Automated Planning Project

Matteo Grisenti Alessando De Vidi

February 2026

Introduction - Interplanetary Museum Vault

This project aims to model the Interplanetary Museum Vault scenario using various planning strategies. Before analyzing the different modeling techniques, we provide a detailed description of the environment and its constraints.

Artifacts

The primary goal is to schedule the behavior of one or more robots to complete specific tasks within a Martian museum complex. The core of these tasks involves the handling and transport of **Artifacts**, which are categorized based on two main properties:

- **Fragility**: Fragile artifacts require careful handling and must be transported using specialized *Anti-Vibration Pods* to prevent damage. Non-fragile artifacts can be handled directly by the robots.
- **Thermal Requirements**: Certain artifacts are temperature-sensitive and must be maintained at low temperatures to prevent degradation.

Environment and Layout

The environment consists of seven distinct locations organized in a **star topology**. The *Maintenance Tunnel* acts as the central hub; all other rooms connect exclusively to this tunnel, meaning robots must pass through it to move between any two locations.

- **Entrance**: The designated starting location for all robots.
- **Maintenance Tunnel**: The central transit hub. Notably, this is the only **unpressurized** area. To navigate this room safely, robots must engage a *sealed* safety modality.
- **Artifact Halls (A and B)**: These rooms serve as the initial storage for all artifacts.
  - **Hall A**: Used to store artifacts that require cold storage. However, as it is not an active chilling room, the planner must monitor these artifacts to ensure they remain at the required temperature.
  - **Hall B**: Stores artifacts that do not require cooling. Due to structural issues, it is currently an *unsafe location* at risk of a Martian quake, which may render the room inaccessible during the mission.
- **Stasis Lab**: A research facility where artifacts are brought for study. It is equipped with cooling tools, serving as a "safe destination" for temperature-sensitive artifacts.

- **Anti-Vibration Pods Room**: A storage deposit containing the pods required for transporting fragile items.
- **Cryo-Chamber**: A specialized chilling room used to rapidly cool down artifacts that require low temperatures before further handling or study.

Objectives

The primary planning objectives are as follows:

1. **Evacuate Hall B**: Move all artifacts from Hall B to Hall A to protect them from a potential structural collapse.
2. **Thermal Management**: Cool down all artifacts that require low temperatures. At the start of the scenario, these artifacts are at ambient temperature; they must be chilled to prevent permanent degradation.
3. **Study the Martian Core**: Transport the **Martian Core** artifact to the Stasis Lab for analysis. The core must be successfully cooled before it arrives at the lab.

Repository Structure

All the files, code, and resources related to this project are available on GitHub at the following repository.

The repository is structured as follows:

```
Inter-Planet-Museumvault/
├── 1.0/
├── 2.1/
├── 2.2/
├── 2.4/
├── 2.5/
└── visualization_tool/
```

- `1.0/`, `2.1/`, `2.2/`, `2.4/`: Directories corresponding to the first four problems. They mainly contain the domain and problem PDDL files, along with directories for experimental variations. You will also find the output plans from tests (typically simple `.txt` or `sas_plan` files) and a `README.md` file with running instructions.
- `2.5/`: Contains the ROS2 module for PlanSys2 related to the fifth problem. It can be directly included in a ROS2 workspace and contains specific instructions for execution.
- `visualization_tool/`: A custom tool developed to graphically represent and debug the domains and problems, with versions built for the first four problems. This tool is explained in the next chapter, and a dedicated `README.md` inside its folder explains how to use it. Additionally, the images featured in this report were generated using this software.

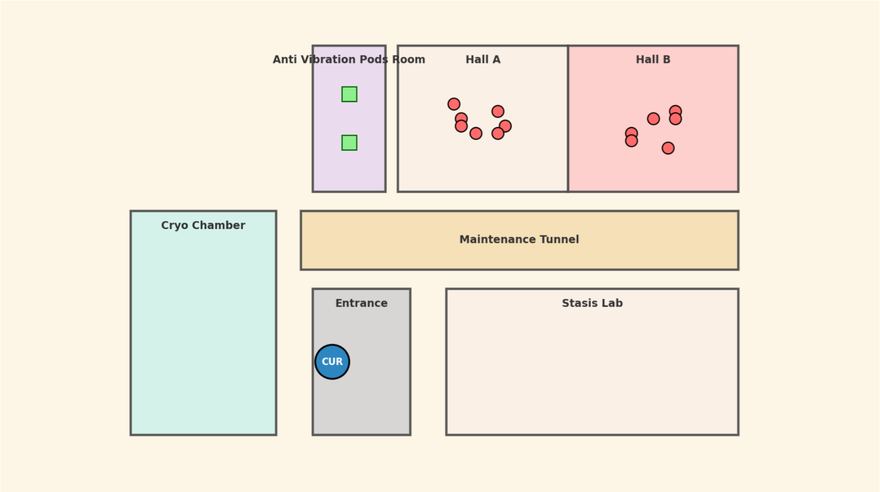


Figure 1: Initial State Shot

Chapter 1

Problem 1 - Single Robot Scenario

This initial phase models the Interplanetary Museum Vault using a single robotic curator. Our approach began with a strictly **"Vanilla "** PDDL implementation. The objective was to utilize as few language requirements as possible, serving as both a technical challenge and a baseline to understand the performance trade-offs of basic logic structures.

Our hypothesis was that simpler logic structures; avoiding complex requirements like `:negative-preconditions` or `:disjunctive-preconditions`; would yield faster computation times, albeit at the cost of more verbose plan lengths and a higher number of specific atoms.

1.1 Vanilla Implementation

The "Vanilla" implementation proved more challenging than expected. Adhering to basic PDDL forces a strictly Boolean reasoning style; for instance, one can only check if a fluent is true. To check if something is false, specific "negative" predicates must be manually maintained (e.g., `hands-empty` vs `carrying`). Furthermore, without high-level operators like `OR`, `WHEN`, or `IMPLY`, we had to distinguish specific sub-cases for each action, leading to a loss of abstraction but an increase in logical transparency.

1.1.1 Domain Type

In the `domain.pddl` file, we defined the following set of types:

- `robot`, `pod`, `location`, `artifact`, and `artifact-type`.

While the first four types are intuitive, the `artifact-type` was a design choice made early in the process. Although it currently serves as a decorative feature without direct utility in the transition rules, we retained it to ensure the model remains extensible for future mission requirements.

1.1.2 Domain Predicates

Following the "Vanilla PDDL" methodology, we have defined a set of predicates that avoid the need for the `:negative-preconditions` requirement. This is achieved by using "complementary pairs";swhere the absence of a state is represented by an explicit positive predicate (e.g., `hands-empty` vs. `carrying`).

Robot State Predicates: These predicates track the physical location, effector status, and safety systems of the robotic agent.

- `(robot-at ?r ?l)`: Defines the current location of robot `?r`.
- `(hands-empty ?r)`: Indicates the robot is not holding any object
- `(carrying ?r ?a)`: The robot is holding a non-fragile artifact directly.
- `(carrying-empty-pod ?r ?p)` / `(carrying-full-pod ?r ?p)`: Explicit states to track container handling without needing complex logical operators.
- `(sealing-mode-on ?r)` / `(sealing-mode-off ?r)`: Tracks the status of the robot's pressure seals, a prerequisite for navigating the Maintenance Tunnel.

Location Predicates

- `(connected ?l1 ?l2)`: Defines the traversable paths between rooms.
- `(is-pressurized ?l)` / `(is-unpressurized ?l)`: Defines the atmospheric state.
- `(is-seismic ?l)` / `(is-unseismic ?l)`: Distinguishes between stable zones and those requiring a non-deterministic safety check.
- `(is-safe ?l)` / `(is-unsafe ?l)`: Store the observed outcome of a seismic check.
- `(is-standard-room ?l)` / `(is-chill-room ?l)`: Defines specialized room logic for dropping artifacts (Standard vs. Cryo-Chamber).

Artifact Predicates

- `(artifact-at ?a ?l)`: The current location of an artifact.
- `(fragile ?a)` / `(no-fragile ?a)`: Determines if an artifact requires a Pod for transport.
- `(warm ?a)` / `(cold ?a)`: Tracks the thermal state. Artifacts must be transitioned to `cold` in a chill-room.
- `(is-type ?a ?t)`: Associates an artifact with a specific category for identification.

Pod Predicates

- `(pod-at ?p ?l)`: The physical location of the pod.
- `(pod-empty ?p)` / `(pod-full ?p)`: Binary state of the container.
- `(pod-contains ?p ?a)`: A link binding a specific artifact inside a specific pod.

1.1.3 Domain Actions

The action schema is engineered to manage movement, safety protocols, and logistics. To adhere to the "Vanilla" PDDL requirements, we explicitly modeled every state transition as a unique action, avoiding high-level abstractions like conditional effects.

Movement: In a vanilla implementation, branching logic must be handled at the action level. Since we cannot use a single `move` action with an `if-pressurized` check, the planner must select the specific action that matches the destination's atmospheric predicates.

- `move-to-pressurized-room`: Facilitates movement into stable, pressurized areas.
- `move-to-unpressurized-room`: Governs entry into the Maintenance Tunnel. It enforces a **hard physical constraint**: the robot must have `sealing-mode-on` in the precondition to successfully transition.

Seismic Model: A seismic event is an exogenous factor that occurs independently of the agent's actions, typically defining a *Dynamic Environment*. Modeling such dynamics in standard PDDL is complex; therefore, we treat the environment as *Static* but with *Uncertainty*. The safety of a room (i.e., the absence of a quake) is only verified when the robot attempts to enter **Hall B**.

We model this check using a **Non-Deterministic** action via the oneof operator. This allows the model to branch based on the environmental outcome:

```
(:action try-to-enter-seismic-room
    :parameters (?r - robot ?to ?from - location)
    :precondition (and
        (robot-at ?r ?from) (connected ?from ?to)
        (is-seismic ?to)
    )
    :effect (oneof
        ;; CASE A: Room is safe - Traversal successful
        (and
            (is-safe ?to)
            (not (robot-at ?r ?from)) (robot-at ?r ?to)
            (not (sealing-mode-on ?r)) (sealing-mode-off ?r)
        )
        ;; CASE B: Room is unsafe - Traversal fails
        (is-unsafe ?to)
    )
)
```

Note: Shifting from classical to non-deterministic planning requires a specialized solver (PRP). Consequently, the solution is not a linear sequence of actions but a **policy**.

Atmospheric Sealing System: The sealing system reflects the robot's interaction with the Martian atmosphere.

- activate-seal: Transitions the robot from off to on to prepare for vacuum exposure.
- deactivate-seal: A safety-critical action. It requires the is-pressurized precondition to prevent accidental decompression within a tunnel.

Manipulation: This category demonstrates the greatest increase in "action verbosity" due to the Vanilla constraints.

- *Pod Management*: We defined separate actions for pick-up-empty-pod and pick-up-full-pod. This is required because the effects on the robot's inventory state differ significantly.
- *Artifact Encapsulation* (put-in-pod): This action bridges the state from carrying an empty pod to carrying a full one by "binding" an artifact to a container at the same location.
- *Unloading Logic*: To model the cooling effect of the Cryo-Chamber without using conditional effects (when), we split the release logic into:
  1. release-artifact / release-artifact-from-pod: For standard locations.
  2. release-artifact-in-cryo / release-artifact-in-cryo-from-pod: These actions explicitly include the (cold ?a) effect and are restricted to is-chill-room.

1.2 PRP

To address the non-deterministic nature of the museum scenario, we utilized the **Planner for Relevant Policies (PRP)** [6]. Unlike classical planners that generate a linear sequence of actions, PRP is designed to find **Strong Cyclic Policies** in Fully Observable Non-Deterministic (FOND) domains. The PRP execution pipeline, as implemented in the official source code [5], is structured into four primary stages:

1. **Translation and Grounding**: The planner accepts the `domain.pddl` and `problem.pddl` files, converting the high-level PDDL into a **SAS+** representation (`output.sas`). This maps the fluents and objects into a structured variable format.
2. **Preprocessing**: The SAS+ output is analyzed to generate an optimized state graph. In this stage, the planner identifies invariants and prunes irrelevant actions to reduce the state-space search complexity.
3. **Weak Plan Generation**: PRP begins by finding a *Weak Plan* a path that succeeds under the most favorable non-deterministic outcomes (e.g., assuming seismic room is always safe). This serves as a heuristic backbone for the policy.
4. **Regression and Policy Refinement**:
   - **Dead-end Detection**: The planner simulates the non-deterministic effects (Case B: `is-unsafe`) to identify states where the goal becomes unreachable.
   - **Rule Generation**: Through a process of *regression*, PRP generates general rules to recover from these failures or to avoid certain states entirely.
   - **The Retry Loop**: If a non-deterministic action fails, the policy guides the robot back to a state where it can attempt a new strategy.

The final output is a **Strong Cyclic Policy** (`policy.out`), which provides the robot with a decision-making framework rather than a fixed path.

1.3 Visualization Tool

The raw output of a PDDL planner is a verbose text-based plan, making it difficult to validate complex state transitions. To address this, we developed a custom visualization tool to bridge the gap between abstract logical states and the physical scenario. To facilitate comprehensive analysis, we implemented two primary visualization modes:

- **Dynamic Animation**: An animated GIF, providing a continuous view of the robot's trajectory and mission execution.
- **Interactive "Comic Book" Viewer**: An interactive interface that displays the visual state alongside its corresponding set of active predicates.

This visual feedback loop significantly accelerated the iterative process of refining action preconditions and seismic logic, ensuring that the final model behaves according to the intended design.

1.4 Evaluation

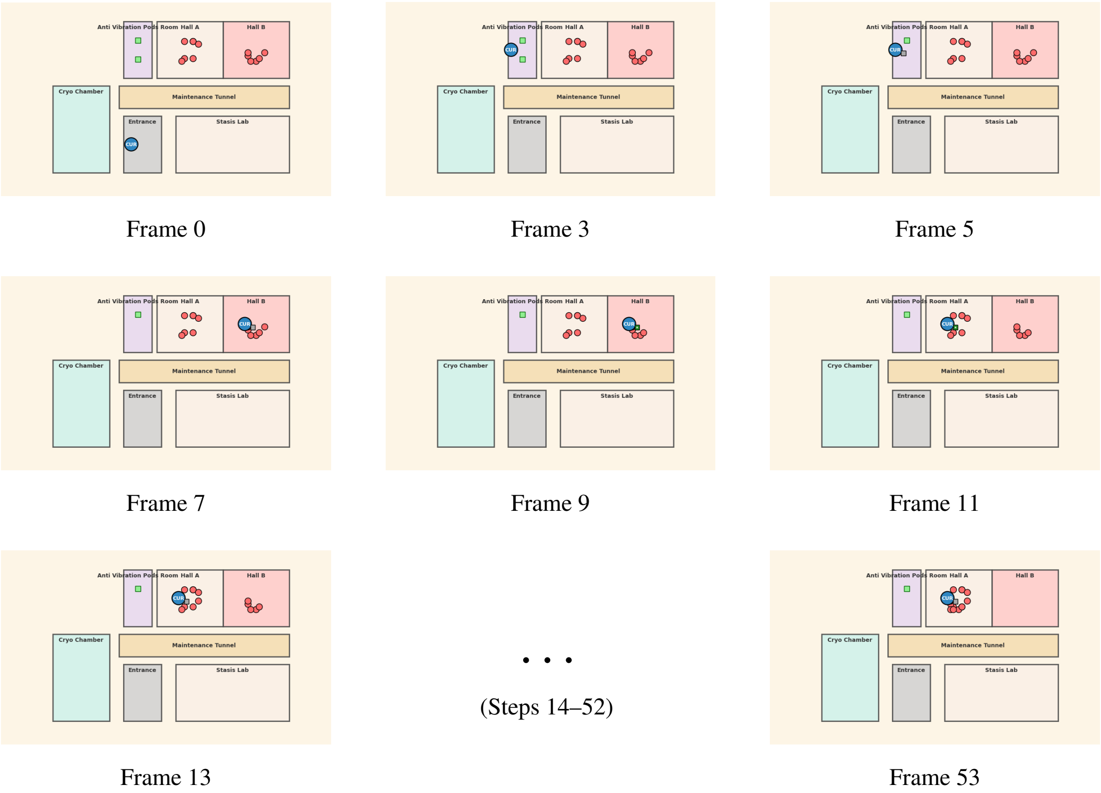
The evaluation of our work focuses on the validity of the domain and problem PDDL formulations, as well as the robustness of the policy generated by the PRP solver.

1.4.1 Domain & Problem Validation

To evaluate the quality of our modeling, we analyze whether the PDDL files accurately represent the intended world physics and logic constraints. We utilize the visual traces generated by our visualization tool, using the *Weak Plan* produced by PRP as a baseline reference for a successful execution path.

Phase 1: Hall B Evacuation

The generated solution begins with the robot navigating to retrieve an anti-vibration pod. The agent then enters an iterative cycle: encapsulating fragile artifacts from Hall B within the pod, transporting them through the maintenance tunnel, and releasing them safely in Hall A. This confirms that the logic correctly enforces the `fragile` constraint, requiring a pod for transport.



Frame 0

Frame 3

Frame 5

Frame 7

Frame 9

Frame 11

Frame 13

. . .

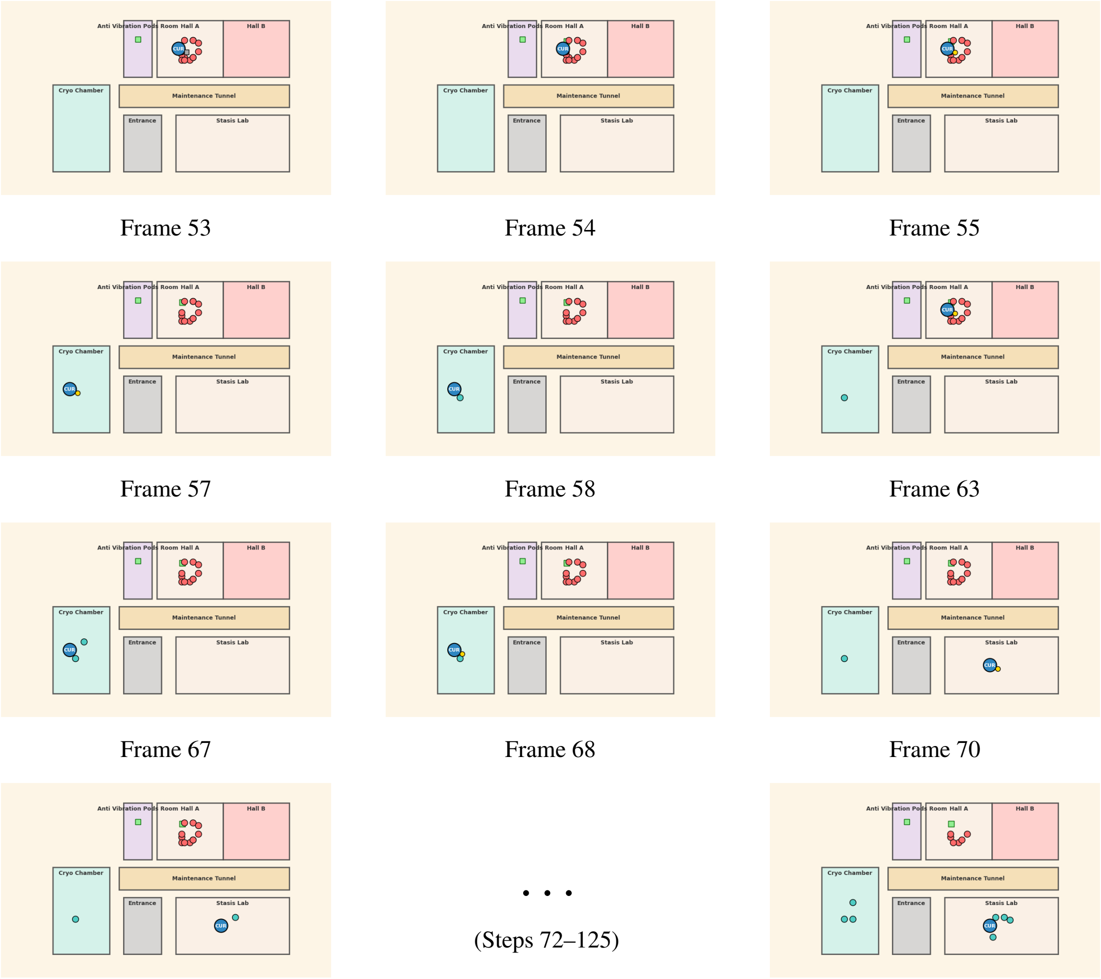
(Steps 14–52)

Frame 53

Figure 1.1: Phase 1 - Hall B Evacuation Sequence

Phase 2: Thermal Stabilization and Delivery

In the subsequent phase, the robot deposits the pod and begins managing the non-fragile artifacts located in Hall A. The agent retrieves these items and transports them to the *Cryo-Chamber*. If the mission goal specifies delivery to the *Stasis Lab*, the robot performs the delivery immediately after the cooling process; otherwise, the artifacts remain in the chamber to maintain their `cold` state.



Frame 53 Frame 54 Frame 55

Frame 57 Frame 58 Frame 63

Frame 67 Frame 68 Frame 70

Frame 71 . . . (Steps 72–125) Frame 126

Figure 1.2: Phase 2 - Thermal Stabilization and Delivery

Results Summary

By analyzing the complete video of the plan (available on our GitHub repository at: `https://github.com/your-repo-link`), it is evident that the expected goals are consistently reached. Furthermore, the execution trace confirms that the robot strictly adheres to environmental constraints, such as pressure sealing and seismic safety protocols.

1.4.2 Policy Evaluation

Similar to the plan files, the policy output generated by PRP is a verbose text file that is difficult to interpret manually. To address this, we extended our visualization tool with a feature capable of generating a **State-Transition Graph** directly from the PRP policy output.

This graph (Figure 1.3) the agent's decision-making process under uncertainty. Specifically, it shows that if Hall B is determined to be `is-unsafe`, the policy instructs the robot to remain in the Maintenance Tunnel. This represents a **Strong Cyclic Solution**: the robot can persist in the tunnel and re-attempt the entry action until **Hall B** is found to be safe, at which point it can proceed with the mission. Under this policy, success is mathematically guaranteed regardless of the non-deterministic outcomes of the seismic checks.

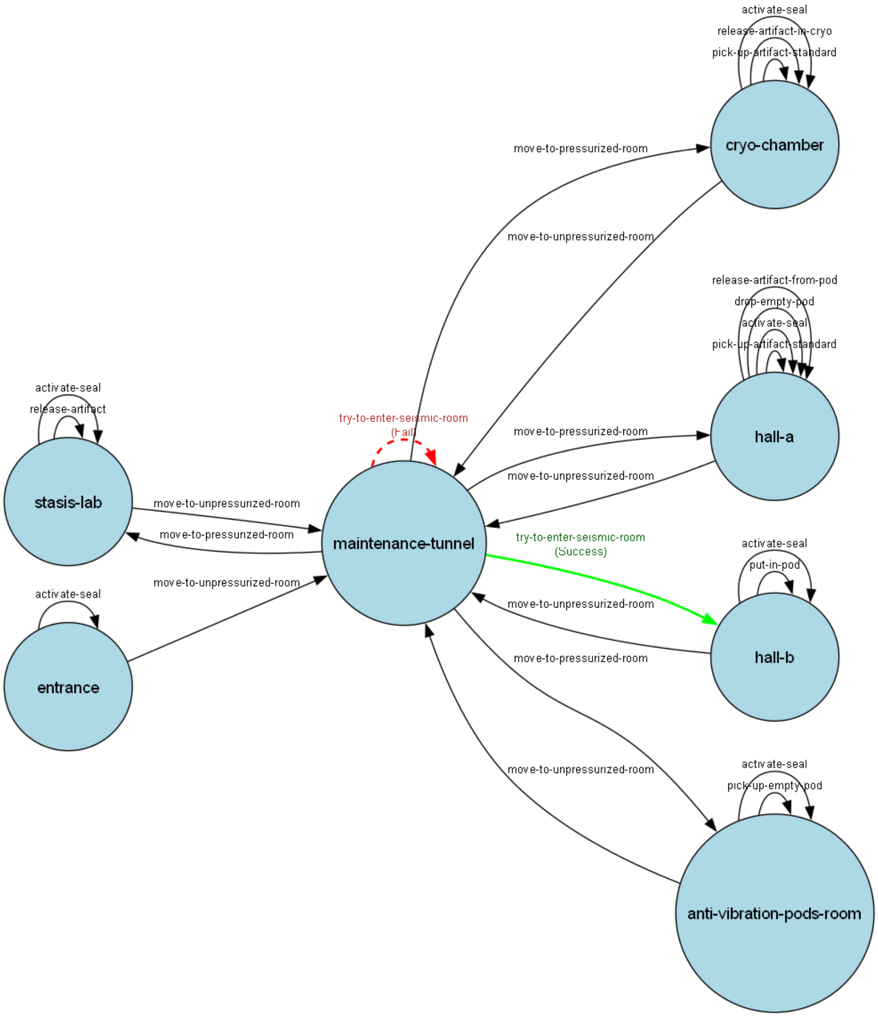


Figure 1.3: Policy State-Transition Graph

1.5 Compact Implementation

Following the initial "Vanilla" implementation, the domain was reformulated using a "Compact" modeling style. This approach leverages PDDL requirements; specifically `:negative-preconditions`, `:conditional-effects`, and `:disjunctive-preconditions`—to achieve a higher level of abstraction and simplify both the domain and problem formulations.

1.5.1 Optimization and Constraints

A primary limitation of this technique became immediately apparent during the first iteration: when utilizing `:existential-preconditions`, the PRP solver failed to compute a solution. This highlights a critical obstacle in advanced PDDL modeling: the trade-off between expressive abstraction and the specific architectural compatibility of planners.

- **Predicate Abstraction and State-Space Reduction:** Redundant predicates (e.g., *hands-empty* and *pod-empty*) were eliminated. Instead, the domain utilizes the negation of primary status fluents, such as $\neg$hands-full(r) and $\neg$pod-full(p). This reduces the number of grounded atoms the solver must track.
- **Operator Unification:** By utilizing `:disjunctive-preconditions` and `:conditional-effects`, specialized actions were merged into generalized operators. This can reduces the branching factor of the search tree.
  - **Robust Navigation:** Movement logic was unified into a single *move* action. A disjunctive precondition allows the robot to navigate both pressurized and unpressurized environments within a single operator:

    $$\text{Precondition: is-pressurized}(to) \vee \text{sealed}(r)$$

  - **Automated Thermal Adaptation:** The *release* actions now utilize conditional effects to automatically apply the *cold* predicate based on the environmental properties of the destination:

    $$\text{Effect: when (is-chill-room}(l)) \implies \text{cold}(a)$$

1.5.2 Evaluation

To rigorously compare the performance of the *Vanilla* and *Compact* implementations, we subjected the PRP solver to a multi-tiered stress test. The problem is scaled across four distinct difficulty levels, incrementally increasing the number of artifacts and, consequently, the complexity of the required plan:

- **Level 0 (Baseline):** Establishes the standard performance metrics for the solver within the previously described environment containing 13 items.
- **Level 1 (Moderate):** Increases the object volume to 23 items. This new batch of 10 artifacts consists of non-fragile items; thus, it increases the problem volume without significantly raising the logical complexity.
- **Level 2 (Heavy):** Increases the object volume to 33 items. This batch introduces 10 artifacts that require thermal processing (cold status). This increases both the volume and the routing complexity, as it necessitates a three-step dependency chain (Pick $\rightarrow$ Cryo $\rightarrow$ Stasis).
- **Level 3 (Stress):** Increases the total object volume to 43 items. This final batch consists of 10 fragile artifacts, increasing both the volume and the logical constraints on the planner.

By evaluating both implementations across these four tiers, we aim to measure how the abstraction provided by the *Compact* style; specifically the reduction in action count and predicate density, impacts the solver's execution time and overall performance.

Table 1.1: Comparative Performance: Vanilla vs. Compact Implementation

Level	**Strategy**	**State Space & Preprocessing**			**Search Performance**	
		Atoms	**Operators**	**Trans. Time (s)**	**Search Time (s)**	**States Gen.**
0	Vanilla	1117	692	**0.127s**	0.14s	**3067**
	Compact	**896**	**509**	0.175s	**0.02s**	3850
1	Vanilla	1842	1182	0.226s	0.10s	**9887**
	Compact	**1471**	**859**	**0.193**s	**0.05s**	10,095
2	Vanilla	2567	1672	0.330s	0.27s	**23,262**
	Compact	**2046**	**1209**	**0.281s**	**0.16s**	24,047
3	Vanilla	3292	2162	0.390s	**0.31s**	**15,755**
	Compact	**2621**	**1559**	**0.230s**	0.41s	48,599

Observation 1: State Space Reduction

As expected, the reduction of predicates and actions in the domain file resulted in a significantly smaller state space. Fewer atoms are required to describe each state, and fewer operators are generated, leading to a lower branching factor.

An unexpected finding, however, is the **Translation Time**. with the exception of Level 0, it is lower in the Compact Implementation. Initially, we hypothesized that because the translator has to handle more complex operators (conditional effects), the process would be slower. In reality, the reduction in the total volume of data to process outweighs the complexity of individual operators.

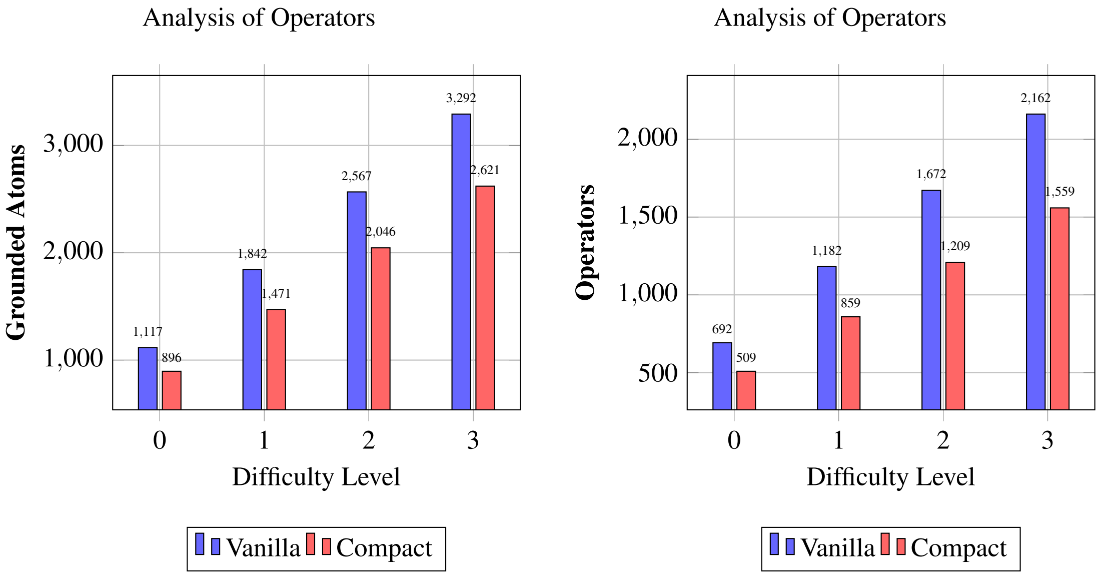


Figure 1.4: State Space Reduction - Colum Plot.

Observation 2: Search Performance Trade-offs

The search performance results yielded three surprising insights:

1. **Heuristic Efficiency:** The Vanilla Implementation consistently requires less exploration, as indicated by the lower number of *Generated States*. This suggests that the exploration is more efficient in the Vanilla version. It implies that the heuristic guidance is less effective when applied to the higher abstraction level of the Compact Implementation.

2. **Computation Speed vs. Exploration:** Even though the Compact Implementation explores more states, the *Search Time* is often lower (except for the last case, where the exploration is three times larger, making it an unfair comparison). This indicates that the lower number of atoms makes the node expansion (and computation) significantly faster. Essentially, the solver searches "more nodes per second" in the Compact domain.

3. **Heuristic Sensitivity (The Level 2 Peak):** We observed a peak in exploration at Level 2 for the Vanilla implementation, which surprisingly decreases at Level 3. We expected this value to increase linearly, but the drop confirms that search performance relies heavily on heuristic quality. The data evidences that in the second level (complex dependency chains), the heuristic was less certain of the optimal path and therefore required more exploration than in the high-volume third level.

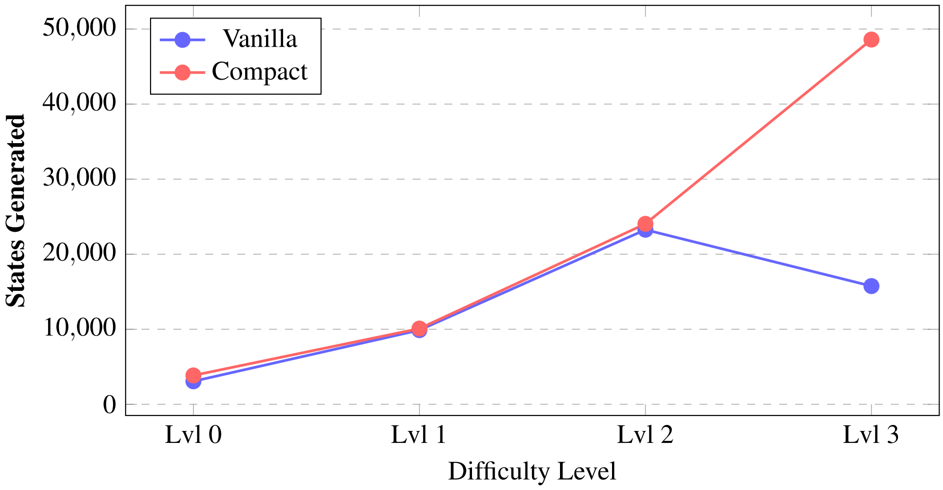


Figure 1.5: Search Effort Analysis

1.6 Output Plan and Annotated Trace

This page presents the **solution trace** for the Single Robot Scenario in the vanilla implementation. You can find all the test and plans in the repository [3]. The following table maps the PDDL actions from the `sas_plan` to the logical requirements of the assignment.

Plan Actions (SAS Output)	**Logical Commentary**

`(activate-seal curator)`
`(move-to-unpressurized-room curator entrance maintenance-tunnel)`
`(move-to-pressurized-room curator maintenance-tunnel anti-vibration-pods-room)`
`(pick-up-empty-pod curator anti-vibration-pods-room pod2)`
	Phase 1: Initialization. The curator activates its internal seal to safely traverse the unpressurized maintenance tunnel. It reaches the pod room to retrieve an anti-vibration container (pod2).

`(try-to-enter-seismic-room_DETDUP_0 curator hall-b maintenance-tunnel)`
`(activate-seal curator)`
`(put-in-pod asteroid-ad29tv-rock-sample hall-b curator pod2)`
`(release-artifact-from-pod curator asteroid-ad29tv-rock-sample hall-a pod2)`
	Phase 2: Fragile Evacuation. Hall B is a high-risk seismic zone. The plan uses the non-deterministic entry check. Fragile items (e.g., `asteroid-rock`) are secured in a pod before being moved to the stable `hall-a`.

`(move-to-pressurized-room curator maintenance-tunnel cryo-chamber)`
`(release-artifact-in-cryo curator mart-east-core-drill cryo-chamber)`
`(pick-up-artifact-standard mart-east-core-drill cryo-chamber curator)`
	Phase 3: Thermal Processing. To satisfy the goal for Martian Core Samples, the robot detours to the `cryo-chamber`. The `release-artifact-in-cryo` action effectively transitions the item state from `warm` to `cold`.

`(move-to-pressurized-room curator maintenance-tunnel stasis-lab)`
`(release-artifact curator mart-east-core-drill stasis-lab)`
`; ... [Plan repeats for remaining samples]`
	Phase 4: Goal Achievement. Finally, the stabilized cores are delivered to the `stasis-lab`. The plan successfully manages the single-item capacity constraint by performing multiple round-trips through the vault.

Observations: The plan's efficiency is limited by the "Vanilla" implementation's requirements: the robot must manually toggle the `sealing-mode` for every tunnel transit and can only carry one artifact (or one pod) at a time. The generated sequence (length: ~70 steps) confirms that the heuristic correctly identified the `hall-a` staging area as the optimal intermediate hub to minimize decompression cycles.

Chapter 2

Problem 2 - Multi Robots with Capabilities Scenario

Following the project specifications, we extended the single-agent scenario to a **Multi-Agent System**. The core objective remains the retrieval and storage of artifacts, but the problem now requires coordination between different robotic agents with different features.

2.0.1 Operational Constraints and Agent Specialization

To transition from a single-agent to a multi-agent environment, we implemented a system of constraints that restricts how robots interact with the vault and the artifacts. These limitations are governed by three main logical dimensions:

- **Spatial Access**: Managed via the `(can-access ?r ?l)` predicate, which defines the security clearance of a robot for a specific room. If a robot lacks this clearance, the move actions are invalidated for that location.
- **Manipulation Permissions**: Defined by `(can-pickup ?r ?at)`, ensuring that only qualified robots can handle sensitive or specialized artifacts types: `scientific`, `technological`, `top-secret`.
- **Payload Capacity**: A physical constraint that distinguishes between standard units (single inventory slot) and heavy-duty units (double inventory slot), modeled through the `can-carry-two` property.

Based on these constraints, the vault staff is now composed of three specialized types of agents:

The Curator

Designed for high-security retrieval of sensitive data and samples.

- **Access:** Full clearance for the main vault areas, including the *Cryo-Chamber*, *Halls A/B*, and the *Pod Room*.
- **Pick-up:** Authorized only for *Scientific* and *Top-Secret* artifacts.
- **Capacity:** Standard (1 item).

The Technician

The "logistics backbone" of the operation, optimized for moving multiple technological components.

- **Access:** Same broad access as the Curator to facilitate transport throughout the museum.
- **Pick-up:** Restricted exclusively to *Technological* artifacts.
- **Capacity: Enhanced**. It is the only robot type capable of carrying two items simultaneously using a secondary inventory slot.

The Scientist

A highly specialized but stationary unit located deep within the facility.

- **Access:** Highly restricted; it is primarily confined to the *Stasis Lab* and the adjacent *Maintenance Tunnel.*
- **Pick-up: Versatile**. It is the only agent with the skills to handle all three artifact types (*Scientific, Technological, and Top-Secret*).
- **Capacity:** Standard (1 item).

2.0.2 Extended Capacity and Inventory Logic

A key requirement of this extension was to model agents with different load capacities. While standard robots (Curator and Scientist) maintain the single-slot inventory logic (`hands-empty` vs `carrying`), the **Technician** robot has been modeled with a higher capacity.

To implement this without complicating the state space for single-slot agents, we introduced a specific set of predicates and actions for the second slot:

- **Predicates:** We added `(can-carry-two ?r)` to flag high-capacity agents, along with `(carrying-second-object ?r ?a)` and `(second-slot-empty ?r)` to track the status of the additional inventory space.
- **Actions:** Specific actions `pick-up-second-object` and `release-second-object` have been implemented. These actions are conditionally available only to robots possessing the `can-carry-two` capability, allowing the Technician to transport two artifacts simultaneously (e.g., carrying a generic object in the primary slot and a second object in the auxiliary slot). Note that the second slot can contain just *non-fragile* objects; this implies that the robot cannot carry 2 pods simultaneously, incentivizing cooperation.

2.0.3 Domain Modifications

To support these extensions, the PDDL action schemas were updated to include the `?r - robot` parameter, ensuring that preconditions and effects are checked against the specific agent performing the action.

Furthermore, the non-deterministic nature of the environment is handled via the `try-to-enter-seismic-room` action, which uses the `oneof` operator. In a multi-agent context, this implies that if a robot fails to cross a seismic room, it may delay the entire fleet, requiring the planner to find robust distinct paths or contingency plans.

The initial state in the problem file places the robots at their starting stations (Curators and Technicians at the Entrance, Scientist at the Stasis Lab), and the goal state remains, as in the previous stage, the successful categorization and storage of all artifacts.

2.0.4 Plan Evaluation and Behavior Analysis

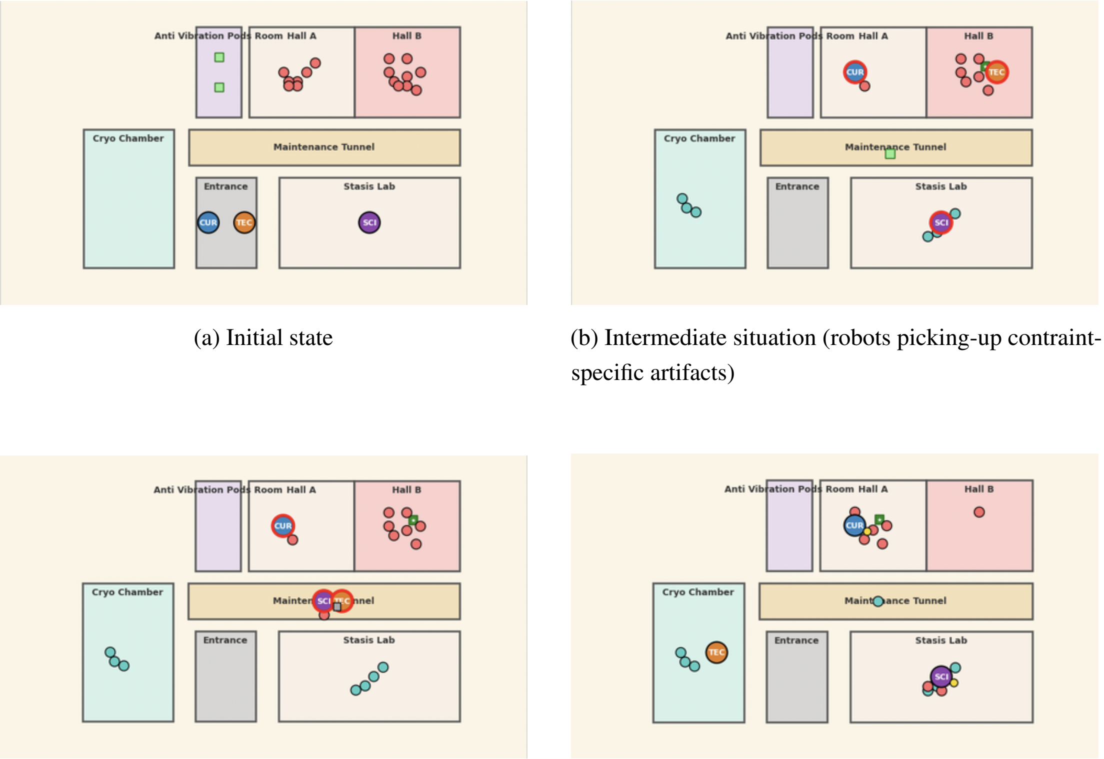
To validate the Multi-Agent extension, we analyzed the linear solution generated by the planner. The resulting plan consists of **166 steps**, significantly longer than the single-agent scenario due to the increased volume of artifacts and the coordination overhead.

Analysis of the execution trace reveals that the domain constraints have successfully induced specialized behaviors and collaboration among the agents:

- **Verification of Role Specialization**: The strict separation of duties defined by the `can-pickup` predicates is respected throughout the plan. The *Curator* is exclusively assigned to retrieve Scientific and Top-Secret items; the *Technician* is dispatched to Hall B solely for Technological artifacts; and the *Scientist* acts as the specialized receiver in the Stasis Lab.

- **Validation of the Dual-Capacity Logic**: The planner successfully exploits the Technician's ability to carry multiple items to optimize retrieval. As observed in the trace, the agent `technician2` enters Hall B and performs sequential pick-ups (e.g., retrieving both `rover-wheel` and `space-suit`) before returning, confirming the functionality of the `pick-up-second-object` action.

- **Emergent Collaboration Strategy**: A "Relay" (or bucket brigade) strategy emerged spontaneously in which the planner optimized the flow by having the Curator drop artifacts in the *Maintenance Tunnel* so the *Scientist*, stationed nearby, enters the tunnel solely to retrieve these items and transport them into the lab, final destination of many artifacts. This minimizes the Curator's travel distance, allowing it to return immediately to the collection points.



(a) Initial state

(b) Intermediate situation (robots picking-up contraint-specific artifacts)

(c) Scientist takes items brought by technician (collaboration)

(d) Goal state

Figure 2.1: Visual execution of the Multi-Agent plan.

2.0.5 Integration of Aerial Units: The Drone Case

To address the inherent instability of specific vault sectors, we introduced a specialized aerial agent: the **Drone**. Unlike ground-based units, the Drone is characterized by the `can_fly` capability, which allows it to navigate through hazardous environments.

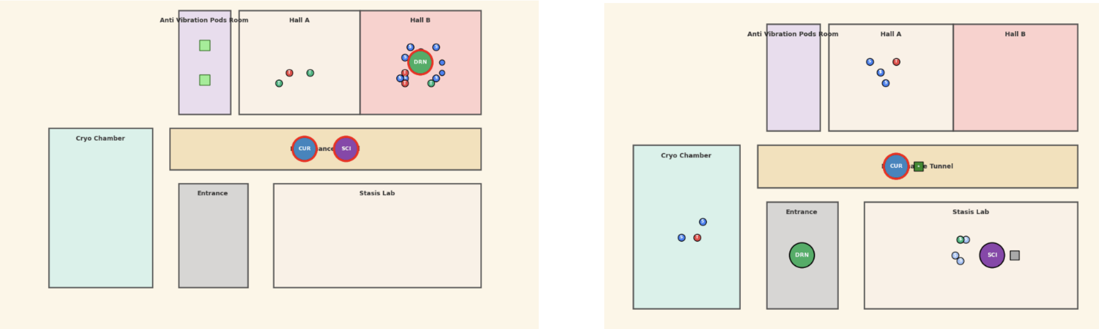
In this model, **Hall B** is designated as permanently `is_seismic` and, by default, `not (is-safe)`. This design choice effectively prevents standard robots from entering the room, avoiding the need for complex negative preconditions which are often poorly supported by certain temporal planners. The Drone bypasses the safety constraint through a specific `fly_into_seismic_room` action; since it flies, the seismic nature of the floor does not impede its operations.

The Drone serves as a high-mobility courier: its operational area includes the *Entrance*, the *Maintenance Tunnel*, and *Hall B*. Its primary role is to retrieve all artifact types from the seismic zone and transport them to the tunnel, where ground robots can safely take over the final delivery. To maximize efficiency, the Drone is equipped with two inventory slots (`can-carry-two`). A final constraint requires the Drone to return to the *Entrance* after completing all tasks.

2.0.6 Drone implementation

Preliminary tests using the **Downward** planner revealed that a four-robot configuration (Curator, Technician, Scientist, and Drone) leads to an exponential increase in the search space, resulting quite higher computation times. To maintain scalability while preserving functionality, we simplified the system to a three-agent model with strict task partitioning:

- **Curator**: Responsible for `technological` and `top-secret` artifacts. Operates in all rooms except Hall B (including the Lab).
- **Drone**: Handles all artifact types within Hall B, the Tunnel, and the Entrance. It is the only agent capable of interacting with the seismic Hall.
- **Scientist**: Dedicated to `scientific` artifacts. Operates between the Tunnel, Lab, and Cryo-chamber.



(a) The drone (green cyrcle) goes in unsafe Hall B to take outside artifacts for other robots.

(b) Drone returns at the entrance after having transported needed artifacts (as defined in the `problem.pddl`).

2.0.7 Comparative Analysis of Plan Efficiency

When comparing the **Drone-based approach** to the standard ground-only configurations under identical initial states and goals, we observe a shift in plan structure and efficiency. In the standard multi-agent scenario, robots often compete for access to stable zones or wait for safety clearances, leading to plan lengths averaging around 260 steps.

Conversely , the **Drone-relay strategy** in the workflow the Drone acts as a specialized buffer that handles all hazardous-zone interactions, allowing other robots to focus on high-throughput transport in safe zones. This specialization, combined with logical pruning, allowed the planner to find highly optimized solutions of approximately 240 steps for the same set of 16 artifacts. This comparison suggests that, while adding agents usually increases search complexity, adding a **specialized** agent with a restricted operational niche can actually guide the heuristic toward more direct and efficient solutions.

2.1 Multi-Agent Plan Execution and Annotated Trace (Base Case)

This section analyzes the execution trace for the Multi-Robot Scenario. The following table (which may span across multiple pages) maps the PDDL actions to the logical requirements of the assignment.

Plan Actions (SAS Output)	**Logical Commentary**

`(activate-seal curator)`
`(activate-seal scientist)`
`(activate-seal technician)`
`(move-to-unpressurized-room technician entrance maintenance-tunnel)`
	Phase 1: Multi-Agent Deployment. All robots synchronize their atmospheric seals. The `technician` moves toward the maintenance tunnel to address high-risk tasks.

`(try-to-enter-seismic-room_DETDUP_0 technician hall-b maintenance-tunnel)`
`(pick-up-slot-1 rusty-lightsaber technological hall-b technician)`
`(pick-up-slot-2 space-suit technological hall-b technician)`
	Phase 2: Specialized Recovery. The `technician` leverages its `can-carry-two` capability to retrieve both the lightsaber and the suit in a single sortie from the seismic zone.

`(drop-pod-slot-1 curator pod2 maintenance-tunnel)`
`(pick-up-pod-slot-1 scientist maintenance-tunnel pod2)`
`(release-artifact-cryo-slot-1 technician rusty-lightsaber cryo-chamber)`
	Phase 3: Agent Coordination. A handover occurs in the tunnel: the `curator` drops a pod for the `scientist`. The technician processes items in the `cryo-chamber`.

Plan Actions (SAS Output)	**Logical Commentary**

`(move-to-pressurized-room scientist maintenance-tunnel stasis-lab)`
`(release-artifact-slot-1 scientist mart-west-core-drill stasis-lab)`
`; ... [Plan continues with remaining samples]`
	Phase 4: Global Goal Fulfillment. The `scientist` delivers scientific-grade samples to the `stasis-lab`. The plan successfully interleaves movements without collisions.

Observations: The multi-agent plan demonstrates a significant reduction in total "makespan". By distributing tasks, the solver utilized the `technician` for hazardous lifting and the `scientist` for sensitive lab deliveries.

Chapter 3

Problem 3 - Hierarchical Task Network

3.1 Delivery Task Decomposition

Building upon the initial implementation, the problem was transitioned to a Hierarchical Task Network (HTN) framework. The core objective is the `delivery` task, defined as the transport of an artifact by a robot between two distinct locations.

```
:task delivery
    :parameters (?r - robot ?a - artifact ?from - location
                 ?to - location ?at - artifact-type)
```

The primary method, `m_delivery_direct`, decomposes this high-level goal into three sequential subtasks. After verifying the initial positions of the artifact and the robot, the task is split into preparation, loading, and transportation phases.

```
:method m_delivery_direct
    :parameters (?r - robot ?a - artifact ?from - location
                 ?to - location ?at - artifact-type ?r_loc - location)
    :task (delivery ?r ?a ?from ?to ?at)
    :precondition (and (artifact-at ?a ?from) (robot-at ?r ?r_loc))
    :subtasks
        (t0 (prepare_robot ?r ?r_loc ?from))
        (t1 (load_artifact ?r ?a ?at ?from))
        (t2 (transport_to_target ?r ?a ?at ?from ?to))
    :ordering (and (t0 < t1) (t1 < t2))
```

- **Prepare Robot**: Ensures the robot is at the pickup location and equipped with a pod if required by the artifact's fragility.
- **Load Artifact**: Handles the specific pickup procedure based on the artifact type and available equipment.
- **Transport to Target**: Manages the movement through the environment, including specialized stops at a cryo-chamber for artifacts requiring low-temperature maintenance.

3.1.1 Robot Preparation

The prepare_robot task ensures the agent is ready for loading. Three distinct methods handle various initial states:

1. **Direct Navigation**: Used for non-fragile artifacts; the robot simply navigates to the artifact's location.

```
:method m_prep_direct
    :task (prepare_robot ?r ?start_loc ?a_loc)
    :subtasks (task0 (get_to ?r ?start_loc ?a_loc))
```

2. **Pod Acquisition**: If a pod is required, the robot first navigates to a free pod, picks it up, and then proceeds to the artifact.

```
:method m_prep_fetch_pod
    :task (prepare_robot ?r ?start_loc ?a_loc)
    :subtasks
        (task0 (get_to ?r ?start_loc ?pod_loc))
        (task1 (pick-up-pod-slot-1 ?r ?pod_loc ?p))
        (task2 (get_to ?r ?pod_loc ?a_loc))
    :ordering (and (task0 < task1) (task1 < task2))
```

3. **Pod Disposal**: If the robot is carrying an unnecessary pod, it must drop the current pod before moving to the target artifact.

```
:method m_prep_drop_pod_first
    :task (prepare_robot ?r ?start_loc ?artifact_loc)
    :subtasks
        (task0 (drop-pod-slot-1 ?r ?p ?start_loc))
        (task1 (get_to ?r ?start_loc ?artifact_loc))
    :ordering (and (task0 < task1))
```

3.1.2 Artifact Loading

The load_artifact task is simplified into two main methods. An artifact can either be loaded into an equipped pod (m_load_into_pod) or picked up directly using the robot's internal slots (m_load_slot_1). For specialized robots such as the *Technician*, a secondary slot method (m_load_slot_2) is also available.

```
:method m_load_into_pod
    :task (load_artifact ?r ?a ?at ?loc)
    :subtasks (task0 (put-in-pod ?a ?at ?loc ?r ?p))

:method m_load_slot_1
    :task (load_artifact ?r ?a ?at ?loc)
    :subtasks (task0 (pick-up-slot-1 ?a ?at ?loc ?r))
```

3.1.3 Transport and Unloading

The transport_to_target task handles the logistics of movement and environmental requirements.

- **Direct Transport**: The robot moves to the destination and executes the unload_artifact task.

```
:method m_transport_direct
    :task (transport_to_target ?r ?a ?at ?curr_loc ?target_loc)
    :subtasks
        (task0 (get_to ?r ?curr_loc ?target_loc))
        (task1 (unload_artifact ?r ?a ?target_loc))
    :ordering (and (task0 < task1))
```

- **Chilled Transport**: If an artifact requires cooling, the robot routes through a cryo-chamber, unloads the artifact to be chilled, reloads it, and then continues to the final destination.

```
:method m_transport_chill
    :task (transport_to_target ?r ?a ?at ?curr_loc ?target_loc)
    :subtasks
        (task0 (get_to ?r ?curr_loc ?cryo_loc))
        (task1 (unload_artifact ?r ?a ?cryo_loc))
        (task2 (load_artifact ?r ?a ?at ?cryo_loc))
        (task3 (get_to ?r ?cryo_loc ?target_loc))
        (task4 (unload_artifact ?r ?a ?target_loc))
    :ordering ( (task0 < task1) (task1 < task2) (task2 < task3)
            (task3 < task4) )
```

The unload_artifact subtask is further decomposed based on room types (standard vs. cryo-chamber) and whether the robot should retain or drop the pod after delivery.

3.1.4 Motion Implementation

Motion in this HTN implementation is divided into two layers of abstraction: the get_to task and the step task.

Path Abstraction: get_to

The get_to task represents high-level navigation between two locations, regardless of the distance or the number of intermediate nodes.

- If the robot is already at the destination, m_get_to_stay triggers a stay-at primitive.
- If the destination is adjacent, m_get_to_direct executes a single step task.
- For distant locations, m_get_to_transit recursively decomposes the path into two intermediate steps, ensuring spatial connectivity is maintained through preconditions.

Primitive Movement: step

The step task handles the physical constraints of moving between two adjacent rooms. This layer abstracts the environmental hazards:

- **Pressurized Environments**: Handled by `m_step_pressurized`, allowing standard movement using the action `move-to-pressurized-room`.
- **Unpressurized Environments**: Requires the `activate-seal` action before entering the room with `move-to-unpressurized-room`.
- **Seismic Zones**: Handled by `m_step_seismic`, which utilizes the `wait-seismic-room-safe` primitive.

For the purpose of this implementation, seismic movement has been determinized. While the original problem involves stochastic risks when entering seismic zones, this HTN model assumes the robot waits for a safe window to ensure a successful transition. This approach avoids the complexity of generating full policies within the HTN, effectively representing a "strong cyclic" solution where the robot persists until the action succeeds.

3.1.5 Problem Formulation

To initialize a specific mission within the `problem.hddl` file, we define an `:htn` block that specifies the high-level tasks to be executed. The following example demonstrates the delivery tasks requirements for transporting a *quantum-chip* from *hall-b* to the *stasis-lab*.

In this scenario we implement a **relay strategy**: the *technician* transports the artifact to the *maintenance-tunnel*, acting as a handover point, where the *scientist* then takes possession to complete the delivery to the *stasis-lab*.

```
:htn
    :parameters ()
    :subtasks
        (t1 (delivery technician quantum-chip hall-b maintenance-tunnel
             technological))
        (t2 (delivery scientist quantum-chip maintenance-tunnel stasis-lab
             technological))
    :ordering (and (t1 < t2))
```

The HTN framework allows for flexible plan generation based on desired final states. For instance, if the artifact must be delivered in a refrigerated state, we specify this requirement in the `:goal` section:

```
:goal
    (cold quantum-chip)
```

When resolving the plan, the solver will automatically select the `m_transport_chill` method rather than the direct transport method to satisfy the `(cold ?a)` predicate. It is important to note that when using this HTN approach, explicitly defining the artifact's final location in the goal section (e.g., via `artifact-at`) is no longer strictly necessary. The successful decomposition and execution of the `delivery` tasks inherently ensure that the artifact reaches its target destination.

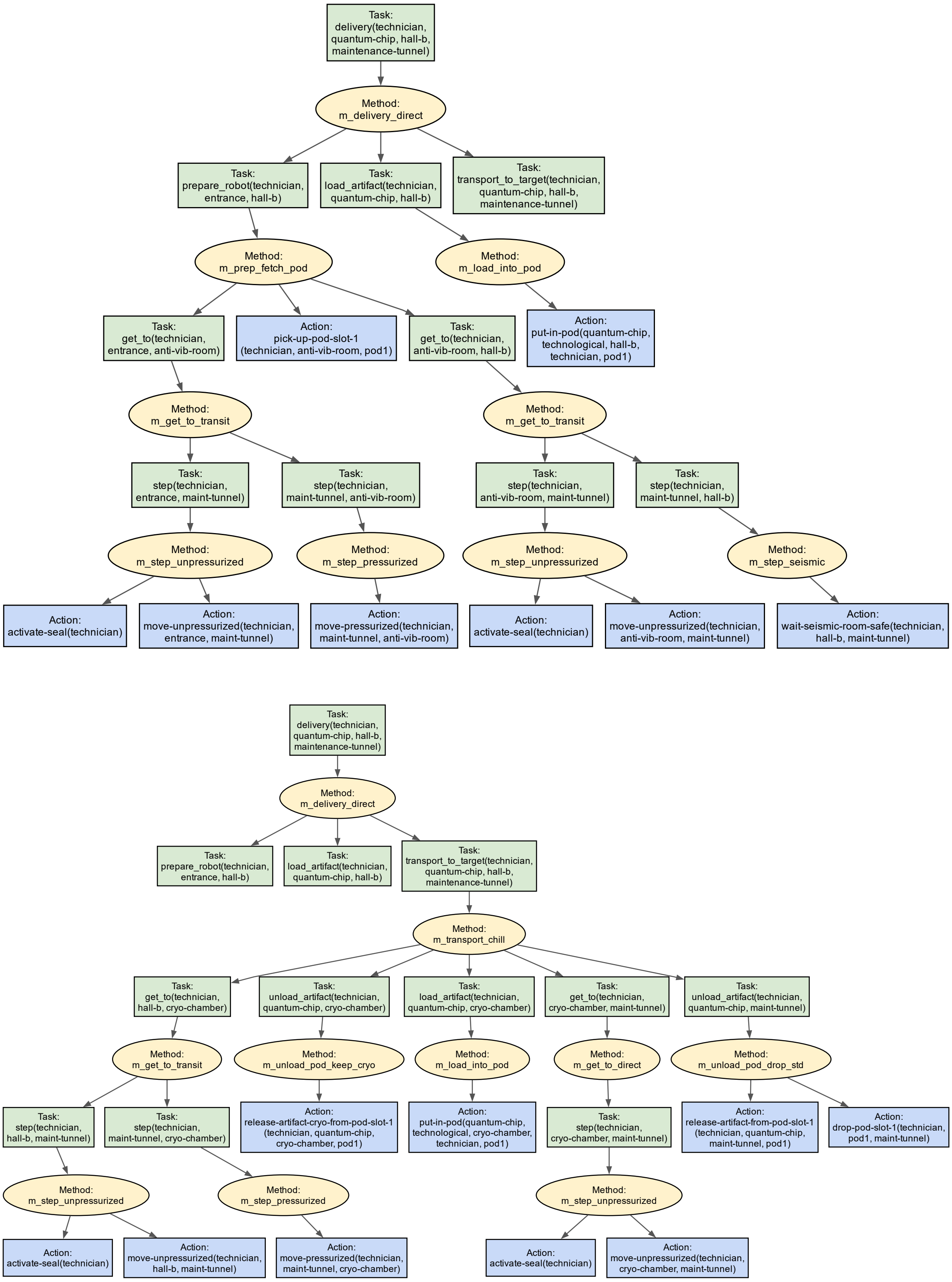

Task: delivery(technician, quantum-chip, hall-b, maintenance-tunnel)
Method: m_delivery_direct
Task: prepare_robot(technician, entrance, hall-b)
Task: load_artifact(technician, quantum-chip, hall-b)
Task: transport_to_target(technician, quantum-chip, hall-b, maintenance-tunnel)
Method: m_prep_fetch_pod
Method: m_load_into_pod
Task: get_to(technician, entrance, anti-vib-room)
Action: pick-up-pod-slot-1 (technician, anti-vib-room, pod1)
Task: get_to(technician, anti-vib-room, hall-b)
Action: put-in-pod(quantum-chip, technological, hall-b, technician, pod1)
Method: m_get_to_transit
Method: m_get_to_transit
Task: step(technician, entrance, maint-tunnel)
Task: step(technician, maint-tunnel, anti-vib-room)
Task: step(technician, anti-vib-room, maint-tunnel)
Task: step(technician, maint-tunnel, hall-b)
Method: m_step_unpressurized
Method: m_step_pressurized
Method: m_step_unpressurized
Method: m_step_seismic
Action: activate-seal(technician)
Action: move-unpressurized(technician, entrance, maint-tunnel)
Action: move-pressurized(technician, maint-tunnel, anti-vib-room)
Action: activate-seal(technician)
Action: move-unpressurized(technician, anti-vib-room, maint-tunnel)
Action: wait-seismic-room-safe(technician, hall-b, maint-tunnel)
Task: delivery(technician, quantum-chip, hall-b, maintenance-tunnel)
Method: m_delivery_direct
Task: prepare_robot(technician, entrance, hall-b)
Task: load_artifact(technician, quantum-chip, hall-b)
Task: transport_to_target(technician, quantum-chip, hall-b, maintenance-tunnel)
Method: m_transport_chill
Task: get_to(technician, hall-b, cryo-chamber)
Task: unload_artifact(technician, quantum-chip, cryo-chamber)
Task: load_artifact(technician, quantum-chip, cryo-chamber)
Task: get_to(technician, cryo-chamber, maint-tunnel)
Task: unload_artifact(technician, quantum-chip, maint-tunnel)
Method: m_get_to_transit
Method: m_unload_pod_keep_cryo
Method: m_load_into_pod
Method: m_get_to_direct
Method: m_unload_pod_drop_std
Task: step(technician, hall-b, maint-tunnel)
Task: step(technician, maint-tunnel, cryo-chamber)
Action: release-artifact-cryo-from-pod-slot-1 (technician, quantum-chip, cryo-chamber, pod1)
Action: put-in-pod(quantum-chip, technological, cryo-chamber, technician, pod1)
Task: step(technician, cryo-chamber, maint-tunnel)
Action: release-artifact-from-pod-slot-1 (technician, quantum-chip, maint-tunnel, pod1)
Action: drop-pod-slot-1(technician, pod1, maint-tunnel)
Method: m_step_unpressurized
Method: m_step_pressurized
Method: m_step_unpressurized
Action: activate-seal(technician)
Action: move-unpressurized(technician, hall-b, maint-tunnel)
Action: move-pressurized(technician, maint-tunnel, cryo-chamber)
Action: activate-seal(technician)
Action: move-unpressurized(technician, cryo-chamber, maint-tunnel)

Figure 3.1: Delivery Task 1, splitted in two to help visualization

3.2 Delivery Tasks Search

The primary limitation of the previous structure is the necessity to manually define the list of delivery tasks, which lacks flexibility. To address this, we introduced a higher level of abstraction by developing a new Hierarchical Task Network (HTN) domain and problem. In this refined architecture, delivery operations act as primitive actions, and the planning objective is to synthesize a valid sequence of these actions autonomously.

Note: The implementation was divided into two distinct domains to mitigate state-space explosion; a unified approach would saturate memory before the planner could yield a solution.

3.2.1 Process Room Task

We define a new abstract macro-task, `process-room`, responsible for managing all artifacts within a specific location based on their assigned destinations.

```
(:task process-room :parameters (?room - location))
```

The decomposition relies on a tail-recursive strategy:

```
(:method m_process_room_single
    :task (process-room ?room)
    :ordered-subtasks (and
        (t1 (deliver ?a ?at ?target))
        (t2 (process-room ?room)) )

(:method m_process_room_double
    :task (process-room ?room)
    :ordered-subtasks (and
        (t1 (double_deliver ?a1 ?at1 ?target1 ?a2 ?at2 ?target2))
        (t2 (process-room ?room)) )

(:method m_process_room_done
    :task (process-room ?room)
    :precondition (not (exists (?a - artifact) (artifact-at ?a ?room)))
    :ordered-subtasks () )
```

Note: The `m_process_room_double` method was implemented to map the `can-carry-two` capability of the technician robot, allowing it to handle two objects simultaneously.

3.2.2 Deliver Task

The `deliver` and `double_deliver` tasks serve as an intermediate abstraction layer between the `process-room` macro-task and the delivery primitive actions. They are utilized to enforce robot access permissions and manage relay (handoff) behaviors when a single robot cannot reach the final destination.

```
(:method m_deliver_direct
    :task (deliver ?a ?at ?target)
    :ordered-subtasks (and (t1 (delivery ?r ?a ?curr_loc ?target ?at))) )
```

```
(:method m_double_deliver_direct
    :task (double_deliver ?a1 ?at1 ?target1 ?a2 ?at2 ?target2)
    :ordered-subtasks (and
        (t1 (double_delivery ?r ?a1 ?a2 ?curr_loc ?target1 ?at1 ?at2))) )

(:method m_deliver_relay
    :task (deliver ?a ?at ?target)
    :ordered-subtasks (and
        (t1 (delivery ?r1 ?a ?curr_loc ?mid_loc ?at))
        (t2 (delivery ?r2 ?a ?mid_loc ?target ?at))) )

(:method m_double_deliver_relay
        :task (double_deliver ?a1 ?at1 ?target1 ?a2 ?at2 ?target2)
        :ordered-subtasks (and
            (t1 (double_delivery ?r1 ?a1 ?a2 ?curr_loc ?mid_loc ?at1 ?at2))
            (t2 (delivery ?r2 ?a1 ?mid_loc ?target1 ?at1))
            (t3 (delivery ?r3 ?a2 ?mid_loc ?target2 ?at2))) )
```

Note: An alternative tail-recursive formulation for the relay task was also evaluated:

```
(:method m_deliver_relay
    :task (deliver ?a ?at ?target)
    :ordered-subtasks (and
        (t1 (delivery ?r1 ?a ?curr_loc ?mid_loc ?at))
        (t2 (deliver ?a ?at ?target)) )
```

While elegant, this formulation proved impractical with suboptimal planners utilizing greedy search strategies. The planner frequently routed artifacts through inefficient intermediate nodes (e.g., the entrance rather than the optimal maintenance tunnel). Consequently, we retained the explicit two-step relay formulation for better heuristic guidance.

3.2.3 Problem Formulation

The initial task network dictates a strictly ordered, room-by-room resolution strategy. The planner is forced to sequence two main macro-tasks: it must completely resolve (process-room hall-a) before it can begin executing (process-room hall-b).

The scenario involves a total of 9 active artifacts, divided by type (scientific, technological, and top-secret), with specific initial states and goal destinations:

- **Hall A (4 artifacts):** Contains two artifacts destined for the *Stasis Lab* and two destined for the *Cryo Chamber*.
- **Hall B (5 artifacts):** Contains three technological artifacts that must be transported to the *Stasis Lab*, and two artifacts (scientific and top-secret) destined for *Hall A*.

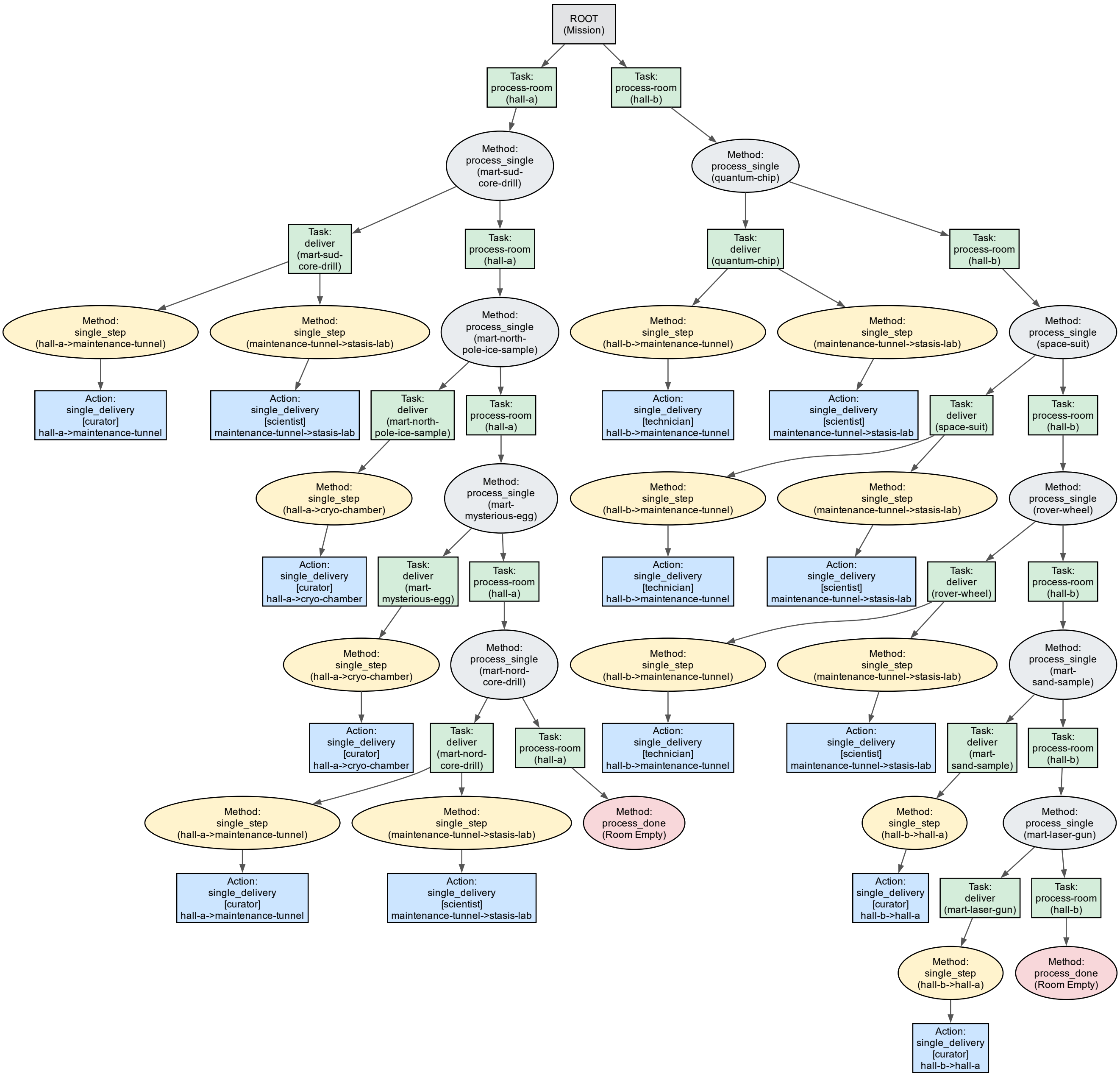

ROOT (Mission)
Task: process-room (hall-a)
Task: process-room (hall-b)
Method: process_single (mart-sud-core-drill)
Method: process_single (quantum-chip)
Task: deliver (mart-sud-core-drill)
Task: process-room (hall-a)
Task: deliver (quantum-chip)
Task: process-room (hall-b)
Method: single_step (hall-a->maintenance-tunnel)
Method: single_step (maintenance-tunnel->stasis-lab)
Method: process_single (mart-north-pole-ice-sample)
Method: single_step (hall-b->maintenance-tunnel)
Method: single_step (maintenance-tunnel->stasis-lab)
Method: process_single (space-suit)
Action: single_delivery [curator] hall-a->maintenance-tunnel
Action: single_delivery [scientist] maintenance-tunnel->stasis-lab
Task: deliver (mart-north-pole-ice-sample)
Task: process-room (hall-a)
Action: single_delivery [technician] hall-b->maintenance-tunnel
Action: single_delivery [scientist] maintenance-tunnel->stasis-lab
Task: deliver (space-suit)
Task: process-room (hall-b)
Method: single_step (hall-a->cryo-chamber)
Method: process_single (mart-mysterious-egg)
Method: single_step (hall-b->maintenance-tunnel)
Method: single_step (maintenance-tunnel->stasis-lab)
Method: process_single (rover-wheel)
Action: single_delivery [curator] hall-a->cryo-chamber
Task: deliver (mart-mysterious-egg)
Task: process-room (hall-a)
Action: single_delivery [technician] hall-b->maintenance-tunnel
Action: single_delivery [scientist] maintenance-tunnel->stasis-lab
Task: deliver (rover-wheel)
Task: process-room (hall-b)
Method: single_step (hall-a->cryo-chamber)
Method: process_single (mart-nord-core-drill)
Method: single_step (hall-b->maintenance-tunnel)
Method: single_step (maintenance-tunnel->stasis-lab)
Method: process_single (mart-sand-sample)
Action: single_delivery [curator] hall-a->cryo-chamber
Task: deliver (mart-nord-core-drill)
Task: process-room (hall-a)
Action: single_delivery [technician] hall-b->maintenance-tunnel
Action: single_delivery [scientist] maintenance-tunnel->stasis-lab
Task: deliver (mart-sand-sample)
Task: process-room (hall-b)
Method: single_step (hall-a->maintenance-tunnel)
Method: single_step (maintenance-tunnel->stasis-lab)
Method: process_done (Room Empty)
Method: single_step (hall-b->hall-a)
Method: process_single (mart-laser-gun)
Action: single_delivery [curator] hall-a->maintenance-tunnel
Action: single_delivery [scientist] maintenance-tunnel->stasis-lab
Action: single_delivery [curator] hall-b->hall-a
Task: deliver (mart-laser-gun)
Task: process-room (hall-b)
Method: single_step (hall-b->hall-a)
Method: process_done (Room Empty)
Action: single_delivery [curator] hall-b->hall-a

Figure 3.2: Delivery Task 1, splitted in two to help visualization

Observations on Planner Behavior

During evaluation with the PANDA planner, the `double_delivery` action was never instantiated, even using the optimal A algorithm. While `m_double_deliver_direct` is pruned by the nature of the problem, we expected the planner to exploit `m_double_deliver_relay`. However, tests with heuristics like `tdg-m` and `number-of-plan-steps` revealed a systemic bias: they often penalize methods that generate more subtasks. Consequently, the three-subtask decomposition of `m_double_deliver_relay` is evaluated as costlier than separate single deliveries, entirely ignoring the actual reduction in physical execution steps.

Chapter 4

Problem 4 - Temporal Planning and Timed Initial Literals

In this phase of the project, the objective was to evolve the multi-agent system from a classical planning model (based on discrete states and instantaneous actions) toward a **Temporal Planning** model using PDDL 2.1.

4.1 From Probability to Temporal Determinism

The most significant shift from the previous model concerns the management of *Hall B*'s instability. In the classical model, uncertainty was handled via the non-deterministic action `try-to-enter-seismic-room`. In the temporal version, we moved to a deterministic approach based on **Timed Initial Literals (TILs)**.

4.1.1 Modeling Earthquakes with TILs

Instead of checking for safety at the moment of entry as a random outcome, safety is now a function of time. In the problem file, we explicitly defined "safety windows" using TILs:

```
;; Hall B starts as safe
(is-safe hall-b)

;; Earthquake Window 1: the room becomes unsafe between t=10 and t=31
(at 10 (not (is-safe hall-b)))
(at 31 (is-safe hall-b))

;; Earthquake Window 2: the room becomes unsafe between t=52 and t=67
(at 52 (not (is-safe hall-b)))
(at 67 (is-safe hall-b))
```

This approach allowed us to simplify the domain by removing the `is-seismic` predicate. The planner (OPTIC [1]) can now anticipate these events and schedule movements only when the room is guaranteed to remain safe for the entire duration of the action.

4.2 Transition to Durative Actions and Technical Challenges

The transition required transforming standard `:action` definitions into `:durative-action`. This evolution involved defining explicit durations, temporal qualifiers, and overcoming specific solver limitations.

4.2.1 Predicate Duality: Handling Negation in OPTIC

A significant modeling challenge arose from the fact that **OPTIC does not support the `not` operator within action conditions**, unlike other solvers such as PRP. This limitation required a re-modeling of artifact properties using **dual predicates**. For example, to define the pick-up of a non-fragile artifact, we could no longer use `(not (fragile ?a))`. Instead, we introduced the explicit predicate `(no-fragile ?a)`. By manually maintaining both states, we ensure the solver can evaluate conditions using only positive literals, which is essential for compatibility with OPTIC's temporal reasoning engine.

4.2.2 Action Merging for Search Optimization

Temporal search is significantly more complex than classical search. To prune the state space and avoid redundant transitions, we optimized the cooling process by merging repetitive sequences. In previous versions, a robot had to release an item in the Cryo-chamber, wait, and then pick it up again. We merged these into single composite durative actions:

- `cool-artifact-while-carrying`
- `cool-artifact-while-carrying-in-pod`

This optimization ensures the robot remains committed to the artifact during the cooling duration, reducing the branching factor and preventing the planner from wasting resources on "release-and-re-pick" cycles.

4.2.3 Temporal Constraints and Qualifiers

We assigned fixed durations to all tasks (e.g., 10 units for movement, 2 units for pressurization). The selection of temporal qualifiers was critical:

- `at start`: Used for conditions that must be true at the beginning (e.g., robot starting position).
- `over all`: Used to ensure a condition persists throughout the execution, such as `(over all (is-safe ?to))` during movement to *Hall B*.
- `at end`: Used for final effects, such as the robot's arrival at a destination.

4.2.4 Modeling Assumptions for Efficiency

Since `over all` conditions are computationally expensive, we used `at start` for all "static" properties (e.g., room pressurization or robot capabilities).

Secondly, we removed the `deactivate-sealing` action. Instead, we implemented an automatic deactivation: once a robot enters a pressurized room (e.g., Hall A/B, Stasis Lab), the **sealing mode is automatically deactivated** at the end of the move action. This structural simplification further decreased the search space size.

4.2.5 Heuristic Tuning and Solver Configuration

To manage the computational overhead, the OPTIC planner was executed with specific flags: `-N -E -W4,1`. This represents a shift toward **Satisficing Planning**:

- **Weighted A Search** (`-W4,1`): By assigning a weight of $W = 4$ to the heuristic, we prioritized finding a feasible goal state quickly over absolute makespan minimality.
- **Exploration Strategy** (`-N -E`): These flags allow the solver to bypass local minima more effectively in the temporal state space.

4.3 Experimental Observations and Visualization

This section analyzes the system's performance, which currently handles problems with up to 6–7 artifacts.

4.3.1 Additional attempts to optimize search

We tested specialized pick-up actions for each artifact type (e.g., `pick-up-scientific`). However, this increased the **branching factor** by proliferating the number of available actions, which actually worsened resolution time. This confirmed that the main bottleneck is temporal coordination, not action selection. Moreover, we tried also to remove the automatic disabling of the sealing mode giving the possibility to have sealing mode always on for the whole problem, but having as a result a significant slow down in finding the solution.

4.4 Drone Integration in the Temporal Domain

The Drone scenario was also implemented within the temporal framework to evaluate its impact on concurrent plan generation. This required a slight modification of the base temporal model: the `is_seismic` predicate (which had been removed in the TIL-based version) was re-introduced, and the aerial-specific capability `can_fly` was added. Crucially, the entry into hazardous zones was modeled as a `durative-action` named `fly_into_seismic_room`.

In this version, while ground robots are constrained by the safety windows dictated by the environment (or restricted from Hall B entirely), the Drone utilizes its specific durative action to operate independently of seismic floor conditions.

4.5 Plan Evaluation and Comparative Analysis

4.5.1 Base implementation

To evaluate the generated temporal plans, we extended our custom visualization tool (Figures 4.1 and 4.2) to support **concurrent durative actions**. This allows for immediate visual confirmation that robots respect the seismic windows defined by the TILs.

The generated plan exhibits a high degree of **concurrent execution**, where the robots operate as a synchronized system rather than independent agents. By analyzing actions we can observe the same **"relay strategy"** in the *Maintenance Tunnel* of Problem 2: the *Curator* and the *Technician* focus on retrieving items from Halls, while the *Scientist* waits at the tunnel to take over the delivery, with the difference that the two robot typically enters in different rooms at the same time (this depends also on how the artifacts are arranged and which is their type). This specialization ensures that robots with higher clearance or specific capabilities remain positioned where they are most effective, minimizing idle time.

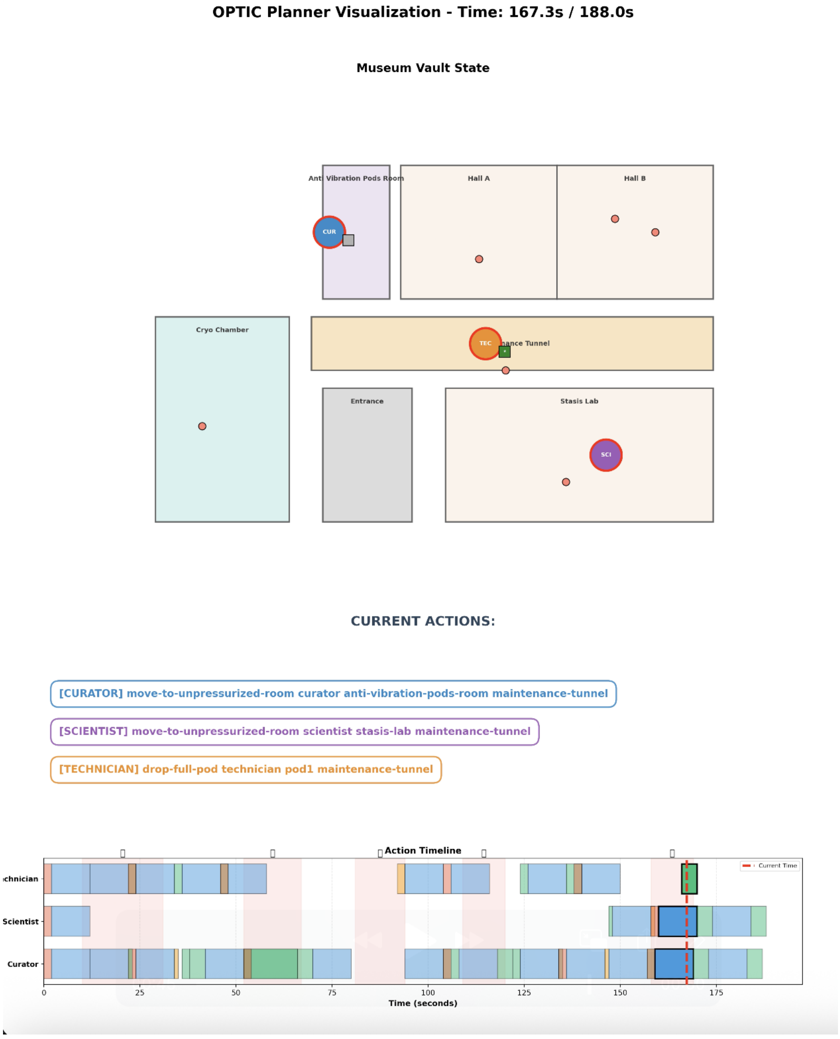
This specialization ensures that the goal—storing technological/top-secret items in Hall A and all others in their designated storage—is reached with significantly higher efficiency. A graphical representation with the temporal diagram is shown in Figure 4.3.

4.5.2 Performance Anomalies and Heuristic Bottlenecks

Experimental results using the **POPF** (Coles et al., 2010) [2] planner (which will be used in Problem 5) revealed a specific performance anomaly regarding artifact types. While the planner effectively manages high-volume goals involving up to 8 `scientific` or `top-secret` artifacts, the introduction of `technological` artifacts causes an exponential spike in search complexity.

Even a goal involving a single technological artifact significantly increases computation time, this means that handling technological elements leads for some reason to a much higher number of states to check during search, probably related to both the design of this solution and, maybe, the planner itself.

By contrast, removing this class of artifacts the artifact pipeline (Drone to Tunnel, Tunnel to Lab/Cryo) is more linear and partitioned, allowing POPF to converge on a solution in significantly shorter timeframes.



OPTIC Planner Visualization - Time: 167.3s / 188.0s

Museum Vault State

CURRENT ACTIONS:

[CURATOR] move-to-unpressurized-room curator anti-vibration-pods-room maintenance-tunnel

[SCIENTIST] move-to-unpressurized-room scientist stasis-lab maintenance-tunnel

[TECHNICIAN] drop-full-pod technician pod1 maintenance-tunnel

Figure 4.1: Integrated visualization: synchronized world state and timeline.

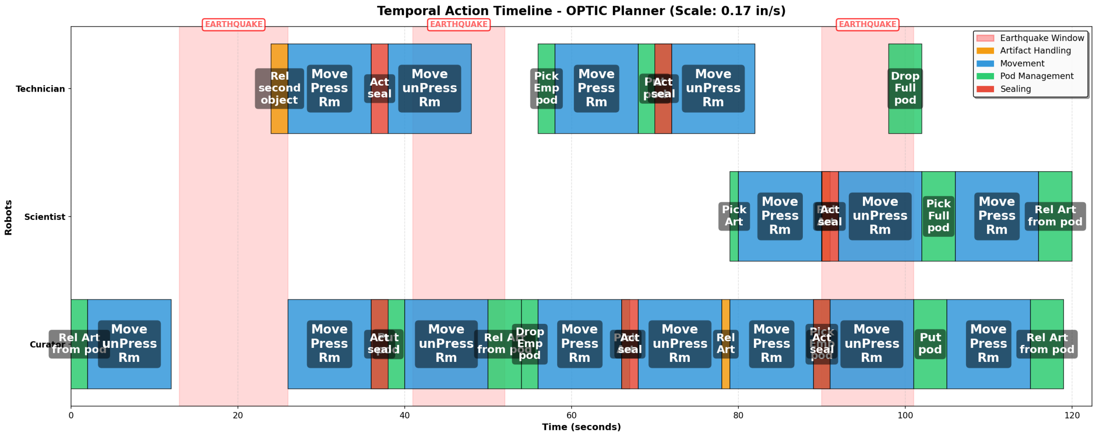
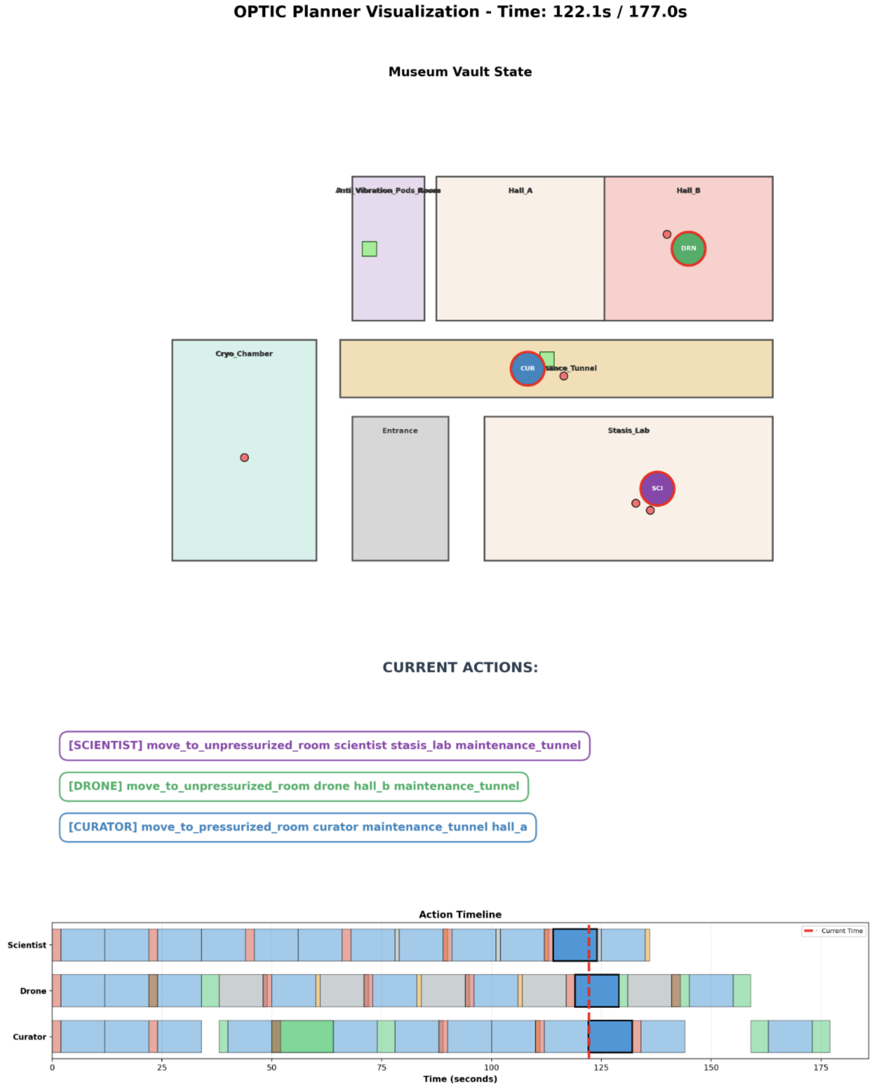


Figure 4.2: Detailed GANTT chart. Vertical light red sections indicate seismic events (TILs), while horizontal bars represent concurrent robot actions.



[SCIENTIST] move_to_unpressurized_room scientist stasis_lab maintenance_tunnel

[DRONE] move_to_unpressurized_room drone hall_b maintenance_tunnel

[CURATOR] move_to_pressurized_room curator maintenance_tunnel hall_a

Figure 4.3: Integrated visualization of drone implementation: synchronized world state and timeline.

4.6 Temporal Plan Execution and Annotated Trace

Temporal Plan Actions (Timestamp: Action)	**Logical Commentary**

`0.001: (activate-seal scientist) [2.000]`
`0.001: (activate-seal technician) [2.000]`
`2.002: (move-to-unpressurized-room scientist stasis-lab maintenance-tunnel) [10.000]`
	Phase 1: Time-Bound Initialization. Actions now have an explicit duration (e.g., `[10.000]` units for movement). Robots synchronize seals immediately to maximize the utility of the first safety window.

`31.001: (move-to-pressurized-room technician maintenance-tunnel hall-b) [10.000]`
`41.002: (pick-up-slot-1 rusty-lightsaber technological hall-b technician) [2.000]`
`43.003: (pick-up-slot-2 space-suit technological hall-b technician) [2.000]`
	Phase 2: Exploit Safety Windows. The technician enters Hall B at `t=31.001`, precisely when the first earthquake TIL ends (`is-safe` becomes true). It utilizes its dual-slot capacity to retrieve both `technological` items before the next seismic event.

`65.005: (release-artifact-cryo-slot-1 technician rusty-lightsaber cryo-chamber) [4.000]`
`69.006: (cool-artifact-while-carrying-slot-1 technician ... ) [2.000]`
`71.007: (pick-up-slot-1 mart-nord-core-drill scientific hall-a curator) [2.000]`
	Phase 3: Concurrent Operations. The plan interleaves the cooling process in the `cryo-chamber` with the retrieval of Martian cores by the `curator`. This concurrency is a key feature of the OPTIC solver to minimize total makespan.

`229.006: (pick-up-empty-pod-slot-1 curator hall-b pod1) [2.000]`
`231.007: (put-in-pod-slot-1 mart-sand-sample scientific hall-b curator pod1) [4.000]`
`269.009: (release-artifact-slot-1 scientist mart-nord-core-drill stasis-lab) [4.000]`
`; ... [Plan finalized for 6 unique artifacts]`
	Phase 4: Goal Stabilization. The trace ends with the delivery of all 6 artifacts (Cores, Sand Samples, Lightsaber, Suit). The scientist and curator coordinate the final staging in the `stasis-lab` while respecting the single-access constraints of the tunnel.

Observations: The shift to PDDL 2.1 highlights the efficiency of the temporal model. By replacing non-deterministic "tries" with deterministic TILs, the planner found a schedule that overlaps robot movements. The total makespan (~280 units) is optimized by ensuring that no robot is idling during the cooling phases of the Martian cores.

Chapter 5

Problem 5 - PlanSys2 Framework Implementation

In this final phase, the objective was to implement **Problem 4** using the **PlanSys2** framework within the **ROS 2** ecosystem. PlanSys2 provides a managed environment for PDDL-based planning where actions are executed as independent ROS 2 nodes. Our implementation followed the foundational architecture described by **Francisco Martín et al.** in their reference example for PlanSys2 [4], adapting it to our specific museum and stasis lab domain.

5.1 System Architecture and Action Implementation

To bridge the gap between high-level planning and execution, we implemented a series of **"fake actions"** in C++. These actions emulate the behavior of the robot's actuators and sensors within the PlanSys2 lifecycle.

Action Nodes: Each durative action defined in the PDDL domain corresponds to a dedicated ROS 2 node that manages the *start*, *execute*, and *end* phases of the task.

Unified Domain: We expanded the `domain.pddl` to include all predicates and durative actions required for both ground robots and drone operations. This consolidated approach allows the planner to handle heterogeneous agents within a single planning request.

5.2 Modeling Constraints: From OPTIC to POPF

A significant departure from the implementation in Problem 4 was the handling of environmental triggers. In the previous version, we utilized the **OPTIC** planner, which supports `:timed-initial-literals` (TILs) to model specific time windows for earthquakes.

However, PlanSys2 integrates **POPF** as its default temporal planner. Since integrating OPTIC into the PlanSys2 stack is notoriously labor-intensive, we adopted a **"lazy" modeling strategy** similar to the one used with the PRP planner in Problems 1 and 2.

The "Always Safe" Assumption: To circumvent the lack of TIL support in POPF, we modeled Hall B as being permanently safe. This effectively skips the mandatory waiting periods during seismic events, assuming the robot only operates when the room is safe. While this simplifies the reality where other actions could occur during an earthquake, it serves as a pragmatic solution to the structural limitations of the default planner while maintaining plan validity.

For the **drone scenario**, the implementation remains identical to the logic presented in **Problem 4**, as that problem was already optimized for POPF-compatible temporal planning.

5.3 Observations and Results

The transition to a real-time execution framework revealed several key performance and behavioral insights:

Planning Efficiency: PlanSys2 demonstrated significantly higher efficiency in generating plans compared to the Planutils environment. The time required to find a valid solution was noticeably reduced.

Plan Optimality: The generated plans are shorter than those seen in Problem 2.4. This is a direct consequence of the "Always Safe" assumption, which removes the temporal bottlenecks caused by seismic windows.

Action Concurrency: To support parallel execution—such as multiple robots moving or picking up artifacts simultaneously—it was necessary to instantiate multiple nodes for the same action. We registered up to **three instances** of each critical action (corresponding to the number of robots) to ensure the PlanSys2 Action Executor could trigger concurrent tasks without resource deadlocks.

Execution Synchronicity and Knowledge Base Latency: During the execution phase via the `run` command, a critical issue was observed regarding the synchronization of the *Knowledge Base*. Although the plan generated by the `get plan` command is formally correct from a PDDL standpoint, the system occasionally enters an `ERROR` state during actual execution.

This behavior appears to be caused by a latency in updating `at end` predicates (such as `robot_at`). The state is not registered quickly enough to satisfy the `over all` requirements of the subsequent action starting at the same timestamp. This misalignment between the theoretical plan and the actual Knowledge Base state suggests a limitation in real-time state management during concurrent execution within PlanSys2 and could be related by the parameters set in the fake actions.

5.4 PlanSys2 Execution Trace: From Initialization to Execution

This section illustrates the three main phases of the PlanSys2 lifecycle: system initialization via the Knowledge Base, plan generation, and the real-time execution of durative actions.

PlanSys2 Terminal / Logs	**Logical Commentary**

`> set instance curator robot`
`> set instance technician robot`
`> set instance mart_nord_core_drill artifact`
`> set predicate (robot_at curator entrance)`
`> set predicate (can_pickup technician technological)`
`> set goal (and (artifact_at mart_nord_core_drill stasis_lab))`
	Phase 1: Initialization. Instances and predicates are injected into the PlanSys2 Knowledge Base. This defines the initial state of the world and the goals that the planner must satisfy.

PlanSys2 Terminal / Logs	Logical Commentary
> get plan 0.001: (activate_seal curator) [2.000] 2.002: (move_to_unpressurized_room curator ...) [10.000] 12.003: (pick_up_slot_1 mart_nord_core_drill ...) [2.000]	**Phase 2: Plan Generation.** Once the goal is set, the POPF planner is invoked. It generates a temporal plan with timestamps and durations, coordinating the multiple agents.
[INFO] [executor]: (activate_seal curator) 0% [INFO] [executor]: (activate_seal curator) 100% [INFO] [executor]: (move_to_unpressurized_room) 0% [INFO] [executor]: (move_to_unpressurized_room) 50% [SUCCESS] Plan executed successfully	**Phase 3: Final Execution.** The PlanSys2 Executor dispatches the actions to the ROS 2 action nodes. Each node emulates the execution (from 0% to 100%). The successful completion of the plan confirms the goal achievement in the simulation.

Observations: The trace highlights the transition from a static PDDL description to a dynamic ROS 2 execution. The use of multiple action instances allowed the executor to handle concurrent robot movements without resource conflicts. Even if minor synchronization errors might occur in the Knowledge Base updates between actions, the overall logic ensures that the preservation mission is completed successfully.

Bibliography

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

[2] Andrew Coles, Amanda Coles, Maria Fox, and Derek Long. Forward-chaining partial-order planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 20, pages 42–49, May 2010.

[3] Matteo Grisenti. Inter-planet-museum. `https://github.com/matteogrisenti/Inter-Planet-Museumvault`, 2026.

[4] Francisco Martín and colleagues. PlanSys2 Tutorials. `https://plansys2.github.io/tutorials/index.html`, 2022. Accessed: 2024-05-20. See page 2 for reference implementation.

[5] Christian Muise. PRP: Planner for Relevant Policies, 2012. Disponibile su GitHub: `https://github.com/QuMuLab/planner-for-relevant-policies/blob/master/src/prp`.

[6] Christian Muise, Sheila A. McIlraith, and J. Christopher Beck. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.