Programming Assignment 1 : Problem Solving Agent

Problem Description : A predefined arrangement of match sticks is required to be rearranged intelligently by a problem solving agent by removing a number of sticks to produce a new arrangement of match sticks. The match sticks are arranged to form squares only. The squares can be of any sizes comprising of 1 match stick each on all four sides, or 2 sticks on each side and so on. The orientation of the match stick can either be vertical or horizontal, independent of the direction of the inflammable material. The intelligent agent has a centralized actuator arm to remove any match stick as a single move. The cost of removing a match stick is one unit. Figure 1 displays an environment with 40 match sticks forming 16 squares of sizes 1x1. The squares can be of sizes 2x2, 3x3 or 4x4 but a larger square cannot have within it a smaller square. A match stick in any arrangement must belong to a square. The goal is to obtain an arrangement of specified number of squares by removing a number of match sticks.

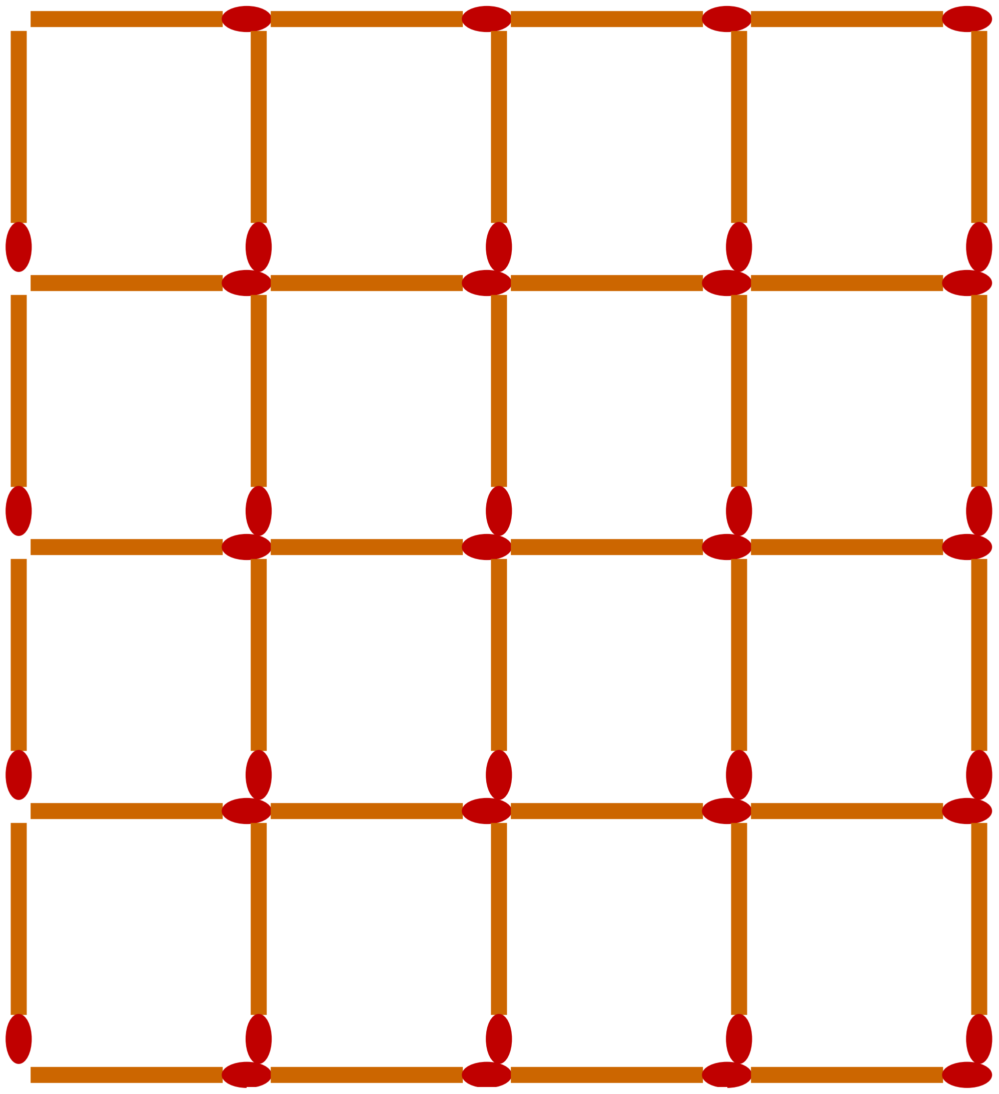

Figure 1: Match stick environment

The intelligent agent's centralized actuator arm (not shown in the figure) is capable of removing one match stick at a time. Though, the intelligent agent's percepts are only local as the environment is not fully observable. Once a match stick is removed, the state of the agent changes. The agent cannot add the removed match stick ever. The agent must know which match sticks are to be removed and in which sequence such that the goal is reached. The goal is defined by the number of non-overlapping squares of any sizes. For example, a goal of obtaining a arrangement with 10 squares is displayed in Figure 2 below. The cost of reaching the goal here is 8 starting with the initial state shown in Fig. 1. Every stick must belong to at least one square and it cannot be left suspended in the air, a square has in its four sides the match sticks.

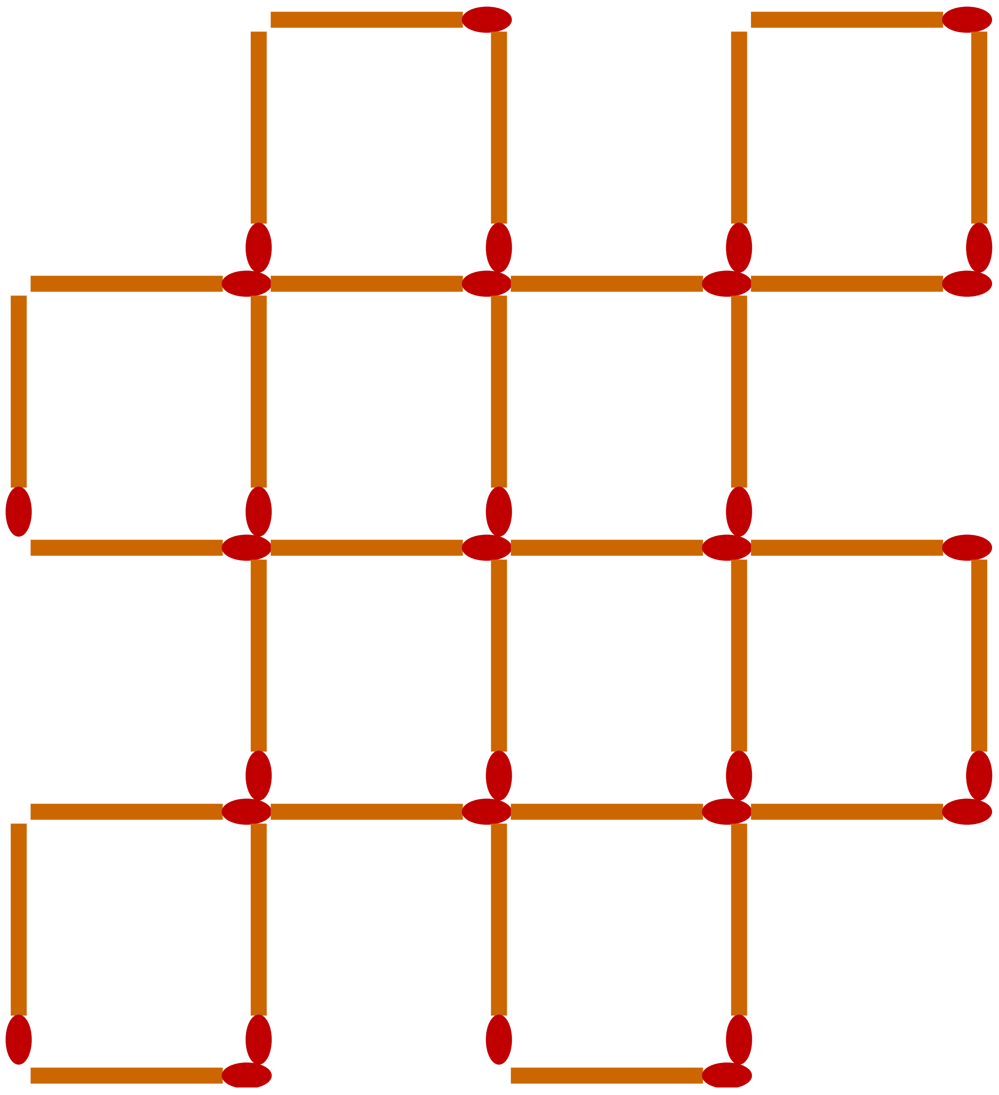
Figure 2: Goal of obtaining 10 squares with path cost of 8

Another arrangement can be as follows (Figure 3) with 9 squares of sizes 1x1 and one square of size 2x2. You can obtain more than one arrangements with 10 squares.

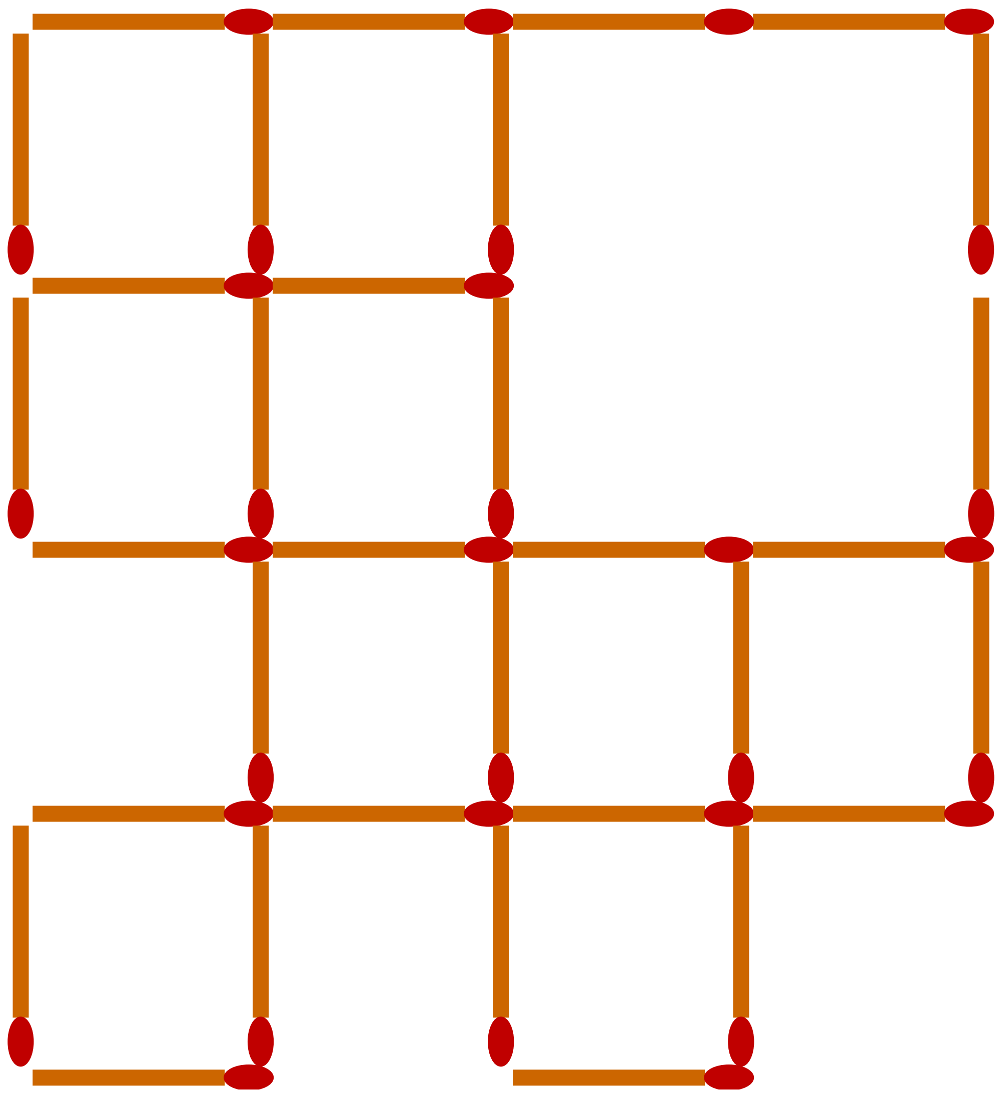

Figure 3: Goal with 10 squares with path cost 8

Write a program in Python programming language to implement the above intelligent agent using uninformed search techniques. Represent the given problem as a search problem, define a state appropriately, define the goal state, action list and obtain the sequence of actions to reach the goal. The initial state is generated randomly. The PEAS descriptions of the intelligent agent are as follows

Performance (P): Path cost
Environment (E): randomly created squares on a 4x4 mesh with each square having match sticks on its four sides. The two neighboring squares share a common match stick
Actuator (A): match stick removing arm where removal is implemented through Graphics
Sensors (S): Match stick position and presence sensors

Implementation

Use Python version 2.7.13 (Windows 10) for implementing your solution. Only standard Python libraries should be used. Support from external libraries such as github will not be accepted in your submissions. Each student must design own solution and write own code. [Refer handout to understand the malpractice policies.]

Modules

You must implement the following in your program.

1. **Function initialStateGenerator (gridSize n, coveragePercentage p):** The function generates a random arrangement of match sticks on a grid of nxn squares. The percentage p describes the coverage of the initial state. For example if n is 4 and p is 50, then the number of squares to be generated randomly will be (nxn)/2. For p =100, all n^2 squares (of sizes 1x1) are generated. For lesser values of p, the function can randomly generate squares of bigger sizes but restricted by the grid size. The created squares are counted for their non-overlapping occurrences. The function ensures that no stick remains suspended without belonging to at least one of the generated squares.
2. **Function GoalTest (state initialState, goal numberOfSquares) function:** Write a function that takes as input an initial state and the number of squares as goal. Notice that there can be a large number of states with the specified number of squares. Maintain a table of such goal states and make your code test against each of these goal states. If any suspended match stick is found then the goal test fails despite meeting the requirement of number of specified squares. The function returns true or false depending on whether the goal is met or not.
3. **Function nextState (state s, action A):** This function takes as input a state s and applies action A. It returns a new next state.
4. **Function createRootNode (state initialState):** This function creates the root node of the search tree for input initial state and returns the node address to be preserved for all tree traversals.

5. **Function addChildNode(treeNode T, state s):** This function first allocates memory to create a node for state s and then adds the node as a sibling of first child of T. [Note that you maintain the nodes/ node addresses for exploration in a separate data structure such as queue or stack.]
6. **Uninformed Search techniques**
   Implement any ***two*** of the following techniques (say T1 and T2)
   - Breadth first search
   - Depth First Search
   - Iterative Deepening Search
7. **Use Turtle graphics**
   Use Python based Turtle graphics to display the actions taken and the movements of the match sticks. First display the initial state of the problem. Once the action path is computed, keep showing the removal of match sticks graphically and mark the goal when reached. Second, a user friendly comprehensive Graphics User Interface (GUI) is required to be created. This GUI must be divided into two partitions (P1 and P2) in the ratio 1:2 vertically. P1 is used for the results in text form and P2 is used to display the graphs. The right side bigger partition (P2) must be further divided into 4 parts. It must have on its upper left quadrant the graphs for T1 (i.e. G1), upper right quadrant, the graph for T2 (i.e. G2). Similarly display G3 in the third quadrant. The details (R1-R12) must appear in one rectangular box in P1 area on the extreme left.
8. **Analysis Module**
   Produce the following analyses and display the resultant values/path etc. on the GUI.
   (a) T1 based analysis
      i. Compute the number of search tree nodes generated till the problem is solved. **[R1]**
      ii. Compute the amount of memory allocated to one tree node. **[R2]**
      iii. Compute the maximum growth of the auxiliary stack or queue used with the search tree. **[R3]**
      iv. Produce the action path in text and in graphics both. For graphics, use color evaporation or flying away of the match stick to show the sequence of actions using T1. **[G1]**
      v. Compute total cost to reach the goal. **[R4]**
      vi. Compute the total time to reach the goal. **[R5]**

   (b) T2 based analysis
      i. Compute the number of search tree nodes generated till the problem is solved. **[R6]**
      ii. Compute the amount of memory allocated to one tree node. **[R7]**
      iii. Compute the maximum growth of the auxiliary stack or queue used with the search tree. **[R8]**
      iv. Produce the action path in text and in graphics both. For graphics, use color evaporation or flying away of the match stick to show the sequence of actions using T2. **[G2]**

v. Compute total cost to reach the goal. **[R9]**
vi. Compute the total time to reach the goal. **[R10]**

(c) Comparative analysis
i. Compare the memory used in T1 and T2. **[R11]**
ii. Run both the techniques 10 times each with randomly generated initial state. Compute average path cost of the path obtained using T1 and T2 respectively. **[R12]**
iii. Plot a graph with two curves displaying the time taken to compute the path by T1 and T2 against the puzzle size varying from 4 x 4 (16 squares mesh) to 8 x 8 (64 squares mesh) in step size of 1 in each direction. **[G3]**

9. **Driver**
The driver must integrate all functionalities and execute the functions appropriately using these options
Option 1: Display the initial environment
Option 2: Find the action sequence and path cost using T1
Option 3: Find the action sequence and path cost using T2
Option 4: Show all results and graphs in the GUI.

Option 1 uses the initialStateGenerator() function and displays the initial environment graphically.
Options 2 and 3 use the appropriate functions and display the result on the console.
Option 4 uses all functions and computes the action sequence using T1 and T2 both and displays all results on the GUI as specified earlier.

Write up
A two page write up illustrating the solution, approach, selected techniques etc. in MS word format (*.docx) is expected to be typed by you. The contents must display the depth of the concepts gained.

Evaluation
The assignment is of 16 marks (8% weight) . The evaluation will be done giving maximum weight to the space and time computation carefully imbibed in the code. No marks will be given for the code alone, if the code does not compile and execute. GUI will have a great impact on presenting your findings, analyses and results. This being the first assignment will be given 5 days extra time to get acquainted with the Python programming language and for experimenting with graphics. Hence the total time given to complete the assignment is 15 days. Remaining five assignments will be of 16 marks each and will be given only 10 days each for completion. As mentioned in the handout, each student will individually work on the assignments and submit through own Nalanda account.

Errata
Students are advised to read the document carefully and inform me any discrepancy, if it exists, immediately. Few corrections, if required will be informed through announcement section of Nalanda.

Submission

Instructions will follow on Nalanda as submission guidelines two days before the deadline.

Vandana
August 26, 2018
