Automated Planning - IMV

Martina Panini
martina.panini@studenti.unitn.it
257851

January 7, 2026

Contents

Chapter 1

Introduction

1.1 Scenario Description

This project addresses the modeling and resolution of automated planning problems within the context of the *Interplanetary Museum Vault* (IMV). The vault is designed to store fragile artifacts from space missions, asteroid excavations, and geological samples.

The operational scenario is driven by a sudden streak of micro-quakes and pressure fluctuations that threaten the integrity of these high-value items. To mitigate this risk, autonomous robotic curators are tasked with executing a series of stabilization and relocation operations. The environment is divided into specific zones, including Artifact Halls (α and β), a Cryo-Chamber, a Maintenance Tunnel, and a Stasis-Lab, each presenting unique constraints such as seismic instability or temperature requirements.

1.2 Objectives

The primary objective of the robotic agents is to secure the artifacts by moving them to safe locations while managing limited resources (e.g., anti-vibration pods) and navigating dynamic environmental hazards. Specifically, the goals are:

- Retrieve artifacts from Hall β, which is subject to periodic seismic instability.
- Secure fragile items inside Anti-Vibration Pods before transport.
- Transfer items from Hall α to the Cryo-Chamber, maintaining specific temperature thresholds.
- Relocate Martian Core Samples to the Stasis-Lab.

1.3 General Assumptions and Design Choices

While the assignment specifications provide the structural framework, several design and implementation choices were made to define the domain logic. The following assumptions apply across all problem instances:

Artifact Destinations

Based on the preservation requirements, specific destinations were assigned to each artifact type:

- Artifact Alpha: The final destination is the Cryo-Chamber. This aligns with the requirement to maintain sub-zero temperatures for temperature-sensitive items found in Hall α.
- Artifact Beta and Mars Core: These items are transported to the Stasis-Lab. As per the specifications, the Stasis-Lab is designated as the final safe destination for all artifacts equipped with advanced preservation technology.

Safe Transfer Protocol

To enforce the requirement that fragile items (specifically from Hall β) must be placed in Anti-Vibration Pods before transport, a predicate-based restriction was implemented:

- Implementation: The `move-carrying` action strictly requires the carried object to possess the `transportable` predicate.
- Logic: `Artifact_beta` is initialized without the `transportable` predicate, whereas `pods` are marked as `transportable`. Consequently, the robot can physically execute a `pick-up` action on the artifact, but it enters a "dead-end" state where movement is impossible. The planner is thus forced to identify the only valid sequence: loading the non-transportable artifact into a transportable pod to enable transit.

Temperature Control Strategy

For `Artifact_alpha`, strictly maintaining the temperature threshold required different modeling approaches depending on the planner's capabilities:

- Sequential (Classical): In non-temporal domains, the robot must perform a distinct `cool-down` action before the transport begins to satisfy preconditions.
- Concurrent (Temporal): In temporal domains (Problem 4), cooling is modeled as a durative action or invariant that must hold during the transport, or must be refreshed if the transport duration exceeds thermal limits.

Robot Behavior and Cyclic Readiness

A boundary condition was added requiring the robotic curator to return to the Entrance after completing all delivery tasks. Although not strictly required for artifact safety, this assumption ensures the system resets to a "ready state," allowing the robot to accept subsequent missions or recharge without obstructing vital vault corridors.

Seismic Activity Simulation

The handling of the unstable Hall β varies by problem type to maintain solver compatibility:

- Static Simulation: For classical planning (Problems 1-3), seismic activity is simulated via static variables or separate simulation runs. This abstraction avoids the complexity of timed events in non-temporal planners.
- Dynamic Windows: For temporal planning (Problem 4), seismic activity is modeled using Timed Initial Literals (TILs), creating dynamic windows of accessibility.

Chapter 2

Problem 1: Classical Planning

This chapter details the specific implementation and solution for the first scenario, building upon the general assumptions outlined in Section 1.1. The focus here is on modeling the domain using classical PDDL strict semantics without temporal features.

2.1 Domain Implementation

The domain was designed to enforce the constraints through strict preconditions rather than soft constraints or temporal penalties. The core logic relies on the interaction between the robot's state and the item properties defined in the type hierarchy.

2.1.1 Temperature Management

For Problem 1, which does not yet support concurrency, the cooling constraint is modeled as a binary state. The `move-carrying` precondition `(or (not (needs-cooling ?i)) (cooled ?i))` enforces a strict sequential order: the `cool-down` action must precede any movement of `art_alpha`.

2.2 Problem Instance and Execution

The problem was encoded in `problem_1.pddl`, reflecting the topology and initial distribution of artifacts and pods. The seismic activity constraint was handled via the static predicate `(not (seismic-active ?l))` in the interaction actions (`pick-up`, `load/unload`), preventing operations in endangered halls.

2.2.1 Solver and Metrics

The problem was solved using the `lama-first` planner via the `planutils` library. This planner was selected for its efficiency in finding satisficing plans in classical domains.

- Planner: LAMA First
- Plan Length: 27 steps

The solution confirms that the boolean preconditions effectively model the safety constraints without the need for numeric fluents or temporal operators at this stage.

2.3 Seismic Activity case

The assignment specifications introduce a dynamic environmental constraint: Hall β is subject to periodic seismic instability and is accessible only outside seismic windows. Separate files of domain and problem are created to simulate the case in which there is seismic activity.

The Static World Assumption

A fundamental challenge in modeling this constraint for Problem 1 is the nature of classical PDDL (STRIPS). Classical planning operates under the Static World Assumption, where the environment changes only as a direct result of the agent's actions. Time does not progress autonomously, and external events (such as the start or end of an earthquake) cannot occur spontaneously in the state space.

Consequently, a literal representation of a time-varying seismic window is impossible without temporal planning features. If the state were initialized with `(seismic-active hall_b)`, and no action existed to modify this predicate, the planner would encounter a dead-end, rendering the goal unreachable.

The "Wait" abstraction

To reconcile the dynamic requirement with the static limitations of Problem 1, I've introduced a modeling abstraction: the explicit wait action.

A predicate `(seismic-active ?l - location)` si defined to mark unsafe zones in the initial state. To simulate the passage of time required for the seismic window to close, was implemented a specific repair action, `wait-for-stability`:

```
(:action wait-for-stability
    :parameters (?l - location)
    :precondition (seismic-active ?l)
    :effect (not (seismic-active ?l))
)
```

While this action appears to give the agent control over the seismic event, logically it represents the agent's ability to sense the environment and delay operations.

- Precondition: The agent identifies that the location is currently unsafe.
- Effect: The agent yields execution time until the environment stabilizes (represented by the negation of the active status).

This approach allows the planner to find a valid solution by paying a unit cost (step) to wait the instability, thereby satisfying the assignment's safety constraint without violating the syntax of classical PDDL.

Chapter 3

Problem 2: Multi-Agent & Optimal Planning

Building upon the first scenario, Problem 2 introduces a multi-agent environment with constrained capacities. This chapter details the transition from satisficing to optimal planning and the necessary domain restructuring required to support strict optimal solvers.

3.1 Scenario Extensions

The second scenario extends the operational requirements by introducing:

- Heterogeneous Agents: The introduction of a Drone alongside the Robotic Curator.
- Capacity Constraints: Agents can carry multiple items, but are limited by a maximum capacity defined by discrete counters (n_0, n_1, n_2).

3.2 Domain Adaptation for Optimal Solvers

A critical trade-off emerged during the modeling phase between expressive power (using ADL features) and solver compatibility. Initially, the domain utilized Advanced Domain Definition Language (ADL) features to maintain compact code, specifically Universal Preconditions to check temperature status and Conditional Effects to handle the side effects of picking up different item types within a single action.

However, the target planner for this problem was Fast Downward configured with the `seq-opt-lmcut` heuristic. During initial testing, the planner failed to parse the ADL domain.

To resolve this and achieve the optimal solution, the domain was converted into a strict STRIPS format.

3.2.1 Action Splitting

Complex interactions were decomposed into mutually exclusive atomic operators based on the item type. The generic `pick-up` action was split into three distinct actions to handle side effects explicitly:

1. `pick-up-standard`: Applies to items that are transportable and already safe (cooled or not requiring cooling). It updates the robot's load without blocking movement.

2. `pick-up-hot`: Applies specifically to items satisfying `(needs-cooling ?i)` but `(not (cooled ?i))`. This action adds a specific effect `(temp-blocked ?a)`, representing the thermal danger.
3. `pick-up-fragile`: Applies to non-transportable items (like `art_beta`). It requires the agent to hold an empty pod and adds the `(blocked ?a)` effect.

3.2.2 Explicit State Blocking

Instead of evaluating complex logic during movement, I've introduced explicit blocking predicates to control the agent's finite state machine. The movement action `move-carrying` is guarded by the preconditions `(not (blocked ?a))` and `(not (temp-blocked ?a))`.

- Thermal Block: If an agent executes `pick-up-hot`, they become `temp-blocked`. This explicitly forces the planner to schedule a `cool-down` action immediately (which removes the predicate) before any movement can occur.
- Physical Block: If an agent executes `pick-up-fragile`, they become `blocked`. The only way to remove this block is to execute `load-pod`, ensuring the fragile artifact is never transported exposed.

3.2.3 Numeric Logic via Predicates

Since optimal classical planners often struggle with numeric fluents, capacity was modeled using a linked-list approach. Was defined `count` objects (n_0, n_1, n_2) and static predicates `(next ?c1 ?c2)`. Preconditions check `(capacity-ok ?a ?current-load)` and effects transition the state from `(load ?a ?n1)` to `(load ?a ?n2)`, effectively implementing a counter entirely within propositional logic.

3.3 Problem Solution and Analysis

The problem was solved using Fast Downward with the `seq-opt-lmcut` heuristic. Fast Downward was selected over LAMA to guarantee optimality. While LAMA found a solution in $0.3s$, it produced a suboptimal plan of 45 steps with redundant moves. Downward required more computation time (approx. $148.83s$) but converged on a significantly more efficient solution of size 29 (Unit Cost).

3.3.1 Execution Analysis

The resulting plan demonstrates the efficiency of the STRIPS modeling and the optimizer's behavior:

1. Single-Agent Efficiency: The planner assigned all tasks to `drone1`. In an optimal sequential plan where agents have identical movement costs, the solver minimizes the search depth by avoiding unnecessary context switching between agents.
2. Pod Recycling: A clear example of optimality is observed in the handling of pods. The drone retrieves `pod1`, uses it to transport `art_beta` to the Stasis Lab, and then immediately reuses the return trip to bring the same pod back to the Pods Room.
3. Constraint Compliance: The split-action logic successfully enforced that `art_alpha` was cooled immediately upon pickup at Hall A before any transport occurred.

This restructuring increased the domain's verbosity compared to Problem 1 but successfully eliminated all axioms and conditional effects, allowing the `lmcut` heuristic to prune the search space effectively and guarantee the global minimum cost.

Chapter 4

Problem 3: Hierarchical Task Network (HTN) Planning

In the third phase of the project, the planning paradigm shifts from a Goal-Based approach (Classical Planning) to a Task-Based approach (Hierarchical Task Network - HTN).

The objective is to execute a high-level plan consisting of three specific tasks: delivering a fragile artifact, delivering a temperature-sensitive artifact, and delivering a standard artifact, enforcing specific ordering constraints where necessary.

4.1 Design Choices and Modeling

The domain was modeled using the HDDL (Hierarchical Domain Definition Language) standard and solved using the Panda planner. The structure separates primitive actions from high-level methods.

4.1.1 Hierarchical Structure

The domain is defined by predicates such as `needs-cooling` and `cooled`, and a set of primitive actions including `move-carrying`, `load-pod`, and `cool-down`.

4.1.2 Method Implementation

The core logic is encapsulated in the decomposition methods, which act as the procedural knowledge of the agents.

A key design choice was handling the `deliver_standard` task using two distinct methods based on the artifact's properties:

1. Normal Delivery (`m_deliver_std`): This method applies when the condition `(not (needs-cooling ?item))` holds. It generates a standard sequence: *Get To* $\rightarrow$ *Pick Up* $\rightarrow$ *Move* $\rightarrow$ *Drop*.
2. Cooled Delivery (`m_deliver_std_cooled`): This method applies when `(needs-cooling ?item)` is true. It enforces a safety step, injecting the `cool-down` action immediately after the pick-up and before any movement occurs. This ensures the constraint is handled at the procedural level rather than via global constraints.

The Fragile Protocol (`m_deliver_frag`) For artifacts deemed non-transportable without protection (like `art_beta`), this method enforces a strict 14-step subtask sequence. The agent must acquire a specific pod, transport it to the artifact, load the artifact (changing its state to `inside`), deliver it to the destination, unload it, and finally return the used pod to its home location (`pod_home`).

Recursive Navigation Navigation is modeled via the `m_drive_to_via` method. Instead of relying on a pathfinding heuristic, the HTN decomposes the movement into recursive steps based on the star topology centered around the `tunnel` location.

4.2 Plan Analysis

The problem was successfully solved by the Panda planner using a progression search algorithm.

- Search Time: 0.021 seconds.
- Nodes Generated: 3,693 nodes.
- Plan Length: The final solution consists of a decomposition tree of 132 steps, resulting in a linear sequence of 31 primitive actions.

4.2.1 Plan Trace

The generated plan adheres strictly to the defined ordering constraints.

1. Drone Operations (Fragile): The `drone1` first handles the fragile `art_beta`. It retrieves `pod1`, moves to `hall_b`, loads the artifact, and delivers it to `stasis`. Crucially, the plan shows the drone immediately returning `pod1` to the `pods_room` before accepting new tasks.
2. Curator Operations (Cooling): The `curator` retrieves `art_alpha`. The log explicitly shows the execution of `cool-down[curator,art_alpha]` at `hall_a` before the robot moves to the tunnel, validating the thermal safety method.
3. Final Transport (Standard): Finally, the drone delivers `mars_core`. Since this artifact does not have the `needs-cooling` predicate, the planner selects the standard `m_deliver_std` method, skipping the cooling step for efficiency.

Chapter 5

Problem 4: Temporal Planning

5.1 Scenario Description and Objectives

The fourth problem marks the transition from classical sequential planning to Temporal Planning.

In this scenario, agents (`Curator` and `Drone`) operate in a concurrent environment where actions have specific durations reflecting real-world physics:

- `move`: 10 seconds (high-cost navigation).
- `load-pod` / `unload-pod`: 8 seconds.
- `cool-down`: 5 seconds.
- `pick-up` / `drop`: 5 seconds.

5.2 Design Choices and Modeling

To handle these requirements, the domain was rewritten using PDDL 2.1 features, specifically `:durative-actions`. The critical modeling challenge involved the definitions of temporal constraints using `at start`, `at end`, and `over all` keywords to ensure logical consistency during concurrent execution.

5.2.1 The "Cool-Down" Constraint and Robustness

A specific focus was placed on the modeling of the `cool-down` action. In the final version, this action is modeled with a strict spatial invariant:

```
(:durative-action cool-down
    :condition (and
        (over all (at ?r ?l)) ... )
)
```

This `over all` condition ensures the agent must remain stationary at the location for the entire duration of the cooling process.

During the testing phase, removing this constraint exposed significant differences in planner robustness:

- LPG-TD: Exploited the missing constraint to generate a physically impossible plan where the drone moved while cooling the artifact to save time.

- Optic: Suffered from search space explosion (infinite loops at high depth), unable to resolve the implicit interference between the unconstrained duration and movement actions.

This highlights that while Optic is more rigorous, LPG-TD is more robust but opportunistic.

5.3 Planner Comparison: LPG-TD vs. Optic

The experiment compared two distinct approaches to temporal planning: LPG-TD (Local Search based on planning graphs) and Optic (Partial Order Planning with temporal heuristics). The results highlight a clear divergence in solution quality and strategy.
The following table summarizes the performance on the corrected domain: LPG-TD gen-

Metric	**LPG-TD**	**Optic (WA*, W=5)**
Makespan	**170.00 s**	180.009 s
Search Time	19.36 s	**13.20**

Table 5.1: Performance comparison between LPG-TD and Optic.

erated a near-optimal plan by maximizing temporal parallelism. The planner split the workload cleanly. The `Drone1` was assigned the longest operation chain: cooling and transporting `art_alpha` followed by retrieving `mars_core`. Simultaneously, the `Curator` was assigned exclusively to the retrieval of `art_beta`.

This division allowed two independent timelines to advance without interference, reducing the critical path to the duration of the Drone's tasks alone.

Optic produced a suboptimal solution due to a phenomenon of capacity saturation. Instead of delegating, the planner instructed the `Curator` to pick up `mars_core` at `Cryo` (utilizing slot $n0 \rightarrow n1$), travel to the `Pods Room` to pick up `art_beta` (utilizing slot $n1 \rightarrow n2$), and finally deliver both.

With the configuration $W = 5$, Optic's heuristic likely overestimated the value of minimizing travel steps by batching transport tasks, ignoring the high temporal cost of serializing these operations. This transformed the Curator into a bottleneck, unnecessarily extending the makespan.

5.3.1 Final discussion about planners comparison

For the `IMV-TEMPORAL` domain, LPG-TD proved superior in both computational speed and solution quality. The capability of Optic to handle complex numerical constraints was not decisive in this specific scenario; conversely, its tendency to saturate agent capacity acted as a detriment to parallelism. To improve Optic's performance, it would be necessary to reduce the heuristic weight W (approaching optimal search at the cost of time) or remove the extra inventory capacity ($n2$) to force a division of labor.

5.4 Handling Dynamic Environments: Seismic Windows

Unlike Problem 1, where the environment was static, Problem 4 requires the agent to respect time-dependent constraints that open and close autonomously.

5.4.1 Modeling with Timed Initial Literals (TILs)

To represent the seismic windows, I've utilized PDDL 2.1 Timed Initial Literals (TILs). TILs allow the specification of facts that become true or false at absolute time points, independent of agent actions. Was defined a scenario where `Hall` β becomes unsafe between $T = 20$ and $T = 60$.

The "Positive Logic" Adaptation for Optic

During the implementation, I've encountered a specific limitation in the `Optic` planner regarding Negative Preconditions in Durative Actions. Defining the constraint as `(over all (not (seismic-active ?l)))` caused segmentation faults or parsing errors due to the complexity of handling negative invariants over intervals.

To resolve this, was inverted the modeling logic from "avoiding danger" to "ensuring safety". So, it is introduced the predicate `(safe ?l)` and enforced the condition `(over all (safe ?l))` in all durative actions (move, pick, drop). This formulation proved numerically stable and robust for the planner.

5.4.2 Plan Analysis

The generated plan demonstrates the Temporal Planner's capability to handle dynamic constraints not just by waiting, but by optimized task interleaving.

In the solution , the Curator is assigned to retrieve Art Beta from Hall B, which is seismically unsafe between $T = 20$ and $T = 60$. Instead of moving to Hall B early and idling (which would violate the safety invariant), the planner scheduled a logistical detour during the danger window:

- At $T = 45$, the Curator moves to the Pods Room.
- It spends the interval $T = 55$ to $T = 60$ retrieving Pod2, a necessary tool for the fragile artifact.
- The return trip to the Tunnel concludes at $T = 70$, exactly $10s$ after the seismic window has closed ($T = 60$).

This sequence ensures that the agent arrives at the entrance of Hall B ($T = 80$) only when it is safe to enter, effectively masking the "wait time" with productive preparatory actions. This confirms that the PDDL 2.1 model successfully forced the planner to reason about future availability windows, preventing unsafe access without requiring explicit "wait" actions.

Chapter 6

Problem 5: PlanSys2 Integration (ROS2)

The final phase of the project bridges the gap between abstract planning algorithms and real-world robotic control. The Temporal PDDL domain, validated in Problem 4, has been integrated into the ROS2 ecosystem using the PlanSys2 framework.

6.1 Design Choices and Modeling

6.1.1 Predicate Adaptation

A significant adaptation was required to ensure compatibility with the PlanSys2 terminal interface.

Initially, a modeling approach based on the negative goal condition `(not (needs-cooling))` was attempted. However, due to specific limitations in parsing negative goals within the PlanSys2/Terminal environment, a more robust strategy was adopted. A positive predicate `(cooled ?x)` was explicitly introduced into both the domain and the problem goal. This modification allows the planner to actively reason about the state change required for thermally sensitive artifacts without encountering parsing ambiguities.

6.2 Implementation Details

6.2.1 Action Nodes

Each PDDL action (e.g., `move`, `pick-up`, `load-pod`) was implemented as a distinct Lifecycle Node in C++. To simulate the execution of long-running tasks without a physical physics simulator, the action nodes utilize a timer-based feedback loop:

1. On Activate: The node receives the action arguments (e.g., `curator`, `entrance`, `tunnel`) and calculates the required duration based on the PDDL definition.
2. Execution Loop: The node enters a loop, publishing a feedback rate (progress from 0% to 100%) while sleeping for the designated duration to simulate the passage of time.
3. Completion: Once the time elapses, the node returns `SUCCESS`, signaling the PlanSys2 executor to proceed to the next step or trigger parallel actions.

6.2.2 Launch System

A Python launch file was developed to orchestrate the system startup. It initializes the PlanSys2 stack, loads the domain file, and spawns all required action nodes, ensuring they are discoverable on the ROS2 network.

6.3 Execution and Validation

The execution flow confirmed the temporal validity of the domain within a distributed environment:

1. Plan Generation: The planner correctly generated a temporal plan ($Cost$: 150.018), consistent with the domain constraints.
2. Parallel Dispatch: The executor successfully dispatched the `(move ...)` actions for `curator` and `drone1` simultaneously. This validated the framework's capability to handle concurrent execution as defined in the PDDL plan.
3. Constraint Satisfaction: The feedback logs confirmed that sequential constraints were respected. For instance, the robots correctly waited for the completion of `load-pod` actions before initiating subsequent movements, validating the synchronization between the abstract plan and the ROS2 node execution.

Chapter 7

Conclusions

The objective of the project was to design autonomous behaviors for robotic agents tasked with securing fragile artifacts in a hostile Martian environment. The work followed a progressive complexity curve, moving from abstract classical modeling to optimal resource management, hierarchical decomposition, temporal concurrency, and finally, integration with a distributed robotic control system.

7.1 Analysis of Problems Results

7.1.1 Classical vs. Optimal Planning (Problems 1 & 2)

In the initial phases, the trade-off between expressivity and optimality became evident.

- While ADL (Advanced Domain Definition Language) features allowed for compact domain definitions in Problem 1, the requirement for strict optimality in Problem 2 necessitated a compilation approach.
- To leverage the `seq-opt-lmcut` heuristic of Fast Downward, complex conditional effects had to be split into atomic actions (e.g., `pick-up-hot` vs. `pick-up-standard`). This demonstrated that solver compatibility often dictates domain design more than pure logical preference.
- The handling of Seismic Activity evolved significantly: from a logical abstraction (an explicit wait action) in the classical domain of Problem 1 to a physically accurate model using Timed Initial Literals (TILs) in the temporal domain of Problem 4.

7.1.2 Procedural vs. Goal-Oriented (Problem 3)

The introduction of HTN (Hierarchical Task Networks) in Problem 3 provided a robust solution for the Fragile Artifact Protocol. By decomposing the task into a strict method (Get Pod $\rightarrow$ Load $\rightarrow$ Transport $\rightarrow$ Return Pod), HTN proved superior to classical planning in enforcing strict procedural compliance, reducing the search space by guiding the solver through predefined sub-tasks rather than open exploration.

7.1.3 Temporal Concurrency (Problem 4)

Problem 4 represented the most significant modeling challenge. Transitioning to PDDL 2.1 required explicit management of physics (durations) and invariants.

- The comparison between LPG-TD and Optic highlighted a divergence in strategy: LPG-TD optimized for parallelism (splitting tasks between Drone and Curator), while Optic tended to saturate the capacity of a single agent.
- A crucial technical finding was the handling of Negative Preconditions in Optic. The planner's inability to handle negative temporal invariants over intervals required a logical inversion of the domain (from `not-seismic` to `safe`), proving that robustness often requires adapting the model to the planner's internal architecture.

7.1.4 From Theory to Deployment (Problem 5)

The final integration with PlanSys2 and ROS2 bridged the gap between theoretical planning and robotics. By encapsulating PDDL actions into C++ Lifecycle Nodes, the system demonstrated that the abstract plans generated in Problem 4 could drive a distributed fleet of agents. The successful synchronization of concurrent actions (e.g., the Drone moving while the Curator waited for a pod to load) validated the feasibility of PDDL 2.1 for real-world high-level control.

Appendix A

Execution Evidence and Logs

In this appendix, I provide visual evidence of the execution metrics and plans generated by the solvers.

A.1 Problem 4

The following chart illustrates the parallel execution of the *Curator* and *Drone1* agents as derived from the `problem_4.pddl.plan` output. It highlights the concurrency during the navigation and cooling phases.

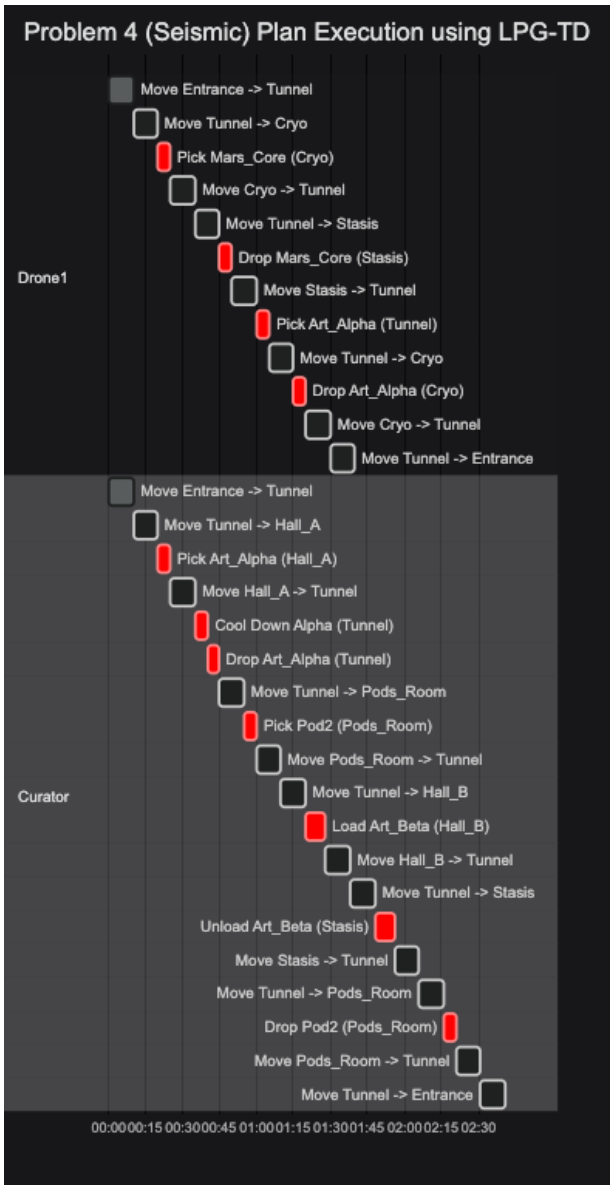
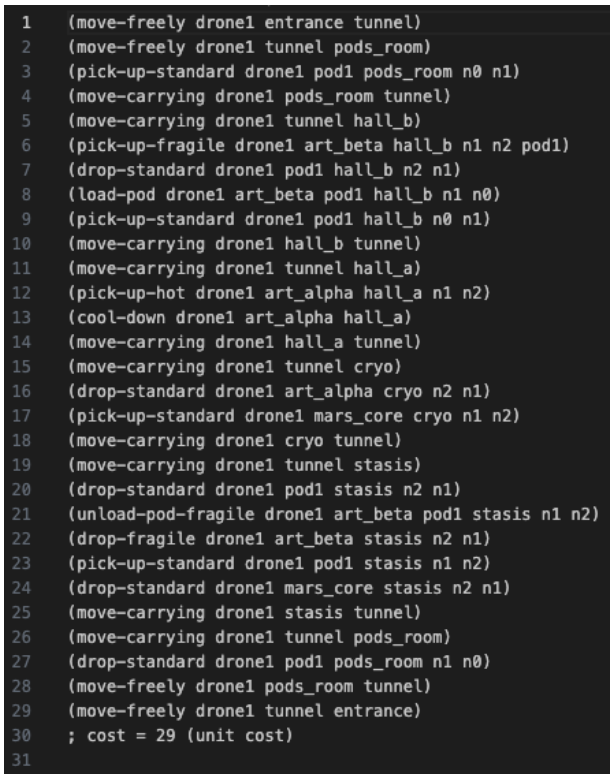


Figure A.1: Gantt Chart reconstructing the concurrent timeline from the Temporal Planner output.

A.2 Planner Output Logs

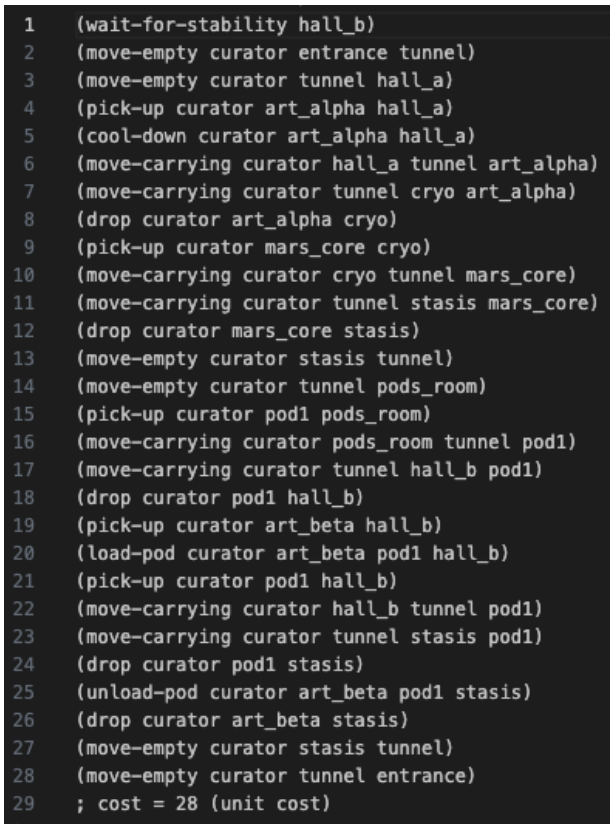
This section contains the raw output from the planners, verifying the optimal cost and step sequences.



```
(move-freely drone1 entrance tunnel)
(move-freely drone1 tunnel pods_room)
(pick-up-standard drone1 pod1 pods_room n0 n1)
(move-carrying drone1 pods_room tunnel)
(move-carrying drone1 tunnel hall_b)
(pick-up-fragile drone1 art_beta hall_b n1 n2 pod1)
(drop-standard drone1 pod1 hall_b n2 n1)
(load-pod drone1 art_beta pod1 hall_b n1 n0)
(pick-up-standard drone1 pod1 hall_b n0 n1)
(move-carrying drone1 hall_b tunnel)
(move-carrying drone1 tunnel hall_a)
(pick-up-hot drone1 art_alpha hall_a n1 n2)
(cool-down drone1 art_alpha hall_a)
(move-carrying drone1 hall_a tunnel)
(move-carrying drone1 tunnel cryo)
(drop-standard drone1 art_alpha cryo n2 n1)
(pick-up-standard drone1 mars_core cryo n1 n2)
(move-carrying drone1 cryo tunnel)
(move-carrying drone1 tunnel stasis)
(drop-standard drone1 pod1 stasis n2 n1)
(unload-pod-fragile drone1 art_beta pod1 stasis n1 n2)
(drop-fragile drone1 art_beta stasis n2 n1)
(pick-up-standard drone1 pod1 stasis n1 n2)
(drop-standard drone1 mars_core stasis n2 n1)
(move-carrying drone1 stasis tunnel)
(move-carrying drone1 tunnel pods_room)
(drop-standard drone1 pod1 pods_room n1 n0)
(move-freely drone1 pods_room tunnel)
(move-freely drone1 tunnel entrance)
; cost = 29 (unit cost)

```

Figure A.3: Raw log of problem 2 output planner



```
(wait-for-stability hall_b)
(move-empty curator entrance tunnel)
(move-empty curator tunnel hall_a)
(pick-up curator art_alpha hall_a)
(cool-down curator art_alpha hall_a)
(move-carrying curator hall_a tunnel art_alpha)
(move-carrying curator tunnel cryo art_alpha)
(drop curator art_alpha cryo)
(pick-up curator mars_core cryo)
(move-carrying curator cryo tunnel mars_core)
(move-carrying curator tunnel stasis mars_core)
(drop curator mars_core stasis)
(move-empty curator stasis tunnel)
(move-empty curator tunnel pods_room)
(pick-up curator pod1 pods_room)
(move-carrying curator pods_room tunnel pod1)
(move-carrying curator tunnel hall_b pod1)
(drop curator pod1 hall_b)
(pick-up curator art_beta hall_b)
(load-pod curator art_beta pod1 hall_b)
(pick-up curator pod1 hall_b)
(move-carrying curator hall_b tunnel pod1)
(move-carrying curator tunnel stasis pod1)
(drop curator pod1 stasis)
(unload-pod curator art_beta pod1 stasis)
(drop curator art_beta stasis)
(move-empty curator stasis tunnel)
(move-empty curator tunnel entrance)
; cost = 28 (unit cost)
```

Figure A.2: Raw log of problem 1 output planner