Artificial Intelligence for Robotics II

Assignment 1 report

Matteo Maragliano, Mattia Musumeci, Davide Leo Parisi,
Daniele Martino Parisi, Sara Sgambato

14 May 2022

Chapter 1

Introduction

The project consists in the planning of robots in a warehouse using an *Artificial Intelligence* approach. The goal is to find a sequence of instructions to synchronize two types of robots, movers and loaders, to move a certain number of crates from their initial position to the loading bay and then load these creates onto a single conveyor belt; the conveyor belt is located at the loading bay.

The two types of robots have the following features:

- **mover**:
  1. it is initially located at the loading bay;
  2. it is able to pick up and move crates around the environment;
  3. it is able to put down a grabbed crate at the loading bay only if a loader is not operating.
- **loader**:
  1. it is always located at the loading bay;
  2. it is able to grab and load crates left at the loading bay onto the conveyor belt.

The ones just descried are the base-actions the problem has to implement in its algorithm; there are some optional extensions that may lead to a better but also more complex solution. In particular:

1. **"Some crates go together"**: some of the crates from the warehouse need to be loaded subsequently on the conveyor belt because they belong to the same order that is leaving. These that need to go together are characterised by the same letter so that the movers and the loader can identify them.
2. **"There are 2 loaders"**: there is a second additional loader which uses the same loading bay as the other one and it can be used while the first is already busy in loading; there is a specification for this additional robot because it is able to grab only light crates.
3. **"Mover robots need recharging"**: 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 actively doing something, for example moving around or moving crates. To recharge, a robot needs to be located at the loading bay and the recharging process does not interfere with loading processes.
4. **"This is fragile!"**: there are some crates that are considered to be fragile, so they need much care; fragile crates always need two movers to be taken to the loading bay and the loader robot works at reduced speed to avoid any potential damage to its content.

1.1 Folder organization

The implementation can be found in these folders:

- *SolutionNoExtensions*: base implementation of the assignment without any extensions;
- *SolutionFullExtensions*: implementation of the assignment with all the extensions.

Chapter 2

Implementation

The code for the automated warehouse is written in *PDDL+* and the used planner is *ENHSP*.
The implementation is divided into:

- domain: all the possibilities are defined here;
- problems: specific situations for which the planner has to find a solution.

2.1 Domain

The domain is divided into:

- *requirements*: section that allows the inclusion of PDDL extensions;
- *types*: definition of the object types used in the problem;
- *predicates*: definition of predicated (which can be interpreted as boolean variables);
- *functions*: definition of fluents (which can be interpreted as decimal variables);
- *actions*: transformations of the state of the world that the planner can exploit to find a plan;
- *processes*: can be thought of as uncontrollable durative actions;
- *events*: can be thought of as instantaneous durative actions;

2.1.1 Actions

Since we worked with the planner *ENHSP*, which does not support durative actions, we had to model these actions with a combination of processes and events. In this assignment four actions are defined:

- retrieve: an action used to make a single mover robot move to a crate, grab it and then bring it at the loading bay.
- retrieve2: basically it has the same role of the simple 'retrieve' action, but it is implemented to make two movers collaborate to bring fragile crates to the loading bay, because they need extra care when being moved around;
- recharge: this is an optional action.
  Like in the real world, the robot's battery runs down during time. With this add-on we want to simulate the process of charging and discharging a battery. For each mover the spot for the recharge is the loading bay. Every time it arrives at the loading bay, the battery is refilled.
- load: with this action the crates are loaded onto the conveyor belt.

First implementation

The first iteration of the domain implementation was meant to be very general, in the sense that it contained more actions to leave more freedom for the planner to be able to find the best solution.

The actions related to a mover robot were:

- move: a mover robot could move to any "object" in the world.
- grab: a mover robot could grab a crate.
- leave: a mover robot could leave a crate at any position.

The actions related to two movers robots were:

- collaborate: two movers could start collaborating.
- move2: two movers could move together to any "object" in the world.
- grab2: two movers could collaborate to grab a crate.
- leave2: two movers could leave a crate at any position.

The actions related to a loader robot were:

- grab: a loader robot could grab a crate.
- load: a loader robot could load a grabbed crate.

The problem with this implementation was that leaving this much freedom of choice to the planner caused the search time to increase up to the point where it was impossible to find a solution in a limited amount of time. Because of this problem we decided to implement a more efficient solution which would simplify the job of the planner.
The solution we adopted was to join different general actions into a more specific one.

The actions related to the movers robot became:

- retrieve: an action which represents the previous sequence of actions:
  - move to crate.
  - grab the crate.
  - move to loading bay.
  - leave create at loading bay.
- retrieve2: an action which represents the previous sequence of actions:
  - start the robot collaboration.
  - move together to crate.
  - grab together the crate.
  - move together to loading bay.
  - leave together the crate at loading bay.

The actions related to a loader robot became:

- load: an action which represents the previous sequence of actions:
  - grab a crate.
  - load the grabbed crate.

Chapter 3

Results

3.1 Observation

As it can be seen from the simulations results, the planner takes lot of time to find a plan for the problems. This is due to the optional extensions which, as previously said, make the problems more difficult to be solved.
For the first three problems the planner was able to find an optimal solution, but for the fourth we had to opt only for a satisfactory solution because, since it is extremely complex, it leads the program to run out of the available memory.
To give a general overview of the results we decided to run all the problems once with full extensions and another time without any of them; for the full extensions we also provided a satisfactory solution and an optimal one for each problem.

3.1.1 Recharge action

The most computationally heavy extension turned out to be the *"Mover robots need recharging"*. This is related to the fact that the planner has to search among a higher number of possibilities with respect to the other optional extensions: the mover, for example, has to take care of the battery status to move to a crate and to bring it to the loading bay. This check has to be done for all movers, thus leading to a larger tree of possibilities to explore.

3.2 Comparison

In the following section we provided the comparisons between the different solutions found for each problem.
We provided results for:

- **elapsed time**: time spent for the simulation;
- **search time**: time the planner spent to find the solution;
- **planning time**: search time plus time to get to the solution;
- **heuristic time**: time spent by the heuristic;
- **expanded nodes**: number of nodes actually expanded during the plan search;
- **evaluated states**: number of nodes predicted to be expanded during the plan search.

3.2.1 Full extensions

Problem 1

- **Satisfactory solution**: we obtained a solution with an expansion of *7880* nodes (and *12298* evaluated states) within the time indicated in Tab 3.1:

Elapsed time	*20.0* ms
Search time	*2242* ms
Planning time	*6023* ms
Heuristic time	*2163* ms

Table 3.1: Problem 1: satisfactory solution.

- **Optimal solution**: we obtained a solution with an expansion of *130861* nodes (and *157267* evaluated states) within the time indicated in Tab 3.2:

Elapsed time	*20.0* ms
Search time	*1038* ms
Planning time	*1239* ms
Heuristic time	*5* ms

Table 3.2: Problem 1: optimal solution.

Problem 2

- **Satisfactory solution**: we obtained a solution with an expansion of *23978* nodes (and *33164* evaluated states) within the time indicated in Tab 3.3:

Elapsed time	*43.0* ms
Search time	*10177* ms
Planning time	*10439* ms
Heuristic time	*9833* ms

Table 3.3: Problem 2: satisfactory solution.

- **Optimal solution**: we obtained a solution with an expansion of *2366888* nodes (and *2489436* evaluated states) within the time indicated in Tab 3.4:

Elapsed time	*36.0* ms
Search time	*18493* ms
Planning time	*18712* ms
Heuristic time	*75* ms

Table 3.4: Problem 2: optimal solution.

Problem 3

- **Satisfactory solution**: we obtained a solution with an expansion of *23597* nodes (and *25761* evaluated states) within the time indicated in Tab 3.5:

Elapsed time	*74.0* ms
Search time	*8243* ms
Planning time	*8492* ms
Heuristic time	*7958* ms

Table 3.5: Problem 3: satisfactory solution.

- **Optimal solution**: we obtained a solution with an expansion of *1992504* nodes (and *2018291* evaluated states) within the time indicated in Tab 3.6:

Elapsed time	*72.0* ms
Search time	*14640* ms
Planning time	*14854* ms
Heuristic time	*56* ms

Table 3.6: Problem 3: optimal solution.

Problem 4

Problem 4 is more complex than the previous three so the planner was not able to find an optimal solution with all the extensions enabled.
In order to find a solution we opted for a general solution, that is one of the possible solution the planner can find to solve the plan and to complete it.

- **Satisfactory solution**: we obtained a solution with an expansion of *2096372* nodes (and *2999498* evaluated states) within the time indicated in Tab 3.7:

Elapsed time	*50.0* ms
Search time	*2192976* ms
Planning time	*8566871* ms
Heuristic time	*2136611* ms

Table 3.7: Problem 4: satisfactory solution.

It can be clearly seen that, despite the solution not being the optimal one, the planner took a lot of time to perform and obtain a solution.

3.2.2 No extensions

Problem 1

- **Optimal solution**: we obtained a solution with an expansion of *2485* nodes (and *2486* evaluated states) within the time indicated in Tab 3.8:

Elapsed time	*19.0* ms
Search time	*56* ms
Planning time	*223* ms
Heuristic time	*0* ms

Table 3.8: Problem 1: optimal solution.

Problem 2

- **Optimal solution**: we obtained a solution with an expansion of *10183* nodes (and *10187* evaluated states) within the time indicated in Tab 3.9:

Elapsed time	*30.0* ms
Search time	*147* ms
Planning time	*324* ms
Heuristic time	*0* ms

Table 3.9: Problem 2: optimal solution.

Problem 3

- **Optimal solution**: we obtained a solution with an expansion of *8016* nodes (and *8018* evaluated states) within the time indicated in Tab 3.10:

Elapsed time	*48.0* ms
Search time	*116* ms
Planning time	*287* ms
Heuristic time	*0* ms

Table 3.10: Problem 3: optimal solution.

Problem 4

- **Satisfactory solution**: we obtained a solution with an expansion of *116783* nodes (and *130029* evaluated states) within the time indicated in Tab 3.11:

Elapsed time	*32.0* ms
Search time	*23714* ms
Planning time	*41184* ms
Heuristic time	*22847* ms

Table 3.11: Problem 4: satisfactory solution.

- **Optimal solution**: we obtained a solution with an expansion of *1125595* nodes (and *1125599* evaluated states) within the time indicated in Tab 3.12:

Elapsed time	*28.0* ms
Search time	*7079* ms
Planning time	*7260* ms
Heuristic time	*31* ms

Table 3.12: Problem 4: optimal solution.