UNIT 2: Search Algorithms

Random search, Search with closed and open list, Depth first and Breadth first search, Heuristic search, best first search, A* algorithm, Game Search.

Random Search ¶

What it is: A method where you **try options at random** to see if they solve the problem.

How it works:

1. **Pick a random choice** from all possible moves or solutions.
2. **Check** if it reaches the goal.
3. If **not**, pick another **random choice** and try again.

Pros: Simple to implement.
Cons: Can take **a long time** if the solution is hard to find.

Open and Closed Lists in Search

- **Open List (To-Do List):**
  These are the **nodes** or **states** your algorithm still needs to explore. Think of it like a **"to-do"** list in search.
- **Closed List (Done List):**
  These are the **nodes** or **states** your algorithm has **already checked**. Think of it like a **"done"** list in search.

Depth-First Search (DFS)

- **Process**: Go **deep** into one branch of the graph or tree before exploring other branches.
- **Analogy**: Imagine **tunneling** straight down in a cave until you can't go further, then coming back up to start digging another tunnel.

Depth-First Search (DFS) Uses

1. **Maze or Puzzle Solving** → DFS dives into one path fully; if it's a dead end, it backtracks.
2. **Detecting Cycles** → DFS helps find loops in networks, like checking for circular dependencies.
3. **Tree-Based Problems** → Many algorithms (e.g., checking each branch of a family tree) use DFS to explore everything in one direction before moving on.

Breadth-First Search (BFS)

- **Process**: Explore all nodes **level by level**, starting from the root, then moving to its neighbors, and so on.
- **Analogy**: Imagine **spreading out** like a wave, visiting everything close first, then moving to the next layer.

Breadth-First Search (BFS) Uses

1. **Social Networks** → Finding the shortest connection between two people (e.g., "friend-of-a-friend").
2. **Route Finding** → In simple maps or grids, BFS quickly finds the shortest path.
3. **Web Crawlers** → BFS visits pages level by level, discovering links in a structured way.

Note: Practice 1-2 problem questions on both Search, we have also covered this in our respective AI classes

---

Heuristics in AI

In Artificial Intelligence, **heuristics** are **rules of thumb** or **educated guesses** used to **guide a search** or **decision-making** process. Instead of checking **every possible path** (which can be huge), a heuristic helps the AI **focus** on **promising** options first.

Heuristics (Very Simple Explanation): A heuristic is like a useful shortcut that helps quickly guess or estimate a good solution instead of looking at every single possibility. It doesn't always give the perfect answer, but it's usually fast and good enough for most purposes.

Why Do We Use Heuristics?

1. **Speed**: They help the AI find **good solutions quickly** rather than getting stuck exploring all possibilities.
2. **Practical**: Many AI problems (like puzzles or route-finding) would take too long to solve with **brute force** methods.
3. **Easy to Apply**: They can be made from **domain knowledge** (e.g., how far we are from a goal, or how many steps remain).

Uses of Heuristics in Various Sectors

1. Healthcare

- **Diagnosis**: Doctors and AI systems use heuristics to quickly narrow down possible diseases.
- **Scheduling**: Hospitals use simple rules to assign staff and arrange appointments efficiently.

2. Banking

- **Fraud Detection**: Heuristic rules spot unusual spending patterns, helping banks catch fraud faster.
- **Loan Approvals**: Banks use quick checks (e.g., credit score thresholds) to decide on lending.

3. E-Commerce

- **Recommendation Systems**: Heuristics pick products likely to interest the customer, based on browsing history.
- **Inventory Management**: Simple guidelines help decide when to reorder stock to avoid shortages.

4. Transportation

- **Route Planning**: Apps use heuristics (like shortest distance) to suggest fast travel routes.
- **Traffic Management**: Cities apply rules to adjust traffic lights or reroute vehicles.

5. Marketing

- **Customer Segmentation**: Heuristics group people by behavior or location for targeted campaigns.
- **Ad Placement**: Quick rules decide which ads to show, based on user interests or keywords.

6. Gaming

- **Computer Players**: Games like chess or tic-tac-toe use heuristics to focus on promising moves first.
- **Difficulty Levels**: Heuristic adjustments make the computer opponent easier or harder.

Example in Search Algorithms

- **A*** algorithm uses a **heuristic function** (h(n)) to guess how close you are to the goal.
- This helps **prioritize** exploring states that look more promising, **speeding up** the search.

Limitations

- **Not Always Perfect**: Heuristics can **overestimate or underestimate**, leading to suboptimal paths.

Conclusion

Heuristics in AI are **simple guidelines** that make problem-solving **faster and more efficient**. They're **key** in **search algorithms**, **game-playing** agents, and **many other AI applications** where exploring all paths is impossible.

A* Algorithm (A Star Algorithm)

The A* algorithm is a popular search and pathfinding algorithm used in many fields, including robotics and game development, due to its performance and accuracy. **It finds the shortest path from a start node to a target node while trying to minimize the total cost (distance, time, etc.).**

---

1. Key Components

- **g(n)**: **Actual cost** from the start node to node **n**.
- **h(n)**: **Heuristic estimate** from node **n** to the goal. Must be **admissible** (never overestimates).
- **f(n) = g(n) + h(n)**: **Total estimated cost** of the path through **n**.

---

2. Basic Steps

1. **Initialization**: Put the **start node** into an **open list** of nodes to explore.
2. **Select Node**: Pick the node in the open list with the **lowest f(n)**.
3. **Goal Check**: If this node **is the goal**, we are done.
4. **Neighbors**: For each neighbor of the chosen node:
    - Calculate its **f(n)**.
    - If this neighbor is **not** in the open list, add it.
    - If it **is** in the open list but now has a **lower** f(n), **update** it.
5. **Move On**: Move the chosen node to a **closed list** so it's not revisited.
6. **Repeat** until the goal is found or the open list is **empty** (no solution).

---

3. Choosing the Heuristic

- A **good heuristic** makes A* faster by guiding the search in a **promising direction**.

Conclusion

A* finds **shortest paths** efficiently if **h(n)** does **not** overestimate. It's widely used in **games**, **navigation apps**, and **robot path planning**.

Follow this link to practice A* problem: https://www.101computing.net/a-star-search-algorithm/ (https://www.101computing.net/a-star-search-algorithm/)

SMA (Simplified Memory-Bounded A)

What is it?

- SMA* is a **pathfinding** method like A*, but it uses **limited memory**.
- It helps in **large problems** where normal A* might **run out of memory**.

How Does It Work?

1. **Store Only Best Paths**: SMA* keeps **the most promising nodes** in memory.
2. **Memory Full?**: If memory is getting full, SMA* **removes** less promising nodes.
3. **Re-Expand**: If it needs a removed node again, it **re-calculates** it.

---

*Why Use SMA?**

- It **prevents memory overload**, which can happen in big searches.
- It **still** finds a **good path**, but it **may take longer** because it sometimes re-does work.
