# AUTOMATED PLANNING THEORY AND PRACTICE PROJECT[*]

Final Assignment of Automated Planning Theory and Practice course[†]

Manh Tuan NGUYEN[‡]
Master Student
Second year of Artificial Intelligence
Avignon University - Trento University
manhtuan.nguyen@studenti.unitn.it
manh-tuan.nguyen@alumni.univ-avignon.fr

Kaleem Ullah Kaleem Ullah[§]
Master Student
Second year of Artificial Intelligence
Trento University
kaleemullah.ullah@studenti.unitn.it

## ABSTRACT

This project contains 4 different problems. Each problem is built base on the previous one. With this project, students have opportunity to apply the knowledge about AI planning with PDDL and Plansys2. This project is a part of *Automated Planning Theory and Practice Course* of *Prof.Marco Roveri*

## 1 INTRODUCTION

Artificial Intelligence (AI) is now one of the most important field in the world. This field brings a lot of potential for human future. This project is aiming for the sub-field of AI - AI planning. AI Planning is used to explore the process of using autonomous techniques to solve planning and scheduling problems. This project is assumed a real life situation where we have certain people at different place who are injured and need some stuff to keep them alive. A robot agent will deliver the necessary stuff to the people. By appling PDDL and PlanSys2 ( a framework is built base on ROS2 ) we manage to solve this problem and output a plan for the robot agent. The project is divided into 4 different problems. Each is built base on the previous one. The source code will be delivered along with this report or you can also find it on github here Each problem will be described detail below

A quick note that since all the problem is a leverage version of the previous one, so we did re-use a lots but we still decided to re-declared all the elements. The new one of each problem will be the last element show in the list. So it will bring the big picture of comparison for reader when they read the report.

## 2 PROBLEM 1

### 2.1 Understanding the problem

The first problem is to make a simple plan where we only involved with one agent who can carry one crate at a time and deliver it to the person who in need. Also we used **fast downward** planner to executed this problem.

### 2.2 Design Choices

*2.2.1 Type Description.* All the subjects involve in the problem will be declared as object type. But since the project want us to be easy to add more type of crate content so i applied the hierarchy knowledge to create the content type as a sub-type of crate. The list of type is here:

- **person** - *object*
- **robot** - *object*
- **crate** - *object*
- **food** - *crate (sub-type )*
- **medicine** - *crate (sub-type )*

*2.2.2 Predicate Description.* With this problem , we come up with 6 different predicates in order to cover all the aspect of the plan. 3 of them is dedicate to position of objects ( localization predicates ), the rest is dedicated to the state of these objects which also review the state of actions ( for example attached will show the state if the crate is attached with agent and also can show the pick up action of the robot agent ) . Below is the list and detail description of each predicate:

- **at** - *True if crate c is at location l*
- **person_at** - *True if person p is at location l*
- **agent_at** - *True if a robot agent is at location l*
- **attached** - *True if a crate c is attached to a robot agent - in another word a robot agent has picked up the crate*
- **person_needs** - *True if a person p is fulfil with their needs crate c - in another words, if a correct crate c has been delivered to the person p*
- **empty** - *True if a robot agent doesn't carry any crate*

*2.2.3 Action Description.* Since the problem has declared that the robot agent can freely move around the place because there is no road map in this problem. This will make the plan is easier to declare. Due to this reason, we came up with 3 different actions is *move_agent, pick_up and deliver* which can cover all the aspect of this problem. Because we only need to know where the robot should go, when to pick up at the depot and when to deliver. The detail description is below:

- **move_agent** - *move a robot agent from 2 adjacent location from and to*
- **pick_up** - *a empty robot agent will pick up a crate c*
- **deliver** - *a carried robot agent will deliver the crate c to a person p*

*2.2.4 Problem Description and Result.* For the problem definition, I create a test case with 4 different people at 4 different locations ( each of them can needs a food or a medicine or don't need anything or need both ) . There are in total 6 different crates divide by 2 types ( food and medicine) is created and at the depot position. A robot agent is empty and also start at depot position.

[*]Produces the permission block, and copyright information
[†]The full version of the author's guide is available as `acmart.pdf` document
[‡]Manh Tuan NGUYEN
[§]Kaleem Ullah Kaleem Ullah

© 2021 Copyright held by the owner/author(s). Published in Proceedings of the 24th International Conference on Extending Database Technology (EDBT), March 23-26, 2021, ISBN 978-3-89318-084-4 on OpenProceedings.org.
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

After run the plan, we got the result of a plan consist of 16 actions ( cost is 16 unit cost ).

## 3 PROBLEM 2

### 3.1 Understanding the problem

This problem is evolved base on the previous one. Inside this problem, there is a new object called carrier which like a container of robot which can be loaded up to 4 different crates at the same time. This problem requires we also need to keep track of the stock of this container. So for this problem, the first idea is using the *numeric fluents* option which has been showed in the lab-session in the class. But since our professor has informed us that the Plansys2 system has problem with numeric fluents requirements so we had to write this plan in normal way by using the mathematics predicate to keep track of stock. This problem we still use the **fast forward** planner.

### 3.2 Design Choices

*3.2.1 Type Description.* This problem re-use every type of previous one but also present new types. First is about the location, since we want to keep track the action that robot can go back to the depot and stock up the container , so we crated 2 different sub-type of location, also we presented the carrier , stock type. The description detail is below:

- **person** - *object*
- **robot** - *object*
- **crate** - *object*
- **food** - *crate (sub-type )*
- **medicine** - *crate (sub-type )*
- **(NEW)carrier** - *object*
- **(NEW)stock** - *object*
- **(NEW)pos** - *location (sub-type ) - this is to indicate all the position of injured people*
- **(NEW)depot** - *location (sub-type ) - indicate the depot position where all the crates are stored and wait to be picked up by robot agent*

*3.2.2 Predicate Description.* We continue to re-use all the predicates from problem 1 ( except empty predicates since it's not necessary to have this. We have tested without empty predicate and the planner still work ). Also we introduce 4 new predicates which will be described below:

- **at** - *True if crate c is at location l*
- **person_at** - *True if person p is at location l*
- **agent_at** - *True if a robot agent is at location l*
- **attached** - *True if a crate c is attached to a robot agent - in another word a robot agent has picked up the crate*
- **person_needs** - *True if a person p is fulfil with their needs crate c - in another words, if a correct crate c has been delivered to the person p*
- **(NEW)inc** - *True if a stock ss is an increase by 1 of stock s*
- **(NEW)dec** - *True if a stock ss is a decrease by 1 of stock s - oposite with inc predicate*
- **(NEW)has_stock** - *True if the carrier cr has the amount of crate inside equal to a stock s*
- **(NEW)carrier_at** - *True if a carrier cr is at a location l*

*3.2.3 Action Description.* Beside re-using all 3 previous actions of problem 1, we introduced new action that is *move_agent_to_depot* so that we can keep track when ever the robot agent needs to be back to depot to stock up the carrier.

- **move_agent** - *move a robot agent from 2 adjacent location from and to*
- **pick_up** - *a empty robot agent will pick up a crate c*
- **deliver** - *a carried robot agent will deliver the crate c to a person p*
- **(NEW)move_agent_to_depot** - *Robot agent will come back to depot to stock up the carrier when they ran out of crate inside carrier*

*3.2.4 Problem Description and Result.* The problem is remaining the same with all the initial conditions and goals. We just need to edit it to adapt with new predicates. So after running the plan, our result is 11 actions or 11 unit cost.

## 4 PROBLEM 3

### 4.1 Understanding the problem

In this problem, we are required to leverage the problem 2 with *durative action*. So at first that should be easy because we only need to change all the actions to be durative one. First we plan to use the **smtplan** because this planner also support not only durative action requirements but also keep support the disjunctive-preconditions and negative-preconditions. But we face a problem that the planner keep output the *Segmentation fault or well known as core dumped problem*. So we have to change the planner to **optic** planner. But this planner doesn't support the disjunctive-preconditions and the negative-preconditions so we have to change our predicate in order to fit with this planner. Inside our source code, we still deliver both optic and smtplan pddl files in order to have a clear view of what is the different. All the information about what planner support what requirements can be found in Planning Wiki All the section below will be described about the change when we use the optic planner.

### 4.2 Design Choices

*4.2.1 Type Description.* We re-used almot every types from the previous one except that we removed the crate sub-type and introduce the new typecrate type one in order to keep track of the type of crate should be delivered.

- **person** - *object*
- **robot** - *object*
- **crate** - *object*
- **carrier** - *object*
- **stock** - *object*
- **pos** - *location (sub-type ) - this is to indicate all the position of injured people*
- **depot** - *location (sub-type ) - indicate the depot position where all the crates are stored and wait to be picked up by robot agent*
- **(NEW)typecrate** - *object - this is to dedicate which is the type of crate when we check with predicate*

*4.2.2 Predicate Description.* Beside of all the predicates that been showed in last 2 problem, we introduced 3 news predicates which is used to check the type of current crate and also the type of crate that people need.

- **at** - *True if crate c is at location l*
- **person_at** - *True if person p is at location l*

- **agent_at** - *True if a robot agent is at location l*
- **attached** - *True if a crate c is attached to a robot agent - in another word a robot agent has picked up the crate*
- **person_needs** - *True if a person p is fulfil with their needs crate c - in another words, if a correct crate c has been delivered to the person p*
- **inc** - *True if a stock ss is an increase by 1 of stock s*
- **dec** - *True if a stock ss is a decrease by 1 of stock s - oposite with inc predicate*
- **has_stock** - *True if the carrier cr has the amount of crate inside equal to a stock s*
- **carrier_at** - *True if a carrier cr is at a location l*
- **(RE-DECLARED)empty** - *True if a carrier cr doesn't carry any crate*

- **(NEW)type**$_n$*eeds* – ***True if a person p need this typecrate t***

*4.2.3 Action Description.* The action is remain the same, only we change it as a durative action and it's really simple. Also we modify a litte to be fit with all the predicates. For the time, we can choose the time randomly ( but need to be logical ) so below we will describe about the time and why

- **move_agent** - *Time is 10s since it should take time to move from a place to another place*
- **pick_up** - *Time is 5s since stock up stuff into a container should take time but will not be long as moving around so we put here 5s*
- **deliver** - *Time is 3s since it will be easy to take 1 crate and give it to person*
- **move_agent_to_depot** - *Time is 10s because it's like the move_agent action in moving*

*4.2.4 Problem Description and Result.* Still declare problem background the same with a modify to fit with all new predicates and types. Since we used 2 different planner so we have 2 different result.

- **optic** - *Successful executed and the result is cost 72*
- **smtplan** - *Fail executed due to Segmentation fault - core dumped problem*

## 5 PROBLEM 4

### 5.1 Understanding the problem

This problem is to practive how to use the **Plansys2** framework. Thanks to *Prof.Roveri* and his Plansys2 example (plansys2_simple_example - we keep the same structure so that if anyone want can re-use this good example ). We managed to input our PDDL file and executed it on Plansys2 - ROS2 based system. The problem is quite easy since the code from professor was clear. We only need to modify and add-on the action declare inside the cpp files and Python launcher. All the command line is declared inside the text file of our repo.

## 6 ACKNOWLEDGMENTS

I would like to express my gratitude to all those who gave me possibility to complete this project in specific as well as academic supporting in general that I received during the time of our studying at the Trento University. I wish to express my sincere thanks to Professor Marco Roveri for providing us with all the necessary facilities for the research.

## ACKNOWLEDGMENTS

To Robert, for the bagels and explaining CMYK and color spaces.

## REFERENCES