# 6.4132/16.413 Principles of Autonomy and Decision Making Final Project

Toya Takahashi

Massachusetts Institute of Technology
Department of Aeronautics and Astronautics

November 30, 2023

## 1 Introduction

Activity planing, motion planning, and trajectory optimization for a robotic arm is challenging. To navigate around an environment, the agent must perform a state space search to plan a sequence of actions and generate a trajectory by either using a sample based planner or modeling the problem as a constraint optimization problem. This project explores 1) defining an activity planning problem with the Planning Domain Definition Language, 2) implementing the Fast-Forward heuristic planner with Enforced Hill Climbing, 3) using RRT with goal-biasing for trajectory generation, and 4) comparing the results with the trajectory generated by constraint optimization. The kitchen simulation environment provides an excellent sandbox for testing and demonstrating the effectiveness of the proposed approach, in addition to learning the benefits and drawbacks of the implemented algorithms.

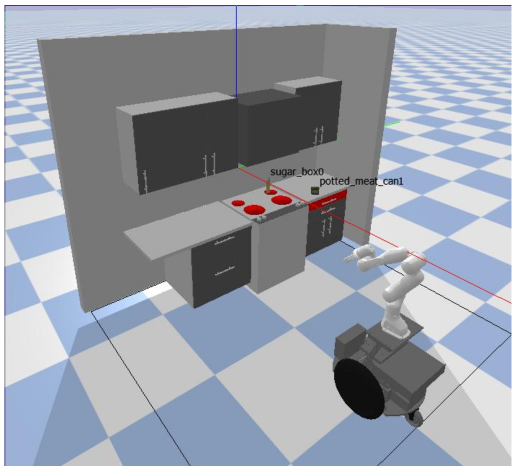

Figure 1: The kitchen simulation environment.

## 2 Activity Planning

### 2.1 PDDL Domain and Problem

The PDDL domain and problem files encode the problem as sets of objects, types, predicates, actions, and goals. There are two main types of objects: the `box` type and the `storage` type. The `box` type represents the sugar and spam boxes which are manipulated by the robot arm, and the `storage` type represents the storage spaces in the kitchen. In addition, I separated the `storage` type into two subtypes: `static`, which are storage spaces without actions (stovetop burner and countertop), and `openable`, which are storage spaces that can be opened or closed (drawer, cabinet).

The following assumptions were made when designing the actions and the initial state:

- The robot can only grab one box at a time
- The robot cannot open drawers or cabinets when gripping an object
- Only one box can be placed on each storage space at a time
- Drawers and cabinets must be open when placing or picking up a box inside them
- Drawers and cabinets must be closed at both the beginning and end of the plan

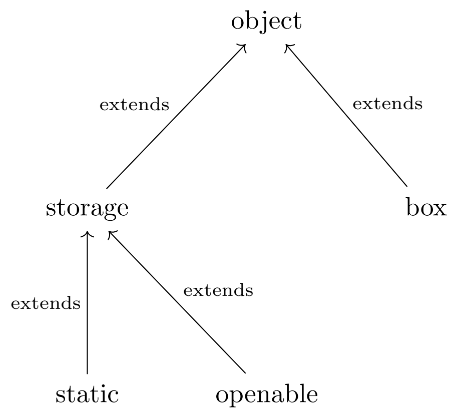

Figure 2: Hierarchical PDDL object types including static/openable storage spaces and boxes

Separating the `storage` type into subtypes allowed separating the action of placing (and picking up) a box on a storage space into `place_static` and `place_openable`, which has an extra `(open ?s)` precondition.

I also did not use any negative preconditions to simplify the implementation of the fast-forward heuristic planner. For actions requiring negative preconditions, I defined a second predicate which is always the negation of the predicate needing to express the negative precondition on (e.g. `(open ?s - openable)` and `(closed ?s - openable)`).

### 2.2 Fast-Forward Heuristic Planner

The `ff_planner` file contains all functions to solve the activity planning problem. The `calculate_hff` function computes the fast-forward heuristic $h_{ff}$ of a given state by counting the number of actions (without delete effects) it takes until the goal condition is a subset of `state`. At each iteration, `new_state` is computed by taking the union of the current state and the add effects of all applicable actions. Because we defined our actions to only have positive preconditions, we can easily determine whether an action is applicable by checking if the precondition is a subset of `new_state`. $h_{ff}$ is an admissible heuristic that always underestimates the number of actions it takes to reach the goal state, hence an alternative algorithm such as A* could have been used to find the optimal path.

In this project, I decided to implement Enforced Hill Climbing in the `solve_ff` function. While EHC is not guaranteed to find the optimal solution, its greedy nature typically leads to higher efficiency and lower memory usage. `solve_ff` initially parses the PDDL files and enumerates all possible actions that can be taken by the agent. Next, it performs a greedy search by immediately adding an action that leads to a strictly smaller $h_{ff}$ to the plan. If no such action is found, there is a "plateau" in the graph, and BFS is used to find a sequence of actions that lead to a smaller $h_{ff}$. If BFS fails to find such action, then the algorithm determines that the activity planning problem is unsolvable.

Comparing the performance of BFS and FF Heuristic Planner, on the contrary to my expectation, I found that BFS consistently runs faster ($650\mu s$ vs. $950\mu s$). This is most likely because the plan graph is rather simple for this problem. For example, the starting state has $h_{ff} = 2$, meaning EHC gets stuck on a "plateau" for most of the time, resorting to BFS. In this case, calculating $h_{ff}$ leads to extra computation time. For more complex problems requiring many actions to be taken, I expect the FF Heuristic Planner to perform significantly faster than BFS.

## 3 Motion Planning

## 4 Trajectory Optimization

[1]

## 5 Conclusion

## References

1. Cui T. Research: The First Science Partner Journal. Research 2018;2018:1.