

# Problem Solving Study Guide

## Artificial Intelligence

### 1 Problems

1. What is a planning problem?

**Solution:** A problem whose solutions are sequences of actions from some initial or current state to a goal state.

2. Describe open-loop and closed-loop control.

**Solution:** In an open-loop control system the agent gets no feedback, i.e., sensor input, after executing an action. If the agent's model is perfect and actions are deterministic, then the agent can operate in an open-loop fashion, simply executing the actions in the solution one after the other.

In a closed-loop control system the agent gets sensory feedback after every action, so it can check whether the action had the expected effect. If the environment is partially observable or actions are nondeterministic, closed-loop control is necessary.

3. Write a problem formulation for a world with three unmarked pitchers – an 8 L pitcher full of water, an empty 5 L pitcher, and an empty 3 L pitcher – where an agent must reallocate the water so that one of the pitchers contains exactly 4 L of water.

**Solution:** One of many possibilities:

- States:  $\mathbf{p} = \langle p_1, p_2, p_3 \rangle$ ,  $p_i \in \mathbb{N}$ ,  $\sum_{i=1}^3 p_i = 8$ ,  $c_1 = 8$ ,  $c_2 = 5$ ,  $c_3 = 3$ ,  $r_i = c_i - p_i$
- Initial state:  $\langle 8, 0, 0 \rangle$
- Actions:  $Actions(\mathbf{p}) = pour(f, t)$  for all  $f, t$  where  $f, t \in \{1, 2, 3\}$  and  $f \neq t$  and  $p_f > 0$  and  $r_t > 0$ . Examples:

$$- actions(\langle 8, 0, 0 \rangle) = \{pour(1, 2), pour(1, 3)\}$$

$$- actions(\langle 3, 5, 0 \rangle) = \{pour(1, 3), pour(2, 1), pour(2, 3)\}$$

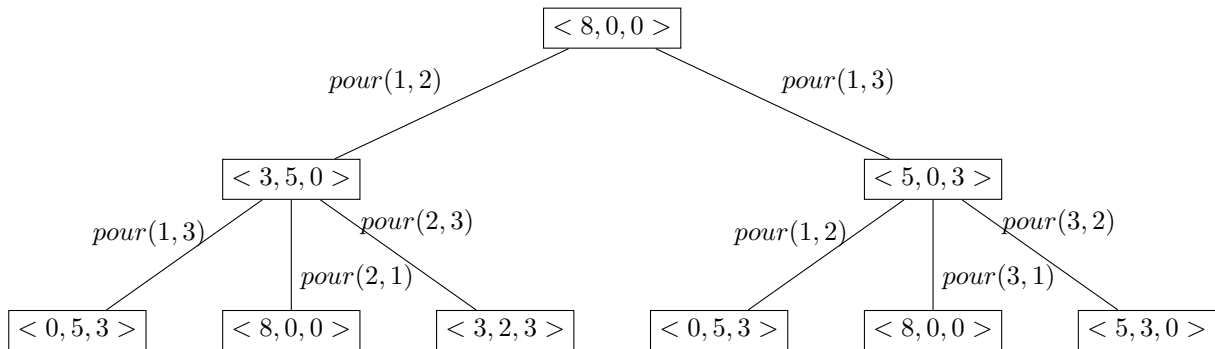
- Transition model:

$$RESULT(pour(f, t)) = \begin{cases} p_t \leftarrow c_t, p_f \leftarrow p_f - r_t & \text{if } r_t \leq p_f \\ p_t \leftarrow p_t + p_f, p_f \leftarrow 0 & \text{if } r_t \geq p_f \end{cases}$$

- Goal states:  $p_i = 4$  for some  $i \in \{1, 2, 3\}$
- Action cost: 1

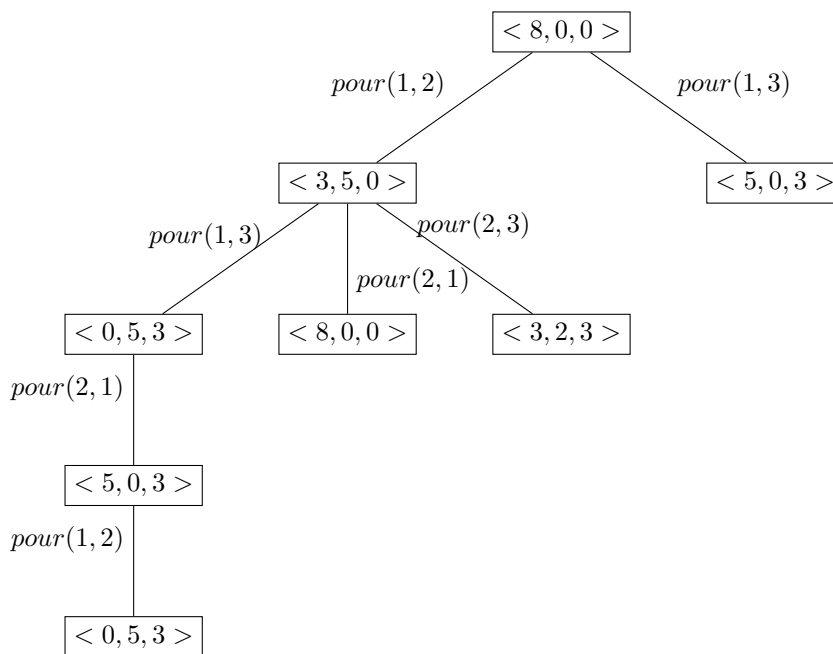
# 1 Uninformed Search

1. Consider the first two levels of a BFS search tree with a start state of  $\langle 8, 0, 0 \rangle$  where the  $expand(problem, node)$  function always enumerates child nodes by choosing actions from “left to right”, that is, choosing the leftmost source pitcher to pour from, and the leftmost target pitcher to pour to.



Discuss the implications of this child node expansion order for Depth-First Search.

**Solution:** In basic Depth-First Search without repeated state-avoidance, the algorithm will get stuck descending an infinite path down the left side of the search tree, as shown in the first four levels of the search tree.



2. Is Breadth-First Search subject to the same problems discussed in the previous question? Why, or why not?

**Solution:** No, because Breadth-First Search expands all the nodes at a particular level in the search tree before moving to the next level.

3. Is Breadth-First Search complete? Why, or why not?

**Solution:** Yes, because it expands all nodes at each level successively, guaranteeing it will not miss a goal state as the depth of the search tree increases.

4. Is Depth-First Search complete? Why, or why not?

**Solution:** No, because even if cycles are avoided, DFS could get stuck searching an infinite infinite state spaces.

5. What is the simplest way to modify Depth-First Search so that it does not descend an infinite path?

**Solution:** Depth-Limited Depth-First Search limits the depth to which Depth-First Search expands nodes. This modification can lead to a complete expansion of the tree to the specified depth-limit.

6. Discuss the most important implication of modifying Depth-First Search so that it does not descend an infinite path.

**Solution:** If there is no goal state within the depth limit for Depth-Limited Depth-First Search, then the algorithm will not find a goal state.

7. Describe an algorithm that uses Depth-First Search but is complete.

**Solution:** Iterative-Deepening Depth-First Search simply runs Depth-Limited Depth-First Search for iteratively increasing depth limits until a goal state is found.

8. What is the primary tradeoff between Bread-First Search and Depth-First Search?

**Solution:** Depth-First Search uses less memory,  $O(bm)$  where  $b$  is branching factor and  $m$  is max depth of tree vs Breadth-First Search, which uses  $O(b^d)$  memory because it stores all the nodes in each level as it expands the search tree.