Report

1. Description of parts of the Model:

I used the extension for fragile objects - crates. Moving by movers and loading by loaders costs time, which increases. The goal is to load objects and minimize time during this action. While loading any other object cannot be on the floor of loading_bay. The actions “pick up” and “put down” don’t cost time.

I have four types:
- PL means place,
- OB object - crate,
- MV mover
- LD loader.

Predicates and functions are also descripted in the PDDL domain.

Predicates:

(fragile ?o - OB)	- Objects can be fragile
(loaded ?o - OB)	- Objects have to be loaded to the conveyor
(at_robby ?m - MV ?p - PL)	- The mover is in place
(at_loader ?l - LD ?p - PL)	- Loader is in place
(at_object ?o - OB ?p - PL)	- The object is in place
(free ?m - MV)	- The mover is free
(ready ?l - LD)	- The loader is ready
(carrying ?m - MV ?o - OB)	- The mover is carrying object
(different ?m1 - MV ?m2 - MV)	- The two grippers are different
(empty_bay ?p - PL)	- While loading the place cannot have objects- crates on the floor

Functions:

(time)	- To measure units of time
(distance ?from - PL ?to - PL)	- Gives us distance between places
(mass ?o - OB)	- Gives weight of the objects

Actions:
- **move_without** - allows the mover to move from one place to another place, only if it doesn't carry any object. Also this action costs time that we increase. Function of time: 0.1 * (distance).
- **move_with_one** - allows one mover to move from one place to another place if only it carries a light object (the second mover doesn’t move and doesn’t carry the object). Cannot move to the loading bay if another object is there. Costs time. Function of time: 0.01 * (distance) * (weight).
- **move_with_two_light** - allows movers to move from place to another place, if they carry a light object. Cannot move to the loading bay if another object is there. This

action costs time, but this function costs less time than using action move_with_one. Function of time: 0.015 * (distance) * (weight).

- **move_with_two_heavy** - allows movers to move from place to another, if they carry a heavy object. Cannot move to the loading bay if another object is there. This action costs time. Function of time: 0.01 * (distance) * (weight)
- **move_with_fragile** - allows movers to move from place to place, if it carries a fragile object. Cannot move to the loading bay if another object is there. This action costs time and doesn't matter the weight of the object. Function of time: 0.01 * (distance) * (weight).
- **pickup_one** - allows one mover to pick up the object, if they are in the same place and the object isn't fragile and the mover is free.
- **pickup_two** - allows both movers to pick up the object if they are in the same place and the object isn't fragile and movers have to be free.
- **pickup_fragile** - allows movers to pick up the object if they are in the same place and the object is fragile and movers are free.
- **putdown_one** - allows the mover to put down an object to place if it carries the object and object isn't fragile.
- **putdown_two** - allows both movers to put down an object to place if they carry the object and the object is not fragile.
- **putdown_fragile** - allows both movers to put down an object to place if they carry the object and the object is fragile.
- **loading_object_normal** - allows the loader to load an object (which is the goal for this model) on the conveyor, if the loader is ready to do it, the object and loader are in the same room and the object isn't fragile and loaded. This action costs time and doesn't matter the weight of the object. Function of time: increase time by 4.
- **loading_object_fragile** - allows the loader to load an object (which is the goal for this model) on the conveyor, if the loader is ready to do it. The object and loader are in the same room and the object is fragile and isn't loaded. This action costs time and doesn't matter the weight of the object. Function of time: increase time by 6.
- **reloading_loader** - this action allows the loader that was used to load an object to reset itself to clear the bay and become ready.

2. Analysis of the performance of the planning engine:

   I used the ENHSP engine for Numeric planning - PDDL 2.1, it is necessary to download and install this engine with java to compile my domain and problem.

   Problem 1.

   In this instance the solution generated *expanded nodes = 169* and *evaluated states = 295*. Solution came quickly without problems. Plan has *21 steps*, but it could have 19 steps. In this instance, the planner makes one mistake by picking up an object by one mover and then putting it down in the same place, so the second mover can come and pick it up with the first mover. It happens because picking up and putting down operations doesn't cost time, which we minimize.

   Problem 2.

   In this instance the solution generated *expanded nodes = 332* and *evaluated states = 892*. Again the solution came quickly without problems. Plan has *29 steps*, but there could be less. It should have about 25 steps. Again in this instance, because of operations pick up and put down that don’t cost time, planner makes 1 unnecessary pick up and put down in the same place to wait for another mover to pick it up together. However in this instance planner have another problem, because of which minimization isn't perfect. It makes a mistake in step 18 by moving from loading_bay to place without object and this operation costs time. In result it also needs to go back to the loading bay and from there move to place with the object. So this mistake cost 2 steps of plan and time.

Problem 3.

In this instance the solution generated *expanded nodes = 645* and *evaluated states = 1352*. The solution came quickly without any problems. Plan is perfect and properly minimized. Solution to this problem is the best possible one. Plan has *26 steps*.

Problem 4.

In this instance the solution generated *expanded nodes* = *2160* and *evaluated states = 5025*. It takes more time to generate this solution, than in previous instances. It is because the problem is more complicated. Again in this instance the plan is perfect and minimized well. Solution to this problem is best possible. Plan has *39 steps*.