Wang Xiyu

National University of Singapore
School of Computing
CS2109S: Introduction to AI and Machine Learning
Semester 2, 2024/2025

Tutorial 1
Problem Formulation & Uninformed Search

Summary of Key Concepts

In this tutorial, we will discuss and explore the following key learning points/lessons:

1. Describing the problem environment with PEAS.
2. Formulating a search problem:
   (a) State representation
   (b) Actions
   (c) Goal state(s)
3. Uniform-cost Search
4. Iterative Deepening Search

A PEAS for AI chess player

1. Determine the PEAS (Performance measure, Environment, Actuators, Sensors) for an AI chess opponent in a mobile chess application.

B Tower of Hanoi Problem

The Tower of Hanoi is a famous problem in computer science, described as follows: You are given 3 pegs (arranged left, middle, and right) and a set of n disks with holes in them (so they can slide onto the pegs). Every disk has a different diameter.

We start the problem with all the disks arranged on the left peg in order of size, with the smallest on top. The objective is to move all the disks from the left peg to the right peg in the *minimum number of moves*. This is done by:

- moving one disk at a time from one peg to another; and
- never placing a disk on top of another disk with a smaller diameter.

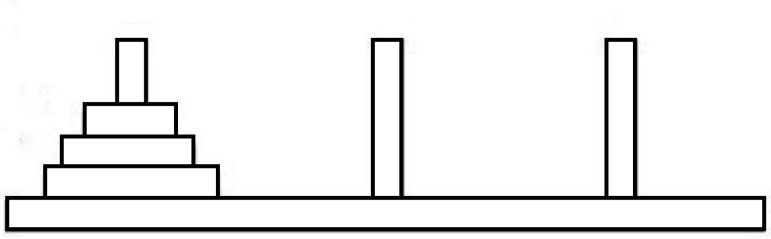

Figure 1: The initial state for $n = 3$

1 1

1.1

a

States: every possible placement scheme of all disks on all poles, represented by a tuple of lists or stacks in the form of {[1, 2, 3, 4, 5], [], []} etc.
Initial State: the starting placement of all disks
Goal States: The designated way of placing the all disks
Use set to make {1, 2} and {2, 1} to be the same state and the later is not a valid state
action: if $state_a$ can be transformed into $state_b$ my moving one disk, $state_b$ is reacheable from $state_a$ in cost 1

Total number of states

Loose bound:
Flatten the list tuple, an action can be seen as a swap between 2 elements in the flattened list. the flattened list can then be partitioned into 3 partitions, by placing 2 separaters, the total number of arranging all disks on all 3 poles is:

$$\binom{n+2}{2}!$$

Tighter bound after considering constraints:
for each state, there are only 3 legal moves. Given state a,

$$S = \{[d_1, d_2, \ldots, d_{a-1}], [d_a, d_{a+1}, \ldots, d_{b-1}], [d_b, d_{b+1}, \ldots d_n]\}(\text{Consider all disks are ordered})$$

only the top of any peg can be placed onto another peg's top. Since all disks are comparable, the tops of all pegs are also comparable. WLOG

$$d_1 < d_a < d_b$$

$\forall S \in \{\text{possible arrangment }\}\exists$ exactly 3 valid movements:

$$move(S, d_1, 1) = \{[d_2, \ldots, d_{a-1}], [d_1, d_a, d_{a+1}, \ldots, d_{b-1}], [d_b, d_{b+1}, \ldots d_n]\}$$

$$move(S, d_1, 2) = \{[d_2, \ldots, d_{a-1}], [d_a, d_{a+1}, \ldots, d_{b-1}], [d_1, d_b, d_{b+1}, \ldots d_n]\}$$

$$move(S, d_a, 2) = \{[d_1, d_2, \ldots, d_{a-1}], [d_{a+1}, \ldots, d_{b-1}], [d_a, d_b, d_{b+1}, \ldots d_n]\}$$

Therefore for in total n disks, there are 3^n valid states.

b

Representation invariant: No larger disk on smaller disks in all possible states. In list tuple representation, all lists are sorted; In set tuple represented there is no need to sort since permutations are regarded as the same.

1.2

Depth-Limit Search(using DFS).

1.3

$k - 1$. as there will be more choices of action from one state to another.

Figure 1 shows the initial state for $n = 3$. To get a better feel for the problem, you can try it here: https://www.mathsisfun.com/games/towerofhanoi.html.

1. (a) Propose a state representation for this problem if we want to formulate it as a search problem.
   (b) Describe the representation invariant of your state (i.e., what condition(s) must a state satisfy to be considered valid). Compare the total number of possible configurations in the actual problem to that of your chosen state representation, with and without considering the representation invariant.
   (c) Specify the initial and goal state(s) of your state representation.
   (d) Define the actions.
   (e) Using the state representation and actions defined above, state the transition function T (i.e., upon applying each of the actions defined above to a current state, what is the resulting next state?).
2. **Additional Question:** Given the nature of the Tower of Hanoi problem, which search algorithm would be most appropriate for finding the solution? Justify your answer.
3. **Additional Question:** if there are k pegs instead of 3, what is the branching factor of the search tree?

C Uniform-Cost Search Analysis

Consider the undirected graph shown in Figure 2. Let S be the initial state and G be the goal state. The cost of each action is as indicated.

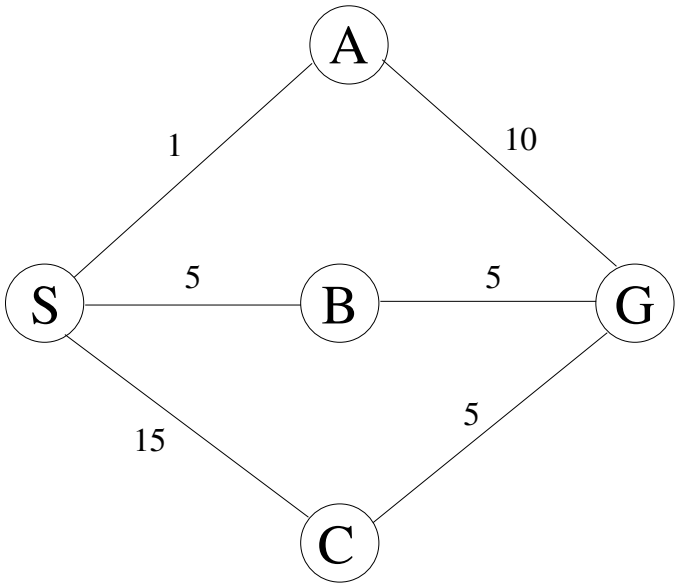

Figure 2: Graph of routes between S and G.

1. Give a trace of uniform-cost search using `SEARCH` (without visited memory) as implemented in Figure 3. If the costs are the same, the tiebreaker is the alphabetical ordering of the letters, with A < Z.

2. Give a trace of uniform-cost search using `SEARCH WITH VISITED MEMORY` as implemented in Figure 4.
3. When A generates G, the goal state, with a path cost of 11 in (b), why doesn't the algorithm halt and return the search result since the goal has been found? With your observation, discuss how uniform-cost search ensures that the shortest path solution is selected.

```
create frontier : priority queue (path cost)

insert Node(initial_state) to frontier
while frontier is not empty:
    node = frontier.pop()
    if node.state is goal: return solution



    for action in actions(node.state):
        next_state = transition(node.state, action)
        frontier.add(Node(next_state))
return failure
```

Figure 3: Uniform cost search (`SEARCH` implementation).

```
create frontier : priority queue (path cost)
create visited : set of states
insert Node(initial_state) to frontier
while frontier is not empty:
    node = frontier.pop()
    if node.state is goal: return solution
    if node.state in visited: continue
    visited.add(node.state)
    for action in actions(node.state):
        next_state = transition(node.state, action)
        frontier.add(Node(next_state))
return failure
```

Figure 4: Uniform cost search (`SEARCH WITH VISITED MEMORY` implementation).

D Comparison of Search Strategies

1. Describe a problem in which iterative deepening search performs much worse than depth-first search.
2. Describe a problem where `SEARCH` performs much better than `SEARCH WITH VISITED MEMORY`.

D

1.4

Linear search, or a tree search with braching factor 1, iterative deepening search perform in worse case scenario costs $O(n^2)$ while dfs costs $O(n)$

1.5

1. In linear search there is no need to keep tracking visited as node will not be visited again whatsoever, thus search without visited memory can save up to O(n) space. For example, search in a linked list.
2. Search on an ordered tree, the search does not need to exhaustively explore nodes thus there is no need to keep track of visited node, saving up to O(n) space. For example, search in a ordered binary tree.