National University of Singapore
School of Computing
CS3243 Introduction to AI

Tutorial 1: Agents, Problems, and Uninformed Search (Solutions)

1. Sudoku is a popular number puzzle that works as follows. We are given a 9×9 square grid. Some cells contain numbers, while some are blank. Our objective is to fill in the blank cells with numbers from $1 - 9$ such that each row, column and the highlighted 3×3 squares contain no duplicate entries (see Figure 1).

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

Figure 1: A simple Sudoku puzzle

Sudoku puzzles can be easily solved by modelling them as a constraint satisfaction problem (which we will cover later in the course). We will instead consider the problem of *generating* Sudoku puzzles.

One possible procedure for doing this is to start with a completed grid (see Figure 2), and iteratively make some cells blank. We will continue blanking out cells as long as the resulting puzzle can be completed in only one way.

2	5	8	7	3	6	9	4	1
6	1	9	8	2	4	3	5	7
4	3	7	9	1	5	2	6	8
3	9	5	2	7	1	4	8	6
7	6	2	4	9	8	1	3	5
8	4	1	6	5	3	7	2	9
1	8	4	3	6	9	5	7	2
5	7	6	1	4	2	8	9	3
9	2	3	5	8	7	6	1	4

Figure 2: Solution to the Sudoku puzzle in Figure 1.

(a) Determine the properties of the above problem from the perspective of an intelligent agent planning a solution. Complete the table below.

Solution: The completed table is as follows.

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

* Note that a typical human player who attempts to solve a Sudoku puzzle would perceive the problem environment as sequential, since each move made would be seen to impact future moves. However, an intelligent agent solving the Sudoku might not. To such an agent, the problem might seem episodic since it would be able to plan all the moves ahead of making any. To such an agent, the Sudoku puzzle is solved by utilising an internal model to simulate and solve the puzzle. Once the plan is devised, a single action is taken to input all the correct values. Thus, solving the problem is episodic.

Ultimately, the definition of the properties of a problem environment are contingent upon an interpretation of the agent's use cases.

(b) Define the search space for the above problem of generating a Sudoku puzzle by completing the following.

- Give the representation of a state in this problem.
- Using the state representation defined above, specify the initial state and goal state(s).
- Define its actions.
- Using the state representation and actions defined above, specify the transition function T.

Solution:

- A state in this problem is a (partial) valid solution of the sudoku puzzle. More formally, it would be a matrix $A \in \{0, \ldots, 9\}^{9\times 9}$ with 0 representing a blank square.
- The initial state is a completely filled out grid of numbers, where the grid is valid: all rows, columns and 3×3 squares contain all numbers between $1, \ldots, 9$. Note that any state in the problem will be a valid goal state.
- An action would be removing a number from the grid. More formally, we take as input a matrix A and a matrix $E_{i,j}(a)$ where $E_{i,j}(a) \in \{0, \ldots, 9\}^{9\times 9}$ is a matrix of all zeros except for a non-zero value $a \in \{1, \ldots, 9\}$ at coordinate (i, j). An action would be setting $A - E_{i,j}(a)$. Note that actions must not result in boards with two or more solutions.
- The transition model would be $T(A, E_{i,j}(a)) = A - E_{i,j}(a)$.

  It is worthwhile to think about how you would make a good puzzle generator. It would probably make sense to randomly remove numbers rather than deterministically doing so. There are many goal nodes here, but it would probably be boring to use your Sudoku generator if its output was always the same puzzle given the same input.

2. (a) Describe the difference between **Tree Search** and **Graph Search** algroithms.

Solution: Graph search algorithms will only explore non-redundant paths – i.e., they only explore nodes (i) with associated states that have not been visited, or (ii) have been visited, but previously via less optimal paths. There are many ways to implement this, with each implementation restricting the some or all of the redundant paths visited. Tree-based algorthms have no such restriction; they consider all paths, including all redundant ones.

(b) Assuming that ties (when pushing to the frontier) are broken based on ascending alphabetical order (e.g., A before B), specify the order of the nodes checked (i.e., via the goal test) by the following algorithms. Assume that S is the initial state, while G is the goal state.

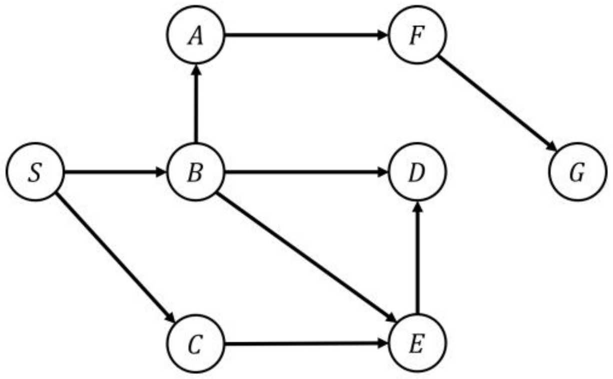

Figure 3: Graph for Question 2b.

You should express your answer in the form S–B–A–F–G (i.e., no spaces, all uppercase letters, delimited by the dash (–) character), which, for example, corresponds to the order S, B, A, F, and G.

i. **Depth-First Search** using a **tree search** implementation
ii. **Depth-First Search** using a **graph search** implementation
iii. **Breadth-First Search** using a **tree search** implementation
iv. **Breadth-First Search** with a **graph search** implementation

Solution:

i. S–C–E–D–B–E–D–D–A–F–G
ii. S–C–E–D–B–A–F–G
iii. S–B–C–A–D–E–E–F–D–D–G
iv. S–B–C–A–D–E–F–G

Do note that this is an academic exercise. We typically only use a graph-implementation with BFS and a tree-implementation with DFS.

3. Prove that the **Uniform-Cost Search** algorithm is optimal as long as each action cost exceeds some small positive constant ϵ. You may also assume the same constraints that make Breadth-First Search complete.

   Solution: We are given the following given assumptions.

   - The branching factor, b, is finite.
   - There is either a solution (i.e., reachable goal state), or the maximum depth, m, is finite.
   - Each action cost exceeds some small positive constant ϵ.

   We may therefore assume that completeness may be assumed. Consequently:

   - Whenever UCS expands a node n, the optimal path to that node has been found. If this was not the case (i.e., the path to n was not optimal - let us denote this path T), there would have to be another frontier node n' on the optimal path from the start node to n (denote this optimal path U). By definition, $g(n)$ via U would be less than $g(n)$ via T, which implies that n' should have been selected first.
   - If action costs are nonnegative, path costs, i.e., g values never get smaller as nodes are added.

   The above two points together imply that UCS expands nodes in the other of the optimal path cost.

4. You are given an n-piece unassembled jigsaw puzzle set (you may assume that each jigsaw piece can be properly connected to either 2, 3, or 4 pieces), which assembles into an $(m \times k)$ rectangle (i.e., $n = m \times k$). There may be multiple valid final configurations of the puzzle. Figure 4 illustrates an example.

   Formulate the above as a search problem. More specifically, define the following:

   - State representation
   - Initial state
   - Actions
   - Transition model
   - Step cost
   - Goal test

   If necessary, you may identify the assumptions you have made. However, assumptions that are contradictory to any instruction in the question, or that are unreasonable, will be invalid.

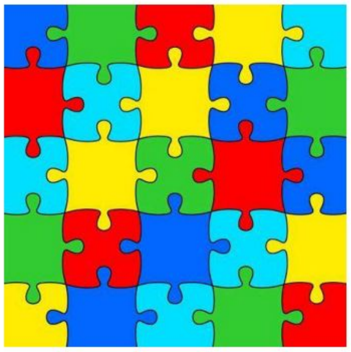

Figure 4: A sample configuration of the jigsaw puzzle.

Solution:

- State representation:
  - Keeps track of the connections (i.e., 2/3/4 per piece) across all the puzzle pieces.
  - Maintain either the set of connected connections or unconnected connections.
- Initial state:
  - Depending on the representation, either the set of connected puzzle pieces is empty, or the set of unconnected puzzle pieces is full. There should also be no connections between puzzle pieces initally.
- Actions:
  - Taking two unconnected puzzle pieces and establishing a connection between them. Note that you must mention that the pair of puzzle pieces picked can be legally connected (i.e., you cannot simply take *any two* puzzle pieces).
- Transition model:
  - Depending on the representation, can either add to the connected set or remove from the unconnected set.
- Step cost
  - Step costs are 1 (or any positive constant value; cannot be 0).
- Goal test
  - Depending on the representation, either the connected set is full, or the unconnected set if empty.
  - Check that for every puzzle piece, there should not be more connections than its number of available legal connections.

5. You have just moved to a strange new city, and you are trying to learn your way around. More specifically, you wish to learn how to get from your home at S to the train station at G.

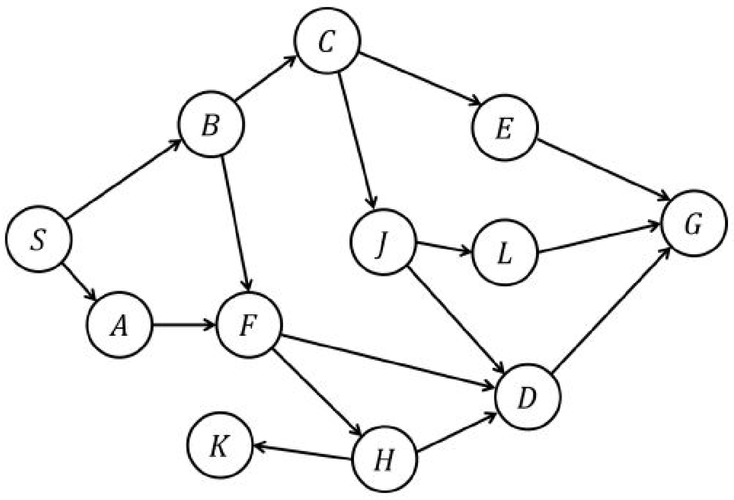

Figure 5: Graphical representation of the city.

Apply the **Depth-First Search** algorithm with a **tree search** implementation. Use ascending alphabetical order to break ties when deciding the priority for pushing nodes into the frontier (i.e., A before B).

Determine the final path found from the start (S) to the goal (G).

Note that you **MUST** express your answer in the form S–B–C–J–L–G (i.e., no spaces, all uppercase letters, delimited by the dash (–) character), which, for example, corresponds to the exploration order of S, B, C, J, L, then G.

Solution: S–B–F–H–D–G