Planning and Aproximate Reasoning: Robot Chef Task

María del Carmen Ramírez, Pedro Agúndez and Antonio Lobo.

November 10, 2024

1 Introduction

Planning an effective work schedule in sectors like catering or the restaurant industry, where numerous tasks must be carried out quickly and efficiently, can be essential to achieving final objectives optimally.

In this work, we will focus on the design and implementation of a planner in PDDL (Planning Domain Definition Language) to modelate how a robot performs tasks equivalent to those carried out by waiters and chefs in an Asian food restaurant. Specifically, a total of three problems will be presented along with their respective domains, and an analysis will be conducted on how optimal solutions are obtained in each scenario in order to meet the objectives set for each of the three problems.

2 Analysis of the problem

We studied several problems, each presenting increasing levels of difficulty. In this section, we provide a detailed description of all of them.

The robot is assumed to hold only one object at a time, whether it is an ingredient, a tool, or a prepared dish. Its storage capacity is limited, which constrains its ability to carry objects. Despite this limitation, we assume that the robot can perform various actions, such as assembling, cooking, cutting, or mixing, while holding an object, as it can be placed inside the robot without restricting its arms, which allows it to perform the aforementioned actions.

Regarding the distribution of the restaurant where the robot works, it consists on seven rooms displaced as it is shown in the Figure 1. The rooms are: preparation area (PA), cooking area (CA), serving area (SVA), dishwashing area (DWA), mixing area (MIXA), cutting area (CTA) and storage area (STA).

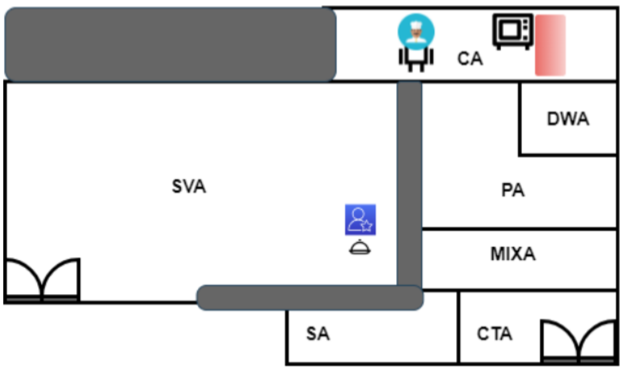

Figure 1: Restaurant structure. Gray areas means that the robot can not move in these areas.

Subsequently, we will introduce the common predicates and actions used across all three scenarios considered in this project.

2.1 Basic problem

It consists of the preparation of a simple sushi recipe. The ingredients were placed in the STA, and each of them had specific requirements to be prepared (e.g., rice needed to be mixed and cooked, fish needed to be cut). Each action was carried out in the area designated for it, and the final goal was to plate the dish in the serving area and return all the tools used to their original place, clean and ready for future use.

The predicates used are listed below.

- (robot-at ?r - robot ?loc - location): Represents the location of a specific robot 'r' in an area of the kitchen 'loc'.
- (ingredient-at ?ingredient - ingredient ?loc - location): This predicate signifies the presence of an 'ingredient' in an area 'loc'.
- (tool-at ?tool - tool ?loc - location): This predicate indicates that the instrument 'tool' is in the area of the kitchen 'loc'.
- (ingredient-prepared ?ingredient - ingredient): This predicate indicates that the ingredient 'ingredient' has been successfully prepared.
- (dish-assembled ?dish - dish): Indicates that the dish 'dish' has been assembled from its ingredients.
- (dish-plated ?dish - dish ?loc - location): Denotes that the dish 'dish' has been plated and placed at location 'loc'.
- (tool-clean ?tool - tool): This predicate denotes that the tool 'tool' is clean.
- (holding-ingredient ?r - robot ?ingredient - ingredient): Indicates that the robot 'r' is holding the ingredient 'ingredient'.
- (holding-dish ?r - robot ?dish - dish): Indicates that the robot 'r' is holding the assembled dish 'dish'.
- (holding-tool ?r - robot ?tool - tool): Indicates that the robot 'r' is holding the tool 'tool'.
- (adjacent ?loc1 - location ?loc2 - location): Indicates that the kitchen areas 'loc1' and 'loc2' are adjacent.
- (used-in ?ingredient - ingredient ?dish - dish): This predicate represents if the ingredient 'ingredient' has been used to prepare the dish 'dish'.
- (need-mix ?ingredient - ingredient): Denotes that the ingredient 'ingredient' requires mixing.
- (need-cook ?ingredient - ingredient): Denotes that the ingredient 'ingredient' requires cooking.
- (need-cut ?ingredient - ingredient): Denotes that the ingredient 'ingredient' requires cutting.

The actions specified are outlined hereafter.

- **pick-up-ingredient:** The robot picks up an ingredient at a specified location.
- **pick-up-tool:** The robot picks up a tool at a specified location.
- **move:** The robot moves from one location to an adjacent location.
- **drop-ingredient:** The robot drops the ingredient it is holding at a specified location.
- **drop-tool:** The robot drops the tool it is holding at a specified location.
- **mix:** The robot mixes an ingredient (e.i. rice) at the mixing area (MIXA), provided the ingredient needs mixing and is not already prepared.
- **cook:** The robot cooks an ingredient (e.i. rice) at the cooking area (CA), provided it has been mixed and needs cooking.
- **cut:** The robot cuts an ingredient at the cutting area (CTA) using a clean tool, provided the ingredient needs cutting.
- **clean-tool:** The robot cleans a tool in the dishwashing area (DWA).

- **assemble-dish:** The robot assembles a dish using prepared ingredients at a specified location.
- **carrying-dish:** The robot carries an assembled dish, provided it is not already holding any other items.
- **plate-dish:** The robot plates an assembled dish at the serving area (SVA).

This case can be found in the attached files $'sushi_simple_domain.pddl'$ and $'sushi_simple_problem.pddl'$.

2.2 Substitution problem

In this scenario, the same sushi recipe as before needs to be plated; nevertheless, an ingredient used in this dish is missing at the STA. Other ingredients were available, and several predicates were given to the robot to inform it about possible substitutions. The robot was able to examine the available ingredients and decide whether to make a substitution in order to complete its task.

The same predicates and actions as those described for the **basic problem** in subsection 2.1 were used. However, one additional predicate and action were included in this PDDL domain to account for the new problem.

Additional predicate:

- (replaceable ?ingredient1 - ingredient ?ingredient2 - ingredient ?*dish* - dish): Represents that 'ingredient1' can be replaced by 'ingredient2' in the preparation of 'dish'.

Additional action:

- **substitution-decision:** The robot makes the decision of changing the 'ingredient1', that is used in the preparation of a dish, by 'ingredient2' when the first one is missing and the second one is suitable for the change.

This case can be found in the attached files $'sushi_substitution_domain.pddl'$ and $'sushi_substitution_problem.pddl'$.

2.3 Priorization order problem

Finally, we step to the stage were the plates were already prepared but they need to be deliver in a certain order. To do so, two functions were created.

The only predicates used from subsection 2.1 are as follows: *robot-at*, *dish-assembled*, *dish-plated*, *holding-dish*, and *adjacent*.

Regarding the actions, the used ones are outlined hereafter.

- **move:** The robot moves from one location to an adjacent location.
- **carrying-dish:** The robot picks up an assembled dish at a specified location, provided it matches the current priority level and the robot is not already holding any other items.
- **plate-dish:** The robot plates an assembled dish at a specified location, provided it matches the current priority level, and then advances the priority level.

The functions used in this problem are specified as follows:

- (priority ?dish - dish): Represents the priority value assigned to each dish, determining the order in which dishes are to be served.
- (current-priority-level): Tracks the current priority level that should be served next, guiding the sequence of dish preparation and serving.

This case can be found in the attached files $'fluents_priority_domain.pddl'$ and $'fluents_priority_problem.pddl'$.

3 Results

In all the scenarios described in the previous section, the results were obtained using a PDDL planner implemented in Julia. Specifically, the A* search algorithm was implemented. The planner initializes the problem state using the initstate function and defines the goals using MinStepsGoal, optimizing for minimal steps to achieve the objectives, while the planning process employs an A∗ planner (AStarPlanner) with the HAddR heuristic to guide the search.

The implementation of Julia was mandatory due to...

The final path obtained for each studied case can be seen in Figure 2, Figure 3 and Figure **??**.

4 Discusion

In this section, it is analyzed the plans obtained in each scenario and how they were achieved. In order to provide a insight into the efficiency and complexity of the searching process, several key performance parameters were analyzed, as it is shown in Table **??**.

Problem	Basic	Substitution	Priorization order
Plan length	41	42	18
Expanded states	33413	34710	2784
Search time (s)	36.0221934	58.560908	0.9147218
Total time (s)	36.0221938	58.5609156	0.9147222

Table 1: Performance of the planners across different test cases.

5 Conclusion

Hola [1]

References

[1] Fu Chang, Chin-Chin Lin, and Chi-Jen Lu. Adaptive prototype learning algorithms: Theoretical and experimental studies. Journal of Machine Learning Research, 7:2125–2148, 2006.

Figure 2: Basic problem final solution.

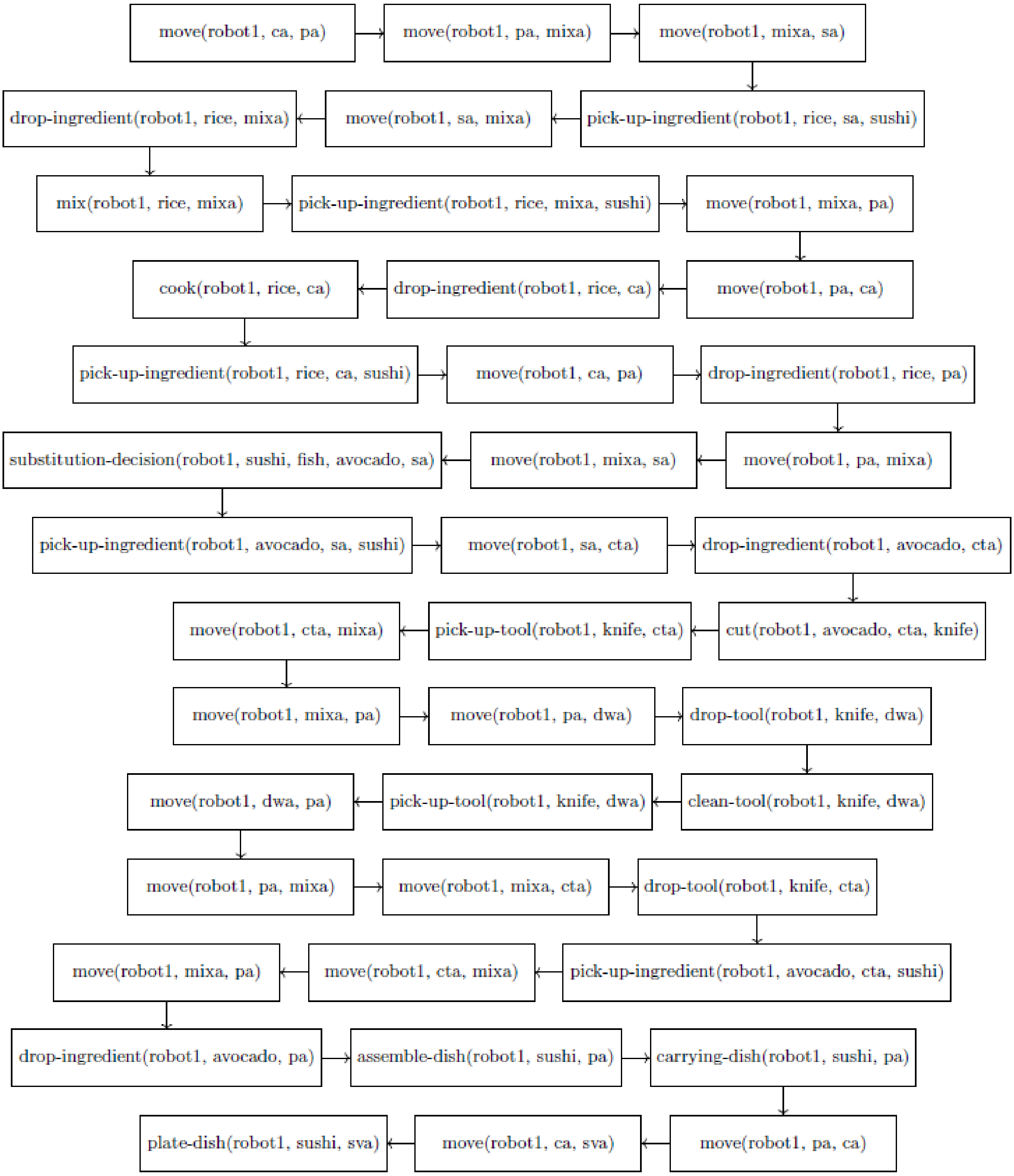

Figure 3: Problem with substitution between ingredients solution.

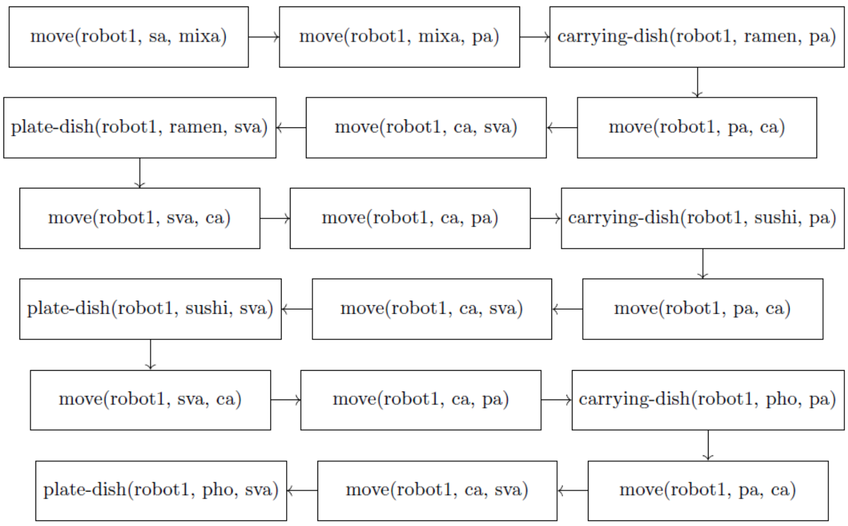

Figure 4: Problem of priorization dishes solution.