UNIVERSITÀ DEGLI STUDI DI GENOVA

DIBRIS

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY,
BIOENGINEERING, ROBOTICS AND SYSTEM ENGINEERING

ARTIFICIAL INTELLIGENCE FOR ROBOTICS II

Assignment

Automated warehouse scenario

Authors:

BEAUJEAN Bertille - S7899816
HALSE Hanna - S7869787
MOMIYAMA Yui - S7784921
OTA Mamoru - S7784976
PHAM DANG Paul - S7899827

Professors:

CARDELLINI Matteo
VALLATI Mauro

May 23, 2025

Contents

1 Model, Domain and Problems

1.1 Components of a Temporal Planning Model

$$\Pi = \langle V_b, V_n, A, I, G \rangle$$

- V_b: Boolean variables. Each $v \in V_b$ takes values from $\{T, \bot\}$.
- V_n: Numeric variables. Each $x \in V_n$ takes values from $\mathbb{Q}$.
- A: Set of durative actions. Each $a \in A$ is defined as:

  $$a = \langle a^{\vdash}, a^{\boxplus}, a^{\dashv}, [L, U] \rangle$$

  where:
  - $a^{\vdash}$: Actions that must be conducted at the **start** of the durative action.
  - $a^{\boxplus}$: Actions that must be conducted **over all** the duration of the action.
  - $a^{\dashv}$: Actions that must be conducted at the **end** of the action.
  - $[L, U]$ is the duration interval, where $L \in \mathbb{Q}^{\geq 0}$ and $U \in \mathbb{Q}^{>0}$ with lower and upper bounds $L \leq U$.

  Each snap action $a \in A$ is defined as:

  $$a = \langle \mathsf{pre}(a), \mathsf{eff}(a) \rangle$$

- I: Initial state, where $I : V_b \to \{T, \bot\}$ and $I : V_n \to \mathbb{Q}$.
- G: Goal conditions. A set of Boolean and numeric conditions.

1.2 Domain Definition

It should be noted that we have not used any already existing PDDL domain models to help us create our PDDL model.

1.2.1 Basic Domain without Extension

In the basic domain **without any extension**, the following Boolean and numeric variables and actions are available.

Boolean Variables V_{b0} Boolean variables for the basic domain is written as:

$$\begin{aligned} V_{b0} = \{&\texttt{at\_loading\_bay(mover)}, \\ &\texttt{at\_pause(mover)}, \\ &\texttt{independent(mover1, mover2)}, \\ &\texttt{empty(robot)}, \\ &\texttt{pickable(crate)}, \\ &\texttt{on\_shelf(crate)}, \\ &\texttt{on\_loading\_bay(crate)}, \\ &\texttt{loaded(crate)}, \\ &\texttt{is\_picked\_by\_loader(loader, crate)}, \\ &\texttt{is\_at\_crate(mover, crate)}, \\ &\texttt{is\_picked\_by\_mover\_single(mover, crate)}, \\ &\texttt{is\_picked\_by\_mover\_dual(mover, crate)}, \\ &\texttt{is\_moving\_crate\_single(mover, crate)}, \\ &\texttt{is\_moving\_crate\_dual(mover, crate)}\} \end{aligned}$$

Most of these variables have explicit names. Generally speaking, predicates beginning with *"at_"* refer to movers, those beginning with *"on_"* refer to crates, and those beginning with *"is_"* represent an interaction between a crate and another object. Table 1 illustrates some detail about the predicates which have less explicit names.

#	Predicate	Description
1	`at_pause(mover)`	A mover is neither going to, picking, moving or dropping any crate.
2	`independent(mover1, mover2)`	Two movers are different objects. Used in dual actions to specify that one mover cannot be used twice in the same action.
3	`pickable(crate)` and `on_shelf(crate)`	A crate is pickable until a mover begins the process of picking it. It is on shelf until it is picked by a mover.

Table 1: Some Basic Domain Predicates

Numeric Variables V_{n0}

$$V_{n0} = \{\texttt{weight(crate)},\ \texttt{distance(crate)}\}$$

Actions A_0 The following actions are available in the basic domain. Actions can be described in the mathematical formula.

$$\textbf{mover\_pick\_single}(m, c) = \langle a^{\vdash}, a^{\boxplus}, a^{\dashv},\ [0.1,\ 0.1]\rangle$$

- $a^{\vdash} = \langle\{\texttt{empty(m)},\ \texttt{on\_shelf(c)},\ \texttt{is\_at\_crate(m,\ c)},\ \texttt{pickable(c)}, \emptyset\}\rangle$
- $a^{\boxplus} = \langle\emptyset, \emptyset\rangle$
- $a^{\dashv} = \langle\{\neg\texttt{pickable(c)}\}, \{\neg\texttt{on\_shelf(c)},\ \neg\texttt{empty(m)}, \texttt{is\_picked\_by\_mover\_single(m,\ c)}\ \}\rangle$
- $[L, U] = [0.1,\ 0.1]$

The list of the name of actions are shown in Table 2.

#	Action Name	Description
1	`move_empty(m,c)`	An empty mover moves to a crate for pickup.
2	`mover_pick_single(m,c)`	An empty mover picks a normal pickable crate from the shelf.
3	`mover_pick_dual_heavy(m1,m2,c)`	Two empty movers pick a heavy pickable crate together from the shelf.
4	`mover_pick_dual_light(m1,m2,c)`	Two empty movers pick a light pickable crate together from the shelf.
5	`move_crate_single(m,c)`	A mover carries a light crate to the loading bay.
6	`move_crate_dual(m1,m2,c)`	Two movers carry a crate to the loading bay.
7	`put_down_single(m,l,c)`	A mover places a crate on the loading bay and returns to pause.
8	`put_down_dual(m,l,c)`	Two movers place a crate on the loading bay and return to pause.
9	`loader_pick(l,c)`	A loader picks a crate on the loading bay.
10	`load_normal_crate(l,c)`	A loader loads a picked normal crate in 4 time units.

Table 2: Basic Domain Actions

If most of the actions have determined durations in the assignment (`move_empty, move_crate, load`), the other actions (`pick, put_down`), which were not explicitly described in the requirements, were set to a reasonable duration of 0.1 units of time, which is not null and short enough to represent those quick actions as physically acceptable and to focus the planning on the other actions, which are more significant.

1.2.2 Additional Components for Extension 1

To address Extension 1, which requires that some crates identified by the same group name be loaded in a specific order onto the conveyor belt to facilitate the delivery workflow, we have introduced new Boolean and numeric variables and actions that enforce the grouping and sequencing constraints.

Boolean Variables V_{b1}

$$V_{b1} = \{\texttt{group\_active(grp)},\ \texttt{no\_group(crate)},\ \texttt{is\_of\_group(grp,\ crate)},\ \texttt{no\_active\_group}\}$$

Numeric Variable V_{n1}

$$V_{n1} = \{\texttt{group-cost}\}$$

This numeric variable is used to define a new metric in the problems: indeed, instead of minimizing only the duration of the solution, the planner will consider the cost of activating or deactivating a group. As this variable is incremented at every activation or deactivation of a group, by minimizing the `group-cost`, the best solution will aim at charging all the crates of one group in a row.

Actions A_1 Additional actions for Extension 1 are as shown in Table 3.

#	Action Name	Description
1	`activate_group(gn,c)`	Activates a group if no group is currently active and if there is at least one crate which belongs to it.
2	`deactivate_groups(ga,c)`	Deactivates the current active group if there is at least one crate which has no group.
3	`switch_group(ga,gn,c)`	Switches from one active group to another if there is at least one crate assigned to the new group.
4	`loader_pick_grouped_crate(l,c,g)`	A loader picks a crate from the loading bay, only if it belongs to a group and this one is active.
5	`loader_pick_ungrouped_crate(l,c,g)`	A loader picks a crate from the loading bay that does not belong to any group, only if no group is active.

Table 3: Group Management Actions

The 3 group-activation actions (`activate_group, deactivate_groups, switch_group`), have decision purpose. Thus, it is considered that their duration is insignificant compared to the actions with movement purpose, and it is set to 0.

Let it be noted that the two new `loader_pick` replace the previous `loader_pick` action which was used in the basic domain.

1.2.3 Additional Components for Extension 2

To incorporate Extension 2, which introduces an additional loader sharing the same loading bay, but with the limitation that it cannot handle heavy crates, we have added the necessary variables and actions to account for the capabilities of each loader and their concurrent use.

Boolean Variables V_{b2}

$$V_{b2} = \{\texttt{idle(loader)}\}$$

#	Predicate Name	Description
1	`idle(loader)`	A loader is idle until a crate is put down on its loading bay. It is empty until it picks a crate.

Table 4: Side Loader and Crate Placement Predicates

Actions A_2 Additional actions for Extension 2 are as shown in Table 5.

Note that, in addition to the actions adapted for the side loader, the existing loading actions were also adjusted to meet the requirements of the full loader, which is capable of handling all types of crates.

1.2.4 Additional Components for Extension 3

To incorporate Extension 3, which introduces battery constraints for mover robots and requires them to recharge at a dedicated station after consuming power during operation, we have added the appropriate variables and actions to represent energy consumption, recharging behavior, and the timing constraints associated with power management.

#	Action Name	Description
1	`put_down_to_side_loader_single (m,l,c)`	A mover places a light crate on the loading bay while knowing that a side loader is available and returns to pause.
2	`put_down_to_side_loader_dual(m,l,c)`	Two movers place a light crate on the loading bay while knowing that a side loader is available and return to pause.
3	`side_loader_pick_ungrouped_crate (sl,sc,fl,fc)`	A side loader picks a light, ungrouped crate from the loading bay, only if a full loader is loading another crate and no group is active.
4	`side_loader_pick_grouped_crate (sl,sc,fl,fc,g)`	A side loader picks a light, grouped crate, only if its group is active and a full loader is loading another crate.
5	`side_load_normal_crate(l,c)`	A side loader loads a picked normal crate onto the conveyor in 4 time units.

Table 5: Side Loader and Crate Placement Actions

Boolean Variables V_{b3}

$$V_{b3} = \{\texttt{is\_loading\_crate(loader, crate)},\ \texttt{free(charging\_station)}\}$$

A charging station is considered free when it is not charging any mover. Thus, a charging station can only charge one mover at a time.

Numeric Variable V_{n3}

$$V_{n3} = \{\texttt{battery(mover)},\ \texttt{battery-capacity(mover)},\ \texttt{battery-vel(charging station)}\}$$

These three numeric variables represent, respectively: the current level of the battery of a mover, the maximum capacity of the battery of a mover, and the rate (or velocity) at which a battery charges on a specific charging station, expressed in units of battery by unit of time.

Action A_3 The additional action for Extension 3 is shown below in mathematical formula.

In this action, a paused mover charges 1 unit of battery at a free station at the loading bay, for a duration based on the station's charging velocity (`charging-vel(charging station)`).

$$\textbf{charge\_mover\_1\_unit}(m, st) = \langle a^{\vdash}, a^{\curlywedge}, a^{\dashv},\ \left[\frac{1}{\texttt{charging\_vel}(st)},\ \frac{1}{\texttt{charging\_vel}(st)}\right]\rangle$$

- $a^{\vdash} = \langle\{\,\texttt{at\_loading\_bay(m)}, \texttt{at\_pause(m)}, \texttt{free(st)}, \texttt{battery(m)} < \texttt{battery\_capacity(m)}\,\}, \emptyset\rangle$
- $a^{\curlywedge} = \langle\emptyset, \emptyset\rangle$
- $a^{\dashv} = \langle\{\,\neg\texttt{free(st)}, \neg\texttt{at\_pause(m)}\,\}, \{\,\texttt{free(st)}, \texttt{at\_pause(m)}, \texttt{increase(battery(m), 1)}\,\}\rangle$
- $[L, U] = \left[\frac{1}{\texttt{charging\_vel}(st)},\ \frac{1}{\texttt{charging\_vel}(st)}\right]$

Regarding the battery usage of a mover, the basic actions have been extended to decrease the battery value every time the movers moves empty (`move_empty`-actions) or move a crate (`move_crate`-actions). To preserve physical sense, these actions require the mover to have more battery units than what is needed for the action. For example, a single mover moving 30 distance units to a light crate (20 kg) and carrying it to the loading bay, requires a battery value of over $30/10 + 20 \times 30/100 = 9$ battery units.

This unit-by-unit implementation enables a reduction in the overall duration of the solution, compared to charging the battery to full capacity each time. However, in `domain-3`, we used the `full_charge_mover` action to make the problem solvable. In that case, the robot was fully charged in a single charging action.

1.2.5 Additional Components for Extension 4

To incorporate Extension 4, the addition of fragile crates which must be transported by two movers and require a longer loading duration, we have added the necessary actions to reflect the special handling requirements and adjusted timing constraints for fragile crates.

Actions A_4 Additional actions for Extension 4 is shown in Table 6.

#	Action Name	Description
1	`load_fragile_crate(l, c)`	A loader loads a picked fragile crate onto the conveyor in 6 time units.
2	`side_load_fragile_crate(l, c)`	A side loader loads a picked fragile crate onto the conveyor in 6 time units.

Table 6: Fragile Crate Loading Actions

In order to handle a fragile crate correctly, subtypes "fragile_crate" and "normal_crate" were incorporated in all `move` actions.

1.2.6 Domain Composition with All Extensions

For the complete version of the domain that includes Extensions 1 through 4, the set of variables and actions satisfies the following conditions.

$$
\begin{aligned}
V_b &= V_{b0} \cup V_{b1} \cup V_{b2} \cup V_{b3} \\
V_n &= V_{n0} \cup V_{n1} \cup V_{n3} \\
A &= A_0 \cup A_1 \cup A_2 \cup A_3 \cup A_4
\end{aligned}
$$

where each subset V_{bi}, V_{ni}, A_i corresponds to the variables introduced or modified by the basic domain ($i = 0$), and Extension i (for $i = 1, 2, 3, 4$). This union represents the comprehensive set of boolean and numeric variables, in addition to actions required to support all the functionalities specified by the extensions.

1.2.7 Domain Usage Overview

Table 7 summarizes the domains we provide, which are used in the evaluation, their included extensions, and their specific roles. The main domain `domain_124` serves as the primary basis for comparison, while others are used to isolate or test the impact of specific extensions and assess baseline performance.

Domain	Extensions	Solvable for	Purpose / Role
`domain_basic`	None	All problems	Contains only the minimum requirements, without any extensions.
`domain_3`	Extension 3	All problems **except** Problem 3	Evaluates the individual impact of extension 3.
`domain_124`	Extensions 1, 2, 4	All problems	Works across all problems. Used for general comparison and analysis.
`domain_1234`	Extensions 1-4 (all)	Problem 0.5, 1	Contains all extensions. Unsolvable for Problems 2–4 (details provided later).

Table 7: Overview of domains used for evaluation, with their respective extensions and purposes.

1.3 Problem Definition

Among the four problems provided, we use **Problem 1** as a representative example to illustrate the definition of the initial state and the goal conditions.

1.3.1 Initial State

Problem 1 consists of three crates with different properties, that must be transported to the loading bay and successfully loaded. One of the crates is fragile and two of the crates are in group A.

The initial state for Problem 1 is described in PDDL as follows:

$I_1 = \{$ `empty(m1), empty(m2), at_pause(m1), at_pause(m2),`
`at_loading_bay(m1), at_loading_bay(m2),`
% Both movers are idle and located at the pause/loading bay position.

`independent(m1, m2), independent(m2, m1),`
% Separates the two movers, and inhibits two occurrences of the same mover

`empty(fl), idle(fl),`
% The full loader (fl) is idle and ready for use.

`empty(sl), idle(sl),`
% The side loader (sl) is idle and available.

`on_shelf(c1), on_shelf(c2), on_shelf(c3),`
% All three crates are initially placed on shelves.

`pickable(c1), pickable(c2), pickable(c3),`
% All crates are available for picking by movers.

`weight(c1)=70, weight(c2)=20, weight(c3)=20,`
% Crate c1 is heavy, while c2 and c3 are lighter.

`distance(c1)=10, distance(c2)=20, distance(c3)=20,`
% Crate c1 is closer to the loading bay than c2 and c3.

`no_active_group, no_group(c1),`
% No group is currently active, and c1 does not belong to any group.

`is_of_group(A, c2), is_of_group(A, c3),`
% Crates c2 and c3 belong to group A.

`group_cost=0,`
% Initialization of the group-cost.

`free(st1), free(st2)`
% Both charging stations are available.

`charging_vel(st1)=2, charging_vel(st2)=2`
% Both charging stations have a charging velocity of 2 units of battery for each unit of time.

`battery-capacity(m1)=20, battery-capacity(m2)=20,`
`battery(m1)=20, battery(m2)=20`
% Both movers have a maximum capacity of 20 battery units and are fully charged.

$\}$

1.3.2 Goal Conditions

The goal of all the problems is to successfully load the crates onto the conveyor belt. The goal conditions for Problem 1 are described in PDDL as follows:

$$G_1 = \{ \texttt{loaded(c1), loaded(c2), loaded(c3)} \}$$

2 Solving, Heuristics and Patterns

2.1 Solving

2.1.1 OPTIC: A Planner for Temporal Planning

OPTIC (Optimal Temporal planner for IPC) is a forward-chaining planner designed for domains involving *durative actions*, *temporal constraints*, and *numeric fluents*. It supports PDDL 2.1.

Key Features

- Handles durative actions with complex temporal structure (e.g., start, over-all, and end conditions and effects).
- Supports numeric fluents, enabling reasoning about continuous values such as time, energy, or weight.
- Capable of generating both optimal and satisfying plans, depending on the search configuration.
- Incorporates heuristic search with temporal-aware heuristics to efficiently explore large temporal spaces.

Why OPTIC is Useful for Durative Planning

In durative planning, actions take time to execute and may overlap with others. Simple classical planners cannot model such interactions or enforce timing constraints. OPTIC is specifically designed to manage these aspects, making it ideal for scenarios such as logistics, robotics, or industrial automation, where:

- Actions must be scheduled with respect to real time (e.g. loading operations that block resources for fixed durations).
- Multiple agents (e.g. robots) need to coordinate over time.
- Optimization over total execution time (makespan) is important.

For our problems, we used A^* because we defined a metric with the line *(:metric minimize (+ (total-time) (group-cost)))*. So, OPTIC searches for a plan that minimizes the makespan.

Another available temporal planning solver is Temporal Fast Downward (TFD). But it has more limitations than OPTIC and we could not find a plan with this solver.

We also tried LPG solver. It works for the domain without extension. However, when we add more extension, we have a warning : ”Problem size too large.”

Thereby, we finally chose the OPTIC solver for the different problems.

2.1.2 Analysis of the performance

Metric	Problem 1	Problem 4
Total Metric	25.4	37.7
Ground Size	76	244
States Evaluated	83	189
Planning Time (sec)	33.8	34.1

Table 8: Planning Statistics of Problem 1 and 4 with domain_124

To evaluate the performance of the OPTIC solver, problems 1 and 4 are used as examples, with the domain file that includes extensions 1, 2, and 4 as shown in Table 8. For problem 1, the solver evaluates **83** states, has a ground size of **76** and finds a plan in approximately 30–40 seconds. In problem 4, the number of evaluated states increases to **189**, the ground size is **244**, and the planning time remains similar.

The number of **evaluated states** reflects how many nodes have been explored during the search to find a valid plan. A lower number typically suggests a more efficient search, as the planner finds a plan with less exploration of the state space. However, this metric must be interpreted alongside the complexity of the planning problem.

One major factor influencing the number of evaluated states is the **ground size**, which represents the total number of grounded actions generated by instantiating action schemas with all valid object combinations in the problem. As shown in Figure 1, the ground size can be computed based on the number of applicable objects

for each action schema. A larger ground size increases the branching factor and expands the search space, which typically leads to more evaluated states.

In this case, problem 4 includes more objects and interactions than problem 1, resulting in a higher ground size (244 vs. 76) and, consequently, more evaluated states (189 vs. 83). Nevertheless, both numbers are relatively low for temporal planning, indicating that the OPTIC solver performs efficiently.

Moreover, the planning time remains consistent across both problems, despite the increased number of evaluated states in problem 4. This suggests that OPTIC's heuristic effectively guides the search by avoiding irrelevant branches and focusing on the most promising parts of the state space. This behavior highlights the planner's robustness and scalability within the tested problem sizes.

Calculation of Ground Size for `move_empty`

Action schema of `move_empty` is shown below:

```
(:durative-action move_empty
 :parameters (?m - mover ?c - crate)
 ...
)
```

Given the problem definition contains **2 movers (`m1, m2`) and 6 crates (`c1,..,c6`)**, the number of grounded instances of `move_empty` is calculated as:

$$\text{Grounded count} = |\texttt{movers}| \times |\texttt{crates}| = 2 \times 6 = \boxed{12}$$

Figure 1: Calculation of ground size for `move_empty`

2.2 Heuristic

2.2.1 Definition of the heuristic

The heuristic function $h : S \to \mathbb{N}$ estimates the minimum time still required to deliver all crates from their current positions to the conveyor belt. It includes:

- **Travel time from robot to crate**: The heuristic assumes that robots move directly without conflicts from the loading bay at a fixed speed of 10 distance units per time unit to the crate.
- **Picking and dropping time of a crate**: When robots pick or drop a crate, it costs 0.1 time unit.
- **Transport time with crate to the loading bay**: For the heuristic, we assume that the transport time is based only on the weight and the distance to the loading bay. For light crates, the faster 2-robot option is always assumed to guarantee the satisfiability of the heuristic. For heavy crates, both robots are always needed.
- **Fixed loading time**: Each crate adds a fixed cost of 0.1 + 4 time units due to the loader robot's operations, which are sequential and cannot overlap with delivery.

Heuristic Function: $h : S \to \mathbb{N}$

The heuristic estimates the total remaining time needed to deliver all undelivered crates. For each undelivered crate, it includes:

1. Time to reach the crate from a robot in the loading bay: $\frac{\text{crate.position}}{10}$
2. Time to pick up the crate: 0.1 time unit
3. Time to transport the crate to the loading bay:
   - If weight ≥ 50kg (heavy): crate.distance $\times$ crate.weight$/100$
   - If weight < 50kg (light, using 2 robots): crate.distance $\times$ crate.weight$/150$
4. Time to drop the crate in the loading bay: 0.1 time unit
5. Fixed pick and loading time: 0.1 + 4 time units

Algorithm 1 HEURISTIC(state)

```
function HEURISTIC(state)
    total_time ← 0
    for each crate in state.crates do
        if not crate.loaded then
            // Step 1: Time to reach crate
            if crate.distance = mover1.position or crate.distance = mover2.position then
                // A robot is by the crate
                time_to_crate ← 0
            else
                // A robot has to reach the crate from the loading bay
                time_to_crate ← crate.position/10,
            end if
            if crate.distance = 0 then
                // No need to pick the crate because it's already at the loading bay
                time_to_crate ← 0
            end if
            // Step 2: Estimate time to transport crate to loading bay
            if crate.weight ≤ 50 then
                time_to_loading_bay ← crate.distance × crate.weight/150
            else
                // Heavy crate: requires two robots
                time_to_loading_bay ← crate.distance × crate.weight/100
            end if
            // Step 3: Time to pick up and/or drop off the crate
            if crate.distance ≠ 0 then
                // The crate is not at the loading bay
                if crate.picked then
                    // Crate has to be dropped
                    time_pick_drop ← 0.1
                else
                    // Crate has to be picked and dropped
                    time_pick_drop ← 0.2
                end if
            else
                // The crate is at the loading bay
                if crate.picked then
                    // Crate has to be dropped
                    time_pick_drop ← 0.1
                else
                    // No need to pick or drop the crate
                    time_pick_drop ← 0
                end if
            end if
            // Step 4: Fixed loading time
            if crate.picked_by_the_loader then
                // Crate has to be only loaded
                loading_time ← 4
            else
                // Crate has to be picked and load by the loader
                loading_time ← 4.1
            end if
            total_time ← total_time + time_to_crate + time_to_loading_bay + time_pick_drop + loading_time
        end if
    end for
    return total_time
end function
```

General Formula:

$$h(S) = \sum_{\text{crate} \in \text{undelivered}} \left(\text{time\_reach\_crate} + \frac{\text{crate.distance} \times \text{crate.weight}}{\text{speed factor}} + \text{pick\_drop\_time} + \text{loading\_time} \right)$$

Where:

- **time_reach_crate** is computed as:

$$\text{time\_reach\_crate} = \begin{cases} 0 & \text{if crate.distance} = \text{mover1.position } \textit{or } \text{crate.distance} = \text{mover2.position} \\ \frac{crate.distance}{10} & \text{otherwise} \end{cases}$$

- **speed factor** = 100 for heavy crates (requiring 2 robots)
- **speed factor** = 150 for light crates (using 2 robots)
- **pick_drop_time** is computed as:

$$\text{pick\_drop\_time} = \begin{cases} 0.2 & \text{if the crate still needs to be picked up and dropped off} \\ 0.1 & \text{if the crate is picked and not yet dropped} \\ 0 & \text{otherwise} \end{cases}$$

- **loading_time** is computed as

$$\text{loading\_time} = \begin{cases} 4 & \text{if crate.picked\_by\_the\_loader} \\ 4.1 & \text{otherwise} \end{cases}$$

2.2.2 Example of the Heuristic Function for Problem 0.5

Problem 0.5 Summary:

- **Crate 1:** 70kg, 10 units from the loading bay (heavy)
- **Crate 2:** 20kg, 20 units from the loading bay (light)

Let's compute the total heuristic value for the initial state.

Crate 1 (70 kg, 10 distance units):

$$\begin{aligned} \text{Time to crate} &= \frac{10}{10} = 1 \\ \text{Transport time} &= \frac{10 \times 70}{100} = 7 \\ \text{Picking and dropping time} &= 0.1 + 0.1 = 0.2 \\ \text{Loading time} &= 0.1 + 4 \\ \text{Total} &= 1 + 7 + 0.2 + 4.1 = 12.3 \end{aligned}$$

Crate 2 (20 kg, 20 distance units):

$$\begin{aligned} \text{Time to crate} &= \frac{20}{10} = 2 \\ \text{Transport time} &= \frac{20 \times 20}{150} = \frac{400}{150} \approx 2.67 \\ \text{Picking and dropping time} &= 0.1 + 0.1 = 0.2 \\ \text{Loading time} &= 0.1 + 4 \\ \text{Total} &= 2 + 2.67 + 0.2 + 4.1 \approx 8.97 \end{aligned}$$

Total Heuristic Value for Initial State:

$$h(S_0) = 12.3 + 8.97 \approx 21.27$$

Usefulness: The heuristic prefers states with fewer or closer crates remaining. For example:

- State S_0: Both crates undelivered, $h(S_0) \approx 21.27$
- State S_1: Only Crate 2 undelivered, $h(S_1) \approx 8.97$
- Therefore, $h(S_1) < h(S_0)$, indicating progress toward the goal

For the next action, we can notice that the only way to reduce the value of h is for the robots to move to the crates.

Effect on Heuristic When Robot m_1 Moves Toward Crate 1

Assume robot m_1 moves from the loading bay to the location of **Crate 1** (70kg, 10 units away). We analyze how this action affects the heuristic value $h(S)$.

In the initial state S_0, both robots are located at the loading bay, meaning the cost to reach Crate 1 is:

$$\text{Step 1: } \frac{10}{10} = 1 \text{ time unit}$$

Now suppose we move robot m_1 to Crate 1's position. In the resulting state S_1, robot m_1 is already at Crate 1's location, and robot m_2 is still at the loading bay.

Step 1's cost for loading Crate 1 is reduced from 1 to 0. The total estimated cost for Crate 1 now becomes:

$$\begin{aligned} \text{Step 1 (to crate)} &= 0 \\ \text{Step 2 (carry back)} &= \frac{10 \times 70}{100} = 7 \\ \text{Step 3 (pick and drop)} &= 0.2 \\ \text{Step 4 (loading)} &= 4.1 \\ \text{Total} &= 0 + 7 + 0.2 + 4.1 = 11.3 \text{ time units} \end{aligned}$$

Step 1's cost for Crate 2 doesn't change.

Compared to the original cost, the heuristic $h(S)$ decreases :

$$h(S_1) = h(S_0) - 1$$

This shows that moving robot m_1 toward Crate 1 reduces the estimated remaining cost and is likely a useful next action in the plan. Since the heuristic is designed to favor states with lower $h(S)$ values, this transition to S_1 will be preferred in informed search algorithms such as A*.

So, the movement of m_1 toward Crate 1 leads to a decrease in the heuristic value due to a reduced robot-to-crate distance. This supports choosing this move as a promising next step during planning.

This shows that the heuristic reflects progress toward the goal state. As robots move closer to completing crate deliveries, the estimated remaining cost decreases accordingly.

2.2.3 Admissibility of our heuristic

To evaluate whether our heuristic function is admissible, we will consider the problem without extension 2, where we only have one loader. First, we want to check that the heuristic never overestimates the real cost. If we look at each step, the first step computes the **minimum** time for a robot to reach the crate from the loading bay. Then, if it's a light crate, the minimal time to reach to loading bay is *crate.distance × crate.weight/150* if it is carried by two robots. If it's a heavy crate, the minimal time is *crate.distance × crate.weight/100*.

Then, to pick, drop and to load a crate, the time is fixed and it costs 0.1, 0.1 and 4.1 time units. We loop on all the crates not delivered. With this method we assure that the cost will never be overestimated, as we always consider the minimum time to perform a task. Indeed, it optimistically assumes that robots are always available and never waiting. It does not consider delays from loading bay contention (e.g. one crate being loaded while another arrives). Nor does it model the sequential dependency of two robots being tied up for a heavy crate.

Finally, our heuristic is fast and easy to compute. The algorithm used to compute the heuristic is simple for a computer and has a linear complexity based on the number of crates. The result is also reasonably accurate, especially for small problems without conflict and waiting at the loading bay.

2.3 Pattern

As we have chosen to model the system with durative actions, constructing a pattern for $problem\ 0.5$ becomes less meaningful, as the process of loading crates onto the conveyor belt is not characterized by recurring actions that can be easily classified into a repeatable pattern. Instead of representing repetitive behaviour, such as issuing the same `move`-action multiple times in sequence, we model a single `move`-action that spans a defined duration. This temporal representation captures the continuity of the action directly, making it unnecessary to define and detect patterns of repetition. The result of this is that making patterns become redundant, because of the abstraction the durative actions already provide.

In addition, all crates are unique in our domain due to their distance to the loading bay, weight, group, and if they are fragile or not. As a result, we need to specify which crate each action is performed on, consequently making each action unique as well. This means that we do not have repeating actions, contributing to the fact that patterns are redundant. If we had modeled our domain differently, where not every crate was unique and

#	Action
1	(move_empty m1 c2)
2	(move_empty m2 c1)
3	(activate_group a c2)
4	(mover_pick_single m1 c2)
5	(move_crate_single m1 c2)
6	(put_down_to_full_loader_single m1 fl c2)
7	(full_loader_pick_grouped_crate fl c2 a)
8	(move_empty m1 c1)
9	(deactivate_groups a c1)
10	(full_load_normal_crate fl c2)
11	(mover_pick_dual m2 m1 c1)
12	(move_crate_dual_heavy m2 m1 c1)
13	(put_down_to_full_loader_dual m2 m1 fl c1)
14	(full_loader_pick_ungrouped_crate fl c1)
15	(full_load_normal_crate fl c1)

Table 9: Pattern for problem 0.5.

some crates were placed in the same location, patterns would have been of more use. In this hypothetical scenario, actions such as `move_to(m1, location1)`, and `pick_single_crate_at(m1, location1)` are examples of actions which could have been repeated once for every crate in this location.

Yet, we have defined a pattern which can be used for solving $problem\ 0.5$. This pattern is illustrated in Table 9 and is based on the domain where extension 1, 2 and 4 are implemented. The pattern includes each type of action needed to successfully transport the crates from their initial location to the conveyor belt. Using this pattern for $problem\ 0.5$ would result in a bound of 1, since both crates would be loaded onto the conveyor belt after one iteration of the pattern.

If the problem were to increase in size by introducing more crates, this pattern would not be sufficient to find a plan. This is because the pattern specifically handles two crates, $c1$ and $c2$, and if a third crate were added, no further iterations would be sufficient to solve the problem. In cases like this, the pattern would need to be extended to include the actions specified for the third crate, to make sure the goal state is reached. In this case, the bound would still be 1.

To summarize; because we have modeled the domain with durative actions, as well as unique crates, the pattern would have to be changed if the problem were to increase in both size and complexity to be able to solve the problem. Consequently, the bound would not change and would remain 1.