**Informed Search Algorithms – Research-Based Tasks**

**🔰 Level 1: Understanding Core Concepts**

**Task 1: What is a Heuristic?**

**In the context of Artificial Intelligence (AI) and search algorithms, a heuristic is a technique or function that helps guide a search algorithm toward the most promising solutions, often using domain-specific knowledge. It is not guaranteed to be perfect or optimal, but it can significantly reduce computation time and effort by prioritizing more likely paths to the goal.**

**✅ What does a heuristic function do?**

**A heuristic function, commonly denoted as h(n), estimates the cost or distance from a given node n to the goal node. It helps the search algorithm make decisions about which node to explore next by ranking nodes based on how "close" they are to the goal—based on the estimate.**

**For example, in a navigation system, the heuristic function might use the straight-line distance between the current location and the destination as an estimate, even though roads may wind or be blocked.**

**How does it differ from path cost?**

* **Path cost (g(n)) refers to the actual cost incurred to reach a node n from the start. This could include the number of steps, time, or distance traveled so far.**
* **Heuristic (h(n)) is an estimated cost from node n to the goal, not necessarily reflecting reality.**

**In algorithms like A\*, both are combined:**

**f(n) = g(n) + h(n)  
Where f(n) is the estimated total cost of the path through n to the goal.**

**Why is it used in informed search?**

**Heuristics are used in informed search algorithms (like A\* or Greedy Best-First Search) because they incorporate knowledge about the problem to make smarter choices during search. Unlike uninformed search (like BFS or DFS), which blindly explores, informed search uses heuristics to prioritize promising paths, speeding up the process and often reducing resource usage.**

**Real-World Examples:**

1. **GPS Navigation**
   * ***Heuristic:* Straight-line distance to destination**
   * ***Path cost:* Actual driving distance so far**
   * ***Use:* Helps find the shortest or fastest route efficiently.**
2. **Chess AI**
   * ***Heuristic:* Evaluates board positions based on material and position**
   * ***Path cost:* Number of moves or sacrifices made so far**
   * ***Use:* Guides the AI toward moves that likely lead to victory.**
3. **Robot Path Planning**
   * ***Heuristic:* Estimated distance to the goal avoiding obstacles**
   * ***Path cost:* Actual distance or time moved so far**
   * ***Use:* Finds efficient routes while avoiding danger.**

**Task 2: Admissible vs Inadmissible Heuristics**

**What is an Admissible Heuristic?**

**An admissible heuristic is a heuristic function h(n) that never overestimates the true cost to reach the goal from node n.**

**🔹 Key rule: h(n) ≤ h\*(n) for all nodes  
Where h\*(n) is the actual minimum cost to reach the goal from node n.**

**Because it never overestimates, an admissible heuristic is always optimistic—it assumes the goal is closer than it might actually be. This is crucial in A\* search, where admissibility guarantees that the algorithm finds the optimal (shortest/cheapest) path.**

**What is an Inadmissible Heuristic?**

**An inadmissible heuristic is one that may overestimate the true cost to the goal.  
That is, it sometimes gives:**

**h(n) > h\*(n)**

**This can lead to faster but non-optimal solutions, because the algorithm might choose a seemingly better path that is actually more expensive.**

**Comparison Table**

| **Feature** | **Admissible Heuristic** | **Inadmissible Heuristic** |
| --- | --- | --- |
| **Overestimates goal?** | **No** | **Yes (sometimes)** |
| **Finds optimal path?** | **Yes (with A\*)** | **Not guaranteed** |
| **Safe to use in A\*?** | **Yes** | **Risky (faster but may be wrong)** |
| **Example** | **Straight-line distance in maps** | **Inflated distance to goal (e.g., adding penalty)** |

**Real-World Examples**

1. **Admissible:**
   * **In a city map, using straight-line distance between cities as a heuristic is admissible because it’s the shortest possible path (roads may be longer).**
2. **Inadmissible:**
   * **A GPS system that estimates time to destination by assuming high traffic everywhere might overestimate the time—this would be inadmissible.**

**Why Does This Matter?**

* **Use admissible heuristics when you want accuracy and guaranteed optimal solutions.**
* **Use inadmissible heuristics when you are okay with possibly faster but non-optimal solutions—common in real-time systems where speed matters more than perfection.**

**Task 3: Consistent (Monotonic) Heuristic**

* **Definition: What is a Consistent Heuristic?**

**A consistent heuristic (also called a monotonic heuristic) is one where the estimated cost from a node n to the goal is always less than or equal to the cost of reaching a neighbor n′ plus the estimated cost from n′ to the goal.**

**This means the heuristic follows a rule similar to the triangle inequality from geometry.**

**Mathematical Condition for Consistency:**

**For a heuristic h(n) to be consistent, the following must hold true for every node n and its successor n′:**

**🔹 h(n) ≤ cost(n, n′) + h(n′)**

**Where:**

* **h(n) = heuristic estimate from node n to the goal**
* **cost(n, n′) = actual cost to go from node n to its neighbor n′**
* **h(n′) = heuristic estimate from neighbor n′ to the goal**

**Also, the goal node must satisfy:**

**h(goal) = 0**

**Admissible but Not Consistent: Example Scenario**

**A heuristic can be admissible but not consistent, meaning it never overestimates the true cost (still optimistic), but it might "jump around" and violate the triangle inequality.**

**Example:**

**Let’s define a graph:**

* **Start (S) → A → Goal (G)**
* **Edge costs:**
  + **S → A = 2**
  + **A → G = 2**
  + **So total path cost from S to G = 4**

**Define a heuristic function:**

* **h(G) = 0**
* **h(A) = 1**
* **h(S) = 3**

**Check admissibility:**

* **True cost from S to G = 4, and h(S) = 3 → Admissible**
* **True cost from A to G = 2, and h(A) = 1 → Admissible**

**Now test consistency:**

* **From S to A:**
  + **h(S) = 3**
  + **cost(S, A) = 2**
  + **h(A) = 1**
  + **Check: 3 ≤ 2 + 1 → OK**
* **From A to G:**
  + **h(A) = 1**
  + **cost(A, G) = 2**
  + **h(G) = 0**
  + **Check: 1 ≤ 2 + 0 → OK**

**But let’s modify the heuristic:**

* **h(A) = 3**
* **h(S) = 3**

**Still admissible because:**

* **h(S) = 3 ≤ true cost 4**
* **h(A) = 3 ≤ true cost 2 → Now it's not admissible and not consistent**

**Try another:**

* **h(S) = 2**
* **h(A) = 1**
* **h(G) = 0**
* **cost(S→A) = 2**
* **Then: h(S) = 2 but cost(S, A) + h(A) = 2 + 1 = 3**
  + **2 ≤ 3   
    Now test h(A) = 2, then: 2 ≤ 2 + 0 = 2 →**

**But if h(A) = 3, 3 ≤ 2 + 0 = 2 → Inconsistent**

**So if h(A) increases too much, it breaks consistency but may still seem admissible in isolated checks.**

**Summary of Findings:**

| **Feature** | **Consistent Heuristic** | **Admissible but Not Consistent** |
| --- | --- | --- |
| **Follows triangle inequality** | **Yes** | **No** |
| **Never overestimates (admissible)** | **Always** | **Yes** |
| **May revisit nodes in A\*** | **No** | **Yes – less efficient** |
| **Example** | **Straight-line distance** | **Custom overestimate between steps** |

**Key Takeaway:**

* **All consistent heuristics are admissible.**
* **Some admissible heuristics may not be consistent, and this can affect the efficiency of algorithms like A\* (may require re-expanding nodes).**

**Task 4: Differences Between Tree Search and Graph Search**

* **What is Tree Search?**

**Tree Search is a search strategy where the algorithm builds a tree of paths starting from the initial state, expanding nodes by generating all possible successors.  
It does not remember previously visited states.**

**Think of it as: “Explore all possible paths, even if you visit the same place more than once.”**

**What is Graph Search?**

**Graph Search improves on tree search by keeping track of visited nodes (usually using a "closed list" or "explored set").  
This helps the algorithm avoid revisiting the same state and prevents infinite loops.**

**Think of it as: “Explore efficiently, don’t go back to places you’ve already been.”**

**Comparison Table**

| **Feature** | **Tree Search** | **Graph Search** |
| --- | --- | --- |
| **Remembers visited nodes?** | **No** | **Yes** |
| **May revisit nodes?** | **Yes (can loop)** | **No (avoids repetition)** |
| **Memory usage** | **Lower** | **Higher (stores explored nodes)** |
| **Time efficiency** | **Slower due to re-exploration** | **Faster due to pruning duplicates** |
| **Completeness** | **Not always (e.g., with cycles)** | **Yes (if branching factor is finite)** |
| **Optimality (with A\*)** | **If no cycles & consistent h(n)** | **Guaranteed with consistent h(n)** |

**Advantages and Limitations**

| **Algorithm** | **Advantages** | **Limitations** |
| --- | --- | --- |
| **Tree Search** | **Simple to implement, uses less memory** | **Can revisit same nodes, may be inefficient** |
| **Graph Search** | **More efficient, avoids redundant work** | **Needs more memory to store visited states** |

**When to Use Which?**

* **Use Tree Search:**
  + **When memory is limited**
  + **When the state space is small or acyclic**
  + **For quick prototyping or when duplicate states are unlikely**
* **Use Graph Search:**
  + **When the search space contains cycles**
  + **When performance and efficiency matter**
  + **When the same states may appear via multiple paths**

**Real-World Example**

* **Tree Search: A naive web crawler that keeps visiting the same page links repeatedly.**
* **Graph Search: A smart GPS system that avoids suggesting paths it has already evaluated**

**🧭 Level 2: Exploring Specific Algorithms**

**Task 5: Greedy Best-First Search (GBFS)**

**How GBFS Works:**

* **Greedy Best-First Search is an informed search algorithm.**
* **It selects the next node to expand based solely on the heuristic estimate of how close that node is to the goal.**
* **It uses:**

**f(n) = h(n)  
where:**

* + **h(n) = estimated cost from node n to goal**

**Node Selection Strategy:**

* **Always expands the node with the lowest h(n) (most promising).**
* **Uses a priority queue (min-heap) ordered by h(n).**

**Strengths:**

* **Fast and memory-efficient in many cases.**
* **Focuses on goal direction → less exploration.**

**Weaknesses:**

* **Not complete (may loop in infinite graphs).**
* **Not optimal – may find a non-shortest path.**
* **Can get stuck in local minima.**

**Diagram: GBFS Node Selection**

**(Simplified example)**

**mathematica**

**CopyEdit**

**A**

**/ | \**

**B C D**

**h=4 h=2 h=6**

**Goal: Reach node with h=0.**

* **Step 1: Start at A**
* **Step 2: Expand C (lowest h=2)**
* **Step 3: Continue with child of C that has lowest h**

***Task 6: A Search Algorithm*\***

***How A Works:*\***

A\* uses both the **cost so far** and **estimated cost to goal**:

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

Where:

* **g(n)** = actual cost from start to node **n**
* **h(n)** = estimated cost from **n** to goal
* **f(n)** = total estimated cost of the cheapest path through **n**

**Roles of Components:**

* **g(n):** Encourages **exploring cheaper paths** so far.
* **h(n):** Guides the search **toward the goal**.

**When is A\* Optimal?**

* A\* is **optimal** if **h(n) is admissible** (never overestimates the true cost).
* **Consistency** of h(n) improves efficiency (no need to revisit nodes).

**Example (Map Problem)**

**Start:** A → Goal: G  
Edges:

* A → B (cost=1), h(B)=4
* B → G (cost=3), h(G)=0
* A → C (cost=2), h(C)=2
* C → G (cost=2)

Step-by-step:

1. f(A) = g=0 + h=5 → f=5
2. f(B) = 1 + 4 = 5
3. f(C) = 2 + 2 = 4 ← expand C first
4. f(G) from C = 4 + 0 = 4 ← goal found with total cost 4

***Task 7: Why A is Optimal (with Admissible Heuristic)*\***

**Why A\* is Guaranteed to Find the Optimal Path:**

* If **h(n)** is **admissible**:
  + It never overestimates → A\* will not ignore better paths.
* A\* explores nodes in **increasing f(n)** order.
* If a suboptimal goal is found, a cheaper path must be in the queue.

**Proof Sketch (Bullet Points):**

* A\* uses f(n) = g(n) + h(n)
* With admissible h(n), f(n) is **lower-bound** of true cost
* A\* always picks node with **lowest f(n)**
* Goal is only picked when no cheaper path exists → optimal

**Why Consistency Helps More:**

* Consistent h(n) ensures:
  + **No need to re-expand nodes**
  + **Efficiency increases**
  + Always satisfies: h(n) ≤ cost(n, n') + h(n')

**Task 8: Comparison Table – BFS, DFS, GBFS, A\***

| **Feature** | **BFS** | **DFS** | **GBFS** | **A\*** |
| --- | --- | --- | --- | --- |
| **Uses Heuristic?** | **No** | **No** | **Yes (h(n))** | **Yes (g(n) + h(n))** |
| **Completeness** | **Yes** | **No (infinite loops)** | **No (may loop)** | **Yes (with finite branching)** |
| **Optimality** | **Yes (uniform cost)** | **No** | **No** | **✅ Yes (if h is admissible)** |
| **Time Complexity** | **O(b^d)** | **O(b^m)** | **O(b^m)** | **O(b^d) (worst case)** |
| **Space Complexity** | **O(b^d)** | **O(m)** | **O(b^m)** | **O(b^d)** |

**Legend:**

* **b = branching factor**
* **d = depth of solution**
* **m = maximum depth**

**🌍 Level 3: Real-World and Heuristic Design**

**Task 9: Heuristics in Real Life**

**GPS Navigation**

* **Heuristic: Straight-line distance (Euclidean or Haversine formula)**
* **Goal: Estimate time/distance to destination**
* **Used in: Google Maps, Waze**

**Game AI (e.g., Chess, Pacman)**

* **Heuristic: Material value of pieces, position score**
* **Goal: Evaluate board states**
* **Used in: Chess engines (like Stockfish), Pacman AI**

**Robotics (e.g., Robot Vacuum)**

* **Heuristic: Distance to dirty areas, battery level, obstacle map**
* **Goal: Efficiently clean with minimal time/energy**

**Task 10: Designing a Heuristic Function (Conceptual)**

**Chosen Scenario: Robot Vacuum Cleaning**

**Heuristic Goal:**

**Estimate how much work (cost) is left to clean the room.**

**Heuristic Function h(n) Could Consider:**

1. **Number of dirty tiles remaining**
2. **Distance to the nearest dirty tile**
3. **Battery level (if battery runs out, cost increases)**
4. **Obstacle density (walls, furniture = slower movement)**
5. **Docking station distance (if recharging is needed)**

**Example Conceptual Heuristic:**

**h(n) = (nearest\_dirty\_distance × 2) + (remaining\_dirty\_tiles × 1.5) - (battery\_level × 0.2)**

**This would help prioritize:**

* **Cleaning nearby dirty areas first**
* **Avoiding low battery situations**
* **Skipping blocked paths**