USN										

INTERNAL ASSESSMENT TEST – II

Sub:	Intelligent Systems and Machine Learning Algorithms							Code:	BEC515A
Date:	14/ 12 / 2024	Duration:	90 mins	Max Marks:	50	Sem:	V	Branch:	ECE

Answer any 5 full questions

		Marks	CO	RBT
1	a) What are the key differences between informed search and uninformed search strategies?	[5] [5]	 CO3	 L2

Informed search in AI is a type of search algorithm that uses additional information to guide the search process, allowing for more efficient problem-solving compared to uninformed search algorithms. This information can be in the form of heuristics, estimates of cost, or other relevant data to prioritize which states to expand and explore. Examples of informed search algorithms include A* search, Best-First search, and Greedy search.

Uninformed search in AI refers to a type of search algorithm that does not use additional information to guide the search process. Instead, these algorithms explore the search space in a systematic, but blind, manner without considering the cost of reaching the goal or the likelihood of finding a solution. Examples of uninformed search algorithms include Breadth-First search (BFS), Depth-First search (DFS), and Depth-Limited search.

Parameters	**Informed Search**	**Uninformed Search**
Known as	It is also known as Heuristic Search.	It is also known as Blind Search.
Using Knowledge	It uses knowledge for the searching process.	It doesn't use knowledge for the searching process.
Performance	It finds a solution more quickly.	It finds solution slow as compared to an informed search.
Completion	It may or may not be complete.	It is always complete.
Cost Factor	Cost is low.	Cost is high.
Time	It consumes less time because of quick searching.	It consumes moderate time because of slow searching.

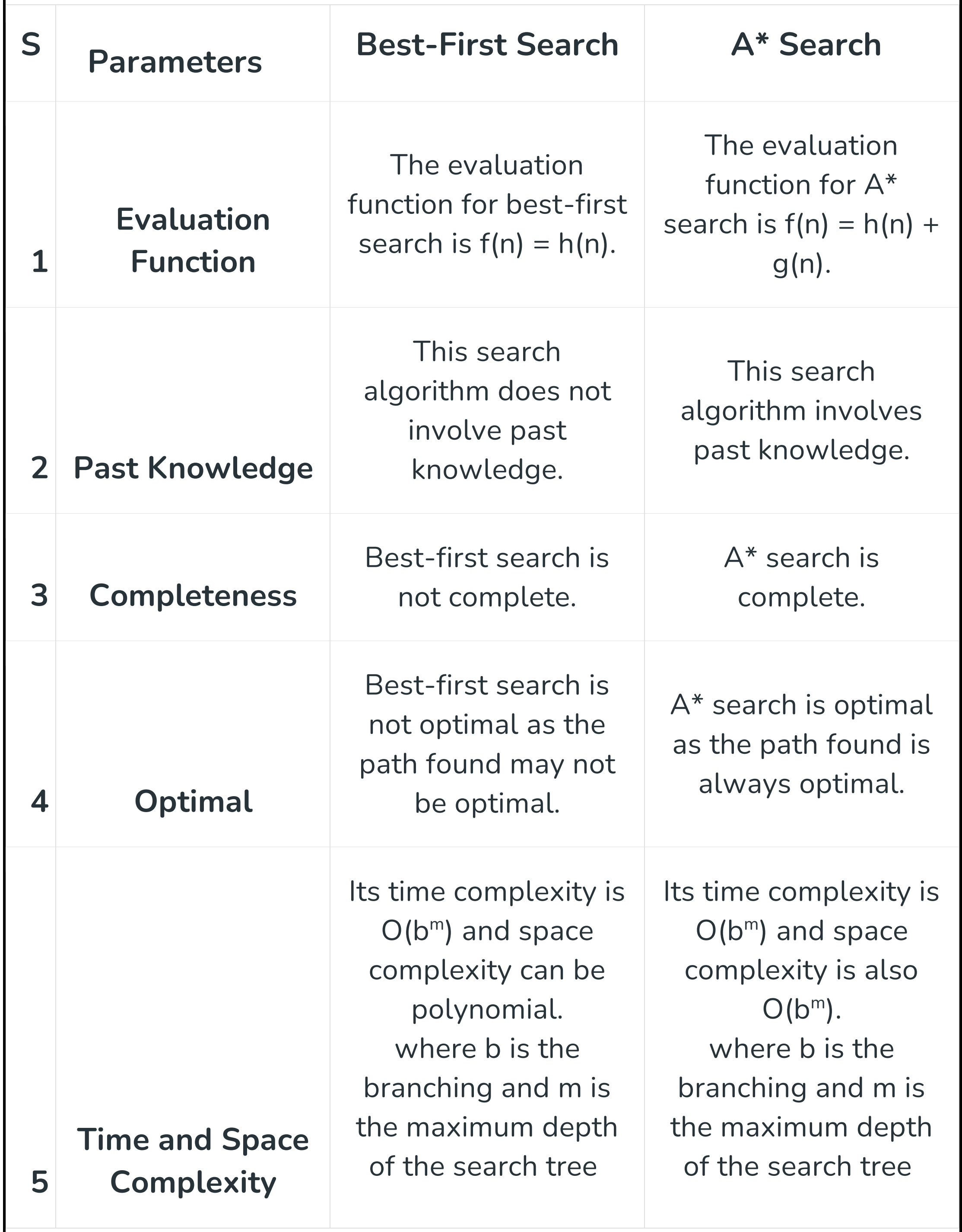
	takes into account cost and performance in the form of heuristic function.	efficient as entire graph/ tree is searched.
Computational requirements	Computational requirements are lessened.	Comparatively higher computational requirements.
Size of search problems	Have a wide scope in terms of handling large search problems.	Solving a massive search task is challenging.
Examples of Algorithms	• Greedy Search • A* Search • AO* Search • Hill Climbing Algorithm	• Depth First Search (DFS) • Breadth First Search (BFS)

b) In what scenarios might greedy best-first search fail to find an optimal solution?

Greedy search algorithms, while efficient in many scenarios, face significant limitations that can hinder their effectiveness in complex problem-solving environments.

1) Getting stuck at local minima
One of the primary challenges of greedy search is the tendency to get trapped in local optima. This occurs when the algorithm consistently selects the locally optimal choice at each step, which may not lead to the best overall solution. For instance, in scenarios where the goal is to minimize costs, a greedy approach might choose the cheapest option available at each decision point. However, this can result in a suboptimal global solution, as it fails to consider
Example: In a travel planning scenario, a greedy algorithm might select the least expensive flight without considering the overall itinerary, potentially leading to higher costs in other areas, such as accommodation or car rentals.
2) Inflexibility in Complex Situations: Greedy algorithms can also exhibit inflexibility when faced with complex problems that involve multiple variables and constraints. In situations where itineraries have numerous segments and interdependencies, relying solely on a greedy approach may not yield satisfactory results. For example, when planning a trip that includes flights, hotels, and car rentals, the initial greedy choice may not account for the cumulative impact of these decisions on the overall travel experience.
3) Risk of Premature Commitment:Another significant challenge is the

	risk of premature commitment to suboptimal solutions. Greedy algorithms often make decisions based on immediate rewards, which can lead to a plateau in performance if the chosen path does not align with long-term goals. For instance, if an algorithm selects a reward function that appears promising initially but fails to deliver sustained benefits, it may lock onto a suboptimal strategy. OR Limitations of greedy best-first search 1. **Incomplete:** GBFS may fail to find a solution if it becomes trapped in a local optimum or dead-end, as it does not consider backtracking. 2. **Non-Optimal:** The algorithm is not guaranteed to find the shortest or best path. It only focuses on immediate gains (greedily choosing the nearest node), which may lead to suboptimal solutions. 3. **Heuristic Dependency:** The efficiency and success of GBFS heavily rely on the quality of the heuristic. A poor heuristic can lead to inefficient searches.			
2	a) How does the f(n)=g(n)+h(n) formula in A* search combine path cost and heuristic value? The core of the A* algorithm is based on cost functions and heuristics. It uses two main parameters: **g(n):** The actual cost from the starting node to any node n. **h(n):** The heuristic estimated cost from node n to the goal. This is where A* integrates knowledge beyond the graph to guide the search. The sum, f(n)=g(n)+h(n) $f(n)=g(n)+h(n)$, represents the total estimated cost of the cheapest solution. The A* algorithm functions by maintaining a priority queue (or open set) of all possible paths along the graph, prioritizing them based on their f(n) values. The steps of the algorithm are as follows: 1. **Initialization:** Start by adding the initial node to the open set with its f(n). 2. **Loop:** While the open set is not empty, the node with the lowest f(n) value is removed from the queue. 3. **Goal Check:** If this node is the goal, the algorithm terminates and returns the discovered path. 4. **Node Expansion:** Otherwise, expand the node (find all its neighbors), calculating g, h, and f values for each neighbor. Add each neighbor to the open set if it's not already present, or if a better path to this neighbor is found. 5. **Repeat:** The loop repeats until the goal is reached or if there are no more nodes in the open set, indicating no available path. b) What is the main distinction between A* search and greedy best-first search? Best-First Search: Best-First search is a searching algorithm used to find the shortest path which uses distance as a heuristic. The distance between the starting node and the goal node is taken as heuristics. It defines the evaluation function for each node n in the graph as f(n) = h(n) where h(n) is heuristics function. A*Search: A*search is a searching algorithm used to find the shortest path which calculates the cost of all its neighboring nodes and selects the minimum cost node. It defines the evaluation function f(n) = g(n) + h(n) where, h(n) is heuristics function and g(n) is the past knowledge acquired while searching.	[5] [5]	CO3	L3

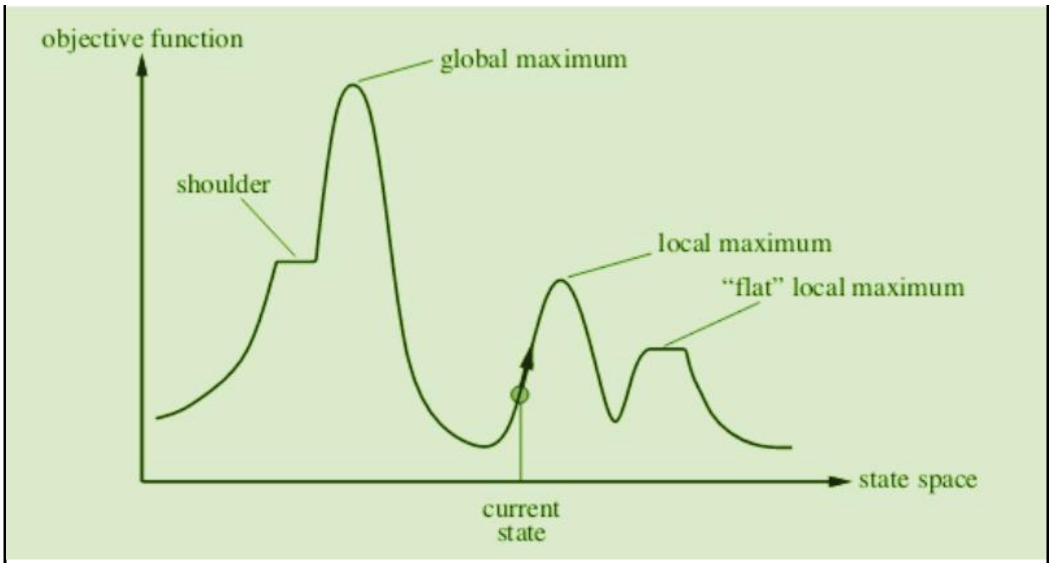
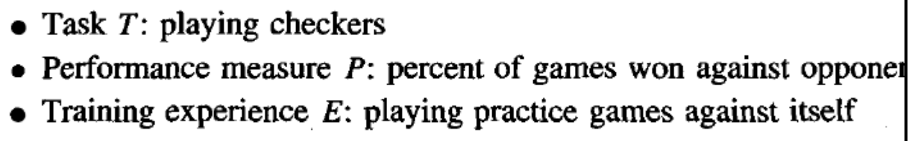


S	Parameters	Best-First Search	A* Search
1	Evaluation Function	The evaluation function for best-first search is f(n) = h(n).	The evaluation function for A* search is f(n) = h(n) + g(n).
2	Past Knowledge	This search algorithm does not involve past knowledge.	This search algorithm involves past knowledge.
3	Completeness	Best-first search is not complete.	A* search is complete.
4	Optimal	Best-first search is not optimal as the path found may not be optimal.	A* search is optimal as the path found is always optimal.
5	Time and Space Complexity	Its time complexity is $O(b^m)$ and space complexity can be polynomial. where b is the branching and m is the maximum depth of the search tree	Its time complexity is $O(b^m)$ and space complexity is also $O(b^m)$. where b is the branching and m is the maximum depth of the search tree

3	a) Explain the common techniques for generating heuristics? i) Generating admissible heuristics from relaxed problems ii) Generating admissible heuristics from subproblems: Pattern databases b) How do local search algorithms differ from global search methods in their approach to finding solutions in state spaces? Local search in AI refers to a family of optimization algorithms that are used to find the best possible solution within a given search space. Unlike global search methods that explore the entire solution space, local search algorithms	[5] [5]	CO3	L2

		focus on making incremental changes to improve a current solution until they reach a locally optimal or satisfactory solution. This approach is useful in situations where the solution space is vast, making an exhaustive search impractical.			
			[5]		

4	a) Explain hill climbing algorithm in AI? **Hill climbing** is a widely used **optimization algorithm** in **Artificial Intelligence (AI)** that helps find the best possible solution to a given problem. As part of the **local search algorithms** family, it is often applied to **optimization problems** where the goal is to identify the optimal solution from a set of potential candidates. It is a form of local search, which means it focuses on finding the optimal solution by making incremental changes to an existing solution and then evaluating whether the new solution is better than the current one. The process is analogous to climbing a hill where you continually seek to improve your position until you reach the top, or local maximum, from where no further improvement can be made. Hill climbing is a fundamental concept in AI because of its simplicity, efficiency, and effectiveness in certain scenarios, especially when dealing with optimization problems or finding solutions in large search spaces. n the **Hill Climbing algorithm**, the process begins with an initial solution, which is then iteratively improved by making small, incremental changes. These changes are evaluated by a **heuristic function** to determine the quality of the solution. The algorithm continues to make these adjustments until it reaches a **local maximum**—a point where no further improvement can be made with the current set of moves. Algorithm: 1. **Initial State**: Start with an arbitrary or random solution (initial state). 2. **Neighboring States**: Identify neighboring states of the current solution by making small adjustments (mutations or tweaks). 3. **Move to Neighbor**: If one of the neighboring states offers a better solution (according to some evaluation function), move to this new state. 4. **Termination**: Repeat this process until no neighboring state is better than the current one. At this point, you've reached a local maximum or minimum (depending on whether you're maximizing or minimizing). b) With the help of state space diagram, explain the different regions in state space. The optimal solution in the state-space diagram is represented by the state where the **objective function** reaches its maximum value, also known as the **global maximum**.	[5] [5]	CO3	L3

	 objective function global maximum shoulder local maximum "flat" local maximum state space current state **Key Regions in the State-Space Diagram** 1. **Local Maximum**: A local maximum is a state better than its neighbors but not the best overall. While its objective function value is higher than nearby states, a global maximum may still exist. 2. **Global Maximum**: The global maximum is the best state in the state-space diagram, where the objective function achieves its highest value. This is the optimal solution the algorithm seeks. 3. **Plateau/Flat Local Maximum**: A plateau is a flat region where neighboring states have the same objective function value, making it difficult for the algorithm to decide on the best direction to move. 4. **Ridge**: A ridge is a higher region with a slope, which can look like a peak. This may cause the algorithm to stop prematurely, missing better solutions nearby. 5. **Current State**: The current state refers to the algorithm's position in the state-space diagram during its search for the optimal solution. 6. **Shoulder**: A shoulder is a plateau with an uphill edge, allowing the algorithm to move toward better solutions if it continues searching beyond the plateau.			
5	a) Define machine learning with respect to experience E, task T and performance measure P. A computer program is said to learn from experience E with respect to some class of tasks ***T*** and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. For example, a computer program that learns to play checkers might improve its performance as ***measured by its abiliry to win*** at the class of tasks involving ***playing checkers games,*** through experience ***obtained*** by ***playing games against itself.*** In general, to have a well-defined learning problem, we must identity these three features: the class of tasks, the measure of performance to be improved, and the source of experience. b) identify the three features E, T and P for the given examples: i) Checkers learning problem  • Task *T*: playing checkers • Performance measure *P*: percent of games won against oppone • Training experience *E*: playing practice games against itself	[3] [7]	CO4	L2

ii) Handwriting recognition learning problem

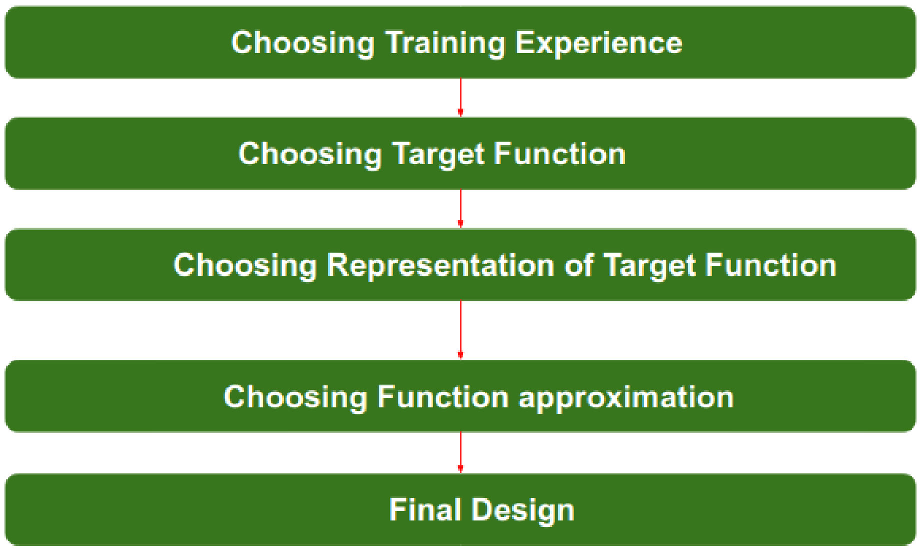
Task T: recognizing and classifying handwritten words within images
Performance measure ***P:*** percent of words correctly classified
Training experience E: a database of handwritten words with given classifications

iii) Robot driving learning problem

- Task T: driving on public four-lane highways using vision sensors
- Performance measure P: average distance traveled before an error (
by human overseer)
- Training experience E: a sequence of images and steering command
ed while observing a human driver

6 Explain the steps involved in designing the learning system in detail [10] CO4 L2

Steps for Designing Learning System are:


Choosing Training Experience
Choosing Target Function
Choosing Representation of Target Function
Choosing Function approximation
Final Design

Step 1) Choosing the Training Experience: The very important and first task is to choose the training data or training experience which will be fed to the Machine Learning Algorithm. It is important to note that the data or experience that we fed to the algorithm must have a significant impact on the Success or Failure of the Model. So Training data or experience should be chosen wisely.

Below are the attributes which will impact on Success and Failure of Data:

- The training experience will be able to provide direct or indirect feedback regarding choices. For example: While Playing chess the training data will provide feedback to itself like instead of this move if this is chosen the chances of success increases.
- Second important attribute is the degree to which the learner will control the sequences of training examples. For example: when training data is fed to the machine then at that time accuracy is very less but when it gains experience while playing again and again with itself or opponent the machine algorithm will get feedback and control the chess game accordingly.
- Third important attribute is how it will represent the distribution of examples over which performance will be measured. For example, a Machine learning algorithm will get experience while going through a number of different cases and different examples. Thus, Machine Learning Algorithm will get more and more experience by passing through more and more examples and hence its performance will increase.

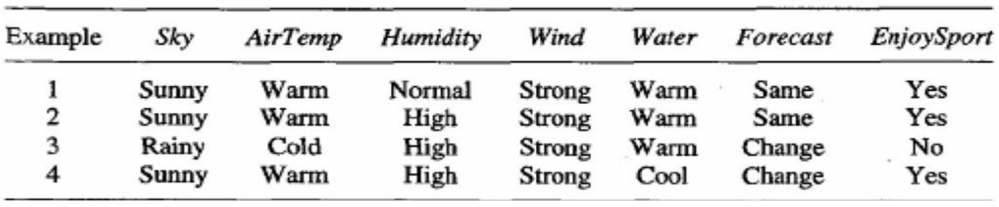
Step 2- Choosing target function: The next important step is choosing the target function. It means according to the knowledge fed to the algorithm the machine learning will choose NextMove function which will describe what type of legal moves should be taken. For example : While playing chess with the opponent, when opponent will play then the machine learning algorithm will decide what be the number of possible legal moves taken in order to get success.

Step 3- Choosing Representation for Target function: When the machine algorithm will know all the possible legal moves the next step is to choose the optimized move using any representation i.e. using linear Equations, Hierarchical Graph Representation, Tabular form etc. The NextMove function will move the Target move like out of these move which will provide more success rate. For Example : while playing chess machine have 4 possible moves, so the machine will choose that optimized move which will provide success to it.

Step 4- Choosing Function Approximation Algorithm: An optimized move cannot be chosen just with the training data. The training data had to go through with set of example and through these examples the training data will approximates which steps are chosen and after that machine will provide feedback on it. For Example : When a training data of Playing chess is fed to algorithm so at that time it is not machine algorithm will fail or get success and again from that failure or success it will measure while next move what step should be chosen and what is its success rate.

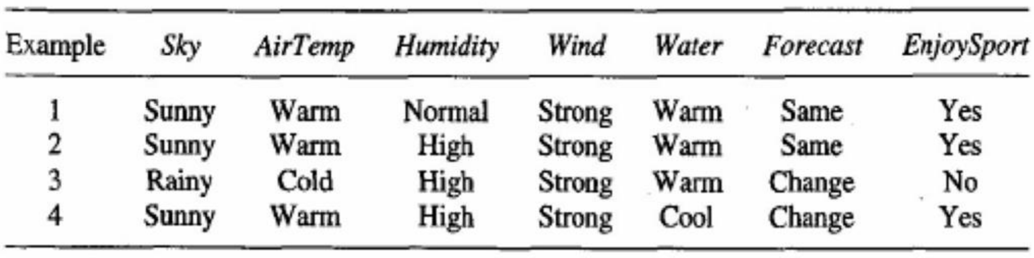
Step 5- Final Design: The final design is created at last when system goes from number of examples , failures and success , correct and incorrect decision and what will be the next step etc. Example: DeepBlue is an intelligent computer which is ML-based won chess game against the chess expert Garry Kasparov, and it became the first computer which had beaten a human chess expert.

7 Find maximally specific hypothesis using Find-S algorithm. How many concepts are possible for this instance space. How many syntactically and semantically distinct hypotheses can be expressed for this example if the attribute values of sky=3, Airtemp=2, Humidity=2, wind=2, water=2 and Forecast=2. [10] CO4 L3

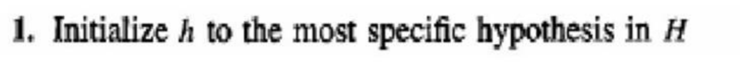


Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

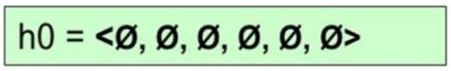
Step – 1 of Find-S Algorithm



Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes



1. Initialize *h* to the most specific hypothesis in *H*



h0 = **<Ø, Ø, Ø, Ø, Ø, Ø>**

Step 2 of Find-S Algorithm First iteration
h0 = (ø, ø, ø, ø, ø, ø, ø)
X1 = <Sunny, Warm, Normal, Strong, Warm, Same>
h1 = <Sunny, Warm, Normal, Strong, Warm, Same>
Step 2 of Find-S Algorithm Second iteration

	h1 = <Sunny, Warm, Normal, Strong, Warm, Same>
	X2 = <Sunny, Warm, High, Strong, Warm, Same>
	h2 = <Sunny, Warm, ?, Strong, Warm, Same>
	Step 2 of Find-S Algorithm Third iteration
	h2 = <Sunny, Warm, ?, Strong, Warm, Same>
	X3 = <Rainy, Cold, High, Strong, Warm, Change> – No
	X3 is Negative example Hence ignored
	h3 = <Sunny, Warm, ?, Strong, Warm, Same>
	Step 2 of Find-S Algorithm Fourth iteration
	h3 = <Sunny, Warm, ?, Strong, Warm, Same>
	X4 = <Sunny, Warm, High, Strong, Cool, Change>
	h4 = <Sunny, Warm, ?, Strong, ?, ?>
	Step 3
	The final maximally specific hypothesis is **<Sunny, Warm, ?, Strong, ?, ?>**			

CCI HOD