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

**Level 1: Understanding Core Concepts**

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

In AI and search algorithms, a **heuristic** is an estimate used to guide decision-making. It helps determine how close a state is to the goal, without guaranteeing the best solution.

**Heuristic Function**

A **heuristic function (h(n))** estimates the cost from a node to the goal. It helps search algorithms decide which paths to explore first.

* **Example:** In GPS navigation, the straight-line distance to the destination is a heuristic.

**Difference from Path Cost**

* **Path cost (g(n))**: Actual cost from the start to node *n*
* **Heuristic (h(n))**: Estimated cost from *n* to the goal
* **A\* search uses:** f(n) = g(n) + h(n)

**Why Use Heuristics in Informed Search?**

Heuristics improve efficiency by reducing the number of explored nodes. Informed search algorithms like A\* use heuristics to find solutions faster than uninformed methods.

**Real-World Examples**

* **GPS apps**: Estimate travel time
* **Chess AI**: Evaluate board positions
* **Robots**: Navigate toward a target

**Task 2: What is an Admissible Heuristic?**

An admissible heuristic **never overestimates** the cost to reach the goal from any node. Formally, for a heuristic h(n)h(n)h(n), it is admissible if:

h(n)≤h∗(n)h(n) \leq h^\*(n)h(n)≤h∗(n)

Where:

* h(n)h(n)h(n) = estimated cost from node nnn to goal
* h∗(n)h^\*(n)h∗(n) = true cost from node nnn to goal

This ensures **optimality** in algorithms like A\*, as the estimated total cost will never be lower than the actual cost.

**❌ What is an Inadmissible Heuristic?**

An inadmissible heuristic may **overestimate** the true cost to the goal:

h(n)>h∗(n)(for at least one node)h(n) > h^\*(n) \quad \text{(for at least one node)}h(n)>h∗(n)(for at least one node)

This can lead A\* to **miss the optimal path**, possibly selecting suboptimal solutions due to misleading estimates.

**Comparison Table**

| **Heuristic Type** | **Definition** | **Example** | **Use in A\*** | **Outcome** |
| --- | --- | --- | --- | --- |
| **Admissible** | Never overestimates actual cost to goal | 1. **Manhattan Distance** (for 4-directional grids) 2. **Euclidean Distance** (for free-space maps) | Yes (ideal case) | Guarantees **optimal** solution |
| **Inadmissible** | May overestimate actual cost to goal | 1. **Manhattan Distance × 2** (doubles cost in grid problems) | Not recommended | May yield **suboptimal** or incorrect paths |

**Consequences of Using an Inadmissible Heuristic in A\***

* **Loss of Optimality**: A\* may return a longer or more costly path.
* **Faster Execution (Sometimes)**: May expand fewer nodes (because of greedier choices), but at the cost of correctness.
* **Risk of Missing Goal**: In edge cases, could miss feasible paths entirely.

**Task 3: Consistent (Monotonic) Heuristic – In Short**

* **Definition**: A heuristic is **consistent** if the estimated cost from a node to the goal is always less than or equal to the cost of reaching a neighbor plus that neighbor’s estimated cost.
* **Formula**:

h(n)≤c(n,n′)+h(n′)(for every node n and successor n′)h(n) \leq c(n, n') + h(n') \quad \text{(for every node \( n \) and successor \( n' \))}h(n)≤c(n,n′)+h(n′)(for every node n and successor n′)

* **Key Point**:  
  All consistent heuristics are **admissible**, but not all admissible heuristics are consistent.
* **Example (Admissible but Not Consistent)**:

S --1--> A --1--> G

h(S) = 2, h(A) = 2, h(G) = 0

* This heuristic is admissible (doesn't overestimate), but:

h(A)=2>1+h(G)=1⇒Not consistenth(A) = 2 > 1 + h(G) = 1 \Rightarrow \text{Not consistent}h(A)=2>1+h(G)=1⇒Not consistent

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

**Written Explanation**

**Tree Search** and **Graph Search** are two fundamental strategies in implementing search algorithms.

* **Tree Search** does **not keep track** of previously visited nodes. It treats every node as new, even if it's been explored before. This may lead to revisiting the same states multiple times and can result in infinite loops in cyclic graphs.
* **Graph Search**, in contrast, maintains a **visited set** (also called "closed list") to track explored nodes and avoids re-expanding them. This makes it more efficient and prevents loops.

**✅ When to Use:**

* **Tree Search** is preferred in **simple, acyclic problems** where revisiting nodes isn’t a concern, and the state space is small.
* **Graph Search** is preferred in **complex or cyclic graphs** where repeated exploration can significantly increase time and space complexity.

**Comparison Table: Tree Search vs Graph Search**

| **Feature** | **Tree Search** | **Graph Search** |
| --- | --- | --- |
| **Revisits Nodes** | Yes (does not check for duplicates) | No (uses visited/closed set) |
| **Memory Usage** | Lower (no closed list) | Higher (stores visited nodes) |
| **Time Efficiency** | May be inefficient in cyclic/large graphs | More efficient due to avoiding re-expansion |
| **Risk of Infinite Loops** | High in cyclic graphs | Avoids infinite loops |
| **Implementation Complexity** | Simpler | Slightly more complex |
| **Use Case** | Tree-structured or small problems | Graph-based or cyclic problems |

**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

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).



📑 A\* Search Algorithm

A\* is one of the most popular and powerful informed search algorithms. It uses both:

• The actual cost so far to reach a node (g(n))

• And an estimated cost to the goal from that node (h(n))

to decide which node to expand next.

📌 How A\* Combines g(n) and h(n)

For each node n, A\* computes:

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

Where:

• g(n) = actual cost from the start node to the current node n

• h(n) = heuristic estimate of the cost from n to the goal

• f(n) = estimated total cost of the cheapest solution through n

The node with the lowest f(n) value is selected for expansion next.

📌 Role of Each Component

Component Role

g(n) Keeps track of the actual cost incurred to reach the current node from the start

h(n) Provides an estimate of the cost from the current node to the goal, guiding the search efficiently

f(n) Represents the total estimated cost of a path passing through n — used to prioritize node expansion

This balanced combination makes A\* both complete and optimally efficient under certain conditions.

📌 When is A\* Optimal?

A\* is guaranteed to find the optimal solution when:

• The heuristic function h(n) is admissible (never overestimates the true cost to reach the goal)

• And ideally consistent (monotonic) (satisfies: h(n)≤c(n,n′)+h(n′)h(n) \leq c(n, n') + h(n'))

When these conditions hold:

• A\* expands the least number of possible nodes necessary to guarantee finding the shortest path.

📊 Example Problem: Pathfinding on a Map

Goal: Find the shortest path from point A to G

Graph:

A

/ \

(1) (4)

B C

| \ |

(3)(1) (2)

D E F

\ | /

(6)(1)(3)

G

Heuristic (h) values:

• h(A) = 7

• h(B) = 6

• h(C) = 5

• h(D) = 4

• h(E) = 2

• h(F) = 1

• h(G) = 0

📌 A\* Step-by-Step (No Coding, Annotated)

Step Current Node Open List (f(n)) Chosen Node

1 A B(1+6=7), C(4+5=9) B

2 B C(9), D(4+4=8), E(2+2=4) E

3 E C(9), D(8), G(3+0=3) G (Goal reached)

Path Found:

A → B → E → G

Actual Cost (g):

1 (A to B) + 1 (B to E) + 1 (E to G) = 3

📌 Why This Worked:

• At each step, A\* prioritized the node with the lowest f(n) value

• Balanced between actual cost so far (g) and estimated remaining cost (h)

• Because h(n) was admissible and consistent, A\* efficiently found the optimal path.

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 Admissible Heuristic)

📌 Reasoning and Proof Sketch (in Bullet Points)

• ✅ A\* Search selects the node with the lowest f(n) = g(n) + h(n) from the open list.

• ✅ If the heuristic h(n) is admissible, it never overestimates the true cost to reach the goal from any node.

• ✅ This ensures that the first time the goal node is selected for expansion, it has the lowest possible total path cost.

• ✅ No other path to the goal can have a lower f(n) than the one already selected, because:

o Any alternative path would either:

 Already be in the open list with a higher or equal f(n)

 Or have been expanded already with a higher g(n)

• ✅ As a result, A\* is complete (it finds a solution if one exists) and optimal (the solution has the lowest possible cost).

📌 Why is Consistency Even Better?

• ✅ A consistent (monotonic) heuristic satisfies:

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

for every node n and its successor n'.

• ✅ This guarantees that the f-values along any path are non-decreasing — meaning once a node is expanded in A\*, it never needs to be revisited.

• ✅ This improves efficiency because:

o It eliminates the need to check or update already expanded nodes.

o It makes the open and closed lists management simpler.

o It preserves optimality just like an admissible heuristic, but with less overhead.

📌 Summary of Key Points

• Admissibility ensures optimality because it prevents A\* from overlooking a cheaper path by guaranteeing heuristic estimates are optimistic.

• Consistency improves efficiency by preventing re-expansions of nodes and ensuring f(n) values along any path are non-decreasing.

• If the heuristic is consistent, it is automatically admissible.

📌 Cited Source

Primary Reference:

Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4th ed.). Pearson.

📖 See Chapter 3 — Solving Problems by Searching, especially section on A\* and properties of heuristics.

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.



📊 Comparison Table: BFS, DFS, GBFS, A\*

Feature Breadth-First Search (BFS) Depth-First Search (DFS) Greedy Best-First Search (GBFS) A\* Search

Use of Heuristic ❌ No ❌ No ✅ Yes (h(n)) ✅ Yes (g(n) + h(n))

Completeness ✅ Yes (if finite branching factor and finite goal depth) ❌ No (may get stuck in infinite path) ❌ No (may get stuck in loops or dead ends) ✅ Yes (if h(n) is admissible and positive cost)

Optimality ✅ Yes (for uniform step cost) ❌ No ❌ No ✅ Yes (if h(n) is admissible)

Time Complexity O(bd)O(b^d) O(bm)O(b^m) O(bm)O(b^m) (worst-case, with poor heuristic) O(bd)O(b^d) (with good heuristic, depends on h(n))

Space Complexity O(bd)O(b^d) O(bm)O(bm) O(bm)O(b^m) O(bd)O(b^d)

Best Use Case Finding shortest path in small, uniform-cost spaces Deep searches with memory constraints Fast approximate searches with good heuristic Finding optimal path efficiently with admissible heuristic

Legend:

• bb = branching factor (average number of successors per state)

• dd = depth of the shallowest solution

• mm = maximum depth of the search tree

🌍 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).



📑 Task 9: Heuristics in Real-World Applications

📍 Slide 1: GPS Navigation (Pathfinding)

🗺 Application:

GPS systems (like Google Maps, Waze) use heuristics to suggest the fastest or shortest route to a destination.

🎯 Heuristic Example:

Straight-line (Euclidean) distance from current location to destination is used as a heuristic estimate of remaining travel.

📌 How it works:

• The system estimates how far a location is "as the crow flies."

• Prioritizes roads leading in that direction, factoring in road type, speed limits, and traffic.

📈 Why it's effective:

It quickly eliminates unlikely routes and focuses on the most promising paths to compute the shortest or fastest travel time.

Image Idea:

• A map with routes and a highlighted straight line from point A to B labeled “Heuristic estimate”.

📍 Slide 2: Game AI (Chess / Pacman)

🎮 Application:

Game AIs use heuristics to make decisions about moves without evaluating every possible future outcome.

🎯 Heuristic Example (Chess):

Material advantage heuristic assigns point values to each piece:

• Queen = 9, Rook = 5, Bishop = 3, Knight = 3, Pawn = 1.

📌 How it works:

• The AI evaluates possible moves by summing the value of remaining pieces.

• Picks moves that improve its position according to this score.

🎯 Heuristic Example (Pacman):

Manhattan distance between Pacman and the nearest dot/ghost to decide movement.

📈 Why it's effective:

It allows fast, strategic decisions without exhaustive computation.

Image Idea:

• Chessboard with point values next to each piece

• Pacman map with dots and ghosts, and arrows showing distances.

📍 Slide 3: Robotics (Autonomous Navigation)

🤖 Application:

Robots use heuristics for real-time pathfinding and obstacle avoidance in physical environments.

🎯 Heuristic Example:

Estimated distance to goal (Euclidean or Manhattan) combined with sensor data for nearby obstacles.

📌 How it works:

• Robot uses sensors (LIDAR, sonar) to map surroundings.

• Heuristic estimates cost of moving toward goal while avoiding obstacles.

📈 Why it's effective:

Balances path efficiency with safety by avoiding areas with high risk or delay.

Image Idea:

• A robot navigating a grid with obstacles

• Path marked based on estimated costs, with goal location highlighted.

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?

Great — let’s pick Food Delivery Path Planning for Task 10 and conceptually design a heuristic for it. I’ll structure it neatly for your submission:



📑 Task 10: Designing a Heuristic Function — Food Delivery Path Planning

📌 📦 Problem Context:

An AI system plans the most efficient path for a food delivery agent to reach a customer's location through city streets, avoiding traffic and delays.

📌 🎯 What Would You Estimate as the Remaining Cost to Goal?

The heuristic would estimate the remaining delivery time from the current location to the customer's address.

h(n)=estimated time (in minutes) to reach the customer from node nh(n) = \text{estimated time (in minutes) to reach the customer from node } n

📌 📊 Factors the Heuristic Would Consider:

Factor Why It Matters

Straight-line distance to goal Basic estimate of how far the location is geographically

Current traffic conditions Affects actual travel time — higher traffic, longer travel

Road types and speed limits Highways and major roads allow faster travel than small streets

Number of traffic signals and junctions Each stop or turn can increase delivery time

Weather conditions (optional) Heavy rain or bad weather slows down movement

Ongoing road closures or diversions May force detours, increasing estimated travel time

📌 📈 Conceptual Heuristic Formula:

A possible simplified conceptual formula:

h(n)=d(n,goal)vavg(n)+traffic\_delay(n)h(n) = \frac{d(n, \text{goal})}{v\_{\text{avg}}(n)} + \text{traffic\\_delay}(n)

Where:

• d(n,goal)d(n, \text{goal}) = straight-line distance to goal

• vavg(n)v\_{\text{avg}}(n) = average effective speed on roads from n (based on road type and traffic)

• traffic\_delay(n)\text{traffic\\_delay}(n) = estimated time lost in traffic signals, jams, or diversions

📌 ✅ Why This Heuristic Would Be Effective:

• Admissible: It estimates real travel time based on distance and traffic but won't overestimate.

• Informed: Uses real-world factors (traffic, speed limits) for accurate route guidance.