Heuristics in Artificial Intelligence

Definition

A **heuristic** is a strategy, technique, or rule of thumb that guides the search process in problem solving and decision making. In Artificial Intelligence (AI), heuristics are used to improve the efficiency of algorithms by making them "intelligent guesses" instead of exhaustively checking all possible solutions.

Mathematically, a heuristic can be represented as an **evaluation function**:

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

Where:

- $f(n)$ = total estimated cost of the solution path through node nn.
- $g(n)$ = cost from the start node to the current node nn.
- $h(n)$ = heuristic estimate of the cost from nn to the goal node.

The function $h(n)$ is what we call the **heuristic function**.

Need for Heuristics

- Real-world problems like Chess, Route Planning, or Speech Recognition have **huge search spaces**.
- Without heuristics, algorithms like BFS or DFS may take impractical amounts of time.
- Heuristics help in **reducing the branching factor** by guiding the search towards promising paths.

Properties of Heuristics

1. **Admissibility**
    - A heuristic is admissible if it **never overestimates** the cost to reach the goal.

- Example: In route planning, using “straight line distance” between cities is admissible because it is always less than or equal to the actual distance.

Mathematically:

$$h(n) \leq h^*(n) \quad \forall n$$

Where $h^*(n)$ is the true cost to reach the goal.

2. **Consistency (Monotonicity)**
    - A heuristic is consistent if the estimated cost is always less than or equal to the step cost plus the estimated cost from the next node.
    - Consistent heuristics ensure that the values of f(n)=g(n)+h(n) are **non-decreasing** along any path.

3. **Accuracy**
    - A heuristic should be as close as possible to the actual cost, but still admissible.
    - A more accurate heuristic reduces the search effort.

Examples of Heuristics

1. **In Pathfinding / Navigation**
    - **Manhattan Distance** (for grids):

$$h(n) = \left|x_{\{goal\}} - x_{\{current\}}\right| + \left|y_{\{goal\}} - y_{\{current\}}\right|$$

    - **Euclidean Distance**:

$$h(n) = \sqrt{(\left(x_{\{goal\}} - x_{\{current\}}\right)^2 + \left(y_{\{goal\}} - y_{\{current\}}\right)^2)}$$

Heuristic Search Algorithms

- **Greedy Best-First Search**
  Uses only heuristic estimate:

$$f(n) = h(n)$$

Fast but not always optimal.

- *A Search**
  Uses both path cost and heuristic:

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

If h(n) is admissible and consistent, A* guarantees optimal solution.

- **IDA*** (Iterative Deepening A*)
  Combines depth-first search with A* heuristic.

Advantages of Heuristics

- Reduces search space.
- Provides near-optimal solutions in complex domains.
- Makes algorithms practical for real-world problems.

Disadvantages of Heuristics

- May not always give the best (optimal) solution.
- Requires domain knowledge to design a good heuristic function.
- Poor heuristics can mislead the search and increase computation time.

Conclusion

Heuristics act as an **intelligent guide** for search algorithms in AI. By using evaluation functions such as:

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

heuristics balance between the known cost g(n)g(n) and the estimated cost h(n)h(n). Admissible and consistent heuristics are crucial for guaranteeing optimal solutions. Overall, heuristics make problem solving feasible in domains where exhaustive search is impossible.

Best First Search (Informed Search)

Best First Search is a heuristic search algorithm that selects the most promising node for expansion based on an evaluation function. It prioritizes nodes in the search space using a heuristic to estimate their potential. By iteratively choosing the most promising node, it aims to efficiently navigate towards the goal state, making it particularly effective for optimization problems.

Approach:

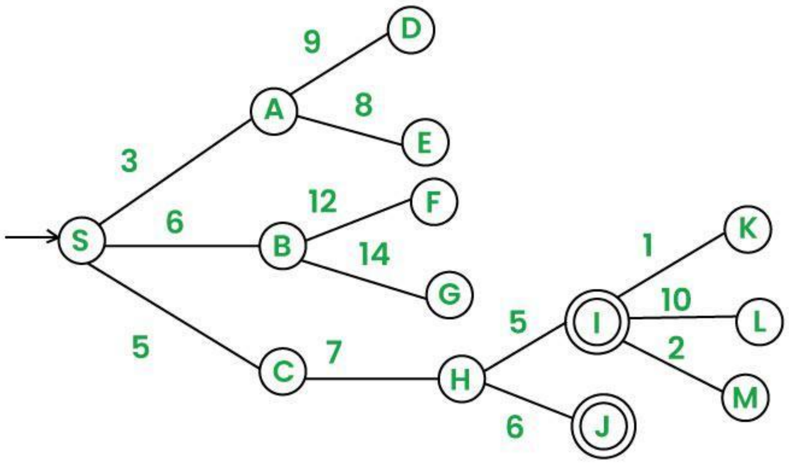
The idea is to use priority queue or heap to store the costs of edges that have lowest evaluation function.

Follow the below given steps:

- Initialize an empty Priority Queue named **pq**.
- Insert the starting node into **pq**.
- While **pq** is not empty:
    - Remove the node **u** with the lowest evaluation value from **pq**.
    - If **u** is the goal node, terminate the search.
    - Otherwise, for each neighbor **v** of **u**: If **v** has not been visited, Mark **v** as visited and Insert **v** into **pq**.
    - Mark **u** as examined.
- End the procedure when the goal is reached or **pq** becomes empty.

Illustration:

- Let us consider the below example:



- *We start from source "S" and search for goal "I" using given costs and Best First search.*
- *pq initially contains S*
    - *We remove S from pq and process unvisited neighbors of S to pq.*
    - *pq now contains {A, C, B} (C is put before B because C has lesser cost)*
- *We remove A from pq and process unvisited neighbors of A to pq.*
    - *pq now contains {C, B, E, D}*
- *We remove C from pq and process unvisited neighbors of C to pq.*
- *pq now contains {B, H, E, D}*
- *We remove B from pq and process unvisited neighbors of B to pq.*
    - *pq now contains {H, E, D, F, G}*
- *We remove H from pq.*
- *Since our goal "I" is a neighbor of H, we return.*

A* Algorithm

The A* algorithm is highly effective and well-known search technique utilized for finding the most efficient path between two points in a graph. It is applied in scenarios such as pathfinding in video games, network routing and various artificial intelligence (AI) applications. It was developed in 1968 by Peter Hart, Nils Nilsson and Bertram Raphael as an improvement on Dijkstra's algorithm.

- While Dijkstra's algorithm explores all possible directions around the starting node uniformly
- A* combines actual travel cost with an estimated future cost(heuristics)

This optimizes the search process, reduces computational load and ensures the most efficient path is found with minimal unnecessary exploration.

Key Components of A* Algorithm

A* uses two important parameters to find the cost of a path:

1. g(n)*:* Actual cost of reaching node *n* from the start node. This is the accumulated cost of the path from the start node to node *n*.
2. h(n)**:** The heuristic finds of the cost to reach the goal from node *n*. This is a weighted guess about how much further it will take to reach the goal.

The function, $f(n)=g(n)+h(n)$ is the total estimated cost of the cheapest solution through node *n*. This function combines the path cost so far and the heuristic cost to estimate the total cost guiding the search more efficiently.

How A* Algorithm Works?

A* is an informed search algorithm, means it uses the $f(n)$ function to prioritize which nodes to explore next. The process can be broken down into the following steps:

1. **Initialization:** The initial node is added to the open set, a collection of nodes that are yet to be explored. The $f(n)$ value for the start node is calculated using the heuristic.
2. **Loop**: A* selects the node with the lowest $f(n)$ value from the open set. This node is expanded and its neighbors are examined.
3. **Goal Check**: If the node being processed is the goal node, the search terminates and the algorithm returns the path to the goal.
4. **Node Expansion**: Each neighbor of the current node is evaluated based on the $g(n)$, $h(n)$ and $f(n)$ values. If a better path to a neighbor is found i.e a lower $f(n)$ then the neighbor is added to the open set or its values are updated.
5. **Repeat**: This process continues until the goal is found or the open set is empty which means there is no solution.

Applications of A* Algorithm

The A* algorithm's ability to find the most efficient path with a given heuristic makes it suitable for various practical applications:

1. **Pathfinding in Games and Robotics:** A* is used in the gaming industry to control characters in dynamic environments as well as in robotics for navigating between points.
2. **Network Routing:** In telecommunications, it helps in finding the shortest routing path that data packets should take to reach the destination.
3. **AI and Machine Learning:** It can be used in planning and decision-making algorithms where multiple stages of decisions and movements need to be evaluated.
4. **Map and Navigation Systems:** GPS navigation systems use A* to find optimal driving or walking routes in real time.
5. **Logistics and Supply Chain:** A* helps in optimizing routes for delivery vehicles and warehouse robots to minimize travel time and costs.

Advantages of A* Algorithm

1. **Optimality:** When equipped with an admissible heuristic, it is guaranteed to find the shortest path to the goal.
2. **Completeness:** It will always find a solution if one exists.
3. **Flexibility:** By adjusting heuristics, it can be adapted to a wide range of problem settings and constraints.
4. **Efficiency:** With a good heuristic, it explores fewer nodes than uninformed algorithms like Dijkstra's, making it faster in many cases.
5. **Wide Applicability:** It works on both weighted and unweighted graphs, grids and other search spaces making it versatile for different domains.

Limitations and Considerations

1. **High Memory Usage:** It stores all open and closed nodes which can become impractical for very large graphs.
2. **Heuristic Sensitivity:** The efficiency and optimality depend heavily on the quality of the chosen heuristic.
3. **Computational Overhead:** In complex or dense graphs, it can still take considerable time to find the path.
4. **Not Always Suitable:** For real-time or memory-constrained systems, simpler algorithms like greedy search may be preferred.