National University of Singapore
School of Computing
CS2109S: Introduction to AI and Machine Learning
Semester I, 2023/2024

Tutorial 1 Solution
Problem Formulation & Uninformed Search

Summary of Key Concepts

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

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

Problems

1. Determine the following environment properties of a Sudoku game.
   - Fully vs Partially Observable
   - Single vs Multi-Agent
   - Deterministic vs Stochastic
   - Episodic vs Sequential
   - Static vs Dynamic
   - Discrete vs Continuous

   Solution:

   The environment properties are:

	Environment Properties	***Sudoku***
1	*Fully vs Partially Observable*	*Fully Observable*
2	*Single vs Multi-Agent*	*Single*
3	*Deterministic vs Stochastic*	*Deterministic*
4	*Episodic vs Sequential*	*Sequential/Episodic*
5	*Static vs Dynamic*	*Static*
6	*Discrete vs Continuous*	*Discrete*

   The fourth property can be episodic or sequential, depending on how the search problem is formulated. If each action consists of filling a single field, the problem would be sequential; If each action involves filling all the boxes simultaneously in an attempt to obtain a solution to the Sudoku puzzle, then the problem would be episodic.

2. Determine the PEAS (Performance measure, Environment, Actuators, Sensors) for the SIRI function of iPhones.

   Solution:

   - *P: Correctly processes users' voice into text based on different language with possible accents. Make correct action with acceptable reaction speed.*
   - *E: iOS device with a functional audio input system. Internet connection and supported language voice.*
   - *A: Speaker or/ and visual output to show answers to user.*
   - *S: Speech Listener/ Microphone. Touchscreen Interactions.*

3. The Tower of Hanoi is a famous problem in computer science, and is 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 into 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 the 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 shows the intial state for $n = 3$. To get a better feel of the problem, you can try it here: https://www.mathsisfun.com/games/towerofhanoi.html.

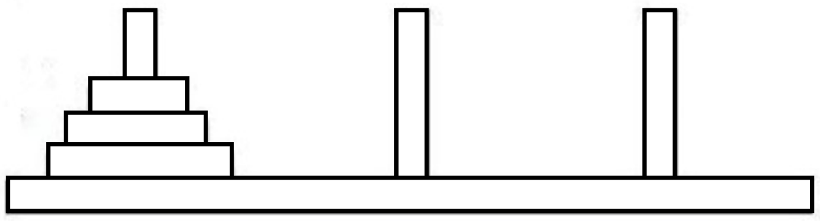

Figure 1: The initial state for $n = 3$

   Complete the following:

   (a) Give the representation of a state in this 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 states 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?).

Solution:

Important note: *While below, we have described a particular state representation for this problem, it is important to note that any state representation is acceptable as long as sufficient justification is given. Minimally, we must require that the state representation is **complete**: For any possible real-world configuration, there must be a corresponding state representation. In other words, a surjection must be formed between the set of all possible **valid** state representations (the validity of which is governed by your representation invariant), and the set of all possible real-world configurations.*

First, we shall assign each disk an index from 1 to n corresponding to its size (i.e. the largest disk will have index n, the second largest disk will have index $n-1$ and so on, with the smallest disk having an index of 1).

In this problem, we can define the state using 3 tuples (L, M, R): each tuple contains the indices of the disks on the left, middle and right pegs respectively, with each index from 1 to n contained in exactly one of these tuples.

*Here, our representation invariant is that the indicies in each tuple must be in **ascending** order. This invariant reduces the total number of unique, valid states of our representation from $n! \times \binom{n+2}{2} = \frac{(n+2)!}{2}$[1] to exactly 3^n,[2] matching the total number of configurations in the actual problem.*

With this state representation in mind, our initial state is $((1, 2, \ldots, n), (), ())$ and the goal state is $((), (), (1, 2, \ldots, n))$.

There are 6 actions in total:

(a) Let a_1 denote the action of 'Transferring a disk from left to middle':
$T(((l_1, \ldots), (m_1, \ldots), (\ldots)), a_1) \to ((\ldots), (l_1, m_1, \ldots), (\ldots)), l_1 < m_1$
$T(((l_1, \ldots), (), (\ldots)), a_1) \to ((\ldots), (l_1), (\ldots))$

(b) Let a_2 denote the action of 'Transferring a disk from left to right':
$T(((l_1, \ldots), (\ldots), (r_1, \ldots)), a_2) \to ((\ldots), (\ldots), (l_1, r_1, \ldots)), l_1 < r_1$
$T(((l_1, \ldots), (\ldots), ()), a_2) \to ((\ldots), (\ldots), (l_1))$

(c) Let a_3 denote the action of 'Transferring a disk from middle to left':
$T(((l_1, \ldots), (m_1, \ldots), (\ldots)), a_3) \to ((m_1, l_1, \ldots), (\ldots), (\ldots)), m_1 < l_1$
$T(((), (m_1, \ldots), (\ldots)), a_3) \to ((m_1), (\ldots), (\ldots))$

(d) Let a_4 denote the action of 'Transferring a disk from middle to right':
$T(((\ldots), (m_1, \ldots), (r_1, \ldots)), a_4) \to ((\ldots), (\ldots), (m_1, r_1, \ldots)), m_1 < r_1$
$T(((\ldots), (m_1, \ldots), ()), a_4) \to ((\ldots), (\ldots), (m_1))$

(e) Let a_5 denote the action of 'Transferring a disk from right to left':
$T(((l_1, \ldots), (\ldots), (r_1, \ldots)), a_5) \to ((r_1, l_1, \ldots), (\ldots), (\ldots)), r_1 < l_1$
$T(((), (\ldots), (r_1, \ldots)), a_5) \to ((r_1), (\ldots), (\ldots))$

(f) Let a_6 denote the action of 'Transferring a disk from right to middle':
$T(((\ldots), (m_1, \ldots), (r_1, \ldots)), a_6) \to ((\ldots), (r_1, m_1, \ldots), (\ldots)), r_1 < m_1$
$T(((\ldots), (), (r_1, \ldots)), a_6) \to ((\ldots), (r_1), (\ldots))$

*Additionally, another way we could represent the state is with 3 **sets**, the difference being by definition, order does not matter for set equality. (e.g. $\{1, 2, 3\} = \{1, 3, 2\} =$*

[1] See Appendix A
[2] See Appendix B

$\{2, 1, 3\}$*). Notice that inherently, the number of possible state representations is exactly* 3^n *even without enforcing a representation invariant.*

Our transition function would be modified slightly. In the case of a_1*:*

$$T((\{l_1, \ldots\}, \{\ldots\}, \{\ldots\}), a_1) \rightarrow (\{\ldots\}, \{l_1, \ldots\}, \{\ldots\}), \forall l \in L(l_1 \leq l) \textit{ and } \forall m \in M(l_1 < m)$$

Similar changes apply for the rest of the actions.

4. 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.

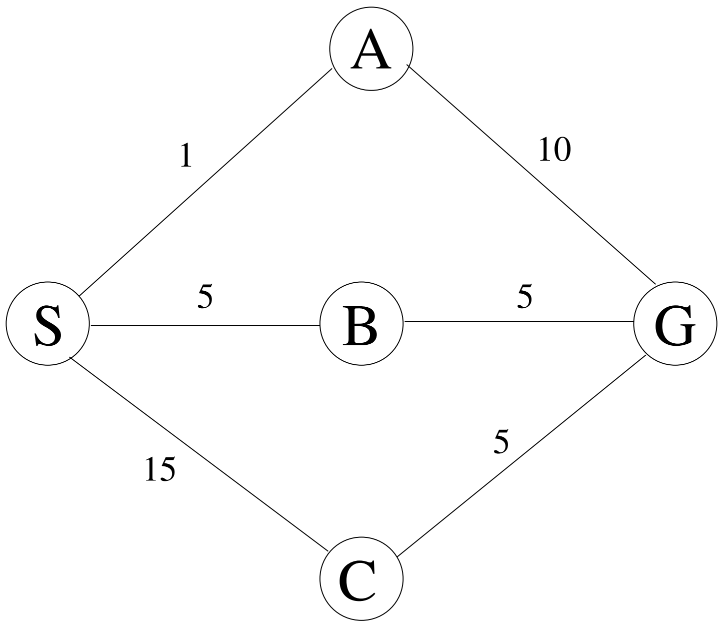

Figure 2: Graph of routes between S and G.

(a) Give a trace of uniform-cost search using *tree* search

Solution:

If we apply the `TREE-SEARCH` *algorithm, the following are the nodes in frontier at the beginning of the loop and at the end of each iteration of the loop: (A node n is followed by its path cost in parenthesis)*

frontier
S(0)
A(1) B(5) C(15)
S(2) B(5) G(11) C(15)
A(3) B(5) B(7) G(11) C(15) C(17)
S(4) B(5) B(7) G(11) G(13) C(15) C(17)
B(5) A(5) B(7) B(9) G(11) G(13) C(15) C(17) C(19)
A(5) B(7) B(9) G(10) S(10) G(11) G(13) C(15) C(17) C(19)
⋮

which is somewhat cumbersome.

(b) Give a trace of uniform-cost search using *graph* search

Solution:

Using the `GRAPH-SEARCH` *algorithm. We obtain the following, which shows the nodes in frontier and explored at the beginning of the loop and at the end of each iteration of the loop (A node n is followed by its path cost in parenthesis):*

frontier	explored
S(0)	
A(1) B(5) C(15)	S
B(5) G(11) C(15)	S, A
G(10) C(15)	S, A, B

(c) 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.

Solution:

After G is generated, it will be sorted together with the other nodes in the frontier. Since node B has lower path cost than G, it is then selected for expansion. Node B then expands to G with a path cost of 10, which gives the solution. By always selecting the node with the least path cost for expansion (i.e., nodes are expanded in increasing order of path cost), uniform-cost search ensures that the first goal node selected for expansion is an optimal (i.e., shortest path) solution.

(d) What's the difference between uniform-cost search (using graph search) and Dijkstra's algorithm?

Solution:

Practically speaking, the 2 algorithms are essentially equivalent! They both traverse the search space in the same manner, using a priority queue to keep track of which nodes/states to visit next. (However, if one were to be pedantic about it, you could argue that Dijkstra finds the shortest path to every node from a single source, while UCS only concerns itself with the shortest path to the goal states. However, in most ways, the algorithms are indistinguishable.)

5. Describe a state space in which iterative deepening search performs much worse than depth-first search.

Solution:

Consider a state space with branching factor b such that all nodes at depth d are solutions and all nodes at shallower depths are not solutions.

Number of search nodes generated by DFS (GRAPH-SEARCH implementation) = $O(bd)$
Number of search nodes generated by IDS = $O(b^{d-1})$

Appendix

Appendix A

The following is by no means a rigorous proof, but rather a sketch of a proof. However, it should still give you a good grasp of how the formula is obtained.

Let us properly define the problem: We want to find the number of ways to arrange n unique elements into 3 tuples: L, M, R. To do so, we shall adopt the following method of construction: "Shuffle" the n elements into a sequence $a_1, a_2, \ldots, a_n$, and split them into 3 partitions such that

$$\begin{aligned} L &= (a_1, a_2, \ldots, a_{n_1}), \\ M &= (a_{n_1+1}, a_{n_1+2}, \ldots, a_{n_1+n_2}), \\ R &= (a_{n_1+n_2+1}, a_{n_1+n_2+2}, \ldots, a_{n_1+n_2+n_3}), \end{aligned}$$

where $n_1 + n_2 + n_3 = n$. (Note that the order of the items matter: $(1, 2, 3) \neq (1, 3, 2)$). The number of unique configurations this method will yield can be calculated as the following:

$$\begin{aligned} &(\text{\# of ways to shuffle } n \text{ elements}) \times (\text{\# of ways to split } n \text{ elements into 3 partitions}) \\ &= n! \times \binom{n+2}{2} \\ &= \frac{(n+2)!}{2} \end{aligned}$$

Furthermore, it can be shown that this method of construction yields a *bijection* with the actual configurations of the problem. Hence, the number of ways to arrange n unique elements into 3 tuples: $\frac{(n+2)!}{2}$.

Appendix B

The following is by no means a rigorous proof, but rather a sketch of a proof. However, it should still give you a good grasp of how the formula is obtained.

In order to show how the number of valid configurations for Tower of Hanoi is 3^n, we shall use a particular method of disk-placing: We shall sequentially place the disks onto any of the 3 pegs *in order from the largest disk to the smallest*. This method of disk-placing will yield 3^n unique configurations: For each disk, there are 3 pegs we can choose to put the disk on, therefore the number of unique configurations using this method: $3 \times 3 \times 3 \times \ldots = 3^n$.

Following this method of construction, we notice 2 things:

- This method of placing will *always* satisfy the constraints of the problem, regardless of which peg you decide to place each disk on
- It can be shown that there will be a *bijection* between this method of disk placing, and the valid configurations of the problem

Therefore, we can assert that the number of possible configurations for Tower of Hanoi is also 3^n.