16782 – Planning and Decision Making for Robotics
HW3: Symbolic Planning

Saurav Kambil – skambil

1. Instruction on running planner:

Windows:

g++ planner.cpp -o planner.exe

To execute:

planner.exe example.txt

Linux:

g++ planner.cpp -o planner.out

To execute:

planner.out example.txt

'example.txt' is the file containing the environment.

2. Code Description and Planning Approach

The first step is to compute all possible actions using the symbols in each environment. The planning algorithm then performs an A* search from the start state to the goal state by iterating through all actions that are valid.

The **heuristic_compute()** function calculates the count of conditions in the goal state that are not yet satisfied in the current state. This approach is straightforward and provides an estimate of how close the state is to the goal. The edge costs between actions are set to 1. However, the heuristic is **inadmissible** and it often times is an overestimate since a since multiple conditions can be satisfied by a single action.

3. Results

The planner calculates the time taken to run each environment. The results of each environment are listed below. The time taken to run each environment is listed in the table below.

a) Blocks Environment:

Plan:

MoveToTable(A,B)

Move(C,Table,A)

Move(B,Table,C)

b) Blocks Triangle Environment

Plan:

MoveToTable(T0,B0)

MoveToTable(T1,B3)

MoveToTable(B0,B1)

Move(B1,B4,B3)

Move(B0,Table,B1)

Move(T1,Table,B0)

c) Fire Extinguisher Environment

Plan:

MoveToLoc(A,B)

LandOnRob(B)

MoveTogether(B,W)

FillWater(Q)

MoveTogether(W,F)

TakeOffFromRob(F)

PourOnce(F)

LandOnRob(F)

MoveTogether(F,W)

Charge(Q)

FillWater(Q)

MoveTogether(W,F)

TakeOffFromRob(F)

PourTwice(F)

LandOnRob(F)

MoveTogether(F,W)

FillWater(Q)

MoveTogether(W,F)

Charge(Q)

TakeOffFromRob(F)

PourThrice(F)

d) Disk Environment
Plan:
MoveToShelf(DiskRed,DiskBlue)
Move(DiskGreen,Shelf,DiskRed)
Move(DiskBlue,Shelf,DiskGreen)

Bonus Environment Description:

In this extended environment, the goal is to rearrange colored disks (Red, Blue, Green, and Yellow) on a shelf, with the added complexity of a robotic arm that must first unclip the disks before they can be moved. This environment is designed to simulate a more complex planning scenario that involves additional steps and constraints.

Key Features:

1. **Disks**: There are four colored disks (Red, Blue, Green, Yellow) that start in a specific stacked arrangement.
2. **Shelf**: The disks need to be rearranged onto this shelf according to the goal conditions.
3. **Robotic Arm**: A crucial element that adds complexity. The arm must unclip a disk before it can be moved. Its availability is a factor in the planning process.
4. **Clipped/Unclipped State**: Each disk begins in a clipped state. The robotic arm must unclip them, transitioning them to an unclipped state, making them eligible for movement.

Initial Conditions:

- The disks are stacked in a specific order, with some on the shelf and others on top of each other.
- Each disk is initially clipped, preventing immediate movement.
- The robotic arm is available for unclipping the disks.

Goal Conditions:

- A specified arrangement of the disks on the shelf, different from the initial setup.

Actions:

- **Unclip**: The robotic arm unclips a disk, allowing it to be moved.
- **MoveToShelf**: Move an unclipped disk from its current position directly onto the shelf.

- **Move**: Stack an unclipped disk on top of another disk.

Challenge:

The primary challenge in this environment is to strategically plan the sequence of actions, considering both the unclipping and moving of disks, to achieve the desired disk arrangement on the shelf. The planner must manage the constraints imposed by the clipping mechanism and the availability of the robotic arm, making the task more akin to real-world scenarios where multiple conditions must be met before an action is performed.

Text File:

Symbols: DiskRed, DiskBlue, DiskGreen, DiskYellow, Shelf, RobotArm, Clipped, Unclipped

Initial conditions: On(DiskRed, DiskBlue), On(DiskBlue, Shelf), On(DiskGreen, Shelf), On(DiskYellow, DiskGreen), Disk(DiskRed), Disk(DiskBlue), Disk(DiskGreen), Disk(DiskYellow), Clear(DiskRed), Clear(DiskYellow), Clipped(DiskRed), Clipped(DiskBlue), Clipped(DiskGreen), Clipped(DiskYellow), RobotArmAvailable(RobotArm)

Goal conditions: On(DiskBlue, DiskGreen), On(DiskGreen, DiskRed), On(DiskRed, Shelf), On(DiskYellow, Shelf)

Actions:

Unclip(d)

Preconditions: Clipped(d), Disk(d), Clear(d), RobotArmAvailable(RobotArm)

Effects: Unclipped(d), !Clipped(d)

MoveToShelf(d,x)

Preconditions: On(d,x), Clear(d), Unclipped(d), Disk(d), Disk(x)

Effects: On(d,Shelf), Clear(x), !On(d,x)

Move(d,x,y)

Preconditions: On(d,x), Clear(d), Clear(y), Unclipped(d), Disk(d), Disk(y)

Effects: On(d,y), Clear(x), !On(d,x), !Clear(y)

4) Performance:

Environment	Time(s)	Plan Length	Expansions	Heuristic
Blocks	0.01	3	5	Yes
Blocks	0.02	3	7	No
BlocksTriangle	0.356	6	103	Yes
BlocksTriangle	1.155	6	205	No
FireExtinguisher	0.663	21	482	Yes
FireExtinguisher	0.909	21	482	No
Disk	0.179	8	65	Yes
Disk	1.094	8	281	No

1. **Completeness**: The planner remains complete because it uses the A* search strategy without a depth limit. This means the planner will exhaustively search the entire space until it finds a solution or determines that no solution exists.
2. **Optimality**: Although the planner uses the A* search algorithm, the optimality of the resulting plan is not guaranteed because the heuristic is inadmissible. An inadmissible heuristic overestimates the distance to the goal, which can cause the A* algorithm to miss shorter paths that it falsely deems less efficient. As a result, the planner may still find a solution, but there's no guarantee that it will be the best possible one.
3. **Domain Independence**: The planner is domain-independent, as evidenced by its performance across various scenarios, from stacking blocks to complex tasks like fire extinguishing. This trait is highly beneficial, as it allows the planner to be applied to a wide range of problems without modification. The general search algorithm and the domain-independent heuristic—which is based on state-goal congruence—contribute to its flexibility and adaptability across different domains.
4. **Quality of the Plan**: The quality of the plan, in terms of the number of steps, may not be the best due to the overestimation by the heuristic. If multiple goal conditions can be satisfied with a single action, the planner might choose a less efficient path because the heuristic suggests that it's closer to the goal than it actually is.
5. **Planning Speed**: An inadmissible heuristic often speeds up the search by aggressively focusing on paths that appear to lead more directly to the goal, potentially at the expense of exploring

other paths that might offer a more optimal solution. In the context of the provided planner, while the planning speed is high and all problems are solved in well under 60 seconds, this comes with the trade-off of potentially suboptimal plans.

In summary, while the planner is complete and efficient in terms of speed, the use of an inadmissible heuristic means that it can no longer guarantee that the plans it generates are optimal. This trade-off between speed and plan quality is a common consideration in the design of planning algorithms, particularly in complex domains where the computational cost of finding the optimal solution may be prohibitively high.

The results show a comparison between runs of a planner using a heuristic ("Yes" in the Heuristic column) and runs of the same planner without a heuristic ("No" in the Heuristic column). Here's an analysis of the improvements gained from using a heuristic in the search:

Planning Speed:

The use of a heuristic has significantly reduced the planning time across all scenarios:

- For simple block stacking problems ("Blocks"), the time is cut in half from 0.02s to 0.01s.
- In more complex scenarios involving both blocks and triangles ("BlocksTriangle"), the heuristic reduces planning time from 1.155s to 0.356s.
- In the "FireExtinguisher" scenario, which likely has a much larger search space, the time reduction is less pronounced but still notable (from 0.909s to 0.663s).
- The "Disk" scenario shows a substantial time reduction when using a heuristic (from 1.094s to 0.179s).

Plan Length:

The plan length remains constant regardless of whether a heuristic is used. This suggests that the heuristic does not affect the outcome in terms of the actions required to achieve the goal but does improve the time taken to find the same solution.

Expansions:

The number of expansions—a measure of how many nodes are processed in the search—shows a clear advantage when using a heuristic:

- In the "Blocks" scenario, using a heuristic reduces the expansions from 7 to 5.
- In the "BlocksTriangle" scenario, expansions are nearly halved (from 205 to 103) when a heuristic is applied.
- The "FireExtinguisher" scenario shows no change in expansions, which is interesting. This might indicate that the heuristic is guiding the search efficiently but the complexity of the scenario prevents a reduction in the number of expansions needed.
- The "Disk" scenario also demonstrates a substantial reduction in expansions (from 281 to 65) when using a heuristic.

Discussion:

The data indicates that using a heuristic in the search yields a significant improvement in planning speed. This is expected because heuristics guide the search towards the goal more directly, reducing the number of nodes that need to be explored. However, the heuristic does not always reduce the number of expansions, as seen in the "FireExtinguisher" scenario, suggesting that the heuristic's effectiveness may be context-dependent.

In this specific case, the heuristic appears to be particularly effective in scenarios with smaller state spaces ("Blocks" and "Disk"). In larger or more complex state spaces ("BlocksTriangle" and "FireExtinguisher"), while the heuristic still improves the time, the benefit to the number of expansions varies.

The constant plan length across runs with and without a heuristic indicates that the heuristic is not affecting the quality of the plan, which remains optimal as guaranteed by the A* algorithm. However, the optimality of the heuristic-based plan is contingent upon the admissibility of the heuristic, which has been noted to overestimate in this planner's case.

Overall, the heuristic improves search efficiency in terms of time, which is a valuable asset in real-world applications where time is a critical factor. The improvement in expansions suggests that the heuristic is generally effective, although its performance may not be uniform across different problem domains. It's also important to note that even though the heuristic can overestimate and therefore is inadmissible, it can still be useful for finding solutions more quickly, even if the guarantee of optimality is lost.

Note:

I tried to implement part(b) the empty delete heuristic but I could not figure out how to increment the heuristic values based on the delete conditions. The code of the heuristic can be found in my planner file on line 982 in planner.cpp