

# 16-782 Planning and Decision-making in Robotics

**Name: Harshad Zade**

**Andrew ID: hvz**

## Homework 3: Symbolic Planning

### **Results:**

The following table summarizes the test summary of the generic symbolic planner over multiple environments that follow the STRIPS representation.

#### **A] With heuristic:**

Data → Environment ↓	Planning Time (ms)	Number of Expanded States	Number of Actions in Plan
Blocks	15	9	3
Blocks and Triangles	2435	371	6
Fire Extinguisher	735	343	21
Warehouse <b>[EXTRA CREDIT]</b>	2918	271	10

#### **B] Without heuristic:**

Data → Environment ↓	Planning Time (ms)	Number of Expanded States	Number of Actions in Plan
Blocks	30	16	3
Blocks and Triangles	3587	470	6
Fire Extinguisher	828	378	21
Warehouse <b>[EXTRA CREDIT]</b>	5329	507	10

# Details about the Algorithm and Code Implementation:

The implemented planner is a heuristic-based A\* search algorithm designed for automated planning in various domains. Here's a brief overview of its key components and functionality:

- **Search Framework:** The planner uses a state-space search framework. It begins from an initial state and explores subsequent states by applying available actions. Each action is contingent on specific preconditions and leads to a new state by producing certain effects.
- **Data Structures:**
  - ◆ **Open List:** A priority queue that stores state is yet to be explored. States are prioritized based on the sum of the cost to reach that state (g-cost) and an estimated cost to reach the goal from that state (heuristic).
  - ◆ **Closed List:** An unordered set of already explored states to avoid redundant searches.
- **State Exploration:**
  - ◆ For each state, the planner evaluates applicable actions based on their preconditions.
  - ◆ Upon finding a valid action, it applies the action's effects to generate a new state.
  - ◆ This new state is then added to the open list if it hasn't been explored before.
- **Goal Checking:** The planner continually checks if the current state satisfies the goal conditions. The goal condition can be just a subset of the current state. If the goal is met, the planner backtracks to construct the sequence of actions (plan) leading to this state.
- **Action Representation:** Actions are represented by their name, arguments, preconditions, and effects. This abstraction allows the planner to be applied to a wide range of domains.
- **Heuristic Function:** The heuristic used is a critical component guiding the search process. A refined heuristic calculation is used to ensure admissibility (the heuristic never overestimates the true cost to reach the goal). It involves scaling down the heuristic value based on the minimum number of literals added by any action in the environment. This method ensures that the heuristic is always equal to or less than the actual cost, aiding in efficient and directed search.
  - ◆ **Calculation Method:**
    - **Minimum Literals Added:** The heuristic first identifies the minimum number of literals added by any action within the domain. This value is crucial as it reflects the smallest incremental progress an action can make toward achieving a goal.
    - **Scaling Down Heuristic:** The primary heuristic value, which is the count of unmet goal conditions, is then scaled down by dividing it by the minimum literals added. This approach adjusts the heuristic

to more realistically reflect the number of actions required to meet the goals, considering the efficiency of the available actions.

◆ **Admissibility and Optimization:**

- **Admissible Estimation:** This heuristic is designed to be admissible. Since it's based on the minimum incremental progress made by the actions and the actual unmet goals, it does not overestimate the true cost to reach the goal.
- **Consistency Considerations:** Consistency, or the heuristic's ability to satisfy the triangle inequality, is maintained under most domain conditions. However, it might vary slightly depending on the specific characteristics of the domain and actions.

◆ **Enhanced Effectiveness:**

- **Practical Efficiency:** This updated heuristic is more aligned with the practical dynamics of the actions in the domain, leading to a more efficient search process.
- **Balance Between Accuracy and Performance:** While it offers a more accurate estimation than the basic count of unmet goals, it also maintains computational performance, avoiding the complexity of deeply analyzing the domain's dynamics.

→ The instructions to compile and run the code in different environments are the same as mentioned in the Homework description.

- ◆ For running with my custom environment, make sure that the Warehouse.txt file is in the same folder as the code and other environments and replace the given environment name with "Warehouse.txt" while running the planner.

## **Analysis of the Planner:**

- **Completeness:** The planner being implemented as A\* guarantees completeness, which means it should find a solution if one exists and report that no feasible plan exists if it doesn't. It systematically explores state space using a priority queue and keeps track of visited states.
- **Domain-Independence:** The planner is designed as a generic search algorithm. It uses the concept of actions, states, preconditions, and effects, which are common in planning and can be defined for any domain. The planner doesn't contain domain-specific logic, suggesting that it can be applied to various problems, like the robot delivery scenario, the blocks world, and even a new scenario I created.
- **Optimality:** The planner guarantees the optimality of the plan as I am using an admissible heuristic (explained earlier). While the heuristic's admissibility increases the likelihood of finding an optimal solution, there may be a balance

between achieving absolute optimality and computational feasibility in more complex scenarios.

- **Speed:** The planner's speed is contingent on the complexity of the domain, the number of actions, and the size of the state space. It can solve problems quickly, potentially within 60 seconds, in smaller, less complex domains. However, its performance may degrade significantly with larger state spaces or more complex domains, as it doesn't optimize for speed and can experience a combinatorial explosion. For all the given environments and start and goal conditions, it can generate the plan well within 60 seconds.
- **Heuristics:** The use of heuristics significantly impacts the planner's performance. A heuristic function guides the search towards the goal more efficiently by estimating the 'distance' or 'cost' from a given state to the goal. Comparing this planner (with heuristic) to one without any heuristic, we notice that:
  - **Improved Performance:** The heuristic-based planner can solve problems faster as it avoids exploring all branches equally and focuses on more promising paths.
    - It can be seen from the results mentioned above that the planner with heuristic was 50.00% faster in the Blocks environment, 32.11% faster in the Blocks and Triangles environment, 11.23% faster in the Fire Extinguisher environment, and 45.24% faster in the custom Warehouse environment. This also shows that although the general trend is the planner with the heuristic is faster than the one without, the delta in the improvement is very much environment and case-dependent. It should also be noted that poorly designed heuristics might mislead the search or provide minimal benefit, so the quality of the heuristic function is crucial.
    - Similarly, as far as the number of states expanded is concerned, we observe a similar trend as that of time. This is consistent logically as the major time is spent in expanding the new states. We get a 43.75% improvement in the Blocks environment, 21.06% improvement in the Blocks and Triangles environment, 9.26% improvement in the Fire Extinguisher environment, and 46.54% improvement in the Warehouse environment when using the planner with heuristics.
    - It is noteworthy that the number of actions that need to be performed in the plan that is generated is the same in both cases.
  - **Efficiency in Complex Domains:** The heuristic-based planner generally performs better than a non-heuristic planner in complex domains.

### **Extra Credit:**

I have created a custom warehouse environment. This environment involves a simple "robot delivery" scenario in a warehouse with multiple locations and packages.

## Environment Specification

### Symbols:

- Locations: L1, L2, L3, L4, L5
- Packages: P1, P2, P3
- Robot: R1

### Initial Conditions:

- At(R1, L1)
- At(P1, L2)
- At(P2, L3)
- At(P3, L4)
- Empty(R1)

### Goal Conditions:

- At(P1, L3)
- At(P2, L4)
- At(P3, L5)

### Actions:

1. **Move(robot, from, to)**
  - Preconditions: At(robot, from)
  - Effects: At(robot, to), !At(robot, from)
2. **PickUp(robot, package, location)**
  - Preconditions: At(robot, location), At(package, location), Empty(robot)
  - Effects: Carrying(robot, package), !At(package, location), !Empty(robot)
3. **PutDown(robot, package, location)**
  - Preconditions: Carrying(robot, package), At(robot, location)
  - Effects: At(package, location), Empty(robot), !Carrying(robot, package)

## Description of the Scenario

- The environment is a warehouse with five locations (L1 to L5) and three packages (P1 to P3).
- The robot (R1) starts at location L1, and each package is initially at a different location.
- The goal is to move the packages to their designated new locations using the robot.

- The robot can move between locations, pick up a package if it's at the same location and it's not carrying anything, and put down a package at its destination.

This environment challenges the planner with tasks of moving around a robot, handling objects (pick up and put down actions), and achieving specific goals related to object locations.

\*\* Performance is presented at the top of the report, along with other environments.