# Automated Planning: Theory and Practice Project

Luca Podavini (257844)
luca.podavini@studenti.unitn.it

Francesco Sorrentino (256151)
francesco.sorrentino@studenti.unitn.it

## I. Introduction

Automated planning is a field of Artificial Intelligence that studies the creation of plans that allow going from an initial state to a desired goal state in a specific scenario. The aim of this project is to design and study different planners and search strategies given an assignment. The work is structured as follows:

- Problem1: Classical planning.
- Problem2: Classical planning with more agents involved.
- Problem3: Planning with Hierarchical task networks.
- Problem4: Temporal planning [1].
- Problem5: Integration with the Plansys2 framework.

### A. Repository

The code can be found at this repository. The directory is organized with different folders for each problem.

*1) Problem1:* The problem1 folder contains:

- Dockerfile: file containing planutils environment.
- domain.pddl: domain file.
- problem_small.pddl: a simple problem to test different planners.
- problem_medium.pddl: a harder problem to make a scalability analysis.
- problem_big.pddl: the problem with the hardest goal for the scalability analysis.
- run_experiment.sh: a bash script to run the entire experiment. After execution a logs folder is generated containing the logs of each planner execution and the generated plans.

*2) Problem2:* The problem2 folder is organized exactly like the problem1 with the same file structure. The main difference is that the domain and problem files are different since there are additional agents that can act.

*3) Problem3:* The problem3 folder contains:

- Dockerfile: file containing planutils environment for singularity and java installation.
- PANDA.jar: HDDL planner.
- domain.hddl: domain file with task and methods implemented.
- problem.hddl: problem file.
- run_experiment.sh: bash script to run the scenario. After execution logs and plans generated are saved in a logs folder.

*4) Problem4:* The problem4 folder contains:

- Dockerfile: file containing planutils environment.
- domain.pddl: domain file containing durative actions.
- problem.pddl: problem file.
- run_experiment.sh: bash script to run the scenario. After execution logs and plans generated are saved in a logs folder.

*5) Problem5:* The problem5 folder has the most complex structure since it follows the Plansys2 [3] example found here.

- launch/: launch directory containing the python script *problem5_launch.py* necessary for creating the ROS nodes and the *commands* file containing the commands that define the problem and generate the plan.
- pddl/: pddl folder contains the *domain.pddl* file with durative actions sligthly modified for Plansys2 [3] integration.
- src/: src folder contains the C++ implementation for the actions defined in the domain file.
- Dockerfile-humble: file containing the configuration for running Plansys2 [3].
- run_experiment.sh: bash script used to execute the framework and open the ros2 terminal. It also creates a launch.log file to monitor the launch process in case of unexpected errors.

This structure was chosen to make the execution simpler since it puts all the files necessary for the execution of that problem in their specific folder.

## II. Scenario

The scenario chosen for the project is the **Interplanetary Museum Vault** scenario that will be called *imv* from now on for simplicity. Note that the following is just one interpretation of the scenario given, with assumptions made to reduce ambiguity as much as possible.

### A. Description

The context of the scenario is that of a subterranean research outpost on Mars that contains artifacts sampled in various regions of the planet. An unexpected event of micro-quakes makes the pressure fluctuate and the various robots inside the structure need to relocate the artifacts to ensure their preservation. Figure 1 shows the structure map in which different zones can be seen:

- Entrance: zone where the robot will start.
- Cryo-Chamber: zone where artifacts can be cooled down.
- Pod-Zone: zone where the anti-vibrational pods are stored.
- Artifact hall $\alpha$: hall where $\alpha$ artifacts are located.
- Artifact hall $\beta$: hall where $\beta$ artifacts are located. It is only accessible when the room is stable and there is no seismic event.

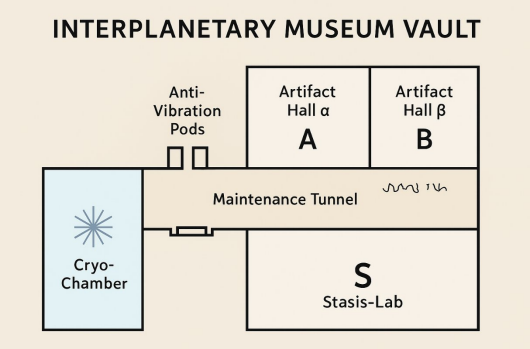


Fig. 1: Map of the imv structure.

- Stasis-lab: zone where artifacts can be preserved.
- Maintenance tunnel: tunnel that connects all other zones. It is an area with low pressure so the robot that passes here needs to be sealed.

Compared to the assignment, the pod zone was interpreted as a single area where the pods are located. Given the map,

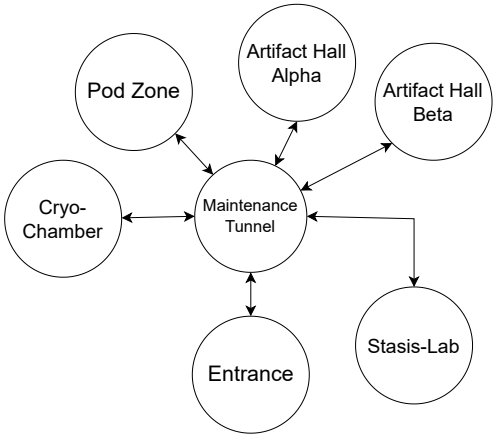


Fig. 2: Graph view of the imv map. The connections where defined starting from the given map.

a formalization was defined for the future modeling steps. A graph was defined where each node contains a zone and the edges show the connection between zones. This was done to clear up any possible ambiguity in the interpretation of the map.

In this environment, different object and agent types can be identified:

- Artifact $\alpha$: artifact to be preserved.
- Artifact $\beta$: artifact to be preserved.
- Martian core samples: artifact to be preserved.
- Robot: agent that must ensure preservation.
- Pod: anti-vibrational pods located at the pod-zone. They are only two available at all times.

### B. Initial state

At the start of the scenario the different objects are scattered in the structure of the imv. Artifact $\alpha$ and $\beta$ start in the respective artifact halls, Martian core samples start in the Cryo-Chamber and the robot starts at the entrance.

### C. Goal state

The goal is to relocate the artifacts while applying also some preservation measures. Specifically, the goal state is reached when:

- Artifact $\alpha$ has been cooled down.
- Artifact $\beta$ is in the pod and is in the Stasis-Lab.
- Martian core samples are in the Stasis-Lab.

### D. Artifact workflows

Given the initial and goal state information, the workflow for each artifact type are:

1) Artifact $\alpha$: Artifact hall $\alpha \rightarrow$ Cyo-Chamber $\rightarrow$ cool down artifact.
2) Artifact $\beta$: Artifact hall $\beta \rightarrow$ Pod-zone $\rightarrow$ place into a pod $\rightarrow$ Stasis-Lab.
3) Martian core samples: Cryo-Chamber $\rightarrow$ Stasis-Lab.

Given this flow, it can be seen that when Artifact $\alpha$ workflow ends the Martian core samples one starts. To have a more compact and clearer domain these two artifact types are merged so that the final goal state definition is:

- Artifact $\alpha$ has been cooled down and is in the Stasis-Lab.
- Artifact $\beta$ is in the pod and is in the Stasis-Lab.

Therefore the workflows with only these two types of artifacts are:

1) Artifact $\alpha$: Artifact hall $\alpha \rightarrow$ Cyo-Chamber $\rightarrow$ cool down artifact $\rightarrow$ Stasis-Lab.
2) Artifact $\beta$: Artifact hall $\beta \rightarrow$ Pod-zone $\rightarrow$ place into a pod $\rightarrow$ Stasis-Lab.

This is a simplification of the given scenario but allows for more clarity and compactness in the future domain modeling steps.

### E. Actions

Once the idea of the artifact workflows are defined, the robot must have a series of actions in order to interact with the environment to follow them. Since this is a step before the actual PDDL files are written, a high-level view of the robot actions is provided and will be further specified in the problem-specific sections. The actions that the robotic curator can do are:

1) movement: move around the museum going from one zone to another.
2) loading/unloading: loading unloading an artifact to change its location.
3) seal/unseal: seal or unsealing itself to withstand the low pressure in the maintenance tunnel.
4) cool artifact: cool down an artifact in the Cryo-Chamber.
5) place in pod: place an artifact inside a pod in the Pod-zone.

6) reset pod: once an artifact $\beta$ in a pod is delivered in the Stasis-Lab the robot can request that another pod is delivered to Pod-zone. This action ensures that there are always two pods outside of Stasis-Lab.
7) end seismic: wait for the end of the seismic events in Artifact hall $\beta$.

### F. Assumptions

Various assumption must be made to clarify properly the scenario:

- The artifact types $\alpha$ and Martian core samples are considered to be the same types to have simpler and more streamlined workflows.
- The state of a pod is reset once its delivered to the Stasis-lab. This is done via the "reset pod" action. This choice is motivated by the fact that the robots should continue visiting pod_zone through the execution and not just greedily carrying around the pods.
- The temperature of artifacts is simply defined with a predicate without an actual numeric value.
- The state of a zone is decided with a predicate for both the stability and the presence of low pressure.
- The artifacts are assume to have all the same weight and the robots to have a predefined maximum capacity.
- Robots must be in sealed mode while passing through the maintenance tunnel but they must be unsealed to have their actuators free to perform every other action.
- The actions are modeled as deterministic. This means that there is no possible failure by the robotic agents.
- When dealing with temporal planning a fixed duration for each action is assigned, even though in reality it may not be the case.

## III. Methodology

This section will discuss the design process and choices made for each problem. A common implementation idea was to keep compatibility and simplicity as the main focus, this was done by only using *strips typing* requirements in the domain files. This choice has the tradeoff of having more predicates and actions since, for example, *negative-preconditions* was not used.

### A. Problem1

The first problem establishes the foundations of the PDDL implementation with only one robotic agent.

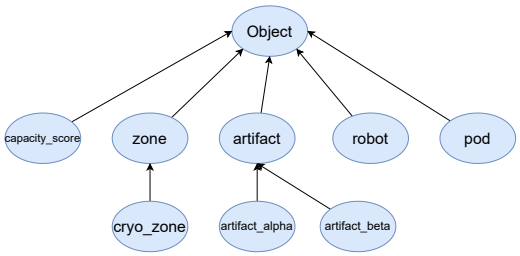


Fig. 3: Graph of the type hierarchy in problem1.

*1) Types:* Different types are defined based on the scenario interpretation:

- *capacity_score*: a utility type to model the robot capacity without fluents enabled.
- *zone*: a location in the museum.
- *cryo_zone*: a location with cryo-based technology for preservation. Only here the cooling of the artifacts can be done.
- *artifact*: an artifact to be preserved.
- *artifact_alpha*: an alpha artifact that needs to be cooled down and then delivered to the Stasis-Lab, as discussed in the scenario interpretation.
- *artifact_beta*: a beta artifact to be put into a pod and delivered to the Stasis-Lab.
- *robot*: robotic agent that acts to preserve the artifacts.
- *pod*: anti-vibrational pod for beta artifacts.

*2) Constants:* Since for every problem the zones and the number of pods are always the same, they are declared in the domain to simplify the declarations in the problem files. The declared constants and their types are:

- hall_alpha hall_beta maintenance_tunnel pod_zone entrance stasis_lab are all of type zone.
- cryo_chamber is of type cryo_zone.
- pod1 pod2 are of type pod.
- n0 n1 n2 n3 n4 are of type capacity score and are used to model the numbers 0, 1, 2, 3 and 4 that indicate the robot capacity. In problem 1 the robot have a maximum capacity of 1 but in subsequent steps there will be agents with a higher one.

*3) Predicates:* The predicates are essential to define the states of the objects, agents and the map structure of the domain. To ensure compatibility, a lot of predicates are used to model the negation of other ones. The state of the map is modeled with the following predicates:

- *connected*: tells if a zone is connected to another. The connections will follow those in the edges of the graph in figure 2.
- *stable/unstable*: tells if a zone is currently unstable due to a seismic event.
- *low_pressure/normal_pressure*: to define the pressure of a zone.

The state of the anti-vibrational pods is modeled with these predicates:

- *pod_at* tells if a pod is in a certain zone.
- *pod_full/pod_empty*: tells if a pod is filled with an artifact or not.
- *in_pod*: tells if an artifact is in a specific pod.

The artifact state is decided by the predicates:

- *artifact_at*: tells if an artifact is in a certain zone.
- *is_cool/is_not_cool*: tells if an artifact alpha has been cooled or not.
- *is_in_pod*: tells if an artifact beta is in a pod.

Finally, the robot state needs his own predicates to be properly defined:

- *robot_at*: tells the zone where a robot is currently.
- *is_sealed/is_unsealed*: tells if a robot is currently sealed or unsealed.
- *is_carrying*: tells if a robot is carrying a specific artifact.
- *is_carrying_pod*: tells if a robot is carrying a specific pod.
- *next_capacity_score*: used to define the order of the capacity scores, for examples n0>n1>n2>n3>n4.
- *robot_capacity*: tells the current robot capacity.
- *robot_can_carry*: tells which capacity scores are available for that robot. For example a robot with capacity 2 can have a capacity score of n0, n1 and n2. In problem1 the maximum capacity of the robot is set to one.

*4) Actions:* The actions are essential to define the transitions between states in the search space. The actions that were created for problem1 are:

- *move_unsealed/move_sealed*: the robot moves respectively to a zone where there is normal pressure or with low pressure.
- *seal/unseal*: the robot enters or leaves sealed mode.
- *load/unload*: the robot loads or unloads an artifact.
- *load_pod/unload_pod*: the robot loads or unloads a pod with an artifact inside.
- *reset_pod*: once a pod with an artifact inside is delivered the robot request in the stasis_lab that a new pod is sent to the pod_zone.
- *cool_artifact*: the robot cools down an artifact in a zone of type cryo_zone.
- *place_in_pod*: the robot places an artifact inside a pod.
- *end_seismic*: the seismic event stops.

Note that the redundancy of the actions is motivated by the need of very simple preconditions, since no disjuctions or negative ones are accepted.

*5) Problem file:* Three different problem files where written with different number of artifacts to test how the scenario scales as the artifact count increases. Table I shows the

| Problem file | Artifacts per type |
|---|---|
| small | 1 |
| medium | 5 |
| big | 10 |

TABLE I: Artifact count per problem file.

different problem configuration that have been defined. Every problem file has Its own initial and goal state. The intial state uses the predicates and objects defined to initialize that problem, also creating the map structure.

$$artifact\_at(alpha1, stasis\_lab) \wedge is\_cool(alpha1) \\ \wedge artifact\_at(beta1, stasis\_lab) \wedge is\_in\_pod(beta1) \quad (1)$$

The goal state has this common structure in every problem file as seen in equation 1.

*6) Design choices:* Most of the design choices can explained by the decision to only use bare bones requirements that require more predicates. There are however some notable exceptions. Firstly the capacity of the agent is set to one in this step, since it was requested by the assignment. In later steps the capacity will be different depending on the type of robot. The seismic events only occur at the start and, since no duration is fixed at this point, stops immediately allowing access to both halls for the whole plan execution. When a pod is delivered correctly, a new pod is placed immediately in the pod_zone for a new delivery of a beta artifact. To make the whole scenario more realistic, the artifacts must be unloaded before cooling them or placing them into pods. Another choice made to make actions more realistic is to allow all other actions except the move_sealed only if the robot is unsealed so that it can use the actuators.

### B. Problem2

Problem2 required the addition of other kinds of robotic agents with different capabilities. Most of the modeling from problem1 is kept intact, here only the differences will be shown. The idea in this step is to define drones that can fly freely to all locations with normal pressure without caring for map connections. The drones are also able to transport artifacts but since they do not have actuators, they cant perform complex actions like cooling down artifacts or putting them inside a pod and transporting it. Another type is that of the ground robots that can perform all actions seen in problem1. Different ground robots can have either 2 or 4 of capacity for light models and heavy models respectively. The expectation behind this is that drones should act as auxiliary agents and the ground ones should only handle complex tasks and pod delivery. This idea is then modeled using PDDL.

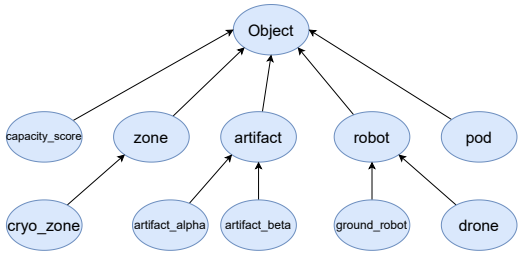


Fig. 4: Graph of the type hierarchy in problem2.

*1) Types:* Figure 4 shows the additional two types added to the domain. Specifically, they are *drone* to model the drones and *ground_robot* to model robots that remain on the ground.

*2) Predicates:* The predicates are keept the same as in problem1 with the difference that the ones regarding sealing mode and carrying a pod are now only related to the ground robots since drones do not have those functionality.

*3) Actions:* Previous actions used in problem1 were all assigned to the *ground_robot* type and therefore new actions must be created for the drones. The new actions are:

- *fly*: the drone can fly freely to other stable zones with normal pressure with no constraints on connections.
- *hook/unhook*: the drone can hook or unhook an artifact to carry it around the museum.

*4) Problem file:* Similarly to problem1, three different problem files were created for a scalability study. The amount of artifacts defined in each problem is shown in the table I. In each problem three different agents were declared: a drone with capacity 1, a ground robot with capacity 2 (called *light1* and a ground robot with capacity 4 (called *heavy*).

*5) Additional assumptions:* The addition of the drone was made to add variability to agents present, adding more strategies to reach the goal. Intuitively, the agents should work together to achieve the goal even faster.

### C. Problem3

Problem3 requires the implementation of a Hierarchical Task Network (HTN) on top of the previous problem. The idea behind this implementation is to integrate domain knowledge into the creation of tasks and methods. The workflows introduced in the introduction can be directly modeled to have more control on the methods used to solve various tasks.

*1) Tasks:* Various tasks are introduced in this step:

- *t_all_artifacts_in_stasis*: root task to complete the goal and preserve all artifacts.
- *t_deliver_all_alpha*/*t_deliver_all_beta*: task to complete artifact preservation for all artifact of a specific type.
- *t_deliver_one_alpha*/*t_deliver_one_beta*: task to preserve one artifact of type beta or alpha.
- *t_transport_artifact*: transport one artifact from a point to another.
- *t_traverse_museum*: task to travel from one zone to another of the museum. The methods of this task are the most complex since starting from actions defined the definition is recursive.
- *t_ensure_sealed*(*t_ensure_unsealed*: task to ensure that the robot is sealed or unsealed. It is used a conditional structure where the robot seals only if it is not already sealed and vice-versa.

*2) Methods:* Various methods are implemented to complete the various tasks. The most interesting implementation is the one of *t_traverse_museum* task. Four methods can be used to complete it:

- *m_traverse_done*: the traversal is over and the task is achieved. Used to end the recursive method.
- *m_traverse_low*: ensure that the robot is sealed and travel to a low pressure zone.
- *m_traverse_normal*: ensure robot is unsealed and travel to a normal pressure zone. The idea of unsealing here is that in normal pressure zone the robot needs to be unsealed to do any other action.
- *m_traverse_recursive*: if the other methods cannot be completed, a a zone between the target zone exists and we launch the *t_traverse_museum* to reach the zone in the middle first and then the destination. Usually the middle zone is the maintenance_tunnel that connects everything together.

This implementation shows the strength of recursive definition in HTN and the potential of including domain knowledge into the plan generation. To properly model the preconditions two auxiliary predicates were added to represent if two zones are disconnected and if two zones are different.

*3) Problem file:* The problem file is the same used in problem2 with 6 artifacts present, half of them alpha and the other half beta artifacts. The drone will not be used in this context since no methods are designed to use its actions.

*4) Modelling choices:* For the purpose of implementing a functioning HTN that was not too complex, only the ground robot actions were taken into account into the implementation of the methods. The expected performance would surely rise thanks to the contribution of the drone agent but it is left to future work.

### D. Problem4

Problem4 introduces time into the previous implementations. For this step durations must be declared for each actions and the generated plan must optimize time taken instead of number of actions.

*1) Durative actions:* The length of the actions previously designed in problem2 are fixed and have been chosen to emulate reasonably a realistic scenario. The specific durations for each action are:

- *fly* has a duration of 2. The idea is that the drone is very light and fast but with limited capacity,
- *hook*/*unhook* have a duration of 1.
- *move_unsealed*/*move_sealed* have a duration of 5.
- *seal*/*unseal*: have a duration of 1.
- *load*/*unload*: have a duration of 3.
- *load_pod*/*unload_pod* have a duration of 3 like when loading/unloading an artifact.
- *reset_pod* has a duration of 1 since the robot just needs to request a new pod and it will be sent to pod_zone.
- *cool_artifact* has a duration of 10.
- *place_in_pod* has a duration of 8.
- *end_seismic* has a duration of 20.

*2) Problem file:* The problem file is similar to Problem2 with four artifacts declared to study how the agents handle multiple artifacts by cooperating.

*3) Modeling choices:* The idea behind the choice of durations is that the drone is very fast in artifact transportation and that the ground robot actions for preservation are slower since their are more complex. An important assumption is that the seismic events occurs only once and does not repeat itself during the execution of the plan. At the start, the seismic event is already started and the robots have to handle the alpha artifacts while waiting for artifact hall beta for be accessible. Some action can occur in parallel if the preconditions allow it. The expected optimal plan would be one where the agents cooperate to achive the most actions in the smallest time possible.

### E. Problem5

Problem5 asks the integration of the temporal planning model into the Planning System 2 framework [3].

*1) PDDL changes:* Some modification have been done on top of the previous step in order to leverage the features of the framework, specifically the use of *fluents* in order to model the capacity of the agents. Another change is in the commands used to generate the problem. For a simpler plan only one ground robot instance of capacity 4 and only two artifacts alpha and beta.

*2) Integration:* The integration was done by creating a ROS2 package following the structure already discussed in the introduction. The launch code is found in the python file in the launch directory and creates executor nodes for the various actions, handling also the PlanSys2 launch with the domain.pddl file. Each action node is implemented in different cpp files by creating a simulated execution using an update rate of 100ms. This is done to simulate real implementation where actual robotic components use the PlanSys2 framework to generate and execute a plan.

*3) Execution explained:* A brief explaination of the process to execute this step is mandatory. First a bash shell script *run_experiment.sh* has been written in order to handle the setup and launch of the environment while also openining the plansys2 terminal. A log also is generate to track the launch progress. After the script is run, the commands file in the launch folder of the package contains the code to create the problem instance and collect the plan. After the plan has been generated, using the run command the action nodes can be executed to test that the plan reaches the end succesfully.

*4) Modeling choices:* The choice to use fluents was to exploit the additional functionalities of the planner used in this step. Another possibility would be to fundamentally restructure the domain file by exploiting additional *adl* features. An important note is that constants in the domain file conflicted with the instances declared in the commands script and so they were deleted.

## IV. Results

This section will show the results of each problem discussing and comparing them.

### A. Problem1

The first problem was studied first by generating plans with different planners and secondly with a scalability analysis.

*1) Comparing planners:* The comparison between different configurations of planners is essential both to study the problem and to better understand the different properties of the planners tested. An example of generated plan can be seen in figure 5 that gives an idea of what is generated by a planner. The planners chosen for comparison are all the aliases available from the Fast Downward [2] planner in planutils.

Table VII shows the comparison between the various alias options when using downward. Some observations can be made from this data. Firstly the *lama-first* finds a suboptimal solution in the fastest time and serves as comparison with other optimal planners. No planner can go below 36 plan length and so its assumed to be the optimal result. The most competitive alias is *seq-opt-bjolp* that finds the optimal planner in the shortest time compared to the others. The absolute worst seems to be *seq-sat-fd-autotune-1* that expands the most nodes. A more in-depth analysis is possible by comparing various search strategies with different heuristic, studying their results for the specific problem.

Table III shows how various configurations have an effect on the generated plans. It can be noticed how A star is guaranteed to find the optimal plan only with admissible heuristics. The

```
(end_seismic hall_beta)
(seal worker)
(move_sealed worker entrance maintenance_tunnel)
(unseal worker)
(move_unsealed worker maintenance_tunnel hall_beta)
(load worker beta1 hall_beta n0 n1)
(seal worker)
(move_sealed worker hall_beta maintenance_tunnel)
(move_sealed worker maintenance_tunnel pod_zone)
(unseal worker)
(unload worker beta1 pod_zone n1 n0)
(place_in_pod worker beta1 pod_zone pod1)
(load_pod worker beta1 pod1 pod_zone n0 n1)
(seal worker)
(move_sealed worker pod_zone maintenance_tunnel)
(unseal worker)
(move_unsealed worker maintenance_tunnel stasis_lab)
(unload_pod worker beta1 pod1 stasis_lab n1 n0)
(reset_pod worker beta1 pod1)
(seal worker)
(move_sealed worker stasis_lab maintenance_tunnel)
(unseal worker)
(move_unsealed worker maintenance_tunnel hall_alpha)
(load worker alpha1 hall_alpha n0 n1)
(seal worker)
(move_sealed worker hall_alpha maintenance_tunnel)
(unseal worker)
(move_unsealed worker maintenance_tunnel cryo_chamber)
(unload worker alpha1 cryo_chamber n1 n0)
(cool_artifact worker alpha1 cryo_chamber)
(load worker alpha1 cryo_chamber n0 n1)
(seal worker)
(move_sealed worker cryo_chamber maintenance_tunnel)
(unseal worker)
(move_unsealed worker maintenance_tunnel stasis_lab)
(unload worker alpha1 stasis_lab n1 n0)
; cost = 36 (unit cost)
```

Fig. 5: Plan generated from lama alias with fast downward [2] The plan found is optimal and shows how the actions can be used to reach the goal state. The worker is how robotic agent is called in problem1.

TABLE II: Performance comparison of downward aliases in problem1.

| Alias | Plan Length | Expanded States | Total Time (ms) |
|---|---|---|---|
| lama | 36 | 21485 | 143.6 |
| lama-first | 40 | 98 | 19.5 |
| seq-opt-bjolp | 36 | 2883 | 27.8 |
| seq-opt-fdss-1 | 36 | 4306 | 29.0 |
| seq-opt-fdss-2 | 36 | 4306 | 32.4 |
| seq-opt-lmcut | 36 | 2130 | 57.8 |
| seq-opt-merge-and-shrink | 36 | 4306 | 29.0 |
| seq-sat-fd-autotune-1 | 36 | 16280 | 261.5 |
| seq-sat-fd-autotune-2 | 36 | 7403 | 129.9 |
| seq-sat-fdss-1 | 36 | 5470 | 37.5 |
| seq-sat-fdss-2 | 36 | 4916 | 35.0 |
| seq-sat-fdss-2014 | 36 | 5451 | 51.8 |
| seq-sat-fdss-2018 | 36 | 4331 | 30.3 |
| seq-sat-fdss-2023 | 36 | 4331 | 34.9 |
| seq-sat-lama-2011 | 36 | 21799 | 156.6 |

TABLE III: Comparison of specific search strategies and heuristics in problem1.

| Algorithm | Heuristic | Plan Length | Expanded States | Total Time (ms) |
|---|---|---|---|---|
| A* | blind | 36 | 5264 | 29.4 |
| | ff | 36 | 2004 | 28.4 |
| | add | 38 | 129 | 19.5 |
| | lmcut | 36 | 2130 | 55.0 |
| | cg | 36 | 140 | 20.0 |
| | cea | 40 | 82 | 18.6 |
| | hmax | 36 | 3923 | 29.5 |
| Eager Greedy | blind | 36 | 5264 | 22.2 |
| | ff | 36 | 61 | 19.0 |
| | add | 48 | 79 | 19.4 |
| | lmcut | 36 | 58 | 19.3 |
| | cg | 42 | 56 | 21.1 |
| | cea | 48 | 77 | 18.2 |
| | hmax | 42 | 672 | 21.9 |
| Lazy WA* (w=5) | blind | 36 | 5697 | 26.0 |
| | ff | 36 | 114 | 18.7 |
| | add | 48 | 155 | 20.0 |
| | lmcut | 36 | 101 | 19.1 |
| | cg | 42 | 136 | 18.8 |
| | cea | 48 | 181 | 18.6 |
| | hmax | 36 | 2131 | 24.7 |

best heuristic is cg for a star and yields in less time the best plan. The standout choice is the lazy WA star strategy with ff heuristic that generates the optimal plan in less time compared to all other configurations.

*2) Scalability analysis:* After studying how different planner configurations behave on the small version of the problem, It is interesting to analyze how the length of a suboptimal plan found by *lama-first*, grows with the number of artifacts. The choice of a planner is motivated by the very long time for convergence when the plan is medium or big, so a suboptimal planner was used as an approximation for this measurement. An important note is also that for every number of artifacts half are of type alpha and the others are of type beta.

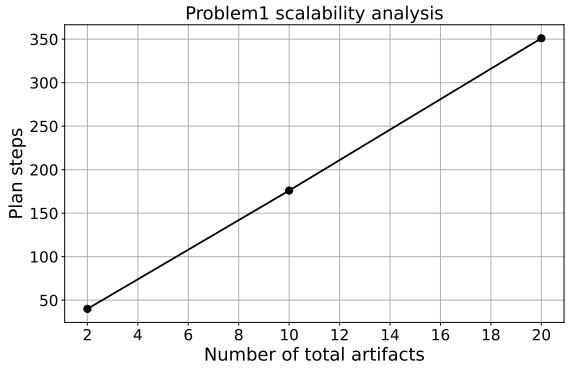


Fig. 6: Scalability analysis in problem1.

The plot in figure 6 shows the growth in plan complexity as more artifacts are introduced. By fitting a linear model on the data, the following parameters are found: $plan\_steps = 17.3 * n\_artifacts + 4.6$. This means that the problem steps grow linearly with a factor of 17.3 as the number of artifacts goes up.

## B. Problem2

The analysis of the second problem follows the blueprint of problem1. The main difference is the introduction of new types of agents that may change the number of steps. The expectation is that with more and varied agents the goals are reached in less time. The problem files in this scenario include a drone of capacity 1 and two ground robots one with capacity 2 and the other with capacity 4. The optimal plan generated can be seen in figure 7, showing how the additional agents can be used to reach the goal faster. It seems that only the ground robot with capacity 4 is exploited with the help of the drone to transport the artifacts.

```
(seal heavy1)
(move_sealed heavy1 entrance maintenance_tunnel)
(end_seismic hall_beta)
(fly drone1 entrance hall_alpha)
(hook drone1 alpha1 hall_alpha n0 n1)
(fly drone1 hall_alpha cryo_chamber)
(unseal heavy1)
(unhook drone1 alpha1 cryo_chamber n1 n0)
(move_unsealed heavy1 maintenance_tunnel cryo_chamber)
(cool_artifact heavy1 alpha1 cryo_chamber)
(load heavy1 alpha1 cryo_chamber n0 n1)
(seal heavy1)
(move_sealed heavy1 cryo_chamber maintenance_tunnel)
(unseal heavy1)
(fly drone1 cryo_chamber hall_beta)
(hook drone1 beta1 hall_beta n0 n1)
(fly drone1 hall_beta pod_zone)
(unhook drone1 beta1 pod_zone n1 n0)
(move_unsealed heavy1 maintenance_tunnel pod_zone)
(place_in_pod heavy1 beta1 pod_zone pod1)
(load_pod heavy1 beta1 pod1 pod_zone n1 n2)
(seal heavy1)
(move_sealed heavy1 pod_zone maintenance_tunnel)
(unseal heavy1)
(move_unsealed heavy1 maintenance_tunnel stasis_lab)
(unload_pod heavy1 beta1 pod1 stasis_lab n2 n1)
(unload heavy1 alpha1 stasis_lab n1 n0)
(reset_pod heavy1 beta1 pod1)
; cost = 28 (unit cost)
```

Fig. 7: Plan generated from lama alias with fast downward [2] for problem2. The plan found is optimal and shows how the actions can be used to reach the goal state.

*1) Comparing planners:* Table IV shows the comparison results between different aliases. Compared to problem1 there are much more states to be explored since the agents are more. As expected the optimal state has less steps than in problem1 since multiple agents can be exploited. The best alias seems to be *merge-and-shrink* reaching the optimal plan in the fastest time.

Table V shows different search strategies and heuristics compared on problem2. Clearly the number of states to be explored is much higher than problem1, requiring also much more time to be solved. The heuristics other than blind in most of the cases while finding a solution faster cannot find the optimal one. The best combination seems to be a star with ff heauristics generating the optimal plan in the fastest time.

*2) Scalability analysis:* The scalability analysis also follows the methodology proposed in problem1. The plot in figure 8 shows the growth in plan steps as more artifacts

TABLE IV: Performance comparison of aliases in problem2.

| Alias | Plan Length | Expanded States | Total Time (ms) |
|---|---|---|---|
| lama | 28 | 619792 | 8277.3 |
| lama-first | 38 | 187 | 24.3 |
| lama-2011 | 28 | 603431 | 9491.4 |
| seq-opt-bjolp | 28 | 152103 | 1035.4 |
| seq-opt-fdss-1 | 28 | 294230 | 624.9 |
| seq-opt-fdss-2 | 28 | 294230 | 634.0 |
| seq-opt-lmcut | 28 | 116130 | 10451.3 |
| seq-opt-merge-and-shrink | 28 | 294230 | 613.5 |
| seq-sat-fd-autotune-1 | 28 | 565777 | 30450.2 |
| seq-sat-fd-autotune-2 | 28 | 633203 | 28935.2 |
| seq-sat-fdss-1 | 28 | 407373 | 6501.1 |
| seq-sat-fdss-2 | 29 | 470094 | 7446.3 |
| seq-sat-fdss-2014 | 28 | 408355 | 8262.5 |
| seq-sat-fdss-2018 | 28 | 624741 | 13375.4 |
| seq-sat-fdss-2023 | 28 | 1136173 | 13827.7 |
| seq-sat-lama-2011 | 28 | 603431 | 9491.5 |

TABLE V: Comparison of specific search strategies and heuristics in problem2.

| Algorithm | Heuristic | Plan Length | Expanded States | Total Time (ms) |
|---|---|---|---|---|
| A* | blind | 28 | 436190 | 811.9 |
| | ff | 28 | 36296 | 730.9 |
| | add | 30 | 15257 | 360.5 |
| | lmcut | 28 | 116130 | 10934.1 |
| | cg | 29 | 3660 | 91.9 |
| | cea | 29 | 15301 | 970.9 |
| | hmax | 28 | 274343 | 2247.5 |
| Eager Greedy | blind | 28 | 436190 | 697.9 |
| | ff | 33 | 579 | 44.0 |
| | add | 40 | 1128 | 46.7 |
| | lmcut | 38 | 3286 | 533.0 |
| | cg | 35 | 965 | 39.1 |
| | cea | 35 | 1525 | 87.4 |
| | hmax | 29 | 219707 | 1634.4 |
| Lazy WA* (w=5) | blind | 28 | 481430 | 1341.1 |
| | ff | 31 | 2021 | 37.5 |
| | add | 38 | 2888 | 41.5 |
| | lmcut | 34 | 26992 | 891.2 |
| | cg | 35 | 3066 | 38.9 |
| | cea | 34 | 6961 | 94.9 |
| | hmax | 29 | 158269 | 1155.1 |

are placed on the map. The choice of using *lama-first* as an approximation is motivated by the fact that if any optimal planner would be used here, the large time to converge on bigger problems would cause the computation to hang indefinitely. Fitting the data to a linear model we obtain that $plan\_steps = 15.5 * n\_artifacts + 6.9$. Compared with problem 1 we see that the linear relationship is less strong looking at the smaller slope coefficient. This makes sense since the introduction of the new agents amortizes the addition of artifacts.

### C. Problem3

Problem 3 involved modeling the same scenario as Problem 2, but using HTNs. For this reason, it was necessary to use planners specifically designed for this type of problem. A wide range of different planners were considered; we tried PANDA, SIADEX, LiloTane, and other IPC2020 planners. In the end we narrowed the final comparison to PANDA and SIADEX in the final comparison because they were the only ones capable of interpreting the domain correctly and to produce valid plans. Other planners failed due to parsing issue, some of them required the use of *ordered-subtasks* and others failed

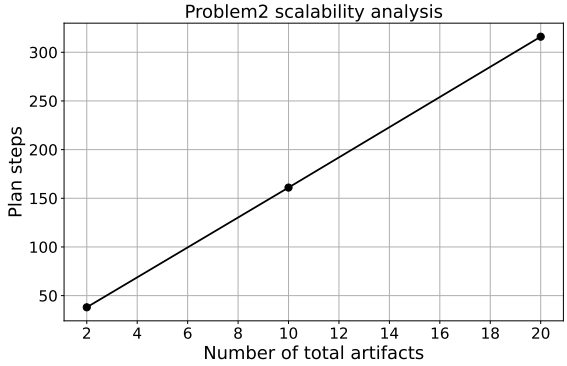


Fig. 8: Scalability analysis in problem2. The number of steps is lower compared to problem1 as expected.

due to tighter constraints on the domain. The final comparison produced the following results:

TABLE VI: Performance comparison of planners in problem3.

| Planner | Plan Length | Expanded States | Total Time (ms) |
|---|---|---|---|
| SIADEX | 105 | - | 786.154 |
| PANDA | 105 | 76130 | 3995.89 |

Although both planners produced optimal plans of equal length (105 steps), they exhibited distinct behaviors in resource allocation and execution time. Analyzing the final plans reveals that SIADEX used a *first-fit* approach, due to its DFS strategy. PANDA uses instead an heuristic search, which allows for a more varied plan that considers many more options. This difference is evident when analyzing the second part of the plan, SIADEX continues to overload the *light1* robot with all the tasks because it is the first valid option found, PANDA instead uses *heavy1* for the Beta artifacts. While both plans are valid, PANDA's solution effectively demonstrates its ability to distribute the workload across the available robots, while SIADEX minimizes search times by focusing on the first feasible task. For PANDA, we also attempted various search configurations (BFS, DFS, Greedy, etc...), but these failed due to memory limits or timeouts, likely caused by the depth of the solution path (greater than 100 steps) creating a combinatorial explosion in the search space. The only PANDA configuration that succeeded was A* search with the Relaxed Composition (RC) heuristic.

In conclusion, for this specific domain, SIADEX offers superior runtime efficiency by exploiting the total ordering of subtasks, while PANDA offers superior plan quality in terms of resource distribution at the cost of higher computational overhead.

### D. Problem4

Problem 4 introduced the timing of each action as a planning variable, so we analyzed the results of several temporal planners and some of their configurations. ENHSP-2020, POPF, and TFD were considered, but for various reasons, they did not

produce valid results. The first two had problems interpreting the domain, while TFD remained stuck in a local optimum, causing timeouts and never returning a concrete result. Optic and LPG-TD were our choices for our temporal planning analysis, reported in the following table:

TABLE VII: Performance comparison of planners in problem4.

| Planner & Configuration | Makespan | Computation Time (s) |
|---|---|---|
| LPG-TD (Incremental) | $70.60 \pm 19.65$ | $77.29 \pm 57.45$ |
| LPG-TD (Quality) | $85.30 \pm 15.15$ | $7.89 \pm 9.78$ |
| LPG-TD (Speed) | $122.20 \pm 35.71$ | $0.59 \pm 0.99$ |
| Optic (Standard) | 58.03 | 0.89 |
| Optic (No Best First) | 58.03 | 0.85 |

From this analysis, we can conclude that Optic is a generally superior and more reliable choice than LPG-TD. This difference is due to the algorithms used in its internal logic: Optic employs a deterministic heuristic search in the state space, while LPG-TD uses a stochastic local search on the planning graphs. This means that Optic predictably explores the problem space, producing a highly optimized plan with each run. In our case, it produced a plan with a makespan of 58.03 in 0.89 s. LPG-TD, on the other hand, generates an initial plan and iteratively attempts to correct it by making randomized, probabilistic changes. While this randomness allows LPG-TD's "incremental" mode to occasionally produce a slightly better plan, it also makes the planner highly unpredictable. Our best run with incremental LPG-DT produced a plan with a makespan of 52.00, but requiring an execution time of 156.63s. However, since the results are probabilistic and on average much worse, for reliable real-world implementation, Optic's deterministic approach is significantly superior than the probabilistic LPG-TD.

We ran Optic with and without Best-Fit, that both generated the same plan, in very similar times. For our specific problem, both approaches managed to guide the algorithm to the same highly efficient plan.

Because LPG-TD is a stochastic algorithm based on randomized choices, evaluating it based on a single run would be highly misleading; therefore, statistical analysis was necessary to capture its true performance profile. We ran each LPG-TD configuration 10 times on the same problem, extracting makespan and computation time, calculating their mean and standard deviation. This methodology highlighted that, although the *incremental* mode provides the best average plan cost (70.60), its computation time is extremely unstable, with a standard deviation of $\pm 57.45$ seconds. Similarly, it highlighted that the *speed* configuration, while fast, represents a huge gamble in terms of plan quality, with a high standard deviation of $\pm 35.71$. This analysis demonstrates that, although LPG-TD can achieve a higher peak optimum, its average reliability is lower than that of a deterministic solver.

### *E. Problem5*

The experiment was run successfully, generating a complete plan that satisfies the constraints. The expected cooperation between the drone and the robot can be observed to work properly. Note also that the planners sometimes choose the fly action not exploiting it fully, moving almost randomly the drone that could reach the goal in fewer moves. The execution of the plan was completed without errors showing that the integration works without any mistake.

```
> get plan
plan:
0:      (fly drone1 entrance cryo_chamber)      [2]
0:      (seal robot1)   [1]
1.001:  (move_sealed robot1 entrance maintenance_tunnel)        [5]
1.002:  (unseal robot1) [1]
2.001:  (fly drone1 cryo_chamber hall_alpha)    [2]
4.002:  (hook drone1 artifact_a hall_alpha)     [1]
4.003:  (fly drone1 hall_alpha cryo_chamber)    [2]
6.002:  (move_unsealed robot1 maintenance_tunnel cryo_chamber)  [5]
6.004:  (unhook drone1 artifact_a cryo_chamber) [1]
6.005:  (fly drone1 cryo_chamber hall_beta)     [2]
8.006:  (hook drone1 artifact_b hall_beta)      [1]
8.007:  (fly drone1 hall_beta pod_zone) [2]
10.008: (unhook drone1 artifact_b pod_zone)     [1]
10.009: (fly drone1 pod_zone cryo_chamber)      [2]
11.003: (cool_artifact robot1 artifact_a cryo_chamber)  [10]
11.004: (seal robot1)   [1]
12.005: (move_sealed robot1 cryo_chamber maintenance_tunnel)    [5]
12.006: (unseal robot1) [1]
12.01:  (hook drone1 artifact_a cryo_chamber)   [1]
12.011: (fly drone1 cryo_chamber stasis_lab)    [2]
14.012: (unhook drone1 artifact_a stasis_lab)   [1]
17.006: (move_unsealed robot1 maintenance_tunnel pod_zone)      [5]
22.007: (place_in_pod robot1 artifact_b pod_zone pod1)  [8]
30.008: (load_pod robot1 artifact_b pod1 pod_zone)      [3]
30.009: (seal robot1)   [1]
31.01:  (move_sealed robot1 pod_zone maintenance_tunnel)        [5]
31.011: (unseal robot1) [1]
36.011: (move_unsealed robot1 maintenance_tunnel stasis_lab)    [5]
41.012: (unload_pod robot1 artifact_b pod1 stasis_lab)  [3]
44.013: (reset_pod robot1 artifact_b pod1 stasis_lab pod_zone)  [1]

> run
seal progress ... [100%]
fly progress ... [100%]
fly progress ... [100%]
move_sealed progress ... [100%]
hook progress ... [100%]
unseal progress ... [100%]
fly progress ... [100%]
move_unsealed progress ... [100%]
unhook progress ... [100%]
fly progress ... [100%]
cool_artifact progress ... [100%]
hook progress ... [100%]
seal progress ... [100%]
fly progress ... [100%]
move_sealed progress ... [100%]
unhook progress ... [100%]
unseal progress ... [100%]
fly progress ... [100%]
move_unsealed progress ... [100%]
hook progress ... [100%]
place_in_pod progress ... [100%]
fly progress ... [100%]
load_pod progress ... [100%]
unhook progress ... [100%]
seal progress ... [100%]
move_sealed progress ... [100%]
unseal progress ... [100%]
move_unsealed progress ... [100%]
unload_pod progress ... [100%]
reset_pod progress ... [100%]
[INFO] [1771264032.851178354] [executor_client]: Plan Succeeded

Successful finished
```

Fig. 9: Plan generation and execution with the Plansys2 [3] framework.

## V. Conclusion

This project showed a full modeling process from classical planning to an actual integration with a framework for robotics. The detailed comparison proposed for problems one and two showed that planners and heuristic are dependent on

the scenario studied and modeling choices. The scalability analysis also showed the proportionality between problem complexity and number of steps in the plan. This is a fundamental aspect to consider when designing complex scenarios since the number of steps in the plan should not cause a blowup in number of action to be more efficient. Problem3 and problem4 showed also some pitfalls of planners not able to handle the complexity of the domain and problems given. Finally, problem5 showed a complete breakdown of the integration with Plansys2 [3] that works as intended.

## REFERENCES


[1] J Benton, Amanda Coles, and Andrew Coles. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, pages 2–10, 2012.

[2] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[3] Francisco Martín, Jonatan Ginés Clavero, Vicente Matellán, and Francisco J Rodríguez. Plansys2: A planning system framework for ros2. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9742–9749. IEEE, 2021.