****

**SET-A**

Subject: - AI

II Year MCA. IV Semester IMid Term Examination, February – 2018

Time: -2 Hrs. [Maximum Marks: -20]

[Min. Passing Marks: 08]

**Part 1**

1. **I)** Predicate logic is a deductive symbolic logical system that allows us to determine valid reasoning and consistency between propositions.
2. **II)** Depth First Search (DFS) searches deeper into the problem space.
3. **III)** Backward chaining (or backward reasoning) is an inference method that can be described colloquially as working backward from the goal(s). It is used in automated theorem proverbs, inference engines, proof assistants and other artificial intelligence applications.

**1. IV)** Artificial intelligence (AI) is an area of computer science that emphasizes the creation of intelligent machines that work and react like humans.

**1. V)** An expert system is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning through bodies of knowledge, represented mainly as if–then rules rather than through conventional procedural code.

**Part 2**

1. **I) Logical Representation Scheme:**

A language with concrete rules

● No ambiguity in representation (may be other errors!)

● Allows unambiguous communication and processing

● Very unlike natural languages e.g. English Many ways to translate between languages

● A statement can be represented in different logics

● And perhaps differently in same logic Expressiveness of a logic

● How much can we say in this language? Not to be confused with logical reasoning

● Logics are languages, reasoning is a process (may use logic)

**2. II) STRUCTURED REPRESNTATION:**

Representing knowledge using logical formalism, like predicate logic, has several advantages. They can be combined with powerful inference mechanisms like resolution, which makes reasoning with facts easy. But using logical formalism complex structures of the world, objects and their relationships, events, sequences of events etc. cannot be described easily.

A good system for the representation of structured knowledge in a particular domain should posses the following four properties:

**(i) Representational Adequacy:-** The ability to represent all kinds of knowledge that are needed in that domain.

**(ii) Inferential Adequacy: -** The ability to manipulate the represented structure and infer new structures.

**(iii) Inferential Efficiency:-** The ability to incorporate additional information into the knowledge structure that will aid the inference mechanisms.

**(iv) Acquisitional Efficiency: -** The ability to acquire new information easily, either by direct insertion or by program control.

**Part 3**

1. **I)** AI has been dominant in various fields such as:
2. **Gaming**

AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where

machine can think of large number of possible positions based on heuristic knowledge.

1. **Natural Language Processing**

It is possible to interact with the computer that understands natural language spoken

by humans.

1. **Expert Systems**

There are some applications which integrate machine, software, and special

information to impart reasoning and advising. They provide explanation and advice to

the users.

1. **Vision Systems**

These systems understand, interpret, and comprehend visual input on the computer.

1. **Speech Recognition**

Some intelligent systems can handle different accents, slang words, noise in the background, change in human’s noise due to cold, etc.

1. **Handwriting Recognition**

The handwriting recognition software reads the text written on paper by a pen or on

screen by a stylus. It can recognize the shapes of the letters and convert it into editable

text.

1. **Intelligent Robots**

Robots are able to perform the tasks given by a human. They have sensors to detect

physical data from the real world such as light, heat, temperature, movement, sound,

bump, and pressure.

1. **II) Water Jug Problem:** Consider the following problem: A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?

**State Representation and Initial State –** we will represent a state of the problem as a tuple (x, y) where x represents the amount of water in the 4-gallon jug and y represents the amount of water in the 3-gallon jug. Note 0 ≤ x ≤ 4, and 0 ≤ y ≤ 3. Our initial state: (0,0)

**Goal Predicate** – state = (2,y) where 0 ≤ y ≤ 3.

**Operators –** we must define a set of operators that will take us from one state to another:

1. Fill 4-gal jug (x,y) → (4,y)

x< 4

2. Fill 3-gal jug (x,y) → (x,3)

y< 3

3. Empty 4-gal jug on ground (x,y) → (0,y)

x> 0

4. Empty 3-gal jug on ground (x,y) → (x,0)

y> 0

5. Pour water from 3-gal jug (x,y) → (4, y - (4 - x))

to fill 4-gal jug 0 <x+y ≥ 4 and y > 0

6. Pour water from 4-gal jug (x,y) → (x - (3-y), 3)

to fill 3-gal-jug 0 <x+y ≥ 3 and x > 0

7. Pour all of water from 3-gal jug (x,y) → (x+y, 0)

into 4-gal jug 0 <x+y ≤ 4 and y ≥ 0

8. Pour all of water from 4-gal jug (x,y) → (0, x+y)

into 3-gal jug 0 <x+y ≤ 3 and x ≥ 0

Through Graph Search, the following solution is found :

Gals in 4-gal jug Gals in 3-gal jug Rule Applied

0 0

1. Fill 4

4 0

6. Pour 4 into 3 to fill

1 3

4. Empty 3

1 0

8. Pour all of 4 into 3

0 1

1. Fill 4

4 1

6. Pour into 3

2 3

**Part 4**

1. **I)** **Branch and Bound**

Depth-first **branch-and-bound** search is a way to combine the space saving of depth-first search with heuristic information. It is particularly applicable when many paths to a goal exist and we want an optimal path. As in A\* search, we assume that h(n) is less than or equal to the cost of a lowest-cost path from n to a goal node.

The idea of a branch-and-bound search is to maintain the lowest-cost path to a goal found so far, and its cost. Suppose this cost is bound. If the search encounters a path p such that cost(p)+h(p) ≥ bound, path pcan be pruned. If a non-pruned path to a goal is found, it must be better than the previous best path. This new solution is remembered and bound is set to the cost of this new solution. It then keeps searching for a better solution.

Branch-and-bound search generates a sequence of ever-improving solutions. Once it has found a solution, it can keep improving it. Branch-and-bound search is typically used with depth-first search, where the space saving of the depth-first search can be achieved. It can be implemented similarly to depth-bounded search, but where the bound is in terms of path cost and reduces as shorter paths are found. The algorithm remembers the lowest-cost path found and returns this path when the search finishes.

1: **Procedure** DFBranchAndBound(G,s,goal,h,bound0)   
2:           **Inputs**  
3:                     G: graph with nodes N and arcs A   
4:                     s: start node   
5:                     goal: Boolean function on nodes   
6:                     h: heuristic function on nodes   
7:                     bound0: initial depth bound (can be ∞ if not specified)   
8:           **Output**  
9:                     a least-cost path from s to a goal node if there is a solution with cost less than bound0   
10:                     or ⊥ if there is no solution with cost less than bound0   
11:           **Local**  
12:                     best\_path: path or ⊥   
13:                     bound: non-negative real   
14:                     **Procedure** cbsearch(⟨n0,...,nk⟩)   
15:                               **if** (cost(⟨n0,...,nk⟩)+h(nk) < bound) **then**   
16:                               **if** (goal(nk)) **then**   
17:                                         best\_path ←⟨n0,...,nk⟩   
18:                                         bound ←cost(⟨n0,...,nk⟩)   
19:                               **else**  
20:                                         **for each** arc ⟨nk,n⟩∈A **do**   
21:                                                   cbsearch(⟨n0,...,nk,n⟩)   
22:           best\_path ←⊥   
23:           bound ←bound0   
24:           cbsearch(⟨s⟩)   
25:           **return** best\_path

Figure 3.11: Depth-first branch-and-bound search

The algorithm is shown in [Figure](http://artint.info/html/ArtInt_63.html#branch-and-bound-fig). The internal procedure cbsearch, for cost-bounded search, uses the global variables to provide information to the main procedure.

Initially, bound can be set to infinity, but it is often useful to set it to an overestimate, bound0, of the path cost of an optimal solution. This algorithm will return an optimal solution - a least-cost path from the start node to a goal node - if there is a solution with cost less than the initial bound bound0.

If the initial bound is slightly above the cost of a lowest-cost path, this algorithm can find an optimal path expanding no more arcs than A\* search. This happens when the initial bound is such that the algorithm prunes any path that has a higher cost than a lowest-cost path; once it has found a path to the goal, it only explores paths whose the f-value is lower than the path found. These are exactly the paths that A\* explores when it finds one solution.

If it returns ⊥ when bound0=∞, there are no solutions. If it returns ⊥ when bound0 is some finite value, it means no solution exists with cost less than bound0. This algorithm can be combined with iterative deepening to increase the bound until either a solution is found or it can be shown there is no solution.

**4. II)** A production system consists of rules and factors. Knowledge is encoded in a declarative from which comprises of a set of rules of the form

Situation ------------ Action

SITUATION that implies ACTION.

Example:-

IF the initial state is a goal state THEN quit.

The major components of an AI production system are

i. A global database

ii. A set of production rules and

iii. A control system

The goal database is the central data structure used by an AI production system. The production system. The production rules operate on the global database. Each rule has a precondition that is either satisfied or not by the database. If the precondition is satisfied, the rule can be applied. Application of the rule changes the database. The control system chooses which applicable rule should be applied and ceases computation when a termination condition on the database is satisfied. If several rules are to fire at the same time, the control system resolves the conflicts.

Four classes of production systems:-

1. A monotonic production system

2. A non monotonic production system

3. A partially commutative production system

4. A commutative production system.

Advantages of production systems:-

1. Production systems provide an excellent tool for structuring AI programs.

2. Production Systems are highly modular because the individual rules can be added, removed or modified independently.

3. The production rules are expressed in a natural form, so the statements contained in the knowledge base should the a recording of an expert thinking out loud.

Disadvantages of Production Systems:-

One important disadvantage is the fact that it may be very difficult analyse the flow of control within a production system because the individual rules don’t call each other.

Production systems describe the operations that can be performed in a search for a solution to the problem. They can be classified as follows.

Monotonic production system :- A system in which the application of a rule never prevents the later application of another rule, that could have also been applied at the time the first rule was selected.

Partially commutative production system:-

A production system in which the application of a particular sequence of rules transforms state X into state Y, then any permutation of those rules that is allowable also transforms state x into state Y.

Theorem proving falls under monotonic partially communicative system. Blocks world and 8 puzzle problems like chemical analysis and synthesis come under monotonic, not partially commutative systems. Playing the game of bridge comes under non monotonic , not partially commutative system.

For any problem, several production systems exist. Some will be efficient than others. Though it may seem that there is no relationship between kinds of problems and kinds of production systems, in practice there is a definite relationship.

Partially commutative , monotonic production systems are useful for solving ignorable problems. These systems are important for man implementation standpoint because they can be implemented without the ability to backtrack to previous states, when it is discovered that an incorrect path was followed. Such systems increase the efficiency since it is not necessary to keep track of the changes made in the search process.

Monotonic partially commutative systems are useful for problems in which changes occur but can be reversed and in which the order of operation is not critical (ex: 8 puzzle problem).

Production systems that are not partially commutative are useful for many problems in which irreversible changes occur, such as chemical analysis. When dealing with such systems, the order in which operations are performed is very important and hence correct decisions have to be made at the first time itself.

****

**SET-B**

Subject: - AI

II Year MCA. IV Semester IMid Term Examination, February – 2018

Time: -2 Hrs. [Maximum Marks: -20]

[Min. Passing Marks: 08]

**Part 1**

1. **I)** Propositional calculus is the branch of logic concerned with the study of propositions (whether they are true or false) that are formed by other propositions with the use of logical connectives.
2. **II)** Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures.
3. **III)** Forward chaining (or forward reasoning) is one of the two main methods of reasoning when using an inference engine and can be described logically as repeated application of modus ponens.
4. **IV)** Artificial intelligence (AI) is an area of computer science that emphasizes the creation of intelligent machines that work and react like humans.
5. **V)** An expert system is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning through bodies of knowledge, represented mainly as if–then rules rather than through conventional procedural code.

**Part 2**

1. **I) Network representation:** This representation captures the knowledge as a graph where knots describe an object or concept of problems encountered, while edge describes the relationship or association between them.

* **Semantic Networks:**

A semantic network or net is a graphic notation for representing knowledge in patterns of interconnected nodes and arcs.

* **Conceptual Graphs:**

Conceptual graphs (CGs) are a system of logic based on the existential graphs of Charles Sanders Peirce and the semantic networks of artificial intelligence.

* **Conceptual Dependency:**

Conceptual dependency theory is a model of natural language understanding used in artificial intelligence systems.

1. **II) Procedural Representation:** Representation describes knowledge as set of instructions to solve a problem. In these knowledge is represented as procedures. Default reasoning and probabilistic reasoning are examples of procedural methods. In these, heuristic knowledge of “How to do things efficiently “can be easily represented.

In practice most of the knowledge representation employs a combination of both. Most of the knowledge representation structures have been developed to handle programs that handle natural language input. There are several types of schemas that have proved useful in AI programs. They include

**(i) Frames:-** Used to describe a collection of attributes that a given object

possesses (e.g.: description of a chair).

**(ii) Scripts:-** Used to describe common sequence of events(e.g.:- a restaurant

scene).

**(iii) Stereotypes: -** Used to described characteristics of people.

**(iv) Rule models:-** Used to describe common features shared among a set of rules

in a production system.

**Part 3**

1. **I)** Hill Climbing is heuristic search used for mathematical optimization problems in the field of Artificial Intelligence .  
   Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum.

* In the above definition, **mathematical optimization problems** implies that hill climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs. Example-[Travelling salesman problem](https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/) where we need to minimize the distance traveled by salesman.
* ‘Heuristic search’ means that this search algorithm may not find the optimal solution to the problem. However, it will give a good solution in **reasonable time.**
* A **heuristic function** is a function that will rank all the possible alternatives at any branching step in search algorithm based on the available information. It helps the algorithm to select the best route out of possible routes.

**Features of Hill Climbing**

1. **Variant of generate and test algorithm :** It is a variant of generate and test algorithm. The generate and test algorithm is as follows :

1. Generatea possible solutions.  
2. Test to see if this is the expected solution.  
3. If the solution has been found quit else go to step 1.

Hence we call Hill climbing as a variant of generate and test algorithm as it takes the feedback from test procedure. Then this feedback is utilized by the generator in deciding the next move in search space.

1. **Uses the**[**Greedy approach**](https://www.geeksforgeeks.org/greedy-algorithms-set-1-activity-selection-problem/)**:** At any point in state space, the search moves in that direction only which optimizes the cost of function with the hope of finding the optimal solution at the end.
2. **II) Missionaries and Cannibals Problem**

The problem is stated as follows:

“Three missionaries and three cannibals are present at one side of a river and need to cross the river. There is only one boat available. At any point of time, the number of cannibals should not outnumber the number of missionaries at that bank. It is also known that only two persons can occupy the boat available at a time.”

The objective of the solution is to find the sequence of their transfer from one bank of river to other using the boat sailing through the river satisfying these constraints.

We can form various production rules as presented in water-jug problem. Let Missionary is denoted by ‘M’ and Cannibal, by ‘C’. These rules are described below:

Rule 1 : (0, M) : One missionary sailing the boat from bank-1 to bank-2

Rule 2 : (M, 0) : One missionary sailing the boat from bank-2 to bank-1

Rule 3 : (M, M) : Two missionaries sailing the boat from bank-1 to bank-2

Rule 4 : (M, M) : Two missionaries sailing the boat from bank-2 to bank-1

Rule 5 : (M, C) : One missionary and one Cannibal sailing the boat from

bank-1 to bank-2

Rule 6 : (C, M) : One missionary and one Cannibal sailing the boat from

bank-2 to bank-1

Rule 7 : (C, C) : Two Cannibals sailing the boat from bank-1 to bank-2

Rule 8 : (C, C) : Two Cannibals sailing the boat from bank-2 to bank-1

Rule 9 : (0, C) : One Cannibal sailing the boat from bank-1 to bank-2

Rule 10 : (C, 0) : One Cannibal sailing the boat from bank-2 to bank-1

All or some of these production rules will have to be used in a particular sequence to find the solution of the problem. The rules applied and their sequence is presented in the following Table.

Table: Rules applied and their sequence in Missionaries and Cannibals problem

After application persons in the persons in the boat position

of rule river bank-1 river bank-2

Start state M, M, M, C, C, C 0 bank-1

5 M, M, C, C M, C bank-2

2 M, M, C, C, M C bank-1

7 M, M, M C, C, C bank-2

10 M, M, M, C C, C bank-1

3 M, C C, C, M, M bank-2

6 M, C, C, M C, M bank-1

3 C, C C, M, M, M bank-2

10 C, C, C M, M, M bank-1

7 C M, M, M, C, C bank-2

10 C, C M, M, M, C bank-1

7 0 M, M, M, C, C, C bank-2

**Part 4**

4. I) A production system consists of rules and factors. Knowledge is encoded in a declarative from which comprises of a set of rules of the form

Situation ------------ Action

SITUATION that implies ACTION.

Example:-

IF the initial state is a goal state THEN quit.

The major components of an AI production system are

i. A global database

ii. A set of production rules and

iii. A control system

The goal database is the central data structure used by an AI production system. The production system. The production rules operate on the global database. Each rule has a precondition that is either satisfied or not by the database. If the precondition is satisfied, the rule can be applied. Application of the rule changes the database. The control system chooses which applicable rule should be applied and ceases computation when a termination condition on the database is satisfied. If several rules are to fire at the same time, the control system resolves the conflicts.

Four classes of production systems:-

1. A monotonic production system

2. A non monotonic production system

3. A partially commutative production system

4. A commutative production system.

Advantages of production systems:-

1. Production systems provide an excellent tool for structuring AI programs.

2. Production Systems are highly modular because the individual rules can be added, removed or modified independently.

3. The production rules are expressed in a natural form, so the statements contained in the knowledge base should the a recording of an expert thinking out loud.

Disadvantages of Production Systems:-

One important disadvantage is the fact that it may be very difficult analyse the flow of control within a production system because the individual rules don’t call each other.

Production systems describe the operations that can be performed in a search for a solution to the problem. They can be classified as follows.

Monotonic production system :- A system in which the application of a rule never prevents the later application of another rule, that could have also been applied at the time the first rule was selected.

Partially commutative production system:-

A production system in which the application of a particular sequence of rules transforms state X into state Y, then any permutation of those rules that is allowable also transforms state x into state Y.

Theorem proving falls under monotonic partially communicative system. Blocks world and 8 puzzle problems like chemical analysis and synthesis come under monotonic, not partially commutative systems. Playing the game of bridge comes under non monotonic, not partially commutative system.

For any problem, several production systems exist. Some will be efficient than others. Though it may seem that there is no relationship between kinds of problems and kinds of production systems, in practice there is a definite relationship.

Partially commutative , monotonic production systems are useful for solving ignorable problems. These systems are important for man implementation standpoint because they can be implemented without the ability to backtrack to previous states, when it is discovered that an incorrect path was followed. Such systems increase the efficiency since it is not necessary to keep track of the changes made in the search process.

Monotonic partially commutative systems are useful for problems in which changes occur but can be reversed and in which the order of operation is not critical (ex: 8 puzzle problem).

Production systems that are not partially commutative are useful for many problems in which irreversible changes occur, such as chemical analysis. When dealing with such systems, the order in which operations are performed is very important and hence correct decisions have to be made at the first time itself.

1. II) **Branch and Bound**

Depth-first **branch-and-bound** search is a way to combine the space saving of depth-first search with heuristic information. It is particularly applicable when many paths to a goal exist and we want an optimal path. As in A\* search, we assume that h(n) is less than or equal to the cost of a lowest-cost path from n to a goal node.

The idea of a branch-and-bound search is to maintain the lowest-cost path to a goal found so far, and its cost. Suppose this cost is bound. If the search encounters a path p such that cost(p)+h(p) ≥ bound, path pcan be pruned. If a non-pruned path to a goal is found, it must be better than the previous best path. This new solution is remembered and bound is set to the cost of this new solution. It then keeps searching for a better solution.

Branch-and-bound search generates a sequence of ever-improving solutions. Once it has found a solution, it can keep improving it. Branch-and-bound search is typically used with depth-first search, where the space saving of the depth-first search can be achieved. It can be implemented similarly to depth-bounded search, but where the bound is in terms of path cost and reduces as shorter paths are found. The algorithm remembers the lowest-cost path found and returns this path when the search finishes.

1: **Procedure** DFBranchAndBound(G,s,goal,h,bound0)   
2:           **Inputs**  
3:                     G: graph with nodes N and arcs A   
4:                     s: start node   
5:                     goal: Boolean function on nodes   
6:                     h: heuristic function on nodes   
7:                     bound0: initial depth bound (can be ∞ if not specified)   
8:           **Output**  
9:                     a least-cost path from s to a goal node if there is a solution with cost less than bound0   
10:                     or ⊥ if there is no solution with cost less than bound0   
11:           **Local**  
12:                     best\_path: path or ⊥   
13:                     bound: non-negative real   
14:                     **Procedure** cbsearch(⟨n0,...,nk⟩)   
15:                               **if** (cost(⟨n0,...,nk⟩)+h(nk) < bound) **then**   
16:                               **if** (goal(nk)) **then**   
17:                                         best\_path ←⟨n0,...,nk⟩   
18:                                         bound ←cost(⟨n0,...,nk⟩)   
19:                               **else**  
20:                                         **for each** arc ⟨nk,n⟩∈A **do**   
21:                                                   cbsearch(⟨n0,...,nk,n⟩)   
22:           best\_path ←⊥   
23:           bound ←bound0   
24:           cbsearch(⟨s⟩)   
25:           **return** best\_path

Figure 3.11: Depth-first branch-and-bound search

The algorithm is shown in [Figure 3.11](http://artint.info/html/ArtInt_63.html#branch-and-bound-fig). The internal procedure cbsearch, for cost-bounded search, uses the global variables to provide information to the main procedure.

Initially, bound can be set to infinity, but it is often useful to set it to an overestimate, bound0, of the path cost of an optimal solution. This algorithm will return an optimal solution - a least-cost path from the start node to a goal node - if there is a solution with cost less than the initial bound bound0.

If the initial bound is slightly above the cost of a lowest-cost path, this algorithm can find an optimal path expanding no more arcs than A\* search. This happens when the initial bound is such that the algorithm prunes any path that has a higher cost than a lowest-cost path; once it has found a path to the goal, it only explores paths whose the f-value is lower than the path found. These are exactly the paths that A\* explores when it finds one solution.

If it returns ⊥ when bound0=∞, there are no solutions. If it returns ⊥ when bound0 is some finite value, it means no solution exists with cost less than bound0. This algorithm can be combined with iterative deepening to increase the bound until either a solution is found or it can be shown there is no solution.