Question 1. **[3 Points].**

Comment the following statement by "True" or "False" with respect to your understanding to the terms.

No.	Text	True/False
1.	Intelligent agents are supposed to maximize their performance measure. As we mentioned in Chapter 2, achieving this is sometimes simplified if the agent can adopt a goal and aim at satisfying it.	T
2.	Goals help organize behaviour by limiting the objectives that the agent is trying to achieve and hence the actions it needs to consider. Goal formulation, based GOAL FORMULATION on the current situation and the agent's performance measure, is the first step in problem solving.	T
3.	The agent's task is to find out how to act, now and in the future, so that it reaches a goal state. Before it can do this, it needs to decide (or we need to decide on its behalf) what sorts of actions and states it should consider.	T
4.	The process of looking for a sequence of actions that reaches the goal is called search. A search algorithm takes a problem as input and returns a solution in the form of an action sequence. Once a solution is found, the actions it recommends can be carried out. This is called the execution phase.	T
5.	A genetic algorithm (or GA) is a variant of stochastic beam search in which successor states are generated by combining two parent states rather than by modifying a single state. The analogy to natural selection is the same as in stochastic beam search.	T
6.	CSP search algorithms take advantage of the structure of states and use general-purpose rather than problem-specific heuristics to enable the solution of complex problems. The main idea is to eliminate large portions of the search space all at once by identifying variable/value combinations that violate the constraints.	T

Question 2. **(7 Marks)**

2.1 A search problem can be defined formally by five components: describe the five components: (5 Marks)

3

	Component	Description
1	Initial state	We start the search from the initial state, it's given
2	Action	each move we make in the path to reach the goal
3	Goal-state	it is given, ~~this is where~~ this is the solution to the search problem, here we terminate
4	the path	the shortest path from start to goal
5	Algorithm	Algorithm to search for the goal

2.2 Before we get into the design of specific search algorithms, we need to consider the criteria that might be used to choose among them. Give the title and describe the four ways that We can use to evaluate an algorithm's performance. (2 Marks)

	Title	Description
1	Time complexity	~~how much time will the search algorithm~~ ~~The time~~ the time it will take to terminate
2	memory ~~space~~	~~to~~ the space it will take in the memory
3	Goal	does it reach the goal or not
4	implementation	~~is it cost~~ ~~to~~ how hard it is to implement

1.5

Question 3. (4 Points)

Background:

To understand local search, we find it useful to consider the **state-space landscape** (as in Figure 4.1). A landscape has both "location" (defined by the state) and "elevation" (defined by the value of the heuristic cost function or objective function). If elevation corresponds to cost, then the aim is to find the lowest valley—**a global minimum**; if elevation corresponds to an objective function, then the aim is to find the highest peak—**a global maximum**. (You can convert from one to the other just by inserting a minus sign.) Local search algorithms explore this landscape.

A **complete** local search algorithm always finds a goal if one exists; an **optimal** algorithm always finds a global minimum/maximum.

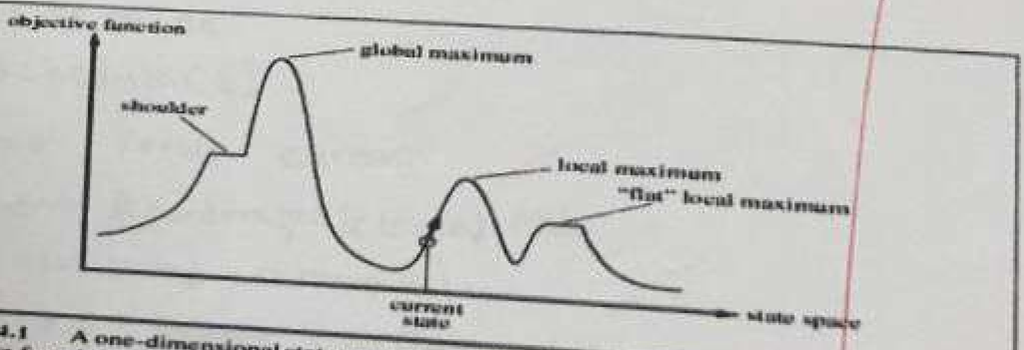

Figure 4.1 A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

The **hill-climbing** search algorithm (**steepest-ascent** version) is shown in Figure 4.2. It is simply a loop that continually moves in the direction of increasing value—that is, uphill. It terminates when it reaches a "peak" where no neighbor has a higher value. The algorithm does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function. Hill climbing does not look ahead beyond the immediate neighbors of the current state. This resembles trying to find the top of Mount Everest in a thick fog while suffering from amnesia.

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current ← MAKE-NODE(problem.INITIAL-STATE)
  loop do
    neighbor ← a highest-valued successor of current
    if neighbor.VALUE ≤ current.VALUE then return current.STATE
    current ← neighbor
```

Figure 4.2 The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor; in this version, that means the neighbor with the highest VALUE, but if a heuristic cost estimate h is used, we would find the neighbor with the lowest h.

Question: A hill-climbing algorithm that *never* makes "downhill" moves toward states with lower value (or higher cost) is guaranteed to be incomplete, because it can get stuck on a local maximum. In contrast, a purely random walk—that is, moving to a successor chosen uniformly at random from the set of successors—is complete but extremely inefficient. Therefore, it seems reasonable to try to combine hill climbing with a random walk in some way that yields both efficiency and completeness. **Simulated annealing** is such an algorithm.

Give the algorithm of the Simulated Annealing:

function Simulated-Annealing ~~(problem)~~ (problem, Schedule) returns a solution
current ← Make-Node (problem, Initial-State)

for t=1 to ∞ do
T= Schedule(t)
if T=0 return current
next ← Randomly-Selected-Node
ΔE = next.Value - current.Value
if ΔE > 0 ~~Return~~ current ← next
else
current ← next only the probability will be $e^{\Delta E/T}$

4

Question 4. (3 Marks)

Like beam searches, GAs begin with a set of k randomly generated states, called the **population**. Each state, or **individual**, is represented as a string over a finite alphabet—most ommonly, a string of 0s and 1s.

Give the general algorithm of the Gas:

function Genetic-Algorithm (Population, Fitness) returns population with one more individual (child)

new_population ← empty-set

loop do {

x ← Randomly-selected (Fitness Fn)

y ← Randomly-selected (Fitness Fn)

if x and y are fit {

Child ← combine (x, y)

new_population ← Child

Break}

}

Population ← Population + new_population

Return Population

3

Question5: [3 Points]

Background:

A constraint satisfaction problem consists of three components, X,D, and C:
X is a set of variables, {X1, . . . ,Xn}.
D is a set of domains, {D1, . . . ,Dn}, one for each variable.
C is a set of constraints that specify allowable combinations of values.
Each domain Di consists of a set of allowable values, {v1, . . . , vk} for variable Xi.
Each constraint Ci consists of a pair **(scope, rel)**, where **scope** is a tuple of variables that participate in the constraint and **rel** is a relation that defines the values that those variables can take on.

Question:

We are looking at a map of Australia showing each of its states and territories (Figure 6.1). We are given the task of coloring each region either red, green, or blue in such a way that no neighboring regions have the same color. **Formulate this as a CSP, by defining the variables with their domains, the set of constraints and by drawing the constraint graph.**

(Figure 6.1)

3

X = {WA, NT, SA, Qu, NE, Vi}

C = color

C = {WAC ≠ NTC, WAC ≠ SAC, NTC ≠ SAC, NTC ≠ QuC, SAC ≠ QuC, SAC ≠ NEC, SAC ≠ ViC, QuC ≠ NEC, NEC ≠ ViC}

D = {Red, green, blue}