E/15/366

S.THINESH

1.Define in your own words the following terms: agent, agent function, agent program, rationality, autonomy, reflex agent, model-based agent, goal-based agent, utility-based agent, learning agent.

- **Agent:** An agent is something that acts. An agent acts on behalf of a person. It is an entity that acts in response to the environmental issues.
- **Agent function:** Agent function is the action performed by an agent in response to the environmental issues.
- **Agent program:** The agent program implements the agent function.
- **Rationality:** Rationality is the property of an agent that selects the action to be performed.
- **Autonomy:** Autonomy is a property of an agent being itself and making decisions of its own.
- **Reflex agent:** A reflex agent is an agent which chooses actions on the basis of current percept.
- **Model-based agent:** An agent that uses the model of the world is called as a model-based agent. The knowledge about "how the world works" is called a model of the world.
- **Goal-based agent:** Goal-based agents are model-based agents which sorts goal information that describes situations.
- **Utility-based agent:** This is an agent that uses an explicit utility function that maximizes the expected utility.
- **Learning agent:** This is an agent that improves its behavior based on its experiences and learning.

2.This exercise explores the differences between agent functions and agent programs.

(a) Can there be more than one agent program that implements a given agent function? Give an example, or show why one is not possible.

Yes, there can be more than one agent program that implements an agent function. For example, the agent function of navigation may be implemented by agent programs which approach this problem through stored tables versus a map plus a general-purpose search.

(b) Are there agent functions that cannot be implemented by any agent program?

Yes, there are functions like the halting problem that can not be implemented by any agent program.

(c) Given a fixed machine architecture, does each agent program implement exactly one agent function?

Yes, given a fixed machine architecture, each agent program implements exactly one agent function. To implement multiple agent functions this would require the agent program to select different actions (or different distributions of actions) given the same percept sequence.

(d) Given an architecture with n bits of storage, how many different possible agent programs are there?

If a is the total number of actions, then the number of possible programs is 2^n, 2^n internal states and a choice for each state. There will be 2^n possible programs.

(e) Suppose we keep the agent program fixed but speed up the machine by a factor of two. Does that change the agent function?

No, that does not change the agent function directly. However, this may allow the program to compress its memory further and to retain a better model of the world.

3. Explain why problem formulation must follow goal formulation.

- Goal formulation is used to steer the agent in the right direction, hence ignoring any redundant actions. Problem formulation must follow this since it is based off of goal formulation which is the process of deciding what actions and states to consider given a certain goal. A goal may be set in stone, but how you achieve it can vary. The most optimal way is chosen usually.

4. The GENERAL-SEARCH algorithm consists of three steps: goal test, generate, and ordering function, in that order. It seems a shame to generate a node that is in fact a solution, but to fail to recognize it because the ordering function fails to place it first.

1. Write a version of GENERAL-SEARCH that tests each node as soon as it is generated and stop immediately if it has found a goal.

```
function General-Search-New(problem, Queuing-FN) returns a solution, or failure

    nodes ß Make-Queue(MAKE-NODE(INITIAL STATE[problem]))

    loop do

            if nodes are empty then return failure

            node ß REMOVE-FRONT(nodes)

            new-node ß EXPAND(node, OPERATORS[problem])
```

```
if GOAL-TEST[*problem*] applied to the STATE of any of new-nodes succeeds **then return** that node
nodes ß QUEUING-FN-FRONT(*nodes*, new-nodes)
    **end**
```

2. Show how the GENERAL-SEARCH algorithm can be used unchanged to do this by giving it the proper ordering function

   We could just call the original GENERAL-SEARCH with a queuing function which contains a call to the goal test function. If a node passes the goal test, it is placed at the very front of the queue; otherwise it is placed at the "normal" position.

5. Give a description of a search space in which Iterative Deepening performs much worse than Depth First search. Give the respective complexities for this domain in Big O notation.

- A tree with branching factor always equal to 1. For n nodes, DFS would visit O(n) states, while IDS would visit O(n(n+1) 2) = $O(n^2)$.
- Regarding a domain in which every state has a single successor, and there is a single goal at depth n. Then depth-first search will find the goal in n steps, whereas iterative deepening search will take 1 + 2 + 3 + … + n = $O(n^2)$ steps.

6. Give a complete problem formulation for each of the following. Choose a formulation that is precise enough to be implemented.

a) Using only four colors (e.g., R/G/B/W), you have to color a planar map in such a way that no two adjacent regions have the same color.

   Initial State: All regions are uncolored.

   Actions: Assign color to an uncolored area.

   Transition Model: The uncolored area is now colored and cannot be colored again.

   Goal Test: All regions have color, and no two adjacent regions have the same color.

   Cost Function: Number of areas.

b) A 3-foot-tall monkey is in a room where some bananas are suspended from the 8- foot ceiling. He would like to get the bananas. The room contains two stackable, movable, climbable 3-foot-high crates.

   Initial State: An 8-foot high room, 2 crates, 1 money, bananas.

   Actions: Monkey moving and stacking boxes to reach bananas.

   Transition Model: the boxes have either moved, been stacked, or both.

Goal Test: The monkey gets the bananas.

Cost Function: Number of actions.

c) You have three jugs, measuring 12 gallons, 8 gallons, and 3 gallons, and a water faucet. You can fill the jugs up or empty them out from one to another or onto the ground. You need to measure out exactly one gallon.

Initial State: Jugs are empty

Actions: Fill jugs up or transfer water between them.

Transition Model: Amount of water in each jug changes.

Goal Test: Is there exactly one gallon?

Cost Function: Number of actions.

8. Both the performance measure and the utility function measure how well an agent is doing. Explain the difference between the two.

A performance measure that is typically imposed by the designer is used to evaluate the behavior of the agent in environment. It tells does agent do what it's supposed to do in the environment whereas a utility function is used by an agent itself to evaluate how desirable states are. Some paths to the goal are more efficient than others which path is the best. The difference is Does agent do what it's supposed to do vs does agent do it in optimal way. The utility function may not be the same as the performance measure. An agent may have no explicit utility function at all, whereas there is always a performance measure.

9. Define in your own words the following terms: state, state space, search tree, search node, goal, action, successor function, and branching factor

- State- It refers to how things are at a particular time period.
- State Space- The complete amount of states which are possible from the initial state through actions of the agent.
- Search Tree- A sequence of actions and states which can be displayed originating from the initial state.
- Search Node- The current state after the action that led up to it.
- Goal- A desired state that the agent wishes to arrive at through its actions.
- Action- The agents influence on its surroundings in an attempt to achieve its goal state.
- Transition model- The description of what the actions do.
- Branching factor- The maximum number of children that a given parent node could have.

10.Does a finite state space always lead to a finite search tree? How about a finite state space that is a tree? What types of state spaces always lead to finite search trees?

A finite search space can result in an infinite search tree if states are repeated. For example, in the 8 puzzles, you can always return to the state of a node's parent by moving the empty tile back. By doing this you can generate an infinitely long chain of search nodes. This can be prevented by avoiding repeated states.

11. Consider a state space where the start state is number 1 and the successor function for state n returns two states, numbers 2n and 2n + 1.

(a) Draw the portion of the state space for states 1 to 15.

- Level 1: 1
- Level 2: 2 3
- Level 3: 4 5 6 7
- Level 4: 8 9 10 11 12 13 14 15

(b) Suppose the goal state is 13. List the order in which nodes will be visited for breadth first search, depth-limited search with limit 3, and iterative deepening search.

- Breadth-first search: 1 2 3 4 5 6 7 8 9 10 11 12 13
- Depth-limited search: 1 2 4 8 9 5 10 11 3 6 12 13
- Iterative-deepening search: 1 1 2 3 1 2 4 5 3 6 7 1 2 4 8 9 5 10 11 3 6 12 13

(c) Would bidirectional search be appropriate for this problem? Why or why not?

Yes. The predecessor function for a node is floor(n/2). The branching factor for searching backward from the goal is 1. The forward branching factor is 2. Since both factors are small and similar, bidirectional search could work. Actually, we will probably be better off just doing the backward search alone since it's branching factor is only 1.

14. we said that we would not consider problems with negative path costs. In this exercise, we explore this decision in more depth.

a. Suppose that actions can have arbitrarily large negative costs; explain why this possibility would force any optimal algorithm to explore the entire state space.

Because you never know whether there exists a negative step cost that can be included into your path to reduce the path cost until you have checked every node in the graph.

b. Does it help if we insist that step costs must be greater than or equal to some negative constant c? Consider both trees and graphs.

<Tree> Yes. If the amount of tree nodes is known then the remaining nodes waiting for checking is known. Suppose that the number of remaining nodes is N, then the possible minimum path cost of these remaining nodes will be -NC. If current path cost is (best path cost + NC), then it is unnecessary to check these remaining nodes. The reason is that it can not

provide a path cost lower than the current best path cost. <Graph> No. If there exists a node containing negative cost, we can always use this node to reduce current best path cost repeatedly, and form another current best path cost.

c. Suppose that a set of actions forms a loop in the state space such that executing the set in some order results in no net change to the state. If all of these actions have negative cost, what does this imply about the optimal behavior for an agent in such an environment?

The agent will keep looping the loop since it can reduce the path cost by that way.

d. One can easily imagine actions with high negative cost, even in domains such as route finding. For example, some stretches of road might have such beautiful scenery as to far outweigh the normal costs in terms of time and fuel. Explain, in precise terms, within the context of state-space search, why humans do not drive around scenic loops indefinitely, and explain how to define the state space and actions for route finding so that artificial agents can also avoid looping.

In real world, we have limited resources, such as limited fuel, limited money, limited time, etc. That's why people can't drive around scenic loops indefinitely. For state-space, we can set factors with limited amount just like real world. Each action can only be executed if there are resource remained, and it will lead to resource consumption. Therefore, when it is out of resource, it can't take any action anymore (stop looping).