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

**🔰 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?
* How does it differ from path cost?
* Why is it used in informed search?

**Deliverable:**  
A one-page written explanation with real-world examples.

### ****What is a Heuristic in AI and Search Algorithms?****

In artificial intelligence (AI), a **heuristic** is a technique or strategy that helps an algorithm make decisions more efficiently by guiding the search process toward the goal. It is often used in **informed search algorithms**, where extra knowledge about the problem helps reduce the number of states explored.

#### ****What Does a Heuristic Function Do?****

A **heuristic function (usually denoted as h(n))** estimates the cost from a given node n to the goal node. It doesn't guarantee the exact cost, but gives a reasonable guess based on available knowledge. The goal is to help the algorithm prioritize which paths are likely to lead to the solution faster.

Example: In a navigation problem (like Google Maps), the heuristic function might be the straight-line distance ("as the crow flies") between the current location and the destination. While it may not represent the actual road distance, it provides a useful estimate.

#### ****How Does It Differ From Path Cost?****

**Path cost (g(n))** is the actual cost incurred so far to reach node n from the starting point (e.g., distance already traveled or time taken).

**Heuristic cost (h(n))** is the estimated remaining cost from n to the goal

In many algorithms like **A\***, both are combined into an evaluation function:  
**f(n) = g(n) + h(n)**

Thus, while path cost reflects the past, heuristics estimate the future.

#### ****Why Is It Used in Informed Search?****

Heuristics are essential in **informed search algorithms** like **Best-First Search** or **A\*** because they:

Help the algorithm make smarter decisions.

Reduce the number of nodes expanded.

Lead to faster and more efficient problem-solving.

In contrast to **uninformed search** (like BFS or DFS), which has no knowledge beyond the problem structure, informed search uses heuristics to guide the path toward the goal more directly.

### ****Real-World Examples****

**GPS Navigation Systems**

**Heuristic**: Straight-line distance to the destination.

**Path cost**: Actual distance traveled on roads.

**Use**: A\* algorithm uses both to find the fastest route.

**Game AI (Chess, Checkers)**

**Heuristic**: Evaluation of board position (e.g., number of pieces, control of the center).

**Path cost**: Number of moves made so far.

**Use**: Helps AI choose the best next move without exploring all possibilities.

**Robot Path Planning**

**Heuristic**: Estimated distance to the goal while avoiding obstacles.

**Use**: Helps robots navigate efficiently in real time.

**Task 2: Admissible vs Inadmissible Heuristics**

**Objective:**

* Research what it means for a heuristic to be admissible.
* Find at least **two examples** of admissible heuristics and **one inadmissible** heuristic.
* Explain the consequences of using an inadmissible heuristic in A\*.

**Deliverable:**  
Table comparing admissible/inadmissible heuristics with examples.

| **Aspect** | **Admissible Heuristic** | **Inadmissible Heuristic** |
| --- | --- | --- |
| **Definition** | A heuristic is **admissible** if it **never overestimates** the true cost to reach the goal from any node. | A heuristic is **inadmissible** if it **can overestimate** the actual cost to the goal. |
| **Guarantee** | Guarantees finding the **optimal solution** in A\*. | May **not find the optimal solution**, but might be faster. |
| **Condition** | For all nodes n: h(n) ≤ h\*(n) (actual cost to goal). | For some nodes n: h(n) > h\*(n). |
| **Use in A\*** | Safe to use; A\* remains **both complete and optimal**. | Risky to use; A\* becomes **non-optimal** (might miss best path). |
| **Example 1** | **Manhattan distance** in a grid with only up/down/left/right moves (no diagonals). |  |
| **Example 2** | **Straight-line (Euclidean) distance** in road maps (like GPS). |  |
| **Example of Inadmissible** |  | A heuristic that adds a constant overestimation (e.g., h(n) = straight-line distance + 5). |
| **Consequences in A\*** | - Expands fewer nodes than uninformed search. - Finds the **shortest or cheapest path**. | - Might miss optimal paths. - Can be **faster**, but **sacrifices correctness**. |

**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.

Here’s a clear and concise summary of your objective on **consistent (monotonic) heuristics**, along with an illustrative example:

📘 Definition: Consistent (Monotonic) Heuristic

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

This property ensures that the estimated total cost (f(n) = g(n) + h(n)) never decreases along a path, which is crucial for optimality in graph-based A\* search.

📐 Mathematical Condition for Consistency

For a heuristic function h(n), consistency requires:

h(n) \leq c(n, n') + h(n')

Where:

* n is any node
* n' is a successor of n
* c(n, n') is the actual cost from n to n'

This is equivalent to satisfying the **triangle inequality** in the context of pathfinding.

⚠️ Admissible but Not Consistent: Example Scenario

A heuristic is **admissible** if it never overestimates the true cost to reach the goal. However, it may still violate consistency.

Example Graph:

A → B → C → Goal

* Costs:
  + c(A, B) = 1
  + c(B, C) = 2
  + c(C, Goal) = 3
* Heuristic values:
* h(Goal) = 0
* h(C) = 0
* h(B) = 0
* h(A) = 2

Analysis:

* Admissibility:
  + All heuristic values are ≤ actual cost to goal → ✅ admissible
* Consistency check:
* h(A) = 2
* c(A, B) + h(B) = 1 + 0 = 1
* Since 2 > 1, this violates consistency → ❌ not consistent

✅ Summary of Findings

* **Consistent heuristics** obey the triangle inequality and ensure non-decreasing f(n) values.
* **Mathematical condition**: h(n) \leq c(n, n') + h(n')
* **Admissible ≠ Consistent**: A heuristic can underestimate the cost (admissible) but still violate monotonicity (not consistent), as shown in the example.

Would you like to explore how this impacts A\* search behavior or dive into real-world applications like route planning or puzzle solving?

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

**Objective:**

* Research the difference between **tree search** and **graph search** in the context of search algorithms.
* What are the advantages and limitations of each?
* When would each be preferred?

**Deliverable:**  
Comparison table + written explanation.

Comparison Table: Tree Search vs. Graph Search

| **Feature** | **Tree Search** | **Graph Search** |
| --- | --- | --- |
| **Explored Set** | ❌ Does not track visited nodes | ✅ Maintains a closed list of visited nodes |
| **Node Repetition** | May revisit the same state multiple times | Avoids revisiting already explored states |
| **Memory Usage** | Lower memory usage | Higher memory usage due to closed list |
| **Time Efficiency** | May be slower due to redundant exploration | Faster by avoiding repeated work |
| **Cycle Handling** | Can get stuck in infinite loops | Prevents cycles by tracking visited nodes |
| **Implementation Simplicity** | Easier to implement | Slightly more complex due to closed list management |
| **Optimality Guarantee** | Only with consistent heuristics | Guarantees optimality with admissible heuristics |
| **Use Case Preference** | Suitable for small or acyclic graphs | Preferred for large or cyclic graphs |

Written Explanation

🌳 Tree Search

* **How it works**: Explores the search space without remembering previously visited states.
* **Advantages**:
  + Simpler to implement.
  + Uses less memory.
* **Limitations**:
* Can revisit the same state multiple times.
* May enter infinite loops in cyclic graphs.
* Less efficient in large or complex graphs.

🔗 Graph Search

* **How it works**: Maintains a closed list to avoid revisiting states.
* **Advantages**:
  + More efficient by avoiding redundant exploration.
  + Handles cycles gracefully.
  + Ensures optimality with admissible heuristics.

· **Limitations**:

* Requires more memory.
* Slightly more complex to implement.

🧠 When to Use Each

* **Tree Search** is ideal when:
  + The state space is small or acyclic.
  + Memory is limited.
  + Simplicity is preferred over efficiency.
* **Graph Search** is preferred when:
* The state space is large or contains cycles.
* Optimality and efficiency are critical.
* You want to avoid redundant computation.

**🧭 Level 2: Exploring Specific Algorithms**

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

**Objective:**

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

**Deliverable:**  
🚀 Summary Report: Greedy Best-First Search (GBFS)

🔍 How GBFS Works

Greedy Best-First Search is an **informed search algorithm** that uses a **heuristic function** h(n) to estimate the cost from a node n to the goal. It always selects the node that appears **closest to the goal**, based solely on this heuristic.

* **Evaluation Function**:
* **Search Strategy**:  
  Expand the node with the **lowest heuristic value** first.

🧭 Node Selection Process

GBFS uses a **priority queue** to manage nodes:

* Nodes are sorted by their heuristic value.
* The node with the **lowest h(n)** is expanded next.
* It **ignores path cost** g(n), which can lead to suboptimal paths.

✅ Strengths

* **Fast exploration**: Quickly reaches the goal in many cases.
* **Simple to implement**: Only requires a heuristic function.
* **Low memory usage**: Stores fewer nodes compared to exhaustive searches.

❌ Weaknesses

* **Not optimal**: May miss the shortest path.
* **Not complete**: Can fail if the heuristic leads into dead ends.
* **Sensitive to heuristic quality**: Poor heuristics can misguide the search.
* **Can get stuck in local minima**: Chooses paths that look good short-term but aren’t globally best.

📈 Diagram: Node Selection in GBFS

Here’s a simplified diagram showing how GBFS selects nodes based on heuristic values:

Start (A) | |-- B (h=6) |-- C (h=3) ← Selected first | |-- D (h=2) ← Selected next | |-- Goal (h=0) ← Final selection

Start (A)

|

|-- B (h=6)

|-- C (h=3) ← Selected first

|

|-- D (h=2) ← Selected next

|

|-- Goal (h=0) ← Final selection

* GBFS chooses **C** over **B** because h(C) = 3 < h(B) = 6
* Then **D** is chosen over other neighbors
* Finally, **Goal** is reached with h = 0

Would you like to compare GBFS with A\* next or explore how different heuristics affect its performance?

***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).

Here’s a complete breakdown of the **A\*** search algorithm, how it works, and when it shines brightest—plus a step-by-step example to bring it to life:

🧠 How A\* Combines g(n) and h(n)

A\* uses an **evaluation function**:

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

Where:

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

This combination allows A\* to balance **exploration** (via h(n)) and **path accuracy** (via g(n)).

🔍 Role of Each Component

| **Component** | **Role** |
| --- | --- |
| **g(n)** | Tracks the cost already incurred to reach node n |
| **h(n)** | Guides the search by estimating remaining cost to goal |
| **f(n)** | Prioritizes nodes with lowest total estimated cost |

* If h(n) = 0, A\* behaves like **Dijkstra’s algorithm** (pure cost-based).
* If g(n) = 0, A\* behaves like **Greedy Best-First Search** (pure heuristic-based).

🏆 When Is A\* Optimal?

A\* is **guaranteed to find the shortest path** if:

* The heuristic h(n) is **admissible** (never overestimates the true cost).
* The heuristic is **consistent** (satisfies the triangle inequality: h(n) \leq c(n, n') + h(n')).

In such cases, A\* is both **complete** (finds a solution if one exists) and **optimal** (finds the best solution).

🗺️ Example Problem: Pathfinding on a Map

Scenario:

Find the shortest path from **Start (A)** to **Goal (G)**.

Graph:

A --1-- B --3-- D --2-- G \ / 4 2 \ / C

Heuristic values (h):

* h(A) = 6
* h(B) = 4
* h(C) = 5
* h(D) = 2
* h(G) = 0

🧭 Annotated Steps of A\*

1. **Start at A**
   * g(A) = 0
   * f(A) = g(A) + h(A) = 0 + 6 = **6**
2. **Expand A → B and C**
   * g(B) = 1, h(B) = 4 → f(B) = 5
   * g(C) = 4, h(C) = 5 → f(C) = 9
   * Choose B (lowest f)
3. **Expand B → D**
   * g(D) = g(B) + 3 = 4, h(D) = 2 → f(D) = 6
   * Choose D
4. **Expand D → G**

* g(G) = g(D) + 2 = 6, h(G) = 0 → f(G) = 6
* Goal reached!

✅ Final Path:

**A → B → D → G**  
Total cost: **6**

***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.

Why A\* Is Optimal with an Admissible Heuristic

* **Admissibility Defined**:  
  A heuristic h(n) is admissible if it **never overestimates** the true cost to reach the goal from node n.  
  Mathematically:

where \( h^\*(n) \) is the actual cost from \( n \) to the goal.

* **Optimality Guarantee**:  
  A\* expands nodes in order of increasing f(n) = g(n) + h(n).  
  Because h(n) is optimistic, A\* will never bypass a cheaper path in favor of a more expensive one.
* **Proof Sketch**:  
  Suppose A\* finds a suboptimal path p' before the optimal path p^\*.  
  Since h(n) \leq h^\*(n), the estimated cost f(p^\*) is **less than or equal to** f(p').  
  Therefore, A\* would have expanded p^\* before p', contradicting the assumption.  
  Hence, A\* must find the optimal path first.

🔁 Why Consistency Is Even Better

* **Consistency Defined**:  
  A heuristic is consistent (or monotonic) if for every node n and successor n':

This is the triangle inequality in pathfinding.

* **Benefits of Consistency**:
  + Ensures that f(n) values **never decrease** along a path.
  + Guarantees that once a node is expanded, its best path has been found—**no need to revisit**.
  + Enables efficient use of a **closed list** in graph search, reducing memory and computation.
* **Relationship to Admissibility**:
* Every consistent heuristic is admissible.
* But not every admissible heuristic is consistent.

📚 Cited Source

You can explore a formal proof and deeper insights in [this lecture from UBC](https://www.cs.ubc.ca/~mack/CS322/lectures/2-Search5.pdf), which outlines the admissibility and optimality of A\* in detail.

Would you like to see a visual example comparing consistent vs. inconsistent heuristics in action? It’s a great way to see how node expansion differs.

Why Consistency Is Even Better

* **Consistency Defined**:  
  A heuristic is consistent (or monotonic) if for every node n and successor n':

This is the triangle inequality in pathfinding.

* **Benefits of Consistency**:
  + Ensures that f(n) values **never decrease** along a path.
  + Guarantees that once a node is expanded, its best path has been found—**no need to revisit**.
  + Enables efficient use of a **closed list** in graph search, reducing memory and computation.
* **Relationship to Admissibility**:
* Every consistent heuristic is admissible.
* But not every admissible heuristic is consistent.

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

**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.

| **Criteria** | **Breadth-First Search (BFS)** | **Depth-First Search (DFS)** | **Greedy Best-First Search (GBFS)** | **A\* Search** |
| --- | --- | --- | --- | --- |
| **Uses Heuristic?** | ❌ No | ❌ No | ✅ Yes (h(n)) | ✅ Yes (f(n) = g(n) + h(n)) |
| **Completeness** | ✅ Yes (if branching factor is finite) | ❌ No (can get stuck in deep or infinite paths) | ❌ No (can get stuck in loops or wrong direction) | ✅ Yes (if h(n) is admissible) |
| **Optimality** | ✅ Yes (for uniform cost) | ❌ No | ❌ No | ✅ Yes (if h(n) is admissible & consistent) |
| **Time Complexity** | **O(b^d)** | **O(b^d)** (worst case) | **O(b^m)** (m = max depth, but often fast) | **O(b^d)** (depends on h(n)) |
| **Space Complexity** | **O(b^d)** | **O(d)** | **O(b^d)** | **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).

### ****Slide 1: GPS Navigation****

**Title:** Heuristics in GPS Navigation

**Real-World Application:** Google Maps, Apple Maps, etc.

**Heuristic Used:** **Straight-line (Euclidean) distance** from current location to destination.

**Purpose:** Estimate how close a location is to the destination to find the shortest or fastest route efficiently.

**Why It Works:** The straight-line distance gives a quick estimate that helps the system prioritize which roads or turns to evaluate first.

**Algorithm Used:** A\* Search (f(n) = g(n) + h(n))

📷 **Suggested Image:**  
Map screenshot showing routes with a pin on start and end point, or arrows showing direct vs. road distance.

### ****🎮 Slide 2: Game AI – Chess / Pac-Man****

**Title:** Heuristics in Game AI

**Real-World Application:** Computer players in **Chess** or **Pac-Man**

**Heuristic Used (Chess):** Piece value, control of center, king safety

**Heuristic Used (Pac-Man):** Distance to nearest pellet or ghost

**Purpose:** Evaluate how good a board or game state is without simulating every future move.

**Why It Works:** The heuristic gives a numeric "score" that helps the AI choose promising moves.

**Algorithm Used:** Minimax + heuristic evaluation function

📷 **Suggested Image:**  
Chessboard with evaluation numbers on positions OR a Pac-Man screen with arrows pointing to nearest pellet/ghost.

### ****🤖 Slide 3: Robotics****

**Title:** Heuristics in Robotics Path Plannin

**Real-World Application:** Warehouse robots (e.g., Amazon Robotics), vacuum robots

**Heuristic Used:** Estimated cost to goal based on **distance** and **obstacles**

**Purpose:** Help robots plan an efficient path in real-time while avoiding obstacles.

**Why It Works:** Reduces computation time by focusing on paths that seem closer to the goal.

**Algorithm Used:** A\*, D\* Lite, RRT, etc.

📷 **Suggested Image:**  
Robot navigating warehouse with path overlay, or grid with obstacles and goal

**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?

### ****Designing a Heuristic Function: Robot Vacuum Cleaning****

#### 🔍 ****Objective:****

Design a heuristic function to estimate the remaining cost for a robot vacuum to clean all dirty spots in a room and return to its charging dock.

### ✅ ****Heuristic Function Concept****

Let’s define the **heuristic h(n)** as:

**h(n) = D + R**

Where

**D** = Estimated distance to the **nearest uncleaned tile**.

**R** = Estimated distance from the **last dirty tile to the charging dock** (if return is required).

### 🔧 ****Factors to Consider in the Heuristic****

| **Factor** | **Why It's Important** |
| --- | --- |
| **Distance to nearest dirt** | Helps vacuum prioritize the closest uncleaned area. |
| **Total remaining dirt patches** | Helps estimate how far it still has to go (e.g., more scattered = more effort). |
| **Room obstacles (walls, furniture)** | Navigation cost is not just straight-line; obstacles affect the real path. |
| **Battery level** | If battery is low, cost should increase to reflect urgency to return to dock. |
| **Charging dock position** | If the robot must return after cleaning, this adds to the final cost. |

### 🎯 ****What Does the Heuristic Estimate?****

**Remaining cleaning effort**, not just distance

The **total expected movement cost** (steps or time) from current state to final goal (clean all and return).

It's an **admissible heuristic** if it never overestimates (e.g., always assumes minimum real-world effort

### 🧠 ****Example Scenario****

Robot is in living room corner.

Three dirt spots scattered across the house.

Furniture blocks straight paths.

Battery is 25%.

The heuristic might estimate:

2 meters to next dirt (D)

5 meters from last dirt to dock (R)

Plus a factor if battery is below threshold