**🔰 Level 1: Understanding Core Concepts**

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

**Objective:**  
Research and define the term *heuristic* in the context of AI and search algorithms.

* What does a heuristic function do?

In artificial intelligence, a **heuristic** is a strategy or rule of thumb used to guide problem-solving and decision-making. It provides an **estimated measure** of how close a given state is to the goal, helping algorithms prioritize which paths to explore. Heuristics are especially vital in **search algorithms**, where they improve efficiency by reducing the number of paths that need to be evaluated.

* How does it differ from path cost?

A **heuristic function**, often denoted as *h(n)*, estimates the cost from a node *n* to the goal. It doesn’t guarantee accuracy but offers a **quick approximation** that helps the algorithm make informed decisions.

* Why is it used in informed search?

**Informed search algorithms**—like A\*, Greedy Best-First Search, and Beam Search—use heuristics to make smarter choices. Instead of blindly exploring every possibility (as in uninformed search), they **focus on promising paths**, saving time and computational resources.

Benefits:

* Faster problem-solving
* Reduced memory usage
* More scalable for complex tasks

Real-World Examples

* **GPS Navigation**: A heuristic might use straight-line (Euclidean) distance to estimate how close a location is to the destination.
* **Puzzle Solving**: In the 8-puzzle, heuristics like the number of misplaced tiles or Manhattan distance help guide moves.
* **Game AI**: In chess, heuristics evaluate board positions to decide the best move without simulating every possibility.

**Task 2: Admissible vs Inadmissible Heuristics**

**Objective:**

* Research what it means for a heuristic to be admissible.

A heuristic is **admissible** if it **never overestimates** the actual cost to reach the goal from any node in the search space. In other words, it always provides a **lower bound** or an exact estimate of the true cost.

Formal Definition:

For a heuristic function *h(n)* to be admissible:

* $$h(n) \leq h^\*(n)$$  
  Where:
* *h(n)* = estimated cost from node *n* to the goal
* *h⁎(n)* = actual minimum cost from node *n* to the goal
* Find at least **two examples** of admissible heuristics and **one inadmissible** heuristic.

**Manhattan Distance (Grid-based pathfinding)**

* **Used in:** 2D grid movement (like sliding puzzles or pathfinding in games), where movement is allowed in four directions.
* **Formula:**

h(n)=∣x1−x2∣+∣y1−y2∣h(n) = |x\_1 - x\_2| + |y\_1 - y\_2|h(n)=∣x1​−x2​∣+∣y1​−y2​∣

**Straight-Line Distance (Euclidean Heuristic)**

* **Used in:** Route-finding problems (like GPS navigation between cities).
* **Formula:**

h(n)=(x1−x2)2+(y1−y2)2h(n) = \sqrt{(x\_1 - x\_2)^2 + (y\_1 - y\_2)^2}h(n)=(x1​−x2​)2+(y1​−y2​)2​

* Explain the consequences of using an inadmissible heuristic in A\*.

**Loss of Optimality**

* A\* may **fail to find the shortest path**.
* Since the heuristic can **overestimate** the cost to the goal, it might **skip better paths** that appear more expensive than they actually are.

**Misguided Search**

* The algorithm may **prioritize wrong nodes**, leading to inefficient exploration.
* It could follow a path that seems promising due to inflated heuristic values but ends up being longer.
* Table comparing admissible/inadmissible heuristics with examples.

| **Aspect** | **Admissible Heuristic** | **Inadmissible Heuristic** |
| --- | --- | --- |
| **Definition** | Never overestimates the true cost to reach the goal | May overestimate the true cost to reach the goal |
| **Formula** | $$h(n) \leq h^\*(n)$$ | $$h(n) > h^\*(n)$$ for some nodes |
| **Guarantee in A\*** | Always finds the **optimal solution** | May return a **suboptimal solution** |
| **Examples** | - **Manhattan Distance** (grid pathfinding) - **Hamming Distance** (tile puzzles) | - **Euclidean Distance × 1.5** (navigation) - **Nilsson’s Sequence Score** (8-puzzle) |
| **Use Case** | When **accuracy and optimality** are critical | When **speed** is prioritized over guaranteed optimality |
| **Risk** | Slower but reliable | Faster but may miss the best path |

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

**Objective:**

* Define a consistent heuristic.
* What is the mathematical condition for consistency?
* Find a situation where a heuristic is admissible but not consistent.

**Deliverable:**  
Summary of findings + example scenario.

Consistent Heuristic

A **consistent heuristic** (also called *monotonic*) ensures that the estimated cost to reach the goal **never decreases** along a path. It satisfies the **triangle inequality**, meaning the heuristic value at a node is always less than or equal to the cost of reaching a neighbor plus the neighbor’s heuristic.

📐 Mathematical Condition:

For every node *n* and successor *n′*: $$ h(n) \leq c(n, n′) + h(n′) $$ Where:

* *h(n)* = heuristic estimate from node *n* to goal
* *c(n, n′)* = actual cost from *n* to *n′*

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

| **Aspect** | **Tree Search** | **Graph Search** |
| --- | --- | --- |
| **Definition** | Explores all paths without tracking visited nodes | Tracks visited nodes to avoid revisiting |
| **Memory Usage** | Lower (no closed list) | Higher (uses closed/explored list) |
| **Efficiency** | May revisit same states multiple times | Avoids redundant exploration |
| **Risk of Loops** | Can get stuck in cycles (in cyclic graphs) | Prevents infinite loops |
| **Optimality** | May find optimal path (with admissible heuristic) | More reliable for optimal path (with consistent heuristic) |
| **Use Case Preference** | Simpler problems or tree-structured spaces | Complex graphs with cycles or repeated states |

When to Use Each

 **Tree Search**: Use when memory is limited or the state space is small and acyclic.

**Graph Search**: Use when avoiding repeated work is critical, especially in large or cyclic graphs.

**🧭 Level 2: Exploring Specific Algorithms**

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

**How GBFS Works:**

**• GBFS uses a heuristic function h(n) to estimate the cost from a node to the goal.**

**• It selects the node with the lowest h(n), ignoring the actual path cost (g(n)).**

**Selection Method:**

**• Among all frontier nodes, pick the one with the smallest h(n) value.**

**Strengths:**

**• Fast and memory-efficient.**

**• Good for simple, goal-directed searches.**

**Weaknesses:**

**• Not optimal (can miss the best path).**

**• Not complete (may loop or miss the goal).**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Diagram (Textual):**

**scss**

**CopyEdit**

**Start**

**├── A (h=5)**

**├── B (h=2) ← selected**

**└── C (h=6)**

**B**

**├── D (h=4)**

**└── E (h=1) ← next selected**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Objective:**

* Research how GBFS works.
* How does it select the next node to expand?
* What are its strengths and weaknesses?

**Deliverable:**  
Summary report + diagram of node selection.

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

**Objective:**

* Explain how A\* combines g(n) and h(n) to compute f(n).
* What is the role of each component?
* In what scenarios is A\* considered optimal?

**Deliverable:**  
Written explanation + one example problem (e.g., path in a map) with annotated steps (no coding).

How it works:

A\* calculates: f(n) = g(n) + h(n)

• g(n): actual cost from start to node n

• h(n): estimated cost from n to goal

Roles:

• g(n) ensures accuracy so far

• h(n) guides future decisions

Optimality:

A\* is optimal when h(n) is admissible (never overestimates) and consistent.

Example (Grid Path):

markdown

CopyEdit

Start (0,0)

└── A (1,0) g=1, h=3 → f=4

└── B (2,0) g=2, h=2 → f=4

└── Goal (3,0) g=3, h=0 → f=3

A\* picks node with lowest f(n) at each step.

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

**Objective:**

* Explore the reasoning or proof sketch of why A\* is guaranteed to find an optimal path if h(n) is admissible.
* Why is consistency even better?

**Deliverable:**  
Write-up in bullet points + cited source.

• A\* finds optimal path if h(n) ≤ true cost to goal (admissible).

• It expands nodes in order of increasing f(n) = g(n) + h(n).

• Optimality is guaranteed because A\* won’t overlook cheaper paths.

• Consistency ensures f(n) is non-decreasing → avoids re-expanding nodes → more efficient.

Source:

Russell & Norvig, Artificial Intelligence: A Modern Approach, 3rd Edition

**Task 8: Comparison Table**

**Objective:**  
Create a table comparing the following algorithms:

* Breadth-First Search
* Depth-First Search
* Greedy Best-First Search
* A\* Search

**Compare on:**

* Use of heuristic
* Completeness
* Optimality
* Time complexity
* Space complexity

**Deliverable:**  
Clean comparison table.

Algorithm Uses Heuristic Complete Optimal Time Complexity Space Complexity

BFS ❌ No ✅ Yes ✅ Yes O(b^d) O(b^d)

DFS ❌ No ❌ No ❌ No O(b^m) O(bm)

GBFS ✅ Yes (h(n)) ❌ No ❌ No O(b^m) O(b^m)

A\* ✅ Yes (g+h) ✅ Yes ✅ Yes O(b^d) (best case) O(b^d)

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

**Task 9: Heuristics in Real Life**

**Objective:**  
Find examples of heuristics used in **real-world applications**:

* GPS navigation
* Game AI (e.g., Chess, Pacman)
* Robotics

**Deliverable:**  
One-slide explanation for each real-world example (include images if possible).

1. **GPS Navigation:**

Uses straight-line distance (h(n)) to estimate travel time.

**2. Game AI (e.g., Chess):**

Heuristics evaluate board positions (e.g., material balance, mobility).

3. **Robotics**:

Robot vacuum uses room layout + dirt level as heuristic to prioritize cleaning.

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

**Objective:**  
Choose one of the following:

* Maze solving
* Food delivery path planning
* Robot vacuum cleaning

Design a heuristic *conceptually*:

* What would you estimate as the remaining cost to goal?
* What factors would your heuristic consider?

Scenario: Food delivery path planning

Heuristic Idea:

• Estimated cost = straight-line distance to customer + traffic delay estimate.

Factors to Consider:

• Real-time traffic

• Road closures

• Customer priority (e.g., hot food first)