x y

5L 3L

→ Jug x can contain 5L

→ Jug y can contain 3L

Goal is to have 4L in Jug x

5 3

(x, y)

<0,0>

<5,0>

<2,3>

<2,0>

<0,2>

<5,2>

<4,3>

Water Jug Problem

- The Water Jug Problem is a classic example often used in computer science and artificial intelligence to demonstrate problem-solving methods such as Means-Ends Analysis. Here's a brief overview:

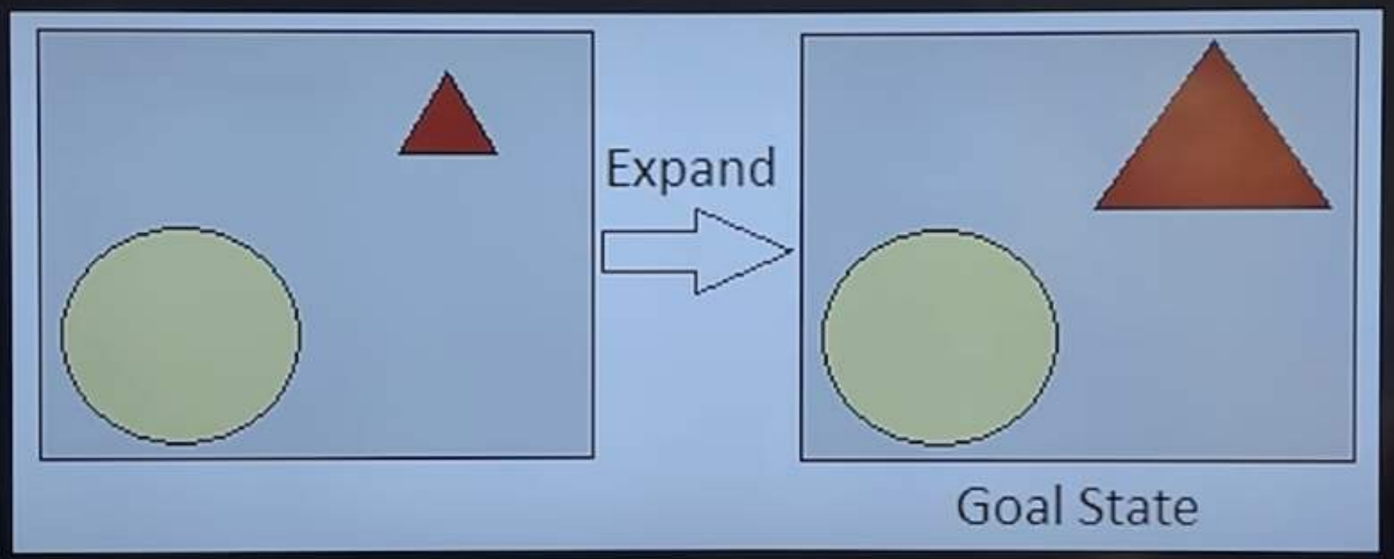
- **Problem Statement**: You have two jugs with different capacities (for example, one jug holds 3 liters and the other holds 5 liters) and a water source. The goal is to measure out a specific amount of water (say 4 liters), you can fill, empty, or transfer water between the jugs and no measurements marking on either jug.

- **Means-Ends Analysis Steps**:
  - **Identify Differences**: Note the discrepancies between the current and goal states.
  - **Set Sub-goals**: Create intermediate goals aimed at reducing identified differences.
  - **Find Operators**: Look for actions that can achieve these sub-goals.
  - **Apply Operators**: Execute the most promising actions. If constraints prevent direct action, set new sub-goals to remove these constraints.
  - **Iterate**: Repeat these steps until the goal is achieved or no further progress can be made.

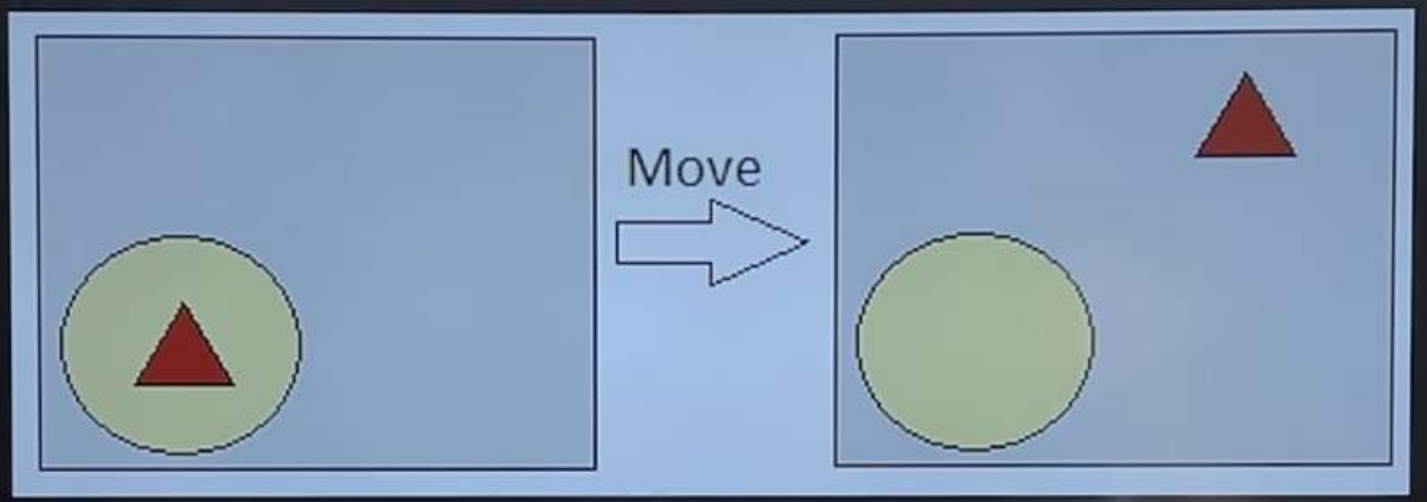
Operator Subgoaling:

- **What it is**: A process within Means-Ends Analysis (MEA) for handling situations where a direct action (operator) cannot be applied to the current state.
- **Purpose**: To establish the necessary conditions (preconditions) for applying an operator by setting up sub-goals that modify the current state.

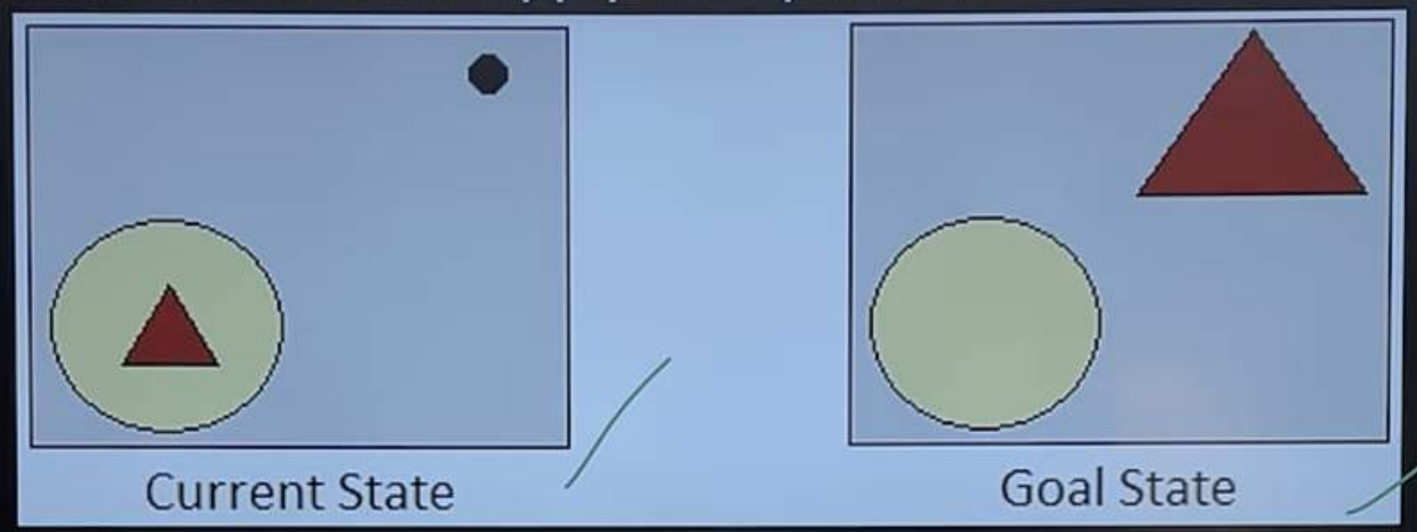
- **Applying Expand Operator:** Now a new state is generated in the third step, and we will compare this state with the goal state. After comparing the states there is still one difference which is the size of the triangle, so, we will apply Expand operator, and finally, it will generate the goal state.



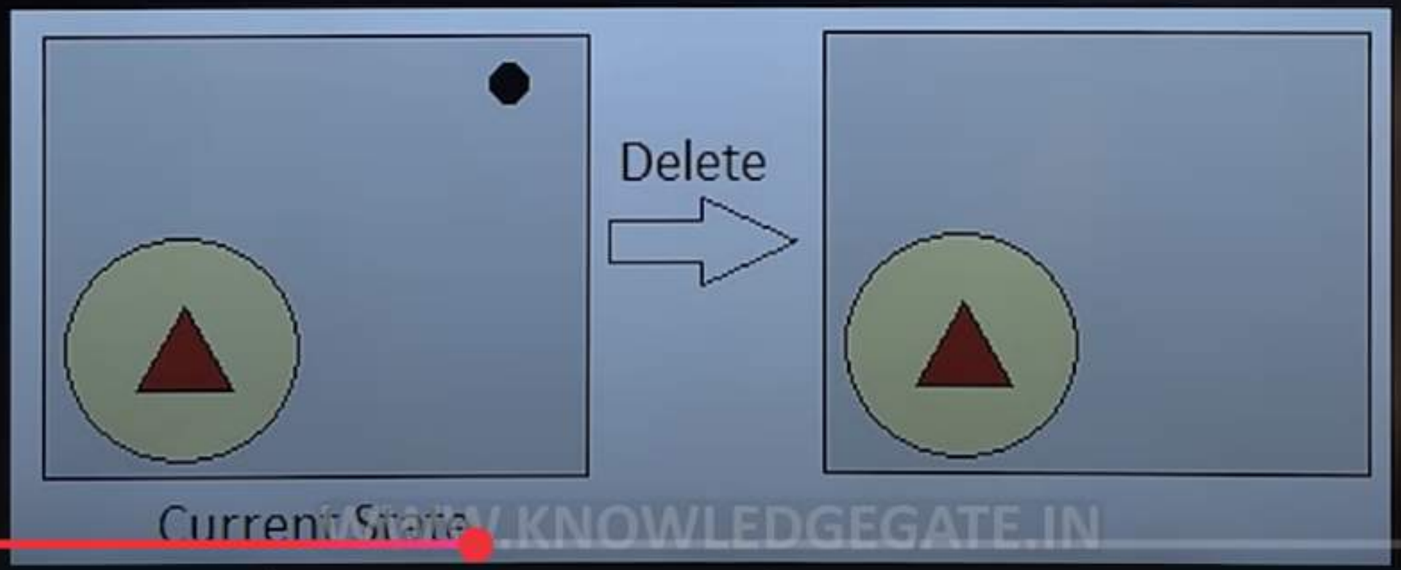
- **Applying Move Operator**: After applying the Delete operator, the new state occurs which we will again compare with goal state. After comparing these states, there is another difference that is the triangle is outside the circle, so, we will apply the Move Operator.



- We first evaluate the difference between the current and the goal state. For each difference, we will generate a new state and will apply the operators.



- **Applying Delete operator:** The first difference that we find is that in goal state there is no dot symbol which is present in the initial state, so, first we will apply the **Delete operator** to remove this dot.



Process:

- **Break down the problem**: Split a large problem into major components and address these first.
- **Address sub-problems**: Once the major components are aligned towards the goal, solve any smaller issues that emerge.
- **Integrate strategies**: Use a blend of forward search (from the starting point) and backward search (from the goal) to guide the problem-solving process effectively.

🔦 2. **Beam Search** (Keeps top-k candidates at each level)

Parameters:

- **Beam width (k)**: 2
- **Start states**: x = -2 and x = 2
- **Neighborhood**: x ± 1

Step-by-Step Table:

Step	Candidates (x)	f(x)	Neighbors (x)	f(neighbors)	Top-k selected
0	-2, 2	4, 4	-3,-1,1,3	9,1,1,9	-3, 3
1	-3, 3	9, 9	-4,-2,2,4	16,4,4,16	-4, 4
2	-4, 4	16,16	-5,-3,3,5	25,9,9,25	-5, 5
3	-5, 5	25,25	-4,4	16,16	-5, 5 (stay)

🔍 **Beam search** maintains the top 2 candidates based on f(x) across the search tree.

⚡ Problem Setup: Maximize $f(x) = x^2$ (Hill Climbing Example)

Search space: $x \in \{-5, -4, -3, \ldots, 5\}$

Objective: Maximize $f(x) = x^2$

🌐 1. **Tabu Search** (Avoiding local loops)

Parameters:

- **Start state**: $x = -3$
- **Neighborhood**: $x \pm 1$
- **Tabu list size**: 3

Step-by-Step Table:

Step	Current x	f(x)	Neighbors (x)	f(neighbors)	Selected x	Tabu List
0	-3	9	-4, -2	16, 4	-4	[-3]
1	-4	16	-5, -3	25, 9	-5	[-3, -4]
2	-5	25	-4	16	-5 (stay)	[-3, -4, -5]
3	-5	25	-4	16	-5 (stay)	[-4, -5]

Means-Ends Analysis

- **What it is**: A strategic approach combining forward and backward reasoning to tackle complex and large problems in AI.
- **Why use it**: It helps in focusing the problem-solving effort on significant discrepancies between the current state and the goal, reducing unnecessary exploration.

Random-restart hill climbing

- To avoid local optima, you can perform multiple runs of the algorithm from different random initial states. This is called random-restart hill climbing and increases the chance of finding a global optimum.
- **Tabu Search:** To prevent the algorithm from getting stuck in a loop or revisiting states, a list of previously visited states can be maintained, which are then avoided in future steps.
- **Local Beam Search:** To tackle plateau and local optima, this variation of hill climbing keeps track of k states rather than just one. It begins with k randomly generated states. At each step, all the successors of all k states are generated, and if any one is a goal, it halts. Else, it selects the best k successors from the complete list and repeats.

Advantages of Hill Climbing:

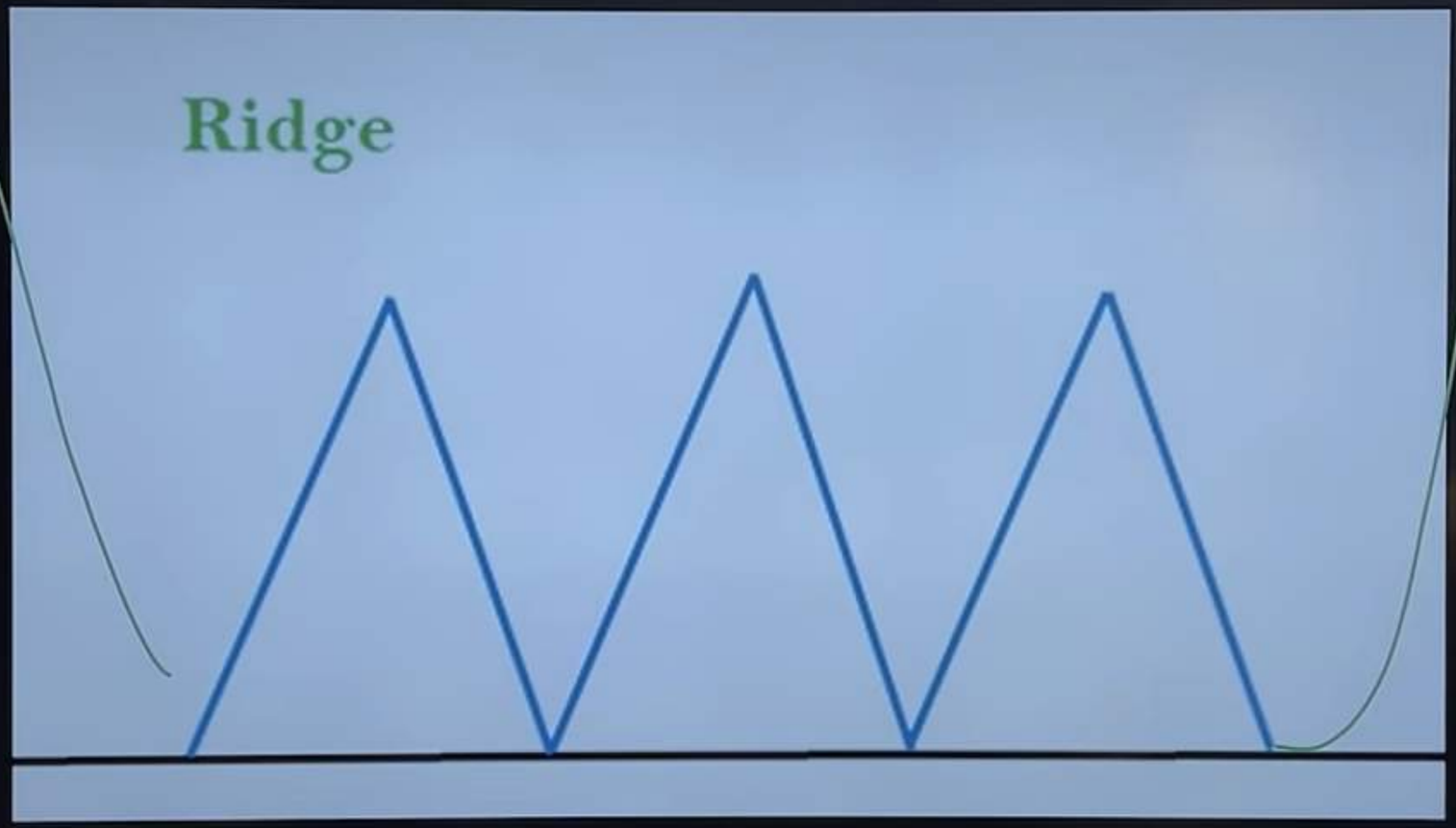
- It's simple to understand and easy to implement.
- It requires less computational power compared to other search algorithms.
- If the heuristic is well chosen, it can find a solution in a reasonable time.

Disadvantages of Hill Climbing:

- It's not guaranteed to find the optimal solution.
- It's highly sensitive to the initial state and can get stuck in local optima.
- It does not maintain a search history, which can cause the algorithm to cycle or loop.
- It can't deal effectively with flat regions of the search space (plateau) or regions that form a ridge.

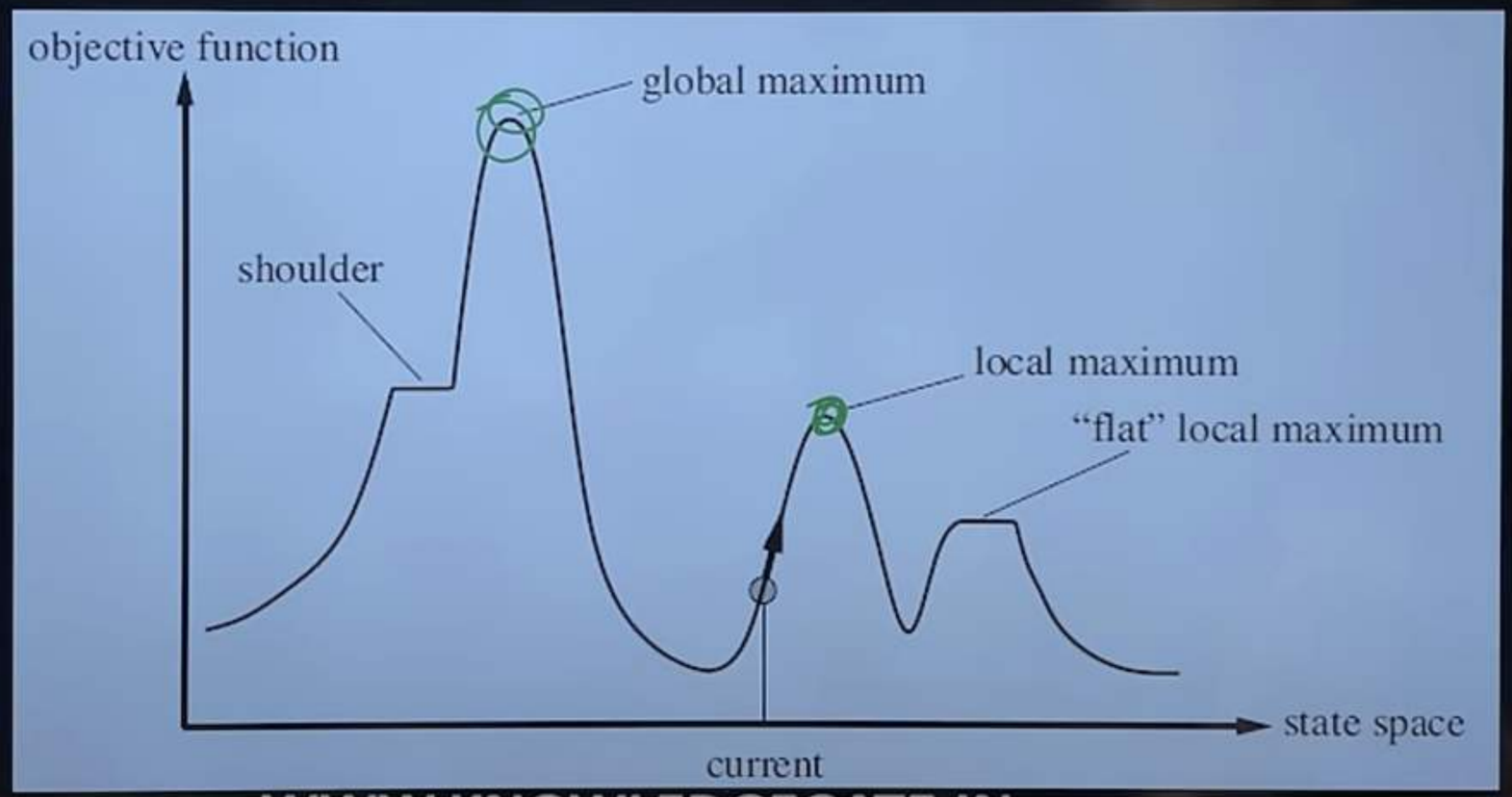
- **Plateau:** A plateau is a flat area of the state-space landscape. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which progress is possible. A hill-climbing search might get lost on the plateau. In each case the algorithm reaches a point at which no progress is being made.

- **Ridges**: Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate.



Problems with Hill Climbing

- **Local maxima**: a local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum. Hill-climbing al that reach the vicinity of a local maximum will be drawn upward towar peak but will then be stuck with nowhere else to go.



Algorithm for Simple Hill climbing

- **Start**: Begin at a random point on the problem landscape.
- **Evaluate**: Assess the current position.
- **Look Around**: Check neighboring points and evaluate them.
- **Can Improve**: Determine if any neighbors are better (higher value) than current position.
- **Move**: If a better neighbor exists, move to that point.
- **Repeat**: Continue the process from the new point.
- **No Improvement**: If no better neighbors are found, you've hit a peak.
- **End**: The process stops when no further improvement can be made.

Hill Climbing Algorithm

- It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making a change, If the change produces a better solution, another incremental change is made to the new solution, and so on until no further improvements can be found.
- Hill climbing is sometimes called **greedy local search with no backtracking** because it grabs a good neighbor state without thinking ahead about where to go next.

- **Completeness and Optimality**
  - AO* is complete, meaning it is guaranteed to find a solution if one exists. It is also optimal, meaning it will find the best solution, provided that the heuristic function is admissible (never overestimates the true cost) and consistent (satisfies the triangle inequality).

- **Time and Space Complexity**
  - The time and space complexity of the AO* algorithm is highly dependent on the problem at hand, particularly the size of the state space, the number of goals, and the quality of the heuristic function. In the worst case, the algorithm will need to visit all nodes, and since each node is stored, the space complexity is also proportional to the number of nodes.

*Advantages of AO Algorithm**

- It can efficiently solve problems with multiple paths due to its use of heuristics.
- It is optimal when the heuristic function is admissible (never overestimates the true cost).

*Disadvantages of AO Algorithm**

- It can consume a large amount of memory, similar to the A* algorithm.
- The performance of AO* is heavily dependent on the accuracy of the heuristic function. If the heuristic function is not well-chosen, AO* could perform poorly.