time, significantly speeding up the results compared to handling only one box at a time. The simpler problem, **p2_simple.pddl**, is essentially a variant of the one in the previous domain, modified to include the carrier with a 3-box capacity, allowing for further testing of the Fast Downward and LAMA planners.

The results demonstrate that the planners efficiently find optimal solutions, with the robots correctly choosing to **load multiple boxes at a time** while respecting their respective carrier capacities, thereby delivering supplies more quickly and efficiently. As previously noted, increasing the problem's complexity results in an exponentially larger search space, especially when handling multiple supplies. This growth leads to significant time challenges when using optimal planners, which can be problematic for testing complex problems that require optimal solutions. As a result, suboptimal planners are often relied upon for these types of scenarios.

Execution details and instructions are provided in the README file, along with the output plans: p1.pddl solved using LAMA-first, and the output for the second problem solved using Fast Downward, as shown in figure 3 in Appendix 6. This format remains consistent for all subsequent problems.

5.3 Problem 3

The third domain is based on a Hierarchical Task Network (HTN) and is tested using the PANDA framework (Planning and Acting in a Network Decomposition Architecture) [5]. The tested problem, present in the folder, is inspired by the main problem from the previous scenarios and involves delivering three supplies and accompanying two patients. Since the order of the tasks is fully specified, the planner can find the optimal solution within a relatively short amount of time (~ 7 minutes (418158 ms)). However, as the problem size increases or the order of the tasks is relaxed or unspecified, the required time and memory to solve the task grow substantially. In conclusion, the domain structure allows the PANDA framework to efficiently find a plan that adheres to the task decomposition and respects the predefined order.

Figure 4 in Appendix 6 presents the planner output for a simplified version of the problem, reduced in complexity due to the solution's length. This version includes a single delivery task and one patient transport task.

5.4 Problem 4

The new domain, incorporating the :durative-actions feature, is tested with two temporal planners: OPTIC [1] and POPF [3]. While OPTIC is well-suited for domains requiring complex constraint handling and multi-objective optimization, POPF serves as the default planner in PlanSys2, making its evaluation crucial to ensure compatibility with the Problem 5 scenario.

The primary problem tested (**p1.pddl**) represents a complex scenario with the goal of delivering four supplies and accompanying three patients, utilizing two box-carrying robots and one patient transport robot. The resulting planner output demonstrates the expected behavior, with robots prioritizing the **loading of as many supplies as possible to minimize total execution time**, as the duration of each action is now specified. Additionally, the **robots operate in parallel to enhance optimization**, and, as correctly modeled, there is no overlap in the actions of the same robot.
As previously mentioned, the size of the goal significantly impacts the planner's time and memory requirements for finding an optimal solution. However, compared to the previously tested domain, the framework demonstrates **improved efficiency** due to better optimization.

Figure 5 in Appendix 6 presents the planner output for a simplified version of the problem, identical to **p2_simple.pddl** from Problem 2, featuring two delivery tasks and two patient transport tasks.

5.5 Problem 5

For the final task, the same domain used in Problem 4 is applied with PlanSys2 [6]. The injected knowledge mimics the p1.pddl scenario from the previous problem. The plan obtained from the PlanSys2 terminal is consistent with the results from the POPF planner in Problem 4.
Once the plan is run, the ROS2 terminal visualizes the corresponding execution, showing that all tasks are performed as expected. The results confirm the successful execution of the goal, with robots delivering supplies and transporting patients to the units, demonstrating the feasibility of the proposed solution within the domain and the correct execution of the simulated environment.

All detailed instructions for configuring both PlanSys2 and running the experiments are provided in the README file. A screenshot of the processes launched in the two terminals for this task is shown in Figure 6 in Appendix 6, highlighting the successful execution and visual feedback from the system.

6 CONCLUSIONS

This work successfully demonstrates the complete modeling and solution of the five progressively complex planning problems in the healthcare logistics scenario. Each domain was formalized and effectively solved using appropriate planning techniques.
The use of STRIPS-based modeling ensured robust and compatible solutions throughout the project, enhancing planner compatibility across all domains. This approach facilitated clear and verifiable state transitions in all actions, simplified debugging and model maintenance. As a result, it is possible to manage increasing complexity without compromising solution clarity.

The first domain established a solid foundation by coordinating robots for single-box deliveries and patient transport. The second domain extended this capability by introducing carriers, enabling multi-box operations while preserving correctness and planner compatibility. In the third domain, the HTN formulation successfully decomposed tasks into hierarchical structures, with the `PANDA` framework effectively solving problems that respected both task decomposition and ordering constraints. The temporal domain proved particularly successful, with both the `OPTIC` and `POPF` planners generating valid solutions that correctly handled durative actions and parallel execution constraints. The final integration with `PlanSys2` demonstrated that these planning models could be seamlessly deployed in a robotics framework, with operations successfully executed.

Testing across a range of problem sizes, from simple scenarios to more complex ones, revealed both the strengths and computational challenges of different planners. These results confirm that, while the domain models scale effectively in terms of expressiveness and correctness, practical applications may require careful consideration of problem size and planner selection to meet specific performance requirements and time constraints.

Key achievements include:

(1) Each domain was correctly modeled and successfully solved by its respective planner.
(2) Progressive complexity was handled without sacrificing solution quality.
(3) All planners found valid solutions within reasonable computational bounds.
(4) The transition from theoretical models to practical implementation was achieved smoothly.

The robustness of these solutions across various problem instances, coupled with the successful execution in `PlanSys2`, validates both the modeling approach and the implementation strategy. This work lays a solid foundation for automated planning in healthcare logistics, with each component thoroughly tested and proven effective for its intended purpose.

REFERENCES

[1] J. Benton, Amanda Coles, and Andrew Coles. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. *Proceedings of the International Conference on Automated Planning and Scheduling* 22, 1 (May 2012), 2–10. https://doi.org/10.1609/icaps.v22i1.13509

[2] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, N. Palomeras, Natalia Hurtos, and Marc Carreras. 2015. ROSPlan: Planning in the Robot Operating System. *Proceedings International Conference on Automated Planning and Scheduling, ICAPS* 2015 (04 2015), 333–341. https://doi.org/10.1609/icaps.v25i1.13699

[3] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. 2010. Forward-Chaining Partial-Order Planning. *Proceedings of the International Conference on Automated Planning and Scheduling* 20, 1 (May 2010), 42–49. https://doi.org/10.1609/icaps.v20i1.13403

[4] M. Helmert. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26 (July 2006), 191–246. https://doi.org/10.1613/jair.1705

[5] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. 2021. The PANDA Framework for Hierarchical Planning. *KI - Künstliche Intelligenz* 35", number = (2021). https://doi.org/10.1007/s13218-020-00699-y

[6] Francisco Martín, Jonatan Ginés, Vicente Matellán, and Francisco J. Rodríguez. 2021. PlanSys2: A Planning System Framework for ROS2. arXiv:cs.RO/2107.00376 https://arxiv.org/abs/2107.00376

[7] Christian Muise, Florian Pommerening, Jendrik Seipp, and Michael Katz. 2022. Planutils: Bringing Planning to the Masses. In *ICAPS 2022 System Demonstrations.*

[8] S. Richter and M. Westphal. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research* 39 (Sept. 2010), 127–177. https://doi.org/10.1613/jair.2972

[9] Marco Roveri. 2024. Assignment for the course *Automated Planning Theory and Practice.* Course material. mailto:marco.roveri@unitn.it Academic Year 2024-2025, University of Trento.

[10] ROS2 Planning System Team. 2020. The first planning package. https://plansys2.github.io/tutorials/docs/simple_example.html Accessed: 2025-02-05.

APPENDIX

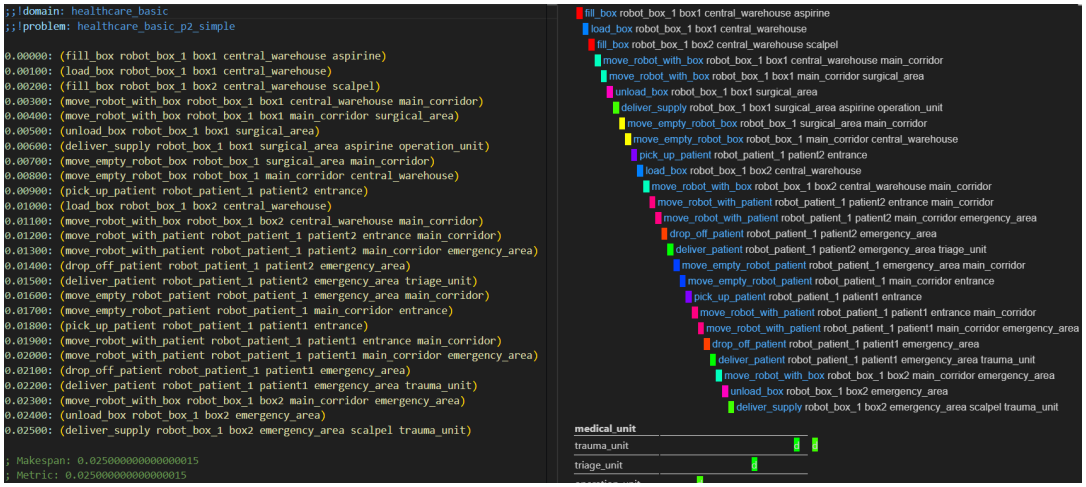


Figure 2: Problem 1. Planner found by `Fast Downward` on `p2_simple.pddl`.

Figure 3: Problem 2. Planner found by `Fast Downward` on `p2_simple.pddl` (robot's carrier has capacity of 2 boxes).

```
SOLUTION SEQUENCE
0: SHOP_methodm_deliver_supply_one_1_precondition(box3,carrier2,emergency_area,central_warehouse,robot_box_2,scalpel,trauma_unit)
1: SHOP_methodm_carrier_is_already_there_11_precondition(robot_box_2,central_warehouse,carrier2)
2: fill_box(robot_box_2,box3,central_warehouse,scalpel)
3: load_first_box(robot_box_2,carrier2,box3,central_warehouse)
4: SHOP_methodm_navigate_carrier_1_via_13_precondition(secondary_corridor,robot_box_2,central_warehouse,carrier2,box3)
5: move_carrier_one_box(robot_box_2,carrier2,box3,central_warehouse,secondary_corridor)
6: SHOP_methodm_navigate_carrier_1_via_13_precondition(main_corridor,robot_box_2,secondary_corridor,carrier2,box3)
7: move_carrier_one_box(robot_box_2,carrier2,box3,secondary_corridor,main_corridor)
8: SHOP_methodm_navigate_carrier_1_direct_12_precondition(box3,main_corridor,carrier2,robot_box_2,emergency_area)
9: move_carrier_one_box(robot_box_2,carrier2,box3,main_corridor,emergency_area)
10: unload_one_box(robot_box_2,carrier2,box3,emergency_area)
11: deliver_supply(robot_box_2,box3,emergency_area,scalpel,trauma_unit)
12: SHOP_methodm_accompany_patient_0_precondition(operation_unit,robot_patient_1,patient2,surgical_area,entrance)
13: SHOP_methodm_robot_patient_is_already_there_6_precondition(robot_patient_1,entrance)
14: pick_up_patient(robot_patient_1,patient2,entrance)
15: SHOP_methodm_navigate_robot_patient_via_8_precondition(robot_patient_1,entrance,patient2,main_corridor)
16: move_robot_with_patient(robot_patient_1,patient2,entrance,main_corridor)
17: SHOP_methodm_navigate_robot_patient_direct_7_precondition(robot_patient_1,main_corridor,patient2,surgical_area)
18: move_robot_with_patient(robot_patient_1,patient2,main_corridor,surgical_area)
19: drop_off_patient(robot_patient_1,patient2,surgical_area)
20: deliver_patient(robot_patient_1,patient2,surgical_area,operation_unit)
```

Figure 4: Problem 3. Planner found by `PANDA` on simplification of `p1.hddl`.

Figure 5: Problem 4. Planner found by `OPTIC` on `p2_simple.pddl`.

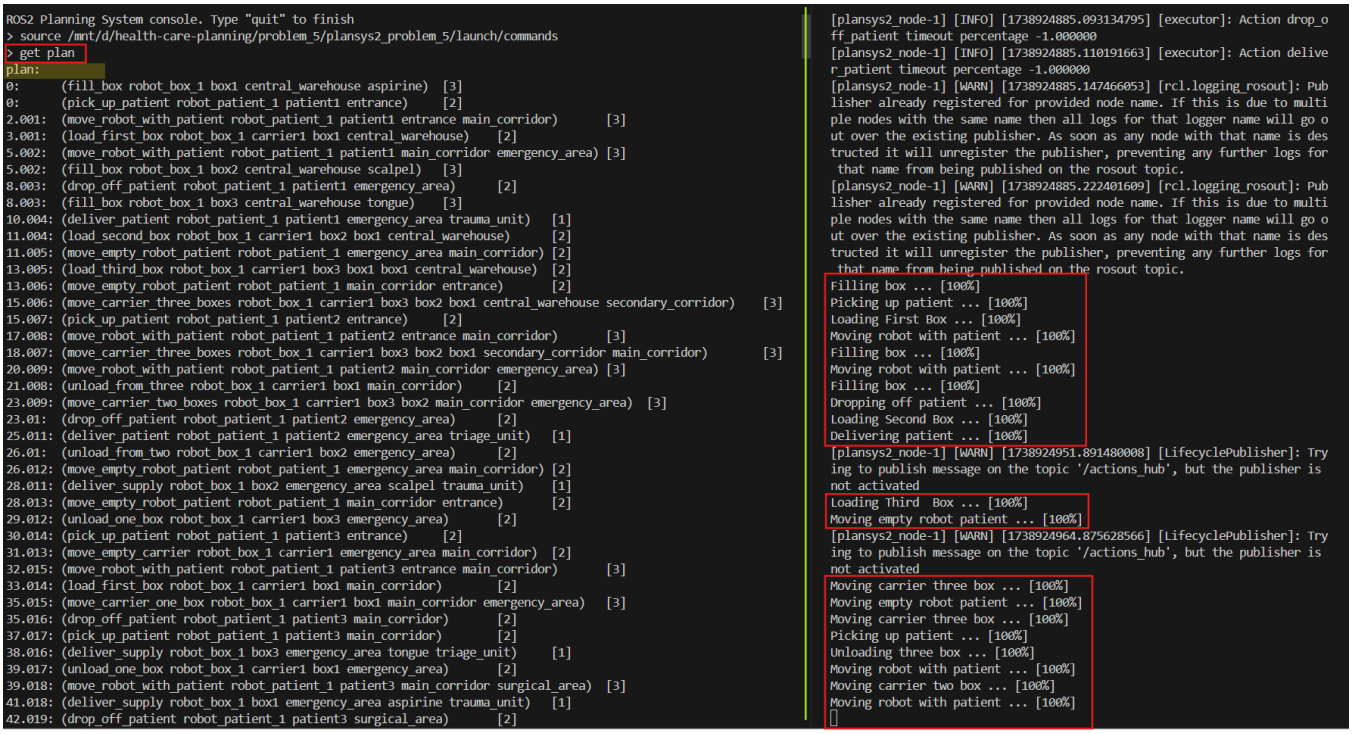


Figure 6: Problem 5. Execution in `PlanSys2`. ROS2 Planning System Control on left terminal and visualization of the plan on the right. Colors and patterns used to highlight key points of interest.