## CSC 479A

## Programming Assignment #1: Uninformed Searching

## 40 pts.

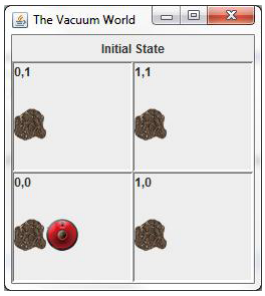
**Background:**

In Module 3 we learned that one common way for an agent to solve a problem is to begin at the problem's initial state and search the entire state-space for a solution, which is defined as a sequence of actions that will lead to a goal state. Searches are commonly represented as trees, with each node representing one possible state of the agent's environment, and the node's children representing all the possible states that would result for each action the agent performs. Depending on the problem and the actions available to the agent, a search tree may be finite or infinite. Module 3 also discussed four uninformed search strategies (breadth-first, depth-first, depth-limited, and iterative deepening), each of which has advantages and disadvantages. In this homework, we will explore some issues involved in performing a depth-limited search using a larger version of the Vacuum World problem discussed in the textbook.

Suppose we have a 2x2 grid for the Vacuum World environment, as shown in Figure 1. An agent in this environment has 5 possible actions available to it (transition model results are shown in parentheses):

1. Suck (removes dirt)
2. Move North (moves agent up 1 location; e.g., from 0,0 to 0,1)
3. Move East (moves agent right 1 location; e.g., from 0,0 to 1,0)
4. Move South (moves agent down 1 location; e.g., from 0,1 to 0,0)
5. Move West (moves agent left 1 location; e.g., from 1,0 to 0,0)

The agent may attempt any of these actions in any location, although obviously some of the actions will have no effect depending on which location the agent is in, and whether or not that location is dirty. Let's assume the path cost function is simple: each action carries a cost of 1.0. The goal state is a state in which all 4 locations contain no dirt, and the agent's location when the goal state is reached is irrelevant.




Actions Possible:

1. Suck (removes dirt)
2. Move North (moves agent up 1 location; e.g., from 0,0 to 0,1)
3. Move East (moves agent right 1 location; e.g., from 0,0 to 1,0)
4. Move South (moves agent down 1 location; e.g., from 0,1 to 0,0)
5. Move West (moves agent left 1 location; e.g., from 1,0 to 0,0)

Figure 1. A 2x2 case of the Vacuum World environment, where the black splotches represent dirt, and the circular red object is an agent designed to clean the environment. In this case, there are 4 possible locations, all of which are dirty, and the agent begins in the location at coordinates 0,0. The actions that are possible for the agent in this environment are listed at the right. The transition model result for each action is shown in parentheses following the action.

Now, suppose the agent is programmed to solve this problem using an uninformed, recursive, depth-limited search, according to the following rules:

1. The agent always considers the actions in the order listed in Figure 1 (Suck, Move North, Move East, Move South, Move West).

2. The agent will consider an action even though that action may not yield a result (e.g., Move West and Move South will have no effect when the agent is in location 0,0).

3. The agent uses a depth limit of 10 for the search.

4. The cost of each action is 1.0.

Figure 2 shows what the search space would look like for the first two levels of the search tree. For the third level, each of the nodes would also have 5 child nodes (Suck, Move North, Move East, Move South, Move West). For the fourth level, each of the nodes created at the third level would have 5 child nodes, and so on.

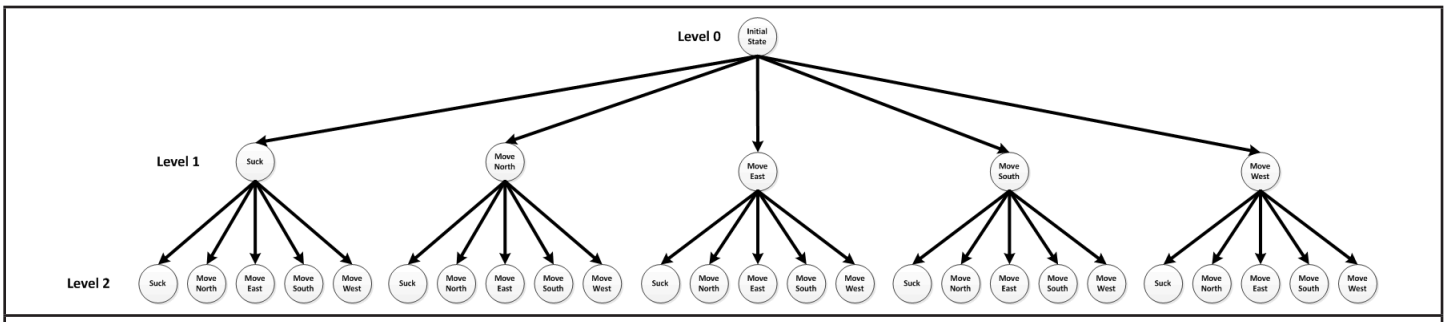



Figure 2. Tree diagram showing the search space for the first two levels of the depth-limited search for the 2x2 Vacuum World environment.

**Online vs. Offline Searching**

There are two ways an agent can deal with a search tree like the one shown in Figure 2. One way is for the agent to actually perform each action on its environment as it traverses the tree. So for example, if the agent encounters a "Suck" action, it would actually suck up any dirt in its current location. Likewise, if the agent encounters a "Move North" action, the agent would physically attempt to move north, and provided there is no obstacle preventing this movement, the agent's location will change. If an agent traverses a search tree, and actually performs each action it encounters, this is known as an online search.

The other approach is for the agent to traverse the tree, but not actually perform each action as it is encountered. Instead, the agent plays out in its mind, so to speak, what the state of its environment would be if the actions were performed in reality. This approach requires keeping track of a virtual copy of the environment that gets updated following each "virtual" action, while leaving the real environment unchanged. If a goal state is found, the agent can then perform the sequence of actions that led to the goal state. This type of search is known as an offline search.

There are pros and cons to each type of search. An online search allows at least the appearance of progress, since the agent is actually doing something. However, if the goal state cannot be achieved, or if a great amount of backtracking is required to reach the goal state for a given search tree, the cost of the search may be prohibitively expensive in terms of energy usage, wear and tear on equipment, etc., since there is a lot of wasted effort exploring paths that ultimately did not lead to a goal state. With an offline search, since no actions are performed until a goal state is found, the cost in terms of those resources will be minimal. However, since the agent doesn't physically do anything until a goal state is found, the agent may give the appearance that it is not working properly, especially if it takes considerable time to find a goal state. In other words, there will only be two outcomes, either nothing will happen, or the agent completes the sequence of actions leading to the goal state, and hence solves the problem.

For this assignment, I want you to implement your search as an offline search. This means your search will need to keep track of a virtual copy of the environment, and the agent will perform actions only on this virtual copy. If your agent needs to backtrack during the search, the virtual environment will need to be reset to its previous state in the search, or your search will not work properly. For example, if your agent moves north by 1 square, and has to backtrack, the agent will have to be returned to its previous location; i.e., it will have to be moved 1 square south, before attempting the next action.

**What you need to do:**

**Part 1 (of 2).**

**Answer the following questions, given the above assumptions about the environment and how the agent performs a depth-limited search:**

1. What is the total possible size of the agent's search space? In other words, assuming the agent searches through the entire search space to a depth limit of 10, how many nodes will the search tree have? Include the initial state node in your answer, but count repeated actions separately; i.e., each "Suck" node is counted as a separate node.

2. Assuming the agent begins the depth-limited search given the initial state shown in Figure 1, and uses a depth limit of 10, will the agent find a sequence of actions that will lead it to the goal state? If so, state how many actions will be in the solution. If not, explain why not. *(Note: you may use the result you obtain from Part 2 to help you answer this question.)*

3. If the agent were solving a 4x4 instance of the Vacuum World, how would that affect the size of the agent's search space?

**Part 2 (of 2).**

**Build an agent to test your hypothesis from Part 1.**

To test your hypothesis for the second question from Part 1, build an agent that solves a 2x2 model of the Vacuum World using a depth-limited search, conforming to the following guidelines:

1. You may use any programming language you wish for this part of the assignment, provided I can compile and run your code on Windows 7 or Windows XP. (You may build the program on another operating system such as Mac OS X or Linux, as long as I can still compile your source code on Windows). If you have any questions about this, please ask.

   ***Note: If you are using Visual Studio, please include your entire project, or solution, not just your source code.***

2. You are free to represent the environment in any way you see fit. You do not have to build a full-fledged, object-oriented, solution for this. The textbook alludes to using a generalized, object-oriented framework for representing problems, agents, etc.; you do not need to be that extensive for this assignment. You can represent the environment as a simple String, for example, and your agent can consist of a few methods for formulating the problem and performing the search. The most difficult part of the assignment may be writing the depth-limited search. Figure 3.17 in the textbook provides excellent pseudocode for this, though.

3. You do **not** need to build a graphical interface for this assignment, unless you wish to do so. A console application that writes text to the standard output will be fine—but please make sure your output is formatted well and understandable. Your output should include the following information:

   a. The number of actions in the solution returned by the search.
   b. The total cost of the solution.
   c. The initial state of the environment.

d. The state of the environment after each action is performed (if practical; if the di covered solution contains more than 100 actions, only display the initial state and the final state).
e. The final state of the environment.
f. In the event no solution is found, your output should indicate this fact, and should show what the final state was when the search terminated.

4. Obviously, the solution you submit must be your own work.

**What you need to turn in:**

A zip file containing the following:

1. Your answers to the 3 questions from Part 1.

2. Your source code from Part 2. If you use Visual Studio to build your program, please submit your entire Visual Studio Solution. If your program uses any third-party or non-standard component libraries, please include those in your submission.

**Rubric:**

| **Item** | **Points** |
|---|---|
| Part 1, Question #1: | 2 |
| Part 1, Question #2: | 6 |
| Part 1, Question #3: | 2 |
| Part 2 (agent correctly performs the depth-limited search): | 30 |
| | |
| **Total:** | **40** |
| | |
| Partial credit will be awarded where it is possible to do so. | |