AIRO2 Assignment 1

Group P

8 May 2022

Contents

1 Introduction

1.1 Initial problem

The planning problem to be solved is to transport some crates, known a priori, from the warehouse to a conveyor belt having two types of robots available: movers and loaders. In particular, there are two movers and one loader.
The movers can move in the environment, instead the loaders are fixed and consist of a robotic arm capable of loading the crates from the bay on the conveyor belt. Crates, of which we know the distance from the bay and the weight, can be light or heavy:

- light (weight ≤ 50): can be transported by one mover in a time

$$\frac{distance \times weight}{100}$$

  or by two movers in a time

$$\frac{distance \times weight}{150}$$

- heavy (weight > 50): can be transported by two movers in a time

$$\frac{distance \times weight}{100}$$

A mover, that is not carrying crates, covers 10 distance units per time unit. The loader can load on the conveyor belt a single crate at a time and this process takes 4 time units. The bay must be free during this process.

1.2 Extensions

The initial problem has been extended with the following requirements:

- **groups**: crates are divided in groups (indicated by a letter) and all the crates belonging to a group must be loaded subsequently on the conveyor belt
- **cheap loader**: there is another loader that uses the same bay of the previous (so the bay is considered usable also if one loader is working). This second loader can load only light crates.
- **battery**: the mover robots have a limited battery capacity of 20 power units. A robot consumes a power unit for each time unit in which it is moving with or without crates. The robot recharges its battery when it's at the bay carrying nothing.
- **fragile crates**: some crates are fragile and so needs two movers also when they are light and they are transported to the bay in a time:

$$\frac{distance \times weight}{100}$$

  and they are loaded in 6 time units.

1.3 Language and planner

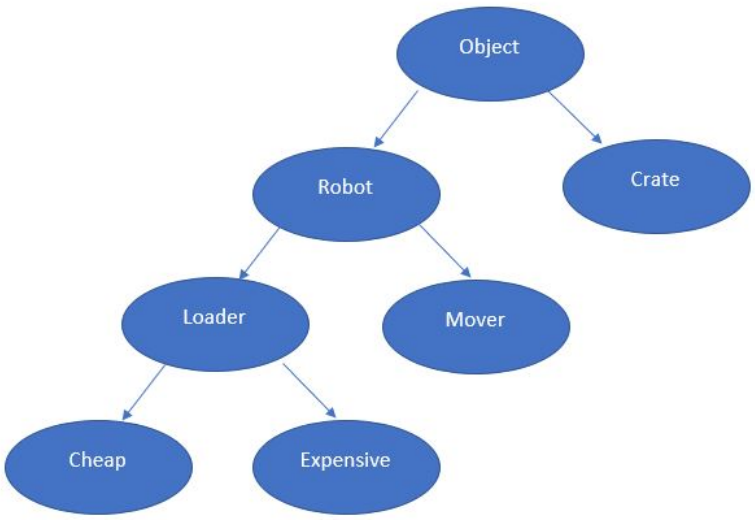
The project is written in PDDL+ and it's designed for ENHSP (`https://sites.google.com/view/enhsp/`) planner that supports numeric planning, but not durative actions.

2 Domain

2.1 Types

The objects involved in the problem can be of type:

- **robot**: general type that includes movers and loaders
- **mover**: robot able to move around in the environment and to carry crates from the warehouse to the bay
- **loader**: robot capable of loading crates from the bay to the conveyor belt
- **expensive**: loader that can put on the conveyor belt any type of crate
- **cheap**: loader that can put on the conveyor belt only light crates
- **crate**: object that must be carried to the conveyor belt from a initial distance



2.2 Predicates

Predicate	Parameters	Description
free	?r - robot	It's true if the robot ?r isn't doing anything (charging, moving or carrying crates if the robot is a mover, or loading crates if the robot is a loader)
mover_at_bay	?m - mover	It's true if the mover ?m is at zero distance from the bay
crate_at_bay	?c - crate	It's true if the crate ?c is on the bay
crate_fragile	?c - crate	It's true if the crate ?c is fragile
free_bay	-	It's true if there are no containers on the bay
mover_at_crate	?m - mover ?c - crate	It's true if the mover ?m is loading the crate ?c
going_to_crate	?m - mover ?c - crate	It's true if the mover ?m is going to catch the crate ?c
going_to_bay	?m - mover	It's true if the mover ?m is going to the bay carrying a crate at normal speed
going_to_bay_fast	?m - mover	It's true if the mover ?m is going to the bay carrying a crate at high speed
crate_on_mover	?m - mover ?c - crate	It's true if the mover ?m is carrying the crate ?c
crate_on_loader	?l - loader ?c - crate	It's true if the loader ?l is loading crate ?c on the conveyor belt
mover_together	?m - mover	It's true if the mover ?m is carrying a crate together with another mover
wait_for_free_bay	?m - mover	It's true if the mover ?m is at the bay carrying a crate, but it can't put the crate on the bay because it's already occupied
loading_crate	?l - loader	It's true if the loader ?l is loading a crate
crate_on_conveyor	?c - crate	It's true if the crate ?c has been loaded on the conveyor belt
crate_at_warehouse	?c - crate	It's true if the crate ?c is at the warehouse
last_crate_loaded	?c - crate	It's true if the crate ?c is the last loaded on the conveyor belt
subsequent_crates	?c1 - crate ?c2 - crate	It's true if the crate ?c1 has been loaded immediately before the crate ?c2 or if the crate ?c2 has been loaded immediately before the crate ?c1
wait_first_crate	-	It's true if no crates has been released on the conveyor belt
crate_need_sequence	?c - crate	It's true if a crate has been just released on the conveyor belt and has to be put in the sequence
going_to_recharge	?m - mover	It's true if the mover ?m has low battery and it's going to the bay for recharging
recharging	?m - mover	It's true if the mover ?m is recharging

2.3 Functions

Function	Parameters	Description
weight	?c - crate	Defines the crate's weight in kg
distance	?o	Defines the distance from an object ?o and the bay
charge	?m - mover	Defines the charge level of the battery of the mover ?m
loading_time	?l - loader	Defines how many seconds have passed since the beginning of the loader ?l loading action
max_charge	-	Defines the maximum charge that a mover's battery can have
max_light_weight	-	Defines the weight threshold to distinguish light and heavy crates

2.4 Actions

2.4.1 Go to crate

This action can be performed when a mover is free at the bay and a crate is at the warehouse, waiting to be carried. After this action the robot will start moving to reach the crate.

```
(:action go_to_crate
    :parameters (?m - mover ?c -crate)
    :precondition (and
        (crate_at_warehouse ?c)
        (mover_at_bay ?m)
        (free ?m))
    :effect (and
        (going_to_crate ?m ?c)
        (not (free ?m))
        (not (mover_at_bay ?m))
    )

)
```

2.4.2 Grab one mover

With this action a free mover grabs a light not fragile crate that is in its same position, but not at the bay. Then the mover will go to the bay carrying the crate.

```
(:action grab_one_mover
    :parameters (?m - mover ?c - crate)
    :precondition (and
        (<= (weight ?c) (max_light_weight))
        (not (crate_fragile ?c))
        (crate_at_warehouse ?c)
        (free ?m)
        (mover_at_crate ?m ?c))
    :effect (and
        (not (crate_at_warehouse ?c))
        (not (mover_at_crate ?m ?c))
        (going_to_bay ?m)
        (crate_on_mover ?m ?c)
        (not (free ?m)))
)
```

2.4.3 Grab two movers

This action is very similar to the previous one, but it involves two movers instead of one. The only difference in the effects are the use of the predicate *mover_ together*, which tells the planner that the mover is working with the other. Two movers are required when the crate is fragile, or when it's too heavy (more than 50 kg).

```
(:action grab_two_movers
    :parameters (?m1 ?m2 - mover ?c - crate)
    :precondition (and
        (free ?m1)
        (free ?m2)
        (or
            (> (weight ?c) (max_light_weight))
            (crate_fragile ?c)
        )
        (crate_at_warehouse ?c)
        (mover_at_crate ?m1 ?c)
        (mover_at_crate ?m2 ?c)
        (not (= ?m1 ?m2)))
    :effect (and
```

```
        (not (crate_at_warehouse ?c))
        (not (mover_at_crate ?m1 ?c))
        (not (mover_at_crate ?m2 ?c))
        (mover_together ?m1)
        (mover_together ?m2)
        (going_to_bay ?m1)
        (going_to_bay ?m2)
        (crate_on_mover ?m1 ?c)
        (crate_on_mover ?m2 ?c)
        (not (free ?m1))
        (not (free ?m2)))
)
```

2.4.4 Grab two movers fast

This action is just the same as the previous one, but here two robots carry a light crate with a faster speed. In order to distinguish between going to bay with a normal speed or with a faster speed, we sets to true the predicate *going_to_bay_fast*, for both movers.

```
(:action grab_two_movers_fast
:parameters (?m1 ?m2 - mover ?c - crate)
:precondition (and
    (free ?m1)
    (free ?m2)
    (<= (weight ?c) (max_light_weight))
    (crate_at_warehouse ?c)
    (not (crate_fragile ?c))
    (mover_at_crate ?m1 ?c)
    (mover_at_crate ?m2 ?c)
    (not (= ?m1 ?m2)))
:effect (and
    (not (crate_at_warehouse ?c))
    (not (mover_at_crate ?m1 ?c))
    (not (mover_at_crate ?m2 ?c))
    (mover_together ?m1)
    (mover_together ?m2)
    (going_to_bay_fast ?m1)
    (going_to_bay_fast ?m2)
    (crate_on_mover ?m1 ?c)
    (crate_on_mover ?m2 ?c)
    (not (free ?m1))
    (not (free ?m2)))
)
```

2.4.5 release one mover

When the mover finally arrives to the bay carrying a crate (which can't be fragile or heavy), it should release the crate. This action simulates this operation, and it's used when the robot isn't moving together with the other. As effect, the mover becomes free at the bay, and the crate is no longer on the mover but stays at the bay, waiting for the loader.

```
(:action release_one_mover
    :parameters (?m - mover ?c - crate)
    :precondition (and
        (<= (weight ?c) (max_light_weight))
        (not (crate_fragile ?c))
        (not (wait_for_free_bay ?m))
        (not (mover_together ?m))
        (crate_on_mover ?m ?c)
        (= (distance ?m) 0))
    :effect (and
        (mover_at_bay ?m)
```

```
        (not (free_bay))
        (crate_at_bay ?c)
        (not (crate_on_mover ?m ?c))
        (free ?m))
)
```

2.4.6 release two movers

This action is the equivalent of *release_one_mover* but it involves the couple of movers. Here the planner check if the two movers have arrived to the bay carrying the same crate, and if it happens the crate is released.

```
(:action release_two_movers
    :parameters (?m1 ?m2 - mover ?c - crate)
    :precondition (and
        (not (wait_for_free_bay ?m1))
        (not (wait_for_free_bay ?m2))
        (crate_on_mover ?m1 ?c)
        (crate_on_mover ?m2 ?c)
        (= (distance ?m1) 0)
        (= (distance ?m2) 0)
        (mover_together ?m1)
        (mover_together ?m2)
        (not (= ?m1 ?m2)))
    :effect (and
        (mover_at_bay ?m1)
        (mover_at_bay ?m2)
        (not (free_bay))
        (crate_at_bay ?c)
        (not (crate_on_mover ?m1 ?c))
        (not (crate_on_mover ?m2 ?c))
        (free ?m1)
        (free ?m2))
)
```

2.4.7 start charging

This action is used to start the charging process. When the mover is at the bay and its charge isn't full, the recharging process is started using the *recharging* predicate.

```
(:action start_charging
    :parameters (?m - mover)
    :precondition (and
        (mover_at_bay ?m)
        (< (charge ?m) (max_charge))
    )
    :effect (and
        (not (free ?m))
        (recharging ?m))
)
```

2.4.8 load crate expensive

This action involves the expensive loader. If a crate is at the bay and the expensive loader isn't carrying anything, the loader start loading this crate in the conveyor belt. This function activates the *load_to_conveyor* process.

```
(:action load_crate_expensive
    :parameters (?l - expensive ?c - crate)
    :precondition (and
        (crate_at_bay ?c)
        (not (loading_crate ?l))
```

```
        )
    :effect (and
        (assign (loading_time ?l) 0)
        (free_bay)
        (not (crate_at_bay ?c))
        (loading_crate ?l)
        (crate_on_loader ?l ?c))
)
```

2.4.9 load crate cheap

This action is the same as the *load_crate_expensive*, but it involves the cheap loader. Since the cheap loader can only load light crates, a check on the weight of the crate is required.

```
(:action load_crate_cheap
    :parameters (?l - cheap ?c - crate)
    :precondition (and
        (<= (weight ?c) (max_light_weight))
        (crate_at_bay ?c)
        (not (loading_crate ?l)))
    :effect (and
        (assign (loading_time ?l) 0)
        (free_bay)
        (not (crate_at_bay ?c))
        (loading_crate ?l)
        (crate_on_loader ?l ?c))
)
```

2.4.10 need recharge one mover

It can happen that the battery level isn't enough to carry a crate to the loading bay in just one movement. This action is executed whenever the mover is near the loading bay but hasn't enough energy to arrive to the bay. This means that $charge = 1$ and the mover distance is greater then the distance that it can cover in one second. When this happens, the crate is released in the warehouse, and the mover goes towards the recharging station. ASSUMPTION: There isn't a crate so heavy and so distant that, when the mover has charge = 1, it will be at distance > 10.

```
(:action need_recharge_one_mover
    :parameters (?m - mover ?c -crate)
    :precondition (and
        (not (mover_together ?m))
        (> (distance ?m) (/ 100 (weight ?c)))
        (= (charge ?m) 1)
        (going_to_bay ?m)
        (crate_on_mover ?m ?c))
    :effect (and
        (assign (distance ?c) (distance ?m))
        (not (crate_on_mover ?m ?c))
        (going_to_recharge ?m)
        (crate_at_warehouse ?c)
    )

)
```

2.4.11 need recharge two movers

This action is very similar to the previous one, and it's used in the same conditions, when the two movers are moving together and one hasn't enough charge to go to the bay. In this case, just for simplicity, all the two movers are sent to the recharging station. It's valid the same assumption as need_recharge_one_mover.

```
(:action need_recharge_two_movers
    :parameters (?m1 ?m2 - mover ?c -crate)
    :precondition (and
        (> (distance ?m1) (/ 100 (weight ?c)))
        (or

            (= (charge ?m1) 1)
            (= (charge ?m2) 1)
        )
        (crate_on_mover ?m1 ?c)
        (crate_on_mover ?m2 ?c)
        (mover_together ?m1)
        (mover_together ?m2)
        (going_to_bay ?m1)
        (going_to_bay ?m2)
        (not (= ?m1 ?m2)))
    :effect (and
        (assign (distance ?c) (distance ?m1))
        (not (crate_on_mover ?m1 ?c))
        (not (crate_on_mover ?m2 ?c))
        (not (mover_together ?m1))
        (not (mover_together ?m2))
        (going_to_recharge ?m1)
        (going_to_recharge ?m2)
        (crate_at_warehouse ?c)
    )

)
```

2.5 Processes

2.5.1 Move to crate

This process is triggered by the action *go_to_crate*. When it is in progress, every second, the battery charge of the mover is decreased by one unit, and the distance of the mover from the bay increases by 10. The process continues as long as the battery is greater than zero and the mover has not yet reached the crate.

```
(:process move_to_crate
    :parameters (?m - mover ?c -crate)
    :precondition (and
        (> (charge ?m) 0)
        (going_to_crate ?m ?c))
    :effect (and
        (decrease (charge ?m) #t)
        (increase (distance ?m) (* #t 10))
    )
)
```

2.5.2 Move to bay normal speed

This process is triggered by the action *going_to_bay*. During it's execution, every second, the battery charge of the mover is decreased by one unit, and the distance of the mover from the bay decreases using the following formula, which depends on the weight of the crate:

$$\Delta distance = \frac{100}{weight}\Delta t$$

The process continues as long as the battery is greater than zero and the mover has not yet reached the crate. Obviously the crate considered should be the one that the mover is carrying.

```
(:process move_to_bay_normal_speed
    :parameters (?m - mover ?c - crate)
```

```
    :precondition (and
        (> (charge ?m) 0)
        (crate_on_mover ?m ?c)
        (going_to_bay ?m))
    :effect (and
        (decrease (charge ?m) #t)
        (decrease (distance ?m) (* #t (/ 100 (weight ?c))))
    )
)
```

2.5.3 Move to bay fast speed

This process, very similar to the previous one, is triggered by the action *going_to_bay_fast*. The only difference at this time is the formula used to compute how much the distance decreases every second, which depends always on the weight of the crate:

$$\Delta distance = \frac{150}{weight}\Delta t$$

This formula is used only when two movers are carrying together a light crate.

```
(:process move_to_bay_fast_speed
    :parameters (?m - mover ?c - crate)
    :precondition (and
        (> (charge ?m) 0)
        (crate_on_mover ?m ?c)
        (going_to_bay_fast ?m))
    :effect (and
        (decrease (charge ?m) #t)
        (decrease (distance ?m) (* #t (/ 150 (weight ?c))))
    )
)
```

2.5.4 wait bay free

This process is triggered by the *wait_for_free_bay* predicate, and it simply waits until the bay becomes free.

```
(:process wait_bay_free
    :parameters (?m - mover)
    :precondition (wait_for_free_bay ?m)
    :effect ()
)
```

2.5.5 recharge battery

This process just recharges the robot battery, increasing the charge level by a unit every time instant.

```
(:process recharge_battery
    :parameters (?m - mover)
    :precondition (and
        (recharging ?m)
    )
    :effect (increase (charge ?m) #t)
)
```

2.5.6 load to conveyor

This process simulates the time required to load a crate on the conveyor belt, depending on its type. Every second the loading time of the loader is increased, and this process will stop when $loading\_time = 4$ if the crate is fragile, otherwise when $loading\_time = 6$ if the crate is not fragile.

```
(:process load_to_conveyor
    :parameters (?l - loader)
    :precondition (and
        (loading_crate ?l))
    :effect (and
        (increase (loading_time ?l) #t)
    )
)
```

2.5.7 move to charge

This process is triggered using the *going_to_recharge* predicate, and every second it reduces the mover charge while decreasing his distance from the bay by 10 units. In our domain, this process is executed for just one second.

```
(:process move_to_charge
    :parameters (?m - mover)
    :precondition (and
        (> (charge ?m) 0)
        (going_to_recharge ?m))
    :effect (and
        (decrease (charge ?m) #t)
        (decrease (distance ?m) (* #t 10))
    )
)
```

2.6 Events

2.6.1 Robot at crate

This event stops the process *move_to_crate*. It happens when a mover, that is going to a crate, reaches it. **WARNING:** If the distance between the crate to be reached and the bay is not a multiple of 10, then this condition (distance of crate = distance of mover) will never be verified and the robot will continue to go on indefinitely.

```
(:event robot_at_crate
    :parameters (?m - mover ?c -crate)

    :precondition (and
        (going_to_crate ?m ?c)
        (= (distance ?m) (distance ?c))
    )
    :effect (and
        (not (going_to_crate ?m ?c))
        (mover_at_crate ?m ?c)
        (free ?m)
    )
)
```

2.6.2 arrived to bay

This event stops the process *move_to_bay* or *move_to_bay_fast*. It happens when a mover that is carrying a crate arrives to the bay, and it's free. Since that the mover distance should be negative (because of the formula that is used to compute the distance made at every time instant), this event sets the distance of the mover to 0.

```
(:event arrived_to_bay
    :parameters (?m - mover ?c -crate)

    :precondition (and
        (free_bay)
```

```
        (crate_on_mover ?m ?c)
        (or
            (going_to_bay ?m)
            (going_to_bay_fast ?m)
        )
        (<= (distance ?m) 0))
    :effect (and
        (assign (distance ?m) 0)
        (not (going_to_bay ?m))
        (not (going_to_bay_fast ?m)))
)
```

2.6.3 arrived to busy bay

This event stops the process *move_to_bay* or *move_to_bay_fast*. It happens when a mover that is carrying a crate arrives to the bay, and it's **not** free. This event also starts a new process that waits until the bay is free, by setting to true the predicate *wait_for_free_bay*. Even in this case, since that the mover distance should be negative, this event sets the distance of the mover to 0.

```
(:event arrived_to_busy_bay
    :parameters (?m - mover ?c -crate)
    :precondition (and
        (not (free_bay))
        (crate_on_mover ?m ?c)
        (or
            (going_to_bay ?m)
            (going_to_bay_fast ?m)
        )
        (<= (distance ?m) 0))
    :effect (and
        (assign (distance ?m) 0)
        (not (going_to_bay ?m))
        (not (going_to_bay_fast ?m))
        (wait_for_free_bay ?m))
)
```

2.6.4 stop waiting for free bay

This event stops the process *wait_bay_free*, and it happens when the mover is currently waiting for the bay to become free and it has become free.

```
(:event stop_waiting_for_free_bay
    :parameters (?m - mover ?c -crate)
    :precondition (and
        (free_bay)
        (wait_for_free_bay ?m))
    :effect (not (wait_for_free_bay ?m))
)
```

2.6.5 charge complete

This event is raised when the mover has fully charged its battery, and stops the process *recharge_battery*.

```
(:event charge_complete
    :parameters (?m - mover)
    :precondition (and
        (recharging ?m)
        (= (charge ?m) (max_charge))
    )
    :effect (and
        (free ?m)
```

```
        (not (recharging ?m)))
)
```

2.6.6 release on conveyor

This event stops the *load_to_conveyor* process if *loading_time* >= 4 and the loader is carrying a non-fragile crate. The crate is then on the conveyor belt, and he needs to be sequenced in order to have some information regarding the crate groups. The last statement corresponds to the predicate *crate_need_sequence*, which is set to True as an effect of this event. Moreover the loading time of the loader is set to 0, and the loader is now able to carry another crate.

```
(:event release_on_conveyor
    :parameters (?l - loader ?c -crate)
    :precondition (and
        (loading_crate ?l)
        (crate_on_loader ?l ?c)
        (not (crate_fragile ?c))
        (>= (loading_time ?l) 4))
    :effect (and
        (not (loading_crate ?l))
        (not (crate_on_loader ?l ?c))
        (crate_on_conveyor ?c)
        (crate_need_sequence ?c)
    )
)
```

2.6.7 release on conveyor fragile

This event is just the same as the previous one, except that it stops the *load_to_conveyor* process if *loading_time* >= 6 and the loader is carrying a fragile crate.

```
(:event release_on_conveyor_fragile
    :parameters (?l - loader ?c -crate)
    :precondition (and
        (loading_crate ?l)
        (crate_on_loader ?l ?c)
        (crate_fragile ?c)
        (>= (loading_time ?l) 6))
    :effect (and
        (not (loading_crate ?l))
        (not (crate_on_loader ?l ?c))
        (crate_on_conveyor ?c)
        (crate_need_sequence ?c)
    )
)
```

2.6.8 create sequence

This event take place when a crate is on the conveyor belt and it is the first that has been released here. Using this event, a sequence of how the crates are released on the conveyor belt is correctly initialized, and the input crate becomes the first element of the sequence.

```
(:event create_sequence
    :parameters (?c - crate)
    :precondition (and
        (wait_first_crate)
        (crate_on_conveyor ?c))
    :effect (and
        (not (wait_first_crate))
        (last_crate_loaded ?c)
        (not (crate_need_sequence ?c))
    )
```

```
)
```

2.6.9 manage sequence

This event happens whenever a new crate has been released on the conveyor belt and needs to be put in the sequence. It takes as inputs the last crate of the sequence and the new crate that isn't in the sequence yet. The two crates are put in sequence using the *subsequent_crates* predicate. After that, the new crate becomes the last element of the sequence, in order to associate it with the next crate that will arrive.

```
(:event manage_sequence
    :parameters (?c1 ?c2 - crate)
    :precondition (and
        (crate_need_sequence ?c2)
        (last_crate_loaded ?c1)
        )
    :effect (and
        (not (crate_need_sequence ?c2))
        (subsequent_crates ?c1 ?c2)
        (subsequent_crates ?c2 ?c1)
        (last_crate_loaded ?c2)
        (not (last_crate_loaded ?c1))
    )
)
```

2.6.10 mover at charge

This event stops the *move_to_charge* process, and is executed when the mover, who needs to recharge, has arrived to the bay. His arrival in notify using the *mover_at_bay* predicate, which in require to enable the charging process.

```
(:event mover_at_charge
    :parameters (?m - mover)

    :precondition (and
        (going_to_recharge ?m)
        (<= (distance ?m) 0)
    )
    :effect (and
        (not (going_to_recharge ?m))
        (mover_at_bay ?m)
    )

)
```

3 Problems

In every problem, in order to find the plan, it is necessary to initialize the light weight threshold and the maximum amount of battery. Also, it is required to tell the planner that there aren't any crate on the conveyor yet, the movers are free to operate and that the bay is free:

```
(:init

        (wait_first_crate)

        (= (max_light_weight) 50)
        (= (max_charge) 20)


        (free mover1)
        (free mover2)
```

```
        (free_bay)
        ...
    )
```

Then it is required to *encode* all the information of the crates, and it is important to tell that every crate is at the warehouse in its initial state.

For the goal, we ask the planner to find a plan that allows all the crates to be on the conveyor belt, asking that the predicate *crate_on_conveyor* for every crate is true. Moreover, in order to manage the *groups* extension, we want that all the crate belonging to a certain group are loaded on the conveyor belt in sequence, using the *subsequent_crates* predicate.

3.1 Problem 1

Crate	**Fragile**	**Weight(kg)**	**Distance**	**Group**
Crate1	No	70	10	None
Crate2	Yes	20	20	A
Crate3	No	20	20	A

3.2 Problem 2

Crate	**Fragile**	**Weight(kg)**	**Distance**	**Group**
Crate1	No	70	10	A
Crate2	Yes	80	20	A
Crate3	No	20	20	B
Crate4	No	30	10	B

3.3 Problem 3

Crate	**Fragile**	**Weight(kg)**	**Distance**	**Group**
Crate1	No	70	20	A
Crate2	Yes	80	20	A
Crate3	No	60	30	A
Crate4	No	30	10	None

3.4 Problem 4

Crate	Fragile	Weight(kg)	Distance	Group
Crate1	No	30	20	A
Crate2	Yes	20	20	A
Crate3	Yes	30	10	B
Crate4	Yes	20	20	B
Crate5	Yes	30	30	B
Crate6	No	20	10	None

4 Performance analysis

4.1 Heuristic Analisys

The tables contain the data for different planner configurations, in order to offer a better overview of the solutions of the problems. Focusing on a particular problem, it is possible to choose the configuration that better suits to the constraints of its request, in terms of computation time needed or memory usage.

Note: (*) after the heuristic name means that with this particular configuration, the planner takes too much time in trying to find the solution. We decided to abort the execution when the planning times exceeded 180s. Aborting the execution, *Elapsed time* and *Heuristic time* are unknown (we represented it with "-").

Note: (**) after the heuristic name means that with this particular configuration, the planner completely filled the java heap space.

Problem 1 with standard planner				
Heuristics	**Elapsed time**	**Planning time(ms)**	**Heuristic time**	**Expanded nodes**
default	26	620	69	107
hradd	26	712	120	242
hmax	24	590	51	122
hrmax	24	646	56	122
hmrp	26	684	74	127
blcost	34	567	0	119

Problem 1 with standard planner, different deltas and heuristics				
Heuristics	**Elapsed time**	**Planning time(ms)**	**Heuristic time**	**Expanded nodes**
default + $\Delta = 0.5$	27.5	1070	398	2469
default + $\Delta = 0.1$	25.8	1464	701	5630
hmax + $\Delta = 0.5$	24	704	97	464
hmax + $\Delta = 0.1$	24.4	2047	1071	20655
blcost + $\Delta = 0.5$	34	516	0	194
blcost + $\Delta = 0.1$	33.4	661	1	791

Problem 2 with standard planner				
Heuristics	**Elapsed time**	**Planning time(ms)**	**Heuristic time**	**Expanded nodes**
default	73	3691	2353	19557
hradd	71	4768	3442	30925
hmax	70	16487	11862	231606
hrmax	67	18211	13913	224315
hmrp	68	23739	17746	377149
blcost	60	586	1	172

Problem 2 with standard planner, different deltas and heuristics				
Heuristics	**Elapsed time**	**Planning time(ms)**	**Heuristic time**	**Expanded nodes**
default + $\Delta = 0.5$	72.5	15356	12060	124515
default + $\Delta = 0.1$*	-	$>$ 180000	-	$>$ 1634138
hradd + $\Delta = 0.5$	69	18816	15094	146834
hradd + $\Delta = 0.1$*	-	$>$180000	-	$>$ 1318038
blcost + $\Delta = 0.5$	60	584	0	294
blcost + $\Delta = 0.1$	60	724	0	1270

Problem 3 with standard planner				
Heuristics	**Elapsed time**	**Planning time(ms)**	**Heuristic time**	**Expanded nodes**
default	129	4512	2792	57832
hradd	130	3425	2187	32848
hmax	124	9142	6545	140062
hrmax	116	10488	7836	142491
hmrp	149	16506	12033	254179
blcost	110	592	0	283

Problem 3 with standard planner, different deltas and heuristics				
Heuristics	**Elapsed time**	**Planning time(ms)**	**Heuristic time**	**Expanded nodes**
default $+ \Delta = 0.5$	127.5	23083	16141	469852
default $+ \Delta = 0.1$*	-	$>$ 180000	-	$>$ 2263982
hradd $+ \Delta = 0.5$	127.5	15763	11833	232399
hradd $+ \Delta = 0.1$*	-	$>$ 180000	-	$>$ 2187915
blcost $+ \Delta = 0.5$	108	666	0	495
blcost $+ \Delta = 0.1$	107.6	663	4	2215

Problem 4 with standard planner				
Heuristics	**Elapsed time**	**Planning time(ms)**	**Heuristic time**	**Expanded nodes**
default	85	43368	35910	266002
hradd	84	27791	23615	123204
hmax*	-	$>$ 180000	-	$>$ 1055202
hrmax*	-	$>$ 180000	-	$>$ 908153
hmrp	87	5730	4099	21995
blcost	66	4839	11	224269

Problem 4 with standard planner, different deltas and heuristics				
Heuristics	**Elapsed time**	**Planning time(ms)**	**Heuristic time**	**Expanded nodes**
hradd + $\Delta = 0.5$	81.5	158198	139144	584097
hradd + $\Delta = 0.1$*****	-	$>$ 180000	-	$>$ 396261
hmrp + $\Delta = 0.5$	79	136135	113554	868542
hmrp + $\Delta = 0.1$	-	$>$ 180000	-	$>$ 963568
blcost + $\Delta = 0.5$	66	27351	85	1562085
blcost + $\Delta = 0.1$******	-	$>$ 137000	-	$>$ 4627528

4.2 Optimal plan

We also tried to find the optimal plan for the four problems. ENHSP has three different optimal planners in its software: *opt-blind,opt-hmax* and *opt-hrmax*. We were able to find the optimal plan for just the first three exercises: all the three optimal planners completely filled the java heap space during their execution.

Here's the data that we collect during the executions.

Note: (**) after the heuristic name means that with this particular configuration, the planner completely filled the java heap space.

Find optimal plan for problem 1			
Planner	**Elapsed time**	**Planning time(ms)**	**Expanded nodes**
opt_blind	20	5025	275169
opt_hmax	24	2241	19327
opt_hrmax	23	2787	19289

Find optimal plan for problem 2			
Planner	**Elapsed time**	**Planning time(ms)**	**Expanded nodes**
opt_blind**	-	$>$ 149000	$>$ 3603447
opt_hmax	52	45006	583884
opt_hrmax	52	45618	583552

Find optimal plan for problem 3			
Planner	**Elapsed time**	**Planning time(ms)**	**Expanded nodes**
opt_blind**	-	> 378000	> 3394858
opt_hmax	52	45286	630152
opt_hrmax	52	52048	629609

Find optimal plan for problem 4			
Planner	**Elapsed time**	**Planning time(ms)**	**Expanded nodes**
opt_blind**	-	> 379000	> 2913608
opt_hmax**	-	> 1250000	> 2454065
opt_hrmax**	-	> 976000	> 2453388

For the first problem we find that the best optimal plan was found using the *opt_ blind* planner, which however wasn't able to find a optimal plan for problems two and three.
For the second an the third problem, *opt_ hmax* and *opt_ hrmax* gives the same result in almost the same time (in problem 3 hmax gives the result seven seconds faster then hrmax) and using the same amount of memory.

4.3 Conclusions

The planner ENHSP with the code of this project always succeeded in founding a plan for the given problems. In general, as you can see from the table reported in the section 3.5, the best heuristic in terms of planning time and expanded nodes resulted *blcost*.
The output files obtained with satisfiability and optimal planners are attached to this documentation.