ARTIFICIAL INTELLIGENCE

Unit II — Problem-Solving & Search Strategies

Topics Covered in this Unit

Problem-Solving Agents: Goal Formulation, Problem Formulation & Search Architecture

Classic Example Problems: Toy Problems (8-Puzzle, 8-Queens, Vacuum World, Missionaries & Cannibals, Water Jug) vs. Real-World Problems (TSP, Routing)

General Search Algorithms: Search Tree vs. Graph Search, Frontier, Explored Set & Redundant Paths

Uninformed (Blind) Search Strategies: BFS, DFS, Uniform-Cost Search (UCS / Dijkstra), Depth-Limited Search (DLS) & Iterative Deepening Search (IDS)

Comprehensive 4-Metric Evaluation: Completeness, Time Complexity, Space Complexity & Optimality

Informed (Heuristic) Search Strategies: Greedy Best-First Search & A* Search Algorithm (f(n) = g(n) + h(n))

Theory of Heuristics: Admissibility (h(n) <= h*(n)), Consistency / Monotonicity & Dominance of Heuristics

Local Search & Optimization Problems: Hill-Climbing Search (Local Maxima, Ridges, Plateaux) & Random-Restart Hill Climbing

Simulated Annealing (Metropolis Criterion & Cooling Schedule) & Local Beam Search

Genetic Algorithms (GA): Population Chromosomes, Fitness Evaluation, Selection, Crossover & Mutation

High-Yield University Exam Question Bank with Detailed Model Answers & Unit Summary

Compiled in a structured, easy-to-understand, textbook style format
Specially curated for B.Tech / B.E. / BCA / MCA / B.Sc Computer Science, AI & Data Science Students

© Copyright — abhyasmitra.in — All Rights Reserved

COURSE SYLLABUS & MAPPING

[UNIT II SYLLABUS — PRESCRIBED UNIVERSITY CURRICULUM]

This section contains the official syllabus for Unit II. Verify with your university curriculum mapping.

Unit II Syllabus Breakdown:

Unit II Problem-solving: Solving Problems by Searching, Problem-Solving Agents, Example Problems, Search Algorithms, Uninformed Search Strategies, Informed (Heuristic) Search Strategies, Heuristic Functions, Search in Complex Environments, Local Search and Optimization Problems

• Course Code: _____ • Semester / Year: _____

TABLE OF CONTENTS

2.1 Problem-Solving Agents & Problem Formulation

Definition: Problem-Solving Agent

A Problem-Solving Agent is a goal-driven intelligent agent that decides what actions to take by finding action sequences that lead to desirable goal states. The process consists of 4 atomic steps:

1. Goal Formulation: Defining the target objectives based on current percepts.

2. Problem Formulation: Formally deciding what actions and states to consider.

3. Search: Simulating action sequences in the state space to find a path to the goal.

4. Execution: Executing the discovered sequence of actions in the physical environment.

Toy Problems vs. Real-World Problems

Problem Type	Specific Problem	State Space & Action Description	Goal Test Formulation
Toy Problem (Benchmarking)	8-Puzzle	State: 3×3 board with 8 numbered tiles and 1 blank space. Actions: Move blank Left, Right, Up, Down. Total states = 9!/2 = 181,440.	Tiles match target numerical matrix: [[1,2,3],[8,blank,4],[7,6,5]].
Toy Problem	8-Queens Problem	State: Any arrangement of 0 to 8 queens on an 8×8 chessboard. Actions: Place a queen in an empty square.	Exactly 8 queens placed such that no two queens attack each other (same row, col, diag).
Toy Problem	Missionaries & Cannibals	State: Number of missionaries, cannibals, and boat on left/right river banks. Actions: Ferry 1 or 2 people across river.	All 3 missionaries and 3 cannibals on the other bank without cannibals outnumbering missionaries.
Real-World Problem	Traveling Salesperson Problem (TSP)	State: Current city visited and set of unvisited cities. Actions: Travel to an unvisited city.	All cities visited exactly once and returned to starting city with minimum total distance.
Real-World Problem	Route-Finding & Navigation	State: Current GPS location/junction. Actions: Transition along road network segments.	Reaching destination junction with minimized travel time/distance (Google Maps).

2.2 Uninformed (Blind) Search Strategies

Definition: Uninformed Search

Uninformed Search (or Blind Search) algorithms have no domain-specific knowledge about how close a state is to the goal. They can only distinguish a goal state from a non-goal state and systematically generate successors according to fixed structural traversal rules.

Detailed Uninformed Search Algorithms

- **1. Breadth-First Search (BFS):** Expands the shallowest unexpanded node first using a First-In-First-Out (FIFO) queue for the frontier. Complete if branching factor b is finite. Optimal if all step costs are equal.

Time Complexity: O(b^d), Space Complexity: O(b^d) (where b is branching factor and d is goal depth)

- **2. Depth-First Search (DFS):** Expands the deepest unexpanded node first using a Last-In-First-Out (LIFO) stack for the frontier. Incomplete in infinite-depth spaces or spaces with cycles (unless graph search with explored set is used). Not optimal.

Time Complexity: O(b^m), Space Complexity: O(b × m) (where m is maximum tree depth; massive space savings)

- **3. Uniform-Cost Search (UCS / Dijkstra's Algorithm):** Expands the node n with the lowest path cost g(n) first using a Priority Queue ordered by g(n). Complete if step costs are strictly positive (ε > 0). Optimal for arbitrary step costs.

Time & Space Complexity: O(b^(1 + ⌊C / ε⌋)), where C* is optimal cost and ε is minimum step cost*

- **4. Depth-Limited Search (DLS):** DFS with a fixed predetermined depth limit l. Avoids infinite paths, but incomplete if goal depth d > l, and non-optimal.
- **5. Iterative Deepening Search (IDS / IDDFS):** Repeatedly executes Depth-Limited Search, gradually increasing the depth limit l from 0, 1, 2, ... up to d. Combines the completeness and optimality of BFS with the modest O(b × d) linear memory footprint of DFS.

Time Complexity: O(b^d) (redundant top-level expansions represent only a negligible fraction of bottom level: b/(b-1)), Space Complexity: O(b × d)

Comprehensive Evaluation Matrix of Uninformed Search

Criterion	Breadth-First (BFS)	Depth-First (DFS)	Uniform-Cost (UCS)	Depth-Limited (DLS)	Iterative Deepening (IDS)
Completeness	YES (if b is finite)	NO (fails in infinite depth)	YES (if step cost ε > 0)	NO (if d > l)	YES (if b is finite)

Criterion	Breadth-First (BFS)	Depth-First (DFS)	Uniform-Cost (UCS)	Depth-Limited (DLS)	Iterative Deepening (IDS)
Time Complexity	O(b^d)	O(b^m)	O(b^(1 + ⌊C*/ε⌋))	O(b^l)	O(b^d)
Space Complexity	O(b^d) [Exponential]	O(b × m) [Linear]	O(b^(1 + ⌊C*/ε⌋))	O(b × l) [Linear]	O(b × d) [Linear]
Optimality	YES (if cost = 1)	NO	YES (for general costs)	NO	YES (if cost = 1)

2.3 Informed (Heuristic) Search Strategies: Greedy & A* Search

Definition: Heuristic Function h(n)

A Heuristic Function h(n) estimates the cheapest cost from node n to the nearest goal state. By definition, if n is a goal state, h(n) = 0. Heuristics inject domain-specific knowledge to guide search efficiently.

1. Greedy Best-First Search

- • **Evaluation Function:** f(n) = h(n) (expands the node estimated to be closest to the goal).
- • **Properties:** Not complete (can get stuck in loops in graph search without visited tracking). Not optimal. Highly susceptible to false leads.

2. A* Search Algorithm

Mathematical Formulation: A* Search Algorithm

A evaluates nodes by combining the exact historical path cost g(n) reached so far with the estimated heuristic cost h(n) to reach the goal:*

$f(n) = g(n) + h(n)$

where g(n) is the cost from start node to n, h(n) is estimated cost from n to goal, and f(n) is total estimated path cost through n.

Optimality Conditions of A* Search

- **1. Admissibility (Required for Tree Search Optimality):** A heuristic h(n) is Admissible if it NEVER overestimates the true cost h*(n) to reach the goal (i.e., h(n) is optimistic):

0 ≤ h(n) ≤ h(n) for all nodes n*

- **2. Consistency / Monotonicity (Required for Graph Search Optimality):** A heuristic h(n) is Consistent if, for every node n and every successor n' generated by action a with step cost c(n, a, n'), the estimated cost satisfies the triangle inequality:

h(n) ≤ c(n, a, n') + h(n')

Theorem: If h(n) is consistent, then f(n) along any path is non-decreasing, and the first time A* expands a state, the path to that state is guaranteed to be optimal.

Heuristic Dominance

If h_2(n) ≥ h_1(n) for all nodes n, and both are admissible, then h_2 Dominates h_1. A* using h_2 will never expand more nodes than A* using h_1 (h_2 is strictly more efficient).

- • **Example in 8-Puzzle:** Manhattan Distance Heuristic (h_2 = sum of horizontal and vertical distances of tiles from goal positions) strictly dominates Misplaced Tiles Heuristic (h_1 = count of tiles not in goal position).

2.4 Local Search Algorithms & Optimization Problems

Concept of Local Search

In many optimization problems (e.g., 8-Queens, VLSI routing, Factory Scheduling), the path to the goal is irrelevant; only the final Goal State configuration matters. Local search algorithms operate using a single current state (rather than maintaining paths), moving only to neighboring states in the state space landscape.

1. Hill-Climbing Search (Greedy Local Search)

- • **Mechanism:** Continuously moves in the direction of increasing elevation (steepest uphill neighbor). Terminates when no neighbor has a higher value.
- • **Failure Pitfalls:**
  - **1. Local Maxima:** A peak that is higher than all its neighbors but lower than the global maximum.
  - **2. Ridges:** A sequence of local maxima where the slope is steep along sides but gradual along the ridge, causing the algorithm to oscillate.

  - **3. Plateaux (Shoulders/Flat Maxima):** A flat area where all neighboring states have identical evaluation, causing random wandering or premature stagnation.
- • **Solution:** Random-Restart Hill Climbing (conducts independent searches starting from randomly generated initial states; complete with probability 1).

2. Simulated Annealing

Definition: Simulated Annealing (Kirkpatrick et al., 1983)
Inspired by the metallurgical annealing process of heating metal and cooling it slowly to form crystalline structures without defects. Simulated Annealing escapes local maxima by allowing occasional downhill moves based on a probability governed by the Metropolis Criterion.

- • **Metropolis Acceptance Probability:** If a neighboring move yields improvement ($\Delta E > 0$), it is always accepted. If the move is worse ($\Delta E \leq 0$), it is accepted with probability:

$$P(Accept) = exp(\Delta E / T)$$

where ΔE is the energy/score difference and T is the current Temperature governed by a cooling schedule. At high T (early stages), the agent explores wildly; as $T \to 0$ (freezing), it behaves like pure Hill-Climbing.

3. Genetic Algorithms (GA)

- • **Population:** Maintains a set of k candidate solution states encoded as discrete strings (chromosomes) over an alphabet (e.g., binary bitstrings).
- • **4-Step Evolutionary Cycle:**
  - **1. Fitness Function:** Evaluates the fitness score of each individual.
  - **2. Selection:** Selects parent individuals with probability proportional to their fitness (Roulette-wheel selection).
  - **3. Crossover (Recombination):** Splits pairs of parents at a random crossover point and splices halves to create offspring.
  - **4. Mutation:** Randomly flips bits with small probability (e.g., 0.001) to introduce novel genetic diversity.

2.5 High-Yield University Exam Question Bank & Model Answers

Part A: Short Answer Questions (2–3 Marks)

- **Q1. What is an Admissible Heuristic? State the condition for A* tree search optimality.**

Answer: A heuristic h(n) is admissible if it never overestimates the true cost to reach the goal from node n (i.e., h(n) ≤ h*(n) where h*(n) is true optimal cost). A* tree search is guaranteed to be optimal if h(n) is admissible.

- **Q2. Why is Iterative Deepening Search (IDS) preferred over BFS and DFS?**

Answer: IDS combines the advantages of both: it achieves the completeness and optimality of BFS while maintaining the modest linear space complexity O(b × d) of DFS, avoiding BFS's catastrophic exponential memory exhaustion.

- **Q3. State the formula for A* Search and define each term.**

Answer: f(n) = g(n) + h(n), where g(n) is the exact cost incurred from start node to node n, h(n) is the heuristic estimated cost from node n to the goal, and f(n) is the total estimated cost of the path passing through n.

- **Q4. Describe the failure pitfalls of Hill-Climbing Search.**

Answer: Hill-climbing fails due to: (1) Local Maxima (sub-optimal peaks with lower scores than global maximum), (2) Ridges (steep side slopes causing zigzag oscillation), and (3) Plateaux (flat regions with zero gradient causing random walk).

Part B: Long Answer Questions (8–10 Marks)

- **Q5. Compare BFS, DFS, UCS, DLS, and IDS across Completeness, Time Complexity, Space Complexity, and Optimality. Solve an 8-Puzzle search problem using A* with Manhattan Distance heuristic.**

Model Answer Outline:

- 1. Comprehensive comparative table evaluating all 5 uninformed algorithms across the 4 formal metrics.
- 2. Mathematical derivation of IDS time complexity proving that repeated top-level expansions add negligible cost: ∑(d-i+1)b^i = O(b^d).
- 3. A* Search on 8-Puzzle: State definition, Manhattan distance calculation h = ∑ (|x_curr - x_goal| + |y_curr - y_goal|).
- 4. Step-by-step search tree trace showing node expansion, f(n) = g(n) + h(n) values, and frontier priority queue states leading to goal state.

- **Q6. Explain Local Search Algorithms in detail: Hill-Climbing, Simulated Annealing, and Genetic Algorithms.**

Model Answer Outline:

- 1. Concept of local search in state space landscapes without path tracking.

- 2. Hill-Climbing: Algorithm, local maxima, ridges, plateaux, and Random-Restart mitigation.
- 3. Simulated Annealing: Metallurgical analogy, temperature cooling schedule T(t), Metropolis acceptance probability P = exp(ΔE / T).
- 4. Genetic Algorithms: Chromosome encoding, Fitness function, Selection (Roulette wheel), Crossover operator, Mutation operator.
- 5. Complete architectural flowchart illustrating genetic algorithm reproduction generation cycles.

2.6 Unit II Summary & Quick Revision Sheet

Search Strategy	Evaluation Function & Key Properties	Complexity & Optimality Summary
Breadth-First (BFS)	Expands shallowest node (FIFO).	Time: O(b^d), Space: O(b^d) [Exponential]. Complete & Optimal for unit costs.
Depth-First (DFS)	Expands deepest node (LIFO stack).	Time: O(b^m), Space: O(b × m) [Linear]. Incomplete & Non-optimal.
Iterative Deepening (IDS)	Iterative DLS with depth limits 0, 1, 2...d.	Time: O(b^d), Space: O(b × d) [Linear]. Complete & Optimal. Best uninformed.
A* Search	f(n) = g(n) + h(n).	Optimal if h(n) is Admissible (Tree search) and Consistent (Graph search). Dominance.
Local Search	Hill-Climbing (local max trap), Simulated Annealing (P=e^(ΔE/T)), Genetic Algorithms (Crossover/Mutation).	