An Automated Warehouse Scenario

Shady Rafat Ibrahim Yousif Abdelmalek[1], Gian Marco Balia[2], Christian Negri Ravera[3], Milad Rabiei[4], and Arian Tavousi[5]

[1-5]Team 2, Artificial Intelligence For Robotics II, DIBRIS - Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genoa

Academic Year 2024–2025

Abstract

Warehouse automation through utilizing robots has gained significant attention in recent decades. Various types of robots are employed for this purpose, particularly robotic arms for local placement, e.g. pick-and-place, and mobile robots for indoor or outdoor package transportation. Besides the technical specifications one must also consider the optimality of using robots for automation, measuring overall cost and time spent versus work done. For such purposes artificial intelligence and planning has been studied and employed for automated planning of predefined tasks and processes. Real-world applications of AI Planning often require a highly expressive modeling language to accurately capture important intricacies of target systems. Hybrid systems are prevalent in the real-world, and PDDL+ serves as the standardized modeling language for capturing such systems as planning domains. PDDL+ enables accurate encoding of mixed discrete-continuous system dynamics, exogenous activities, and numerous other features exhibited in realistic scenarios [4]. This study proposes a specific domain for 5 warehouse automation problems each with a different study case; The details of which are expanded in the introduction. Part I focuses on the implementation and practical results of the solution. Later in Part II, we provide a specific heuristic for the planner, and analyze how symbolic patterns help solve the problem with a new approach.

Contents

1 Part I: Models and Design

1.1 Setup Enviroment

Before the explanation of the domain and the problem, we present how to run and test the implementation using *ENHSP*. Our solution runs on *ENHSP v19*, with the following command, and can be run locally using the binary java run file:

```
java -jar enhsp-19_path.jar -o domain_file_path.pddl -f
   problem_file_path.pddl -delta x (optional)
```

We also express that our problem cannot work with ENHSP-20, but the reason is still unknown. Also the project is available on this [GitHub link].

It is noteworthy to mention that with a time_delta of 1 crates can be accessed only if the initial positions of crates are multiples of the velocity, e.g. speed of 10. However, we can modify the time_delta to be able to generalize about this limitation, without contradicting other aspects of the project. Considering a regular domain, of course, decreasing the time_delta would result in an exploration of a larger state space in hybrid planning, and a relatively huge delay in plan generation. For the current study, time_delta has been set to both 1 and 0.5, which needed an accepted amount of time to plan.

1.2 Methodology

The mover robots navigate the nearly continuous (quantum) unidimensional inventory space. Certain procedures, such as moving or loading crates from the loading bay to the conveyor belt, require a specific time frame. Additionally, there are one-time actions with binary preconditions and effects. Considering these factors, it is evident that a hybrid planning approach combining numeric, temporal, and classic planning is employed. Durative actions, which are regular actions with a specific time commitment, were introduced from PDDL2.1. However, some general planners do not support these actions, e.g. ENHSP. Consequently, the sequence of Action/Event, Process, Action/Event is carefully crafted. Furthermore, this study aims to reduce the number of predicates and functions utilized to minimize the dimensionality of the state space and facilitate planning, thereby reducing time and resource allocation.

For clarity, some actions, processes, and events currently use "or" conditions in their definitions. While logically correct, this approach can complicate the state space for the planning solver, making it less efficient. A better solution would be to separate each "or" condition into distinct actions, processes, or events, where each new element handles only one specific condition at a time. This simplification can lead to a more manageable search space and potentially faster plan generation.

The project's gradual development strategy started from [2]. Through usage of predicates and functions, state configurations are specified.

1.3 Domain Description

The movement of movers can be specified in three states: `moving_forward`, `moving_backward`, and `stopped`. The minimum number of predicates required for these three states is two, moving and `to_positive`. Alternatively, a moving predicate can be used, and the sign of the velocity determines the direction, which is less descriptive. To address the requirements of the problem, several procedures were developed to provide a high-level overview of the basic movements the agent can perform within the environment. The main tasks are the following:

- `move_to_crate`;
- `pickup_crate`;
- `move_crate_to_loader`;
- `handover_to_loader`;
- `load_crate_onto_belt`.

Moving State	**Predicates**
Moving Forward	`moving` $:= \top$ `to_positive` $:= \top$
Moving Backward	`moving` $:= \top$ `to_positive` $:= t$
Stopped	`moving` $:= \bot$ `to_positive` $:= \{\top, bot\}$

Table 2:

1.4 Base problem description

1.4.1 Move to crate

The `move_to_crate` procedure facilitates the agent's movement from its current position to a target crate. These are the steps:

1. `Start_forward` action allows the agent to move forward;

Parameters	Preconditions	Effects
mover	- The mover is not currently moving - The mover is at position 0 - The mover is not carrying anything - The mover is free	- The mover starts moving - The movement direction is set to positive (forward)

2. `Goto_pickup` [process]

Parameters	Preconditions	Effects
mover	- The mover is moving - The direction of motion is positive (forward) - The mover is not carrying anything - The mover is free	- The position of the mover is increased at each step according to its velocity

3. `stop_at_ball` [action]

Parameters	Preconditions	Effects
mover	- The mover is moving - The direction of motion is positive (forward) - The mover's position is equal to the crate's position	- The mover stops moving
crate	- The crate is not yet loaded - The crate's position is greater than 0	

With a time step (time_delta) of 1 and a constant integer velocity (e.g., 10 units per time step), an agent can only reliably reach positions that are exact multiples of its velocity, assuming its initial position is also a multiple of the velocity or zero. Consequently, if a crate's location is not a multiple of the agent's velocity (and the starting position isn't appropriately offset), the agent, with time_delta = 1, will inevitably fall short of or overshoot the crate, never landing precisely on it.

1.4.2 Pickup crate

The core functionality for crate pickup by the robot includes considerations for the crate's weight. Recognizing that two movers can collaboratively transport any object, we established two individual actions to address this capability.

1. `pickup` [action] This action allows a single robot to pick up a crate only if all the following conditions are met:

Parameters	Preconditions	Effects
mover	- The mover is at the same position as the crate - The mover is currently not moving - The movement direction is set to positive (forward) - The mover is free	- The mover's state is updated to ”moving” - The mover starts carrying the crate - The mover is no longer free
crate	- The crate is not already loaded - The crate weighs 50 kg or less - The crate is on the ground inside the warehouse	- The crate is no longer on the ground

2. `pickup_by_two` [action] This action is used for crates heavier than 50 kg. It requires two distinct robots to pick up a crate.

Parameters	Preconditions	Effects
mover1, mover2	- The movers are distinct from each other - Both are at the same position as the crate - Both movers are not moving - Both movers are moving forward - Both movers are free and not carrying anything	- Both movers enter the ”moving” state - Both movers start carrying the crate - Both movers are no longer free
crate	- Crate is not already loaded - Crate is on the ground inside the warehouse	- The crate is no longer on the ground - The velocity coefficient flag is enabled: `coeff_set` := true

The action, after execution, sets the flag `coeff_set` := true, which will trigger the decision for the coefficient to apply for the velocity.

1.4.3 Move crate to loader

To understand how the mover returns to the loading bay after picking up a crate, we must first define a coefficient used to calculate its return velocity. This velocity will then determine the mover's movement back to the loading bay. Once a crate is grasped, the time required for this return journey is directly influenced by the crate's weight, as detailed below.

- For heavy and light crates (carried by a single mover):

$$time = \frac{weight * pos_{mover}}{100} \tag{1}$$

- For light crates carried by two movers:

$$time = \frac{weight * pos_{mover}}{150} \tag{2}$$

To express this behavior, the change in time is instead represented as a change in velocity. This is done to leverage existing information—namely, the maximum velocity of each robot—without introducing additional variables, thus avoiding increased complexity in the domain model. In essence, the movers' perception of time is encoded in their velocity computation. According to the first law of motion, we have:

$$x = v * t \tag{3}$$

Where $x$is the position of the mover,v is its velocity, and t is the time. We can isolate t and compare it with the previously defined time:

$$t = \frac{\text{pos}_{\text{mover}}}{V} = \frac{\text{weight} \times \text{pos}_{\text{mover}}}{coeff}, \quad \text{where } coeff = 100, 150 \tag{4}$$

By simplifying:

$$V = \frac{coeff}{\text{weight}} \tag{5}$$

Therefore, we can define a mapping from a time value to an equivalent velocity value. Any time update will result in a corresponding velocity update:

$$\text{time} = \frac{\text{weight} \times \text{pos}_{\text{mover}}}{100} \quad \Leftrightarrow \quad V = \frac{100}{\text{weight}}, \quad \text{for heavy and light crates} \tag{6}$$

$$\text{time} = \frac{\text{weight} \times \text{pos}_{\text{mover}}}{150} \quad \Leftrightarrow \quad V = \frac{150}{\text{weight}}, \quad \text{for light crates carried by two movers} \tag{7}$$

Below the two event in charge to compute the velocity coefficient:

1. `coeff_changer_light` [event]

Parameters	Preconditions	Effects
mover1, mover2	- The movers must be different from each other - Both movers must be carrying the crate - The flag `coeff_flag` is true	- Each mover's velocity is set to: $v = \frac{150}{\text{weight}}$
crate	- The crate weighs less than 50 kg	- The flag `coeff_flag` is set to false

2. `coeff_changer_weight` [event]

Parameters	Preconditions	Effects
mover1, mover2	- The agents are distinct from each other - Both agents must be carrying the crate - The flag `coeff_flag` is true	- Each mover's velocity is set to: $v = \frac{100}{\text{weight}}$
crate	- The crate weighs more than 50 kg	- The flag `coeff_flag` is set to false

We designed two distinct processes to handle the different carrying behaviors of a single robot based on whether the crate weighs less than or more than 50 kg.

- `backto_loader` [process]

Parameters	Preconditions	Effects
mover	- The mover is moving in a backward (negative) direction - The mover is carrying a crate, thus it is not free	- The mover's position is decreased at each time step according to its velocity: $mover_{position} -=$ `velocity(m)`
crate	- The crate weighs less than 50 kg	

- `backto_loader_by_two` [process]

Parameters	Preconditions	Effects
mover1, mover2	- Both movers are moving in a backward (negative) direction - Both movers are carrying a crate, thus they are not free - Movers' velocities are the same (to allow carrying the payload)	- Each mover's position is decreased at each time step according to its velocity: $mover1_{position} -=$ `velocity(mover1)` $mover2_{position} -=$ `velocity(mover2)`
crate	- The crate weighs less than 50 kg	

Note: the condition "greater than 50 kg" is omitted here, since even light crates can be carried by two movers

1.4.4 Handover to loader

The procedure allows the agent to hand over the carried crate at the loading bay. In our problem, we have simplified this by assuming the robot stops directly at the loading bay to transfer the crate to the loader. If the loader is currently busy, the mover will wait until the loading process is complete. To manage the varying carrying behaviors of a single robot based on the crate's weight (above or below 50 kg), we designed two distinct actions.

- `stop_handover` [event]

Parameters	Preconditions	Effects
mover	- The mover's position is less than or equal to 0 - The mover is moving - The mover's direction is backward (not positive) - The mover is carrying	- The mover stops moving - The mover's direction is set to positive - The mover is no longer carrying - The mover is set to free - The mover's position is set to 0 - The mover's velocity is set to its maximum velocity
loader	- The loader is not busy loading - The loader is free	- The loader is set to busy loading - The loader is no longer free
crate	- The crate weighs less than or equal to 50 kg - The crate is not fragile - Either the crate belongs to no group (ID=0), or it belongs to the currently active group.	- The crate's position is set to 0

A specific challenge arises when the agent returns to the loading bay. Due to the discrete time steps (initially T = 1), the ratio of the agent's position to its velocity might not be an integer. Consequently, there might not be an integer number of time steps that allows the agent to reach exactly position zero. This can lead the agent to overshoot the loading bay, causing an error in the system.

To mitigate this overshooting problem, we attempted to reduce the time_delta to 0.1 and 0.01, aiming for finer-grained time sampling. However, in both these cases, the solver required an excessively long time to find a solution due to the dramatically increased number of states that needed to be explored. While a sampling rate of T = 0.5 allowed for a more feasible planning time, it still did not guarantee the desired precision down to the second decimal place.

To effectively address the overshooting issue without the prohibitive computational cost of very small time_delta values, we implemented a condition that teleports the agent directly to the zero position if it happens to move past it during its return journey. This ensures the agent reliably reaches the loading bay without introducing errors related to the limitations of time discretization and without sacrificing computational efficiency.

- `Stop_handover_by_two` [event]

Parameters	Preconditions	Effects
mover1 mover2	- The movers are distinct from each other - Both movers' positions are less than or equal to 0 - Both movers are moving - Both movers' direction is backward (not positive) - Both movers are carrying	- The movers stop moving - The direction is set to positive for both movers - Both movers are no longer carrying ?b (crate) - Both movers are set to free - Both movers' positions are set to 0 - Both movers' velocities are updated to their maximum velocity
loader	- The loader is not busy loading	- The loader is set to busy loading
crate	- The crate weighs less than or equal to 50 kg - The crate is not fragile - The crate belongs to a group (ID ¿ 0) - The ID of the crate's group matches the currently considered group - Either no group is currently active, or the crate belongs to the currently active group.	- The crate's position is set to 0

NOTE:
The precondition ”Either no group is currently active, or the crate belongs to the currently active group” in the two actions, while logically sound, was included primarily for clarity. However, this ”or” condition can increase the complexity for the planning solver. A more efficient approach would be to decompose this precondition by creating separate actions for each case within the ”or” statement. This would result in two distinct actions: one triggered when no group is active, and another triggered when the crate belongs to the currently active group. This separation, while increasing the number of actions, can simplify the individual action definitions and potentially reduce the search space complexity for the solver.

1.4.5 Load crate onto belt

The procedure enables the loader to load the crate onto the conveyor belt after having obtained the crate from the mover

- `load` [process]

Parameters	Preconditions	Effects
loader	- The loader must be busy (it must have received the crate from a mover) - 4 seconds must not have passed - Loadertimer is the variable tracking elapsed loading time	- Increase the timer counter Loadertimer at each time step
crate		

- `doneload` [event]

Parameters	Preconditions	Effects
loader	- The loader must be busy (it must have received the crate from a mover) - The time passed is equal to 4 - `Loadertimer` tracks elapsed loading time	- Reset the timer `Loadertimer` - Set the loader to not busy
crate		- Set the crate as loaded

.

1.5 Extentions

After defining and verifying the domain against all mandatory specifications, we now present the required extensions and our proposed approach for their assessment.

For each of the following extensions, we will detail its features and any potentially added components.

1.5.1 Extention 1 description – Grouped Crates

Crates belonging to specific groups require unified loading, isolated from crates of other groups. For improved scalability, these groups are represented numerically in the domain (starting from 1 onwards and number 0 is assigned to crates without groups.), not necessarily implying any physical separation. Our main focus was to distinguish the execution flow for individual crates versus grouped crates and to ensure no priority is given to any specific group. When a group's processing begins, a flag is activated (TRUE) and the id of the group is stored, blocking the processing of other carted belonging to other groups and allowing the execution of the crates belonging to the current group. This flag is deactivated (FALSE) only after all members of the current group have been processed and the stored id stored is reset to 0, enabling the selection of another group.

1. Data structures:
   - Belong_to: maps each crate to the ID of its corresponding group.
   - Number_of_group: maps a group to its numerical ID.
   - elementspergroup: keeps track of how many crates still need to be loaded for each group. Group 0 is defined as the default group for crates that do not belong to any group.
2. State flags and variables:
   - currentgroupset: a boolean flag that indicates whether a group is currently being loaded.
   - currentgroup: a variable storing the numeric ID of the group currently in progress
3. Modified actions:
   - Basic Pickup Actions (pickup, pickup_by_two) are restricted to crates belonging to group 0. They now include preconditions:
     - (not (currentgroupset))
     - (= (belong ?b) 0)
     - (= (currentgroup) 0)

     This ensures that basic pickups don't interfere with ongoing group-based operations.

4. Some extra elements were added:

- `switch_group` [process]

Parameters	Preconditions	Effects
group	- All crates in the group have been loaded - The group in input matches the current group - The flag `currentgroupset` is true	- The flag `currentgroupset` is reset (set to false), allowing a new group to begin

- `pickup_per_gruppo` [action]

Parameters	Preconditions	Effects
mover	- The mover is at the same position as the crate - The mover is currently not moving - The movement direction is set to positive (forward) - The mover is free	- The mover's state is updated to "moving" - The mover starts carrying the crate - The mover is no longer free
crate	- The crate is not already loaded - The crate weighs 50 kg or less - The crate is on the ground inside the warehouse - The crate belongs to a group (ID > 0) - The id of the group of the crate is the same id of the group considered - Either no group is currently active, or the crate belongs to the currently active group.	- The crate is no longer on the ground
group	- The id of the group of the crate is the same id of the group considered - Either no group is currently active, or the crate belongs to the currently active group.	- Sets `currentgroup` to the crate's group. - Activates `currentgroupset`. - Decreases `elementspergroup`.

- `pickup_by_two_per_gruppo` [action] Equivalent to pickup_by_two, but for grouped crates with the same conditions and effects as pickup_per_gruppo.

Parameters	Preconditions	Effects
mover1 mover2	- Both movers are at the same position as the crate - Both movers are currently not moving - The movement direction is set to positive (forward) - Both movers are free	- Both movers' states are updated to "moving" - Both movers start carrying the crate - Both movers are no longer free
crate	- The crate is not already loaded - The crate weighs 50 kg or less - The crate is on the ground inside the warehouse - The crate belongs to a group (ID ¿ 0) - The id of the group of the crate is the same id of the group considered - Either no group is currently active, or the crate belongs to the currently active group.	- The crate is no longer on the ground
group	- The id of the group of the crate is the same id of the group considered - Either no group is currently active, or the crate belongs to the currently active group.	- Sets `currentgroup` to the crate's group. - Activates `currentgroupset`. - Decreases `elementspergroup`.

NOTE:
Crates belonging to group 0 are not allowed to trigger group pickup actions. This ensures that such crates do not affect the state of the current group. The basic pickup actions explicitly require that ¬`currentgroupset` and that the crate belongs to group 0 (i.e., $\texttt{belong}(?b) = 0$), thus preventing any interference with group management.

In the problem definition, the flag `currentgroupset` is initially set to `false`. This means that at the beginning, if both stray crates and grouped crates are present, the non-grouped crates will be processed first.

NOTE:
The precondition "Either no group is currently active, or the crate belongs to the currently active group" in the two "per gruppo" actions, while logically sound, was included primarily for clarity. However, this "or" condition can increase the complexity for the planning solver. A more efficient approach would be to decompose this precondition by creating separate actions for each case within the "or" statement. This would result in two distinct actions: one triggered when no group is active, and another triggered when the crate belongs to the currently active group. This separation, while increasing the number of actions, can simplify the individual action definitions and potentially reduce the search space complexity for the solver.

1.5.2 Extention 2 description – Cheap loader

The system includes two loaders: a standard one and a specialized loader for crates weighing less than 50 kg, identified by the "ischeap" attribute. We manage multiple loaders by specifying the intended loader in the handover, load, and doneload events. This enables movers to select a particular loader for handover, which then becomes busy during loading.

- (Ischeap ?l), variable which, given a loader tells is it is cheap or not

1.5.3 Extention 3 description – Battery consumption

To simulate realistic operation, movers possess a limited power capacity that diminishes during movement. Should a mover's battery run out, it can be replenished by returning to the initial loading point. While moving away from the loading bay, the battery of each active mover decreases by time_delta at each time step, with the granularity of this reduction determined by time_delta. When a mover is stationary and free at the loading bay with a charge below its maximum, it is immediately restored to full power in a single action. For a more detailed simulation, this could be refined into a sequence involving events and a charging process. This power constraint feature is designed as a modular extension and can be easily excluded from the simulation by commenting out its code.

1. State flags and variables:
    - (battery ?m) which keeps track of the battery of each mover
    - (Maxchargebattery ?m) reference to the max battery that each mover can have
2. Some extra elements were added:
    - `Drain_battery` [process]

Parameters	Preconditions	Effects
mover	- The mover is moving - The battery level > 0 - The position of the mover > 1 (to indicate it is moving)	- At each time step, the battery level of the mover is decreased by 1

- `Charging_battery` [event]
  The event is in charge of restoring the battery of the mover.
  The charging position is 0.

Parameters	Preconditions	Effects
mover	- The mover is not moving - The mover is at position 0 - The battery level is lower than the maximum	- Mover's battery level is set to its maximum

In order not to increase the complexity of plan generation, it was decided battery charging is not instantaneous,

1.5.4 Extention 4 description – Fragile crates

The domain differentiates fragile crates in their pickup and loading procedures. To enforce that a single mover cannot lift them, the condition (not (fragile ?b)).
Furthermore, the extended loading time takes 4 seconds if the crate is not fragile and 6 seconds if it is.

1. State flags and variables:
   - Isfragile to distinguish a fragile crate from a normal one
2. Modified actions:
   - The pickup and pickup_per_gruppo actions were modified to include the condition: (not (isfragile ?b))
   - This ensures that fragile crates are only handled through the pickup_by_two or pickup_by_two_per_gruppo actions, which implicitly allow multiple robots to manage them.
   - backto_loader was modified to include the condition (not (isfragile ?b)), in this way all crates with "is fragile" attribute will be redirected to the backto_loader_by_two
   - Stop_handover was modified to include the condition (not (isfragile ?b)), in this way all crates with "is fragile" attribute will be redirected to the Stop_handover_by_two
   - load and doneload were modified to include the condition which checks if the timer is either arrived to 4 and the payload is not fragile or the timer has reached 6 and the crate is fragile

NOTE:
The current use of "or" conditions to manage the 4-second and 6-second loading times is logically sound and accurately reflects the intended behavior. However, this approach increases the complexity the planning solver must handle. A more optimal solution would be to create separate events and processes for each loading duration (one set for the 4-second case and another for the 6-second case). This separation would allow the solver to deal with each timing constraint independently, potentially simplifying the planning process and improving efficiency.

1.6 Experiments and Results

The the assignment problems and their generated plans have been posted on this GitHub page in the plans folder. On a regular PC with Intel Core-i5 U-series CPU results of all problems were generated under 4 seconds. Problem 3 was not solvable under the battery consumption extension rules.

2 Part II: Solving, Heuristics, and Patterns

2.1 Solving

In automated planning, domain and problem descriptions are modeled using states and actions, analogous to nodes and edges in a graph [3]. Solving a planning problem involves moving from an initial state through a series of actions (a path) to reach a goal state. While Part I employed general-purpose problem solvers, Part II focuses on informed search and the use of heuristic functions to guide this process. Later, symbolic pattern planning is also discussed and a pattern to solve a

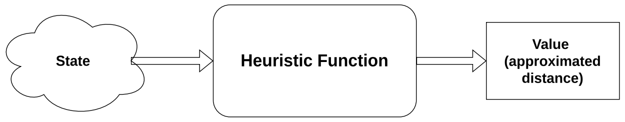

Figure 1:

problem of the specified domain is proposed. One approach to symbolic planning is best-first search, with A* being the most well-known variant. A* evaluates nodes using a cost function that combines the cost to reach the node and an estimated cost to reach the goal from that node. Provided the heuristic function h(n) is admissible (never overestimates the true cost) and consistent, A* guarantees completeness and optimality. A heuristic function estimates the remaining path cost toward the goal and ideally decreases as the state approaches the goal. Unlike uninformed strategies that explore the state space blindly, heuristics enable planners to prioritize more promising paths, thereby enhancing efficiency. An effective heuristic must be computationally efficient, informative enough to differentiate between states, and admissible when optimality is required. To construct such a heuristic, we must understand the structure of the state space and the nature of its state variables. In the given domain, these variables are expressed as predicates (boolean expressions) and functions (numerical values).

2.2 Heuristic

A well-structured heuristic function is an important part in problem solving for planning – not only it can guarantee the problem solution but also increase the performance of the solver and exploit human intuition to important predicates and functions for the problem. The aim in this case was to estimate a heuristic, which is consistent then admissible, such that the *Problem0.5* can be solved. In this report is proposed an heuristic function $h(s)$, Equation 2, that takes into account some aspect of the various states that are keypoints to understand how much is the cost to reach the goal.

The heuristic function $h(s)$ maps each state s to a non-negative value, Figure 1: $f(s) = g(s) + h(s)$, where:

- $g(s)$: the accumulated cost from the initial state to the current state, and
- $h(s)$: the estimated cost from the current state to the goal.

Problem Statement – *Problem 0.5* Given two crates:

- Crate 1: $70\,kg$ weight, 10 units distance from loading bay;
- Crate 2: $20\,kg$ weight, 20 units distance from loading bay, Group A member.

The task involves defining a heuristic function $h(s)$ that estimates the cost to reach the goal from any given state s, adhering to the principles of admissibility and consistency. Admissibility: guarantees $h(s) \leq h*(s)$ Consistency: guarantees $h(s) \leq c(a) + h(s')$, where $c(a)$ is the cost to pass from s to s'.

To ensure computational efficiency and interpretability, we aim to identify a minimal set of state variables that provide relevant, non-redundant information for goal progression. We begin with the full set of state variables

Free	Carry	Moving	To-positive	Busy loading
Is loaded	Equal	Ischeap	Isfragile	currentgroupset
freeloader	atcompany	velocity	max_vel	at-robby
position	Loading Time	weight	belong	currentgroup
Battery	max_battery	numberofgroup	elemntspergroup	

To relax the problem, one should eliminate variables that do not influence the outcome of *Problem 0.5*:

- Variables that are constant throughout execution;
- Some variables whose values are reset at the end (e.g., carry, freeloader);
- Light crates cannot be carried by two movers at the same time: this makes the solution more predictable and deterministic;
- The only group-related information taken into account is the number of elements unloaded at each point in time.

To construct the heuristic, we adopt a modular approach—integrating individual state variables (or combinations thereof) as terms of the function. The goal is to ensure that the heuristic:

- Provides a directional guide toward the goal;
- Is monotonically decreasing along valid paths (i.e., it never increases as progress is made);
- Is computationally simple and memory-efficient.

A monotonically decreasing heuristic aligns with the property of consistency. It ensures that with each valid action taken, the estimated cost to the goal does not increase. This guarantees that the total cost function $f(s) = g(s) + h(s)$ is non-decreasing along any valid path, enabling efficient and optimal search using algorithms like $A*$.

Heuristic function is a function of the state variables. We strongly emphasize the geometry of the heuristic function when analyzing it. If we consider each variable to have a dimension in the overall function, then at each point in the state space there is (hopefully) a path towards the minimum state; The heuristic function is supposed to show the direction towards the goal, i.e. the state with a heuristic of zero. Also, as we incorporate more variables into the heuristic function, the aim is to help the solver find new paths over the function plane towards the goal.

It is trivial that the sum of monotonically decreasing functions is still decreasing. It is also important to note that the heuristic function should not include negative values, a point which has been taken into account for each term. Below is the proposed heuristic function and the analysis of its monotonicity, and effect on consistency, for each term:

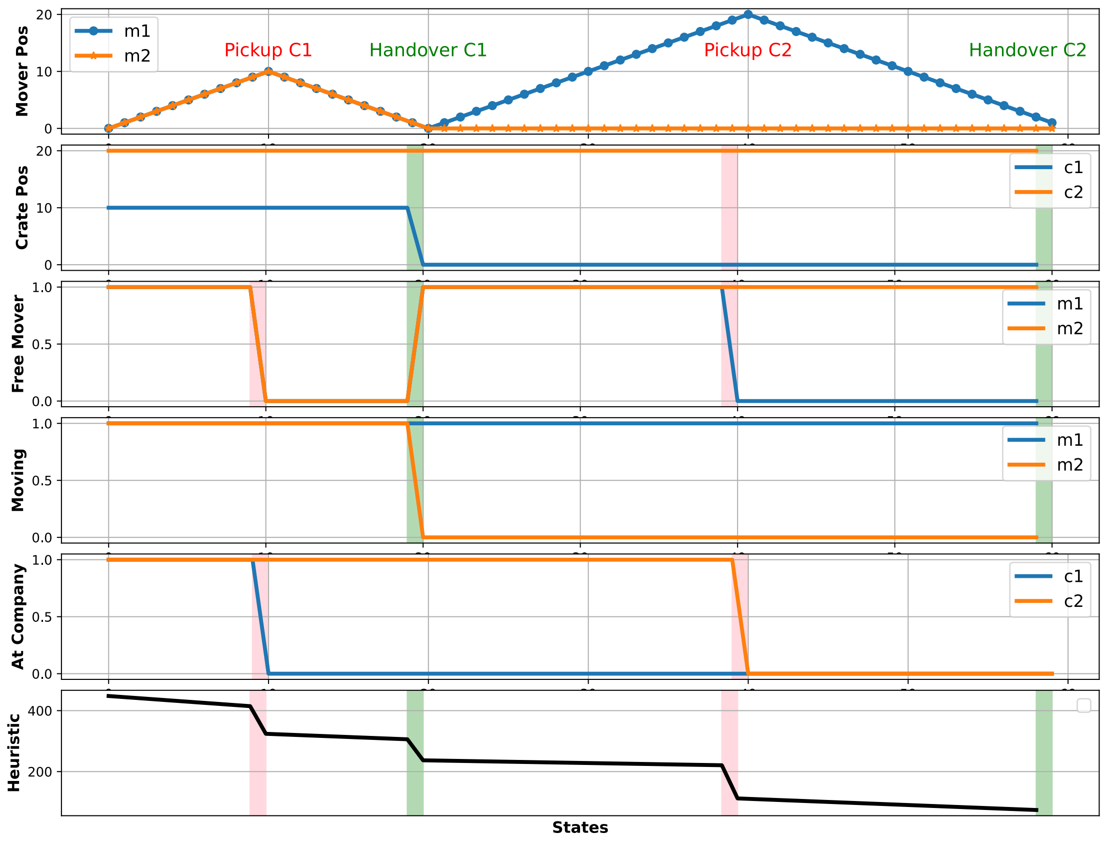

Figure 2: Graphical representation of the state variables and proposed heuristic along the plan of *Probelem0.5*

$$
\begin{aligned}
h(s) = \frac{1}{2} \cdot \sum_{i=1}^{\#\text{crates}} \texttt{position}(c_i) + 200 \cdot \texttt{at\_comp}(c_i) + 25 \cdot \sum_{i=1}^{\#\text{crates}} (\neg\texttt{loaded}(c_i)) \cdot \texttt{at\_comp}(c_i) \\
+ 2 \cdot \sum_{j=1}^{\#\text{movers}} \texttt{rob\_position}(m_j) \cdot (\neg\texttt{free}(m_j)) \\
+ \frac{1}{5} \cdot \sum_{i=1}^{\#\text{crates}} (\min_{m \in \text{Movers}} (|\texttt{position}(c_i) - \texttt{rob\_position}(m)|) \cdot \texttt{free}(m)) \cdot \texttt{at\_comp}(c_i) \cdot (\neg\texttt{loaded}(c_i)) \\
+ 60 \cdot \sum_{i=1}^{\#\text{crates}} (\neg\texttt{loaded}(c_i)) + \sum_{i=1}^{\#\text{crates}} \sum_{j=1}^{\#\text{movers}} |\texttt{position}(c_i) - \texttt{rob\_position}(m_j)| \cdot \texttt{at\_comp}(c_i)
\end{aligned} \tag{8}
$$

A heuristic is monotonically decreasing if for any state n and any successor state n' it holds $h(n') \leq h(n)$. A heuristic is consistent if for every node n and every successor node n' with actual cost $c(n, n')$ it holds $h(n) \leq c(n, n') + h(n')$. Lastly, a theorem declares that consistency brings admissibility.

A monotonically decreasing heuristic function supports the consistency property essential for informed search algorithms, such as $A*$. Consistency ensures that the estimated total cost does not increase along a proper path. A decreasing heuristic naturally aligns with this condition by reducing the estimated cost as the agent progresses toward the goal. Intuitively, we observed that the difference in heuristic values between two states tends to reflect the transition cost, suggesting that this difference should not exceed the actual cost $c(n, n')$. This relationship reinforces the idea that a well-designed heuristic should not only decrease with proximity to the goal but also respect the local cost structure of the state space to maintain consistency and support optimal search performance.

The following is an explanation of different terms of the proposed heuristic and why they are useful:

1. Term $h_1(s) = \frac{1}{2} \cdot \sum_{i=1}^{\#\text{crates}} \texttt{position}(c_i) + 200 \cdot \texttt{at\_comp}(c_i)$

   Variables `position` of the crates and predicate `at_comp` of crates.

   Usefulness This term combines two crucial aspects of crate delivery. $\texttt{position}(c_i)$, which measures the distance of the crate from its destination (assuming the destination is position 0 at base). This is a fundamental estimate of the remaining travel work. $200 * \texttt{at\_comp}$, which assigns a high fixed penalty if the crate is still at its initial "company" location. This strongly penalizes untouched crates.

   Monotonically decreasing The crate being `at_comp`, i.e. untouched, it is a huge negative point since intuition holds on crates picked up and delivered. Therefore, `at_comp` crates will get a penalty relative to their `position`.

2. Term $h_2(s) = 25 \cdot \sum_{i=1}^{\#\text{crates}} (\neg\texttt{loaded}(c_i)) \cdot \texttt{at\_comp}(c_i)$

   Variables Predicates `at_comp` and `loaded` of crates

   Usefulness Crates shift state from (`at_comp`, ¬`loaded`) to (¬`at_comp`, ¬`loaded`) to (¬`at_comp`, `loaded`). Therefore the next worst penalty is given to crates in the second state, attracting the planner to move to the third.

   Monotonically decreasing Following the crates' state towards the goal, this term as an extension of the first term will represent the penalties of the three states gradually and smoothly.

3. Term $h_3(s) = 2 \cdot \sum_{j=1}^{\#\text{movers}} \texttt{rob\_position}(m_j) \cdot (\neg\texttt{free}(m_j))$

   Variables `rob_position` and the `free` predicate of the movers.

   Usefulness This term only penalizes the movers when they have picked up crates, and relative to how far they are from the base, attracting the planner to move towards the base after pickup.

Monotonically decreasing This value will be zero when movers are `free`, which will be compensated by other terms. When not zero, it shows a decreasing behavior as the position of the mover decreases.

4. Term $h_4(s) = \frac{1}{5} \cdot \sum_{i=1}^{\#\text{crates}} (\min_{m \in \text{Movers}}(|\texttt{position}(c_i) - \texttt{rob\_position}(m)|) \cdot \texttt{free}(m)) \cdot \texttt{at\_comp}(c_i) \cdot (\neg \texttt{loaded}(c_i))$

   Variables `rob_position` and `free` of the movers and `position` and predicate `at_comp` and `loaded` of the crates.

   Usefulness Here the idea is to link movers and crates by considering how far is the free mover on the way to pick up the `at_comp` (and not `loaded`) crate.

   Monotonically decreasing As the mover approaches the crate, the distance and therefore heuristic value decrease. The predicates mostly define the specifications of the mover and crate. The main idea is to penalize the distance, which should decrease as they get close.

5. Term $h_5(s) = 60 \cdot \sum_{i=1}^{\#\text{crates}} (\neg \texttt{loaded}(c_i))$

   Variables Predicate `loaded` of the crates.

   Usefulness The most important feature of the crates is their `loaded` status. As such we force discouragement on the crates which are not loaded.

   Monotonically decreasing If a crate becomes `loaded`, its corresponding contribution to this term becomes 0, therefore it decreases. Noteworthy that while the contribution is vital to the path towards the goal, the term includes flat areas which are not ideal.

6. Term $h_6(s) = \sum_{i=1}^{\#\text{crates}} \sum_{j=1}^{\#\text{movers}} |\texttt{position}(c_i) - \texttt{rob\_position}(m_j)| \cdot \texttt{at\_comp}(c_i)$

   Variables `rob_position`, `position` and `at_comp` predicates of the movers.

   Usefulness This term is similar to the one before, yet here we check for all movers. The idea is that movers can cooperate for each crate, either to pick them up by two or for timing efficiency.

   Monotonically decreasing As the two movers cooperate more, this term flattens or decreases. Otherwise there could be successive states which have increasing heuristic values. Only `at_comp` crates are considered here, as to not confuse the solver after each delivery with the new crate position.

2.3 Patterns

Patterns are a recent and innovative method to enhance symbolic planning to decrease planning complexity. Indeed the plan's bound generally decrease till the number of pattern repetitions. In order to find our pattern in the sequence of actions it was searched an order in variables in the *Problem0.5*. Also if commonly a pattern is found with relaxations, losing some constrains, it was chosen to not do it because otherwise the pattern will not be correct. Because hybrid planning $\Pi = \langle V_B, V_N, I, A, E, P_\delta \rangle$ was used in Part 1.2 it was considerate appropriate to use *Asymptotic Relaxed Planning Graph* (*ARPG*) approach with layers of states and happenings $h \in A \cup E \cup P_\delta$, Appendix 3. Where were considers only actions to compose the lifted pattern:

$$\begin{aligned}\prec = \langle &\texttt{start\_forward}, \texttt{stop\_at\_crate}, \texttt{pickup\_by\_two},\\ &\texttt{pickup\_by\_two\_per\_group}, \texttt{pickup}, \texttt{pick\_per\_group},\\ &\texttt{stop\_handover\_by\_two}, \texttt{stop\_handover}\rangle\end{aligned}$$

But the interesting part is the grounded pattern [1] that has a bound 2:

$$\begin{aligned}\prec = \langle &\texttt{start\_forward}(mover1), \texttt{start\_forward}(mover2), \texttt{stop\_at\_crate}(mover1, crate1),\\ &\texttt{stop\_at\_crate}(mover2, crate1), \texttt{stop\_at\_crate}(mover1, crate2),\\ &\texttt{stop\_at\_crate}(mover2, crate2), \texttt{pickup\_by\_two}(mover1, mover2, crate1),\\ &\texttt{pickup\_by\_two}(mover2, mover1, crate1), \texttt{pickup\_by\_two\_per\_group}(mover1, mover2, crate2),\\ &\texttt{pickup\_by\_two\_per\_group}(mover2, mover1, crate2),\\ &\texttt{pickup}(mover1, crate2), \texttt{pickup}(mover2, crate2), \texttt{pick\_per\_group}(mover1, crate2),\\ &\texttt{stop\_handover\_by\_two}(mover1, mover2, crate1, loader1), \texttt{stop\_handover}(mover1, crate2, loader1),\\ &\texttt{stop\_handover}(mover1, crate2, loader2), \texttt{stop\_handover}(mover2, crate2, loader1),\\ &\texttt{stop\_handover}(mover2, crate2, loader2)\rangle\end{aligned}$$

The bound was calculated on how many times the solver is supposed to rethink its sequence over the provided pattern and with two patterns the bound is calculated as two.

Two notes have to be taken into account. First, actions which take as argument two objects of the same type can be doubled in the grounded pattern, since they can be interchanged when using the action. Second, the provided pattern can be further made simple by concatenating the delivery sequence of the two crates in one pattern, yet with loss of generality.

3 Conclusion and Future Work

This article was a study of automated planning for warehouse automation with mover and loader robots. The main problem and domain definitions were discussed in detail in part I. This study approached the problem through hybrid planning with PDDL+, exposing actions, events, and processes, as well as predicates and functions. Some implemented extensions on the main problem include grouped delivery, loader types, mover battery consumption, and fragile crates. Later in part II a more theoretical approach was analyzed to gain more perspective into how solvers exploit informed search to achieve goals from initial states. Symbolic pattern planning is also noted in the penultimate section, a novel approach to planning as satisfiability with reduced complexity. The authors propose pattern planning with lifted representation as a future research idea. Also studies with increased number of movers and more complex dynamics may be interesting, pushing the boundary towards more industrial settings.

Appendices

Asymptotic Relaxed Planning Graph

s_I	a_0	s_0	a_1	s_1	a_2	s_2
free$m_1 := \top$	start_forward	free$m_1 := \top$	goto_pickup	free$m_1 := \top$	drain_battery	free$m_1 := \top$
free$m_2 := \top$		free$m_2 := \top$		free$m_2 := \top$	stop_at_crate	free$m_2 := \top$
rob_pos$m_1 \in [0,0]$		rob_pos$m_1 \in [0,0]$		rob_pos$m_1 \in [1,+\infty)$		rob_pos$m_1 \in [1,+\infty)$
rob_pos$m_2 \in [0,0]$		rob_pos$m_2 \in [0,0]$		rob_pos$m_2 \in [1,+\infty)$		rob_pos$m_2 \in [1,+\infty)$
moving$m_1 := \bot$		moving$m_1 := \top$		moving$m_1 := \top$		moving$m_1 := \{\top,\bot\}$
moving$m_2 := \bot$		moving$m_2 := \top$		moving$m_2 := \top$		moving$m_2 := \{\top,\bot\}$
topositive$m_1 := \bot$		topositive$m_1 := \top$		topositive$m_1 := \top$		topositive$m_1 := \top$
topositive$m_2 := \bot$		topositive$m_2 := \top$		topositive$m_2 := \top$		topositive$m_2 := \top$
battery$m_1 \in [20,20]$		battery$m_1 \in [20,20]$		battery$m_1 \in [20,20]$		battery$m_1 \in (-\infty,20]$
battery$m_2 \in [20,20]$		battery$m_2 \in [20,20]$		battery$m_2 \in [20,20]$		battery$m_2 \in (-\infty,20]$
velocity$m_1 \in [10,10]$		velocity$m_1 \in [10,10]$		velocity$m_1 \in [10,10]$		velocity$m_1 \in [10,10]$
velocity$m_2 \in [10,10]$		velocity$m_2 \in [10,10]$		velocity$m_2 \in [10,10]$		velocity$m_2 \in [10,10]$
isloaded$m_1 := \bot$		isloaded$m_1 := \bot$		isloaded$m_1 := \bot$		isloaded$m_1 := \bot$
isloaded$m_2 := \bot$		isloaded$m_2 := \bot$		isloaded$m_2 := \bot$		isloaded$m_2 := \bot$
coef_set $:= \bot$		coef_set $:= \bot$		coef_set $:= \bot$		coef_set $:= \bot$
at_company$c_1 := \top$		at_company$c_1 := \top$		at_company$c_1 := \top$		at_company$c_1 := \top$
at_company$c_2 := \top$		at_company$c_2 := \top$		at_company$c_2 := \top$		at_company$c_2 := \top$
elementspergroup$A \in [1,1]$		elementspergroup$A \in [1,1]$		elementspergroup$A \in [1,1]$		elementspergroup$A \in [1,1]$
currentgroupset $:= \bot$		currentgroupset $:= \bot$		currentgroupset $:= \bot$		currentgroupset $:= \bot$
currentgroup $\in [0,0]$		currentgroup $\in [0,0]$		currentgroup $\in [0,0]$		currentgroup $\in [0,0]$
loadertimer$l1 \in [0,0]$		loadertimer$l1 \in [0,0]$		loadertimer$l1 \in [0,0]$		loadertimer$l1 \in [0,0]$
loadertimer$l2 \in [0,0]$		loadertimer$l2 \in [0,0]$		loadertimer$l2 \in [0,0]$		loadertimer$l2 \in [0,0]$
busyloading$l1 := \bot$		busyloading$l1 := \bot$		busyloading$l1 := \bot$		busyloading$l1 := \bot$
busyloading$l2 := \bot$		busyloading$l2 := \bot$		busyloading$l2 := \bot$		busyloading$l2 := \bot$
freeloader$l1 := \top$		freeloader$l1 := \top$		freeloader$l1 := \top$		freeloader$l1 := \top$
freeloader$l2 := \top$		freeloader$l2 := \top$		freeloader$l2 := \top$		freeloader$l2 := \top$
carry$c_1m_1 := \bot$		carry$c_1m_1 := \bot$		carry$c_1m_1 := \bot$		carry$c_1m_1 := \bot$
carry$c_1m_2 := \bot$		carry$c_1m_2 := \bot$		carry$c_1m_2 := \bot$		carry$c_1m_2 := \bot$
carry$c_2m_1 := \bot$		carry$c_2m_1 := \bot$		carry$c_2m_1 := \bot$		carry$c_2m_1 := \bot$
carry$c_2m_2 := \bot$		carry$c_2m_2 := \bot$		carry$c_2m_2 := \bot$		carry$c_2m_2 := \bot$
position$c_1 \in [10,10]$		position$c_1 \in [10,10]$		position$c_1 \in [10,10]$		position$c_1 \in [10,10]$
position$c_2 \in [20,20]$		position$c_2 \in [20,20]$		position$c_2 \in [20,20]$		position$c_2 \in [20,20]$

a_3	s_3	a_4	s_4
pickup_per_group	free$m_1 := \{\top,\bot\}$	switch_group	free$m_1 := \{\top,\bot\}$
pickup_by_two	free$m_2 := \{\top,\bot\}$	coeff_changer_light	free$m_2 := \{\top,\bot\}$
pickup_by_two_per_group	rob_pos$m_1 \in [1,+\infty)$	coeff_changer_heavy	rob_pos$m_1 \in (-\infty,+\infty)$
	rob_pos$m_2 \in [1,+\infty)$	backto_loader	rob_pos$m_2 \in (-\infty,+\infty)$
	moving$m_1 := \{\top,\bot\}$		moving$m_1 := \{\top,\bot\}$
	moving$m_2 := \{\top,\bot\}$		moving$m_2 := \{\top,\bot\}$
	topositive$m_1 := \{\top,\bot\}$		topositive$m_1 := \{\top,\bot\}$
	topositive$m_2 := \{\top,\bot\}$		topositive$m_2 := \{\top,\bot\}$
	battery$m_1 \in (-\infty,20]$		battery$m_1 \in (-\infty,20]$
	battery$m_2 \in (-\infty,20]$		battery$m_2 \in (-\infty,20]$
	velocity$m_1 \in [10,10] \cup [\frac{100}{\texttt{weight}c_2}, \frac{100}{\texttt{weight}c_2}]$		velocity$m_1 \in [10,10] \cup [\frac{100}{\texttt{weight}c_2}, \frac{100}{\texttt{weight}c_2}] \cup [\frac{150}{\texttt{weight}c_2}, \frac{150}{\texttt{weight}c_2}] \cup [\frac{100}{\texttt{weight}c_1}, \frac{100}{\texttt{weight}c_1}]$
	velocity$m_2 \in [10,10] \cup [\frac{100}{\texttt{weight}c_2}, \frac{100}{\texttt{weight}c_2}]$		velocity$m_2 \in [10,10] \in [10,10] \cup [\frac{100}{\texttt{weight}c_2}, \frac{100}{\texttt{weight}c_2}] \cup [\frac{150}{\texttt{weight}c_2}, \frac{150}{\texttt{weight}c_2}] \cup [\frac{100}{\texttt{weight}c_1}, \frac{100}{\texttt{weight}c_1}]$
	isloaded$m_1 := \bot$		isloaded$m_1 := \bot$
	isloaded$m_2 := \bot$		isloaded$m_2 := \bot$
	coef_set $:= \{\top,\bot\}$		coef_set $:= \{\top,\bot\}$
	at_company$c_1 := \{\top,\bot\}$		at_company$c_1 := \{\top,\bot\}$
	at_company$c_2 := \{\top,\bot\}$		at_company$c_2 := \{\top,\bot\}$
	elementspergroup$A \in (-\infty,1]$		elementspergroup$A \in (-\infty,1]$
	currentgroupset $:= \{\top,\bot\}$		currentgroupset $:= \{\top,\bot\}$
	currentgroup $\in [0,0] \cup [\texttt{belong}c_2, \texttt{belong}c_2]$		currentgroup $\in [0,0] \cup [\texttt{belong}c_2, \texttt{belong}c_2]$
	loadertimer$l1 \in [0,0]$		loadertimer$l1 \in [0,0]$
	loadertimer$l2 \in [0,0]$		loadertimer$l2 \in [0,0]$
	busyloading$l1 := \bot$		busyloading$l1 := \bot$
	busyloading$l2 := \bot$		busyloading$l2 := \bot$
	freeloader$l1 := \top$		freeloader$l1 := \top$
	freeloader$l2 := \top$		freeloader$l2 := \top$
	carry$c_1m_1 := \{\top,\bot\}$		carry$c_1m_1 := \{\top,\bot\}$
	carry$c_1m_2 := \{\top,\bot\}$		carry$c_1m_2 := \{\top,\bot\}$
	carry$c_2m_1 := \{\top,\bot\}$		carry$c_2m_1 := \{\top,\bot\}$
	carry$c_2m_2 := \{\top,\bot\}$		carry$c_2m_2 := \{\top,\bot\}$
	position$c_1 \in [10,10]$		position$c_1 \in [10,10]$
	position$c_2 \in [20,20]$		position$c_2 \in [20,20]$

a_5	s_5	a_6	s_6	a_7	s_7
charging_battery	free$m_1 := \{\top,\bot\}$	load	free$m_1 := \{\top,\bot\}$	doneload	free$m_1 := \{\top,\bot\}$
switch_group	free$m_2 := \{\top,\bot\}$		free$m_2 := \{\top,\bot\}$		free$m_2 := \{\top,\bot\}$
stop_handover	rob_pos$m_1 \in (-\infty,+\infty)$		rob_pos$m_1 \in (-\infty,+\infty)$		rob_pos$m_1 \in (-\infty,+\infty)$
stop_handover_by_two	rob_pos$m_2 \in (-\infty,+\infty)$		rob_pos$m_2 \in (-\infty,+\infty)$		rob_pos$m_2 \in (-\infty,+\infty)$
	moving$m_1 := \{\top,\bot\}$		moving$m_1 := \{\top,\bot\}$		moving$m_1 := \{\top,\bot\}$
	moving$m_2 := \{\top,\bot\}$		moving$m_2 := \{\top,\bot\}$		moving$m_2 := \{\top,\bot\}$
	topositive$m_1 := \{\top,\bot\}$		topositive$m_1 := \{\top,\bot\}$		topositive$m_1 := \{\top,\bot\}$
	topositive$m_2 := \{\top,\bot\}$		topositive$m_2 := \{\top,\bot\}$		topositive$m_2 := \{\top,\bot\}$
	battery$m_1 \in (-\infty,20]$		battery$m_1 \in (-\infty,20]$		battery$m_1 \in (-\infty,20]$
	battery$m_2 \in (-\infty,20]$		battery$m_2 \in (-\infty,20]$		battery$m_2 \in (-\infty,20]$
	velocity$m_1 \in [10,10] \in [10,10] \cup [\frac{100}{\texttt{weight}c_2}, \frac{100}{\texttt{weight}c_2}] \cup [\frac{150}{\texttt{weight}c_2}, \frac{150}{\texttt{weight}c_2}] \cup [\frac{100}{\texttt{weight}c_1}, \frac{100}{\texttt{weight}c_1}] \cup [\texttt{maxvel}, \texttt{maxvel}]$		velocity$m_1 \in [10,10] \in [10,10] \cup [\frac{100}{\texttt{weight}c_2}, \frac{100}{\texttt{weight}c_2}] \cup [\frac{150}{\texttt{weight}c_2}, \frac{150}{\texttt{weight}c_2}] \cup [\frac{100}{\texttt{weight}c_1}, \frac{100}{\texttt{weight}c_1}] \cup [\texttt{maxvel}, \texttt{maxvel}]$		velocity$m_1 \in [10,10] \in [10,10] \cup [\frac{100}{\texttt{weight}c_2}, \frac{100}{\texttt{weight}c_2}] \cup [\frac{150}{\texttt{weight}c_2}, \frac{150}{\texttt{weight}c_2}] \cup [\frac{100}{\texttt{weight}c_1}, \frac{100}{\texttt{weight}c_1}] \cup [\texttt{maxvel}, \texttt{maxvel}]$
	velocity$m_2 \in [10,10] \in [10,10] \cup [\frac{100}{\texttt{weight}c_2}, \frac{100}{\texttt{weight}c_2}] \cup [\frac{150}{\texttt{weight}c_2}, \frac{150}{\texttt{weight}c_2}] \cup [\frac{100}{\texttt{weight}c_1}, \frac{100}{\texttt{weight}c_1}] \cup [\texttt{maxvel}, \texttt{maxvel}]$		velocity$m_2 \in [10,10] \in [10,10] \cup [\frac{100}{\texttt{weight}c_2}, \frac{100}{\texttt{weight}c_2}] \cup [\frac{150}{\texttt{weight}c_2}, \frac{150}{\texttt{weight}c_2}] \cup [\frac{100}{\texttt{weight}c_1}, \frac{100}{\texttt{weight}c_1}] \cup [\texttt{maxvel}, \texttt{maxvel}]$		velocity$m_2 \in [10,10] \in [10,10] \cup [\frac{100}{\texttt{weight}c_2}, \frac{100}{\texttt{weight}c_2}] \cup [\frac{150}{\texttt{weight}c_2}, \frac{150}{\texttt{weight}c_2}] \cup [\frac{100}{\texttt{weight}c_1}, \frac{100}{\texttt{weight}c_1}] \cup [\texttt{maxvel}, \texttt{maxvel}]$
	isloaded$m_1 := \bot$		isloaded$m_1 := \bot$		isloaded$m_1 := \{\top,\bot\}$
	isloaded$m_2 := \bot$		isloaded$m_2 := \bot$		isloaded$m_2 := \{\top,\bot\}$
	coef_set $:= \{\top,\bot\}$		coef_set $:= \{\top,\bot\}$		coef_set $:= \{\top,\bot\}$
	at_company$c_1 := \{\top,\bot\}$		at_company$c_1 := \{\top,\bot\}$		at_company$c_1 := \{\top,\bot\}$
	at_company$c_2 := \{\top,\bot\}$		at_company$c_2 := \{\top,\bot\}$		at_company$c_2 := \{\top,\bot\}$
	elementspergroup$A \in (-\infty,1]$		elementspergroup$A \in (-\infty,1]$		elementspergroup$A \in (-\infty,1]$
	currentgroupset $:= \{\top,\bot\}$		currentgroupset $:= \{\top,\bot\}$		currentgroupset $:= \{\top,\bot\}$
	currentgroup $\in [0,0] \cup [\texttt{belong}c_2, \texttt{belong}c_2]$		currentgroup $\in [0,0] \cup [\texttt{belong}c_2, \texttt{belong}c_2]$		currentgroup $\in [0,0] \cup [\texttt{belong}c_2, \texttt{belong}c_2]$
	loadertimer$l1 \in [0,0]$		loadertimer$l1 \in [0,6]$		loadertimer$l1 \in [0,6]$
	loadertimer$l2 \in [0,0]$		loadertimer$l2 \in [0,6]$		loadertimer$l2 \in [0,6]$
	busyloading$l1 := \{\top,\bot\}$		busyloading$l1 := \{\top,\bot\}$		busyloading$l1 := \{\top,\bot\}$
	busyloading$l2 := \{\top,\bot\}$		busyloading$l2 := \{\top,\bot\}$		busyloading$l2 := \{\top,\bot\}$
	freeloader$l1 := \{\top,\bot\}$		freeloader$l1 := \{\top,\bot\}$		freeloader$l1 := \{\top,\bot\}$
	freeloader$l2 := \{\top,\bot\}$		freeloader$l2 := \{\top,\bot\}$		freeloader$l2 := \{\top,\bot\}$
	carry$c_1m_1 := \{\top,\bot\}$		carry$c_1m_1 := \{\top,\bot\}$		carry$c_1m_1 := \{\top,\bot\}$
	carry$c_1m_2 := \{\top,\bot\}$		carry$c_1m_2 := \{\top,\bot\}$		carry$c_1m_2 := \{\top,\bot\}$
	carry$c_2m_1 := \{\top,\bot\}$		carry$c_2m_1 := \{\top,\bot\}$		carry$c_2m_1 := \{\top,\bot\}$
	carry$c_2m_2 := \{\top,\bot\}$		carry$c_2m_2 := \{\top,\bot\}$		carry$c_2m_2 := \{\top,\bot\}$
	position$c_1 \in [10,10] \cup [0,0]$		position$c_1 \in [10,10] \cup [0,0]$		position$c_1 \in [10,10] \cup [0,0]$
	position$c_2 \in [20,20] \cup [0,0]$		position$c_2 \in [20,20] \cup [0,0]$		position$c_2 \in [20,20] \cup [0,0]$

References

[1] Matteo Cardellini. *Symbolic Pattern Planning*. PhD thesis, Politecnico di Torino, 2024.

[2] IPC Committee. Gripper domain from the international planning competition. PDDL domain definition used in IPC, 1998. Available from https://ipc.informatik.uni-freiburg.de/.

[3] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010.

[4] Enrico Scala et al. Ai planning for hybrid systems. In *IJCAI*, pages 7045–7050, 2023.