Assignment for the course Artificial Intelligence

Prof. Francesco Scarcello
Ing. Antonio Bono

Academic year 2023/2024

Introduction

The aims of this assignment are threefold:

Task 1: use the PDDL language to model a classical planning problem (7 points);

Task 2: implement your own search algorithm and heuristics for solving the provided instances of the problem (16 points).

Task 3: show how a temporal model could be integrated in a real-world robotic software architecture (7 points).

For the second task, students can implement the algorithm by whatever means they find most appropriate (an example is the `PDDL4J` library [3]). For the third task, the student will refer to `planutils` [2] to solve the problem and the `PlanSys2` [1] package presented during the lectures and available at https://github.com/PlanSys2/ros2_planning_system. All the mentioned software and tools are already installed in the VM provided in the Teams course page.

This assignment can be performed alone or in group of *at most three* students.

1 Modeling

Let us consider a scenario inspired by industrial manufacturing services where there are a number of work stations at given known locations that to perform the assigned work need something to be positioned at their premises. The objective of the planning systems is to orchestrate the activities of a set of different robotic agents to deliver boxes containing needed supplies to each work station. Let's consider these assumptions:

1. Each work station is at a specific location.
2. Each box is initially at a specific location (e.g., the warehouse) and can be filled with a specific content such as valve or bolt or tool, if empty. Box contents shall be modeled in a generic way, so that new kind of contents can easily be introduced in the problem instance (i.e., we shall be able to reason on the content of each box).
3. Each work station has or does not have a box with a specific content. That is, you must keep track of whether they have e.g., valves or not, whether they have bolts or not, whether they have tools or not, and so on.
4. There can be more than one work station at any given location, and therefore it isn't sufficient to keep track of where a box is in order to know which work station have been given boxes!
5. Each robotic agent can:
   (a) fill a box with a content, if the box is empty and the content to add in the box, the box and the robotic agent are at the same location;
   (b) empty a box by leaving the content to the current location and given work station, causing the work station to then have the content;
   (c) pick up a single box and load it on the robotic agent, if it is at the same location as the box;
   (d) move to another location, considering that if the robotic agent is loaded with a box, then also the box moves with the

agent itself, and if the agent is not loaded with a box, then only the agent moves to the new location;

(e) deliver a box to a specific workstation who is at the same location.

6. The robotic agents can move only between connected locations as typical of an industrial plant (there is a "road-map" with specific connections that must be taken).
7. Since we want to be able to expand this domain for multiple robotic agents in the future, we expect to use a *separate type* for robotic agents, which currently happens to have a single member in all problem instances.

Model such a scenario using the PDDL 1.2 language.

2 Classical Planning

Use *your own planner* to solve the following problem instances. Your solution must report the amount of time used (max 10 minutes to be considered), the numbers of states evaluated and optionally the amount of used memory.

2.1 Instance 1

2.1.1 Initial Condition

- Initially all boxes are located at a single location that we may call the central warehouse.
- All the contents to load in the boxes are initially located at the central warehouse.
- There are no work stations at the central warehouse.
- A single robotic agent is located at the central warehouse to deliver boxes.

There are no particular restrictions on the number of boxes available, and constraints on reusing boxes! These are design modeling choices left unspecified and that each student shall consider and specify in her/his solution.

2.1.2 Goal

The goal should be that:

- certain work station have certain supplies (e.g., bolt, valve, tool)
- some work station might not need supply (e.g., bolt, valve, tool)
- some work station might need several supplies (e.g., bolt and valve, or bolt and tool, or three of them and so on)

This means that the robotic agent has to deliver to needing work stations some or all of the boxes and content initially located at the central warehouse, and leave the content by removing the content from the box (removing a content from the box causes the work station at the same location to have the unloaded content).

Remark: work stations don't care exactly which content they get, so the goal should not be (for example) that `WorkStation1` has `bolt3` and `valve5`, merely that `WorkStation1` has a `bolt` and a `valve`.

2.2 Instance 2

Use your own planner to solve the same problem of Instance 1 with the following extensions, where we will have to consider an alternative way of moving boxes (e.g., drones or different kind of robots), and the fact that the robotic agents have a carrier to load the boxes.

- each robotic agent has a carrier with a maximum load capacity (that might be different from each agent);
- the robotic agents can load boxes in the carrier up to the carrier maximum capacity;
- the robotic agent and the boxes to be loaded in the robotic agent carrier shall all be at the same location;
- the robotic agent can move the carrier to a location where there are work stations needing supplies;
- the robotic agent can unload one or more box from the carrier to a location where it is;
- the robotic agent may continue moving the carrier to another location, unloading additional boxes, and so on, and does not have to return to the `central_warehouse` until after all boxes on the carrier have been delivered;

- though a single carrier is needed for the single robotic agent, there should still be a separate type for carriers;
- for each robotic agent we need to count and keep track of i) which boxes are on each carrier; ii) how many boxes there are in total on each carrier, so that carriers cannot be overloaded.
- the capacity of the carriers should be problem specific. Thus, it should be defined in the problem file.

It might be the case additional actions, or modifications of the previously defined actions are needed to model loading/unloading/moving of the carrier (one can either keep previous actions and add new ones that operates on carriers, or modify the previous actions to operate on carriers). The initial condition and the goal are the same as the problem discussed in Section 2.1

2.2.1 Initial Conditions

- Initially all boxes are located at a single location that we may call the `central_warehouse`.
- All the contents to load in the boxes are initially located at the `central_warehouse`.
- There are no work station needing supplies at the `central_warehouse`.
- The robotic agents are located at the `central_warehouse` to deliver boxes.
- Each robotic agent is initially empty.
- Fix the capacity of each robotic agent to be a value greater than 1.

2.2.2 Goal

The goal should be that

- certain work stations have certain supplies (e.g. bolt, tool, valve);
- some workstation might not need supplies;
- some workstation might need more than one supply;

3 Temporal Planning & Robotics

3.1 Temporal Planning

We leverage the problem instance reported in paragraph 2.2 with the following extensions.

- Convert the domain defined in paragraph 2.2 to use durative actions. Choose arbitrary but reasonable durations for the different actions.
- Consider the possibility to have actions to be executed in parallel when this would be possible in reality. For example, a robotic agent cannot pick up several boxes at the same time, or pick up a box and if it is a drone fly to a destination at the same time.

The initial condition and the goal are the same as the problem discussed in Section 2.2. To solve this problem *you should use one of the temporal planners provided by* `planutils` (popf, tfd, etc.).

3.2 Robotics Planning

The final problem consists in implementing within the `PlanSys2` the last problem resulting as outcome of Section 3.1 using fake actions as discussed in the tutorials of `PlanSys2` available at [4].

Remark: To generate the plan do not use the inner temporal solvers of `PlanSys2` but rely on the plan you generated in the previous step. In such a way you do not have to face the current limitations of `PlanSys2` as the the inability to use hierarchical types in the domain definition. In order to use the generated plan in `PlanSys2` it must have the same sintax of a plan generated by popf, i.e.,

```
<time>:  (<action>) [<duration>]
```

Once you edit correctly your plan, it is sufficient to create the problem instance with the `PlanSys2` terminal commands (possibily sourcing a file with them listed) and then execute

```
run plan-file <your_plan.txt>
```

If everything is well settled, you will see the actions progress on the terminal.

4 Deliverables

The deliverable shall consist of a unique archive containing at least the following contents:

1. The PDDL files (domain and problems) for each of the problems discussed in Sections 2.1, 2.2 and 3.1.

2. The *commented* code of your planner with the instructions to run in on the proposed problem instances.

3. A folder containing all the code to execute the `PlanSys2` problem as discussed in Section 3.2.

4. A report in PDF describing the approach followed, justifying and document in all the design choices of both the modeling and the solver, possible additional assumptions (for the part left underspecifed), and in case of deviation from the specification in this document a clear justification. Finally, the document shall include evidence of the attempts to solve the PDDL problems formulated, and running the final version within `PlanSys2` (in form of screenshots and/or output generated by the planner).

    - The report shall also discuss the content of the archive and how it has been organized.

    - The report shall also contain a critical discussion of the results obtained.

References

[1] Francisco Martın et al. "PlanSys2: A Planning System Framework for ROS2". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - October 1, 2021*. IEEE, 2021.

[2] Cristian Muise and other contributors. *PLANUTILS*. General library for setting up linux-based environments for developing, running, and evaluating planners. 2021. URL: https://github.com/AI-Planning/planutils.

[3] D. Pellier and H. Fiorino. "PDDL4J: a planning domain description library for java". In: *Journal of Experimental & Theoretical Artificial Intelligence* 30.1 (2018), pp. 143–176. URL: http://pddl4j.imag.fr/.

[4] Francisco Martin Rico and colleagues. *Plansys2 tutorials*. 2022. URL: https://plansys2.github.io/tutorials/index.html.