

A* Search Algorithm

Definition:

A* is an intelligent search algorithm used to find the **shortest path** between a start node and a goal node in a graph or grid. It combines the actual cost from the start (**g(n)**) with a heuristic estimate of the remaining cost (**h(n)**) to prioritize which nodes to explore.

Why A* in the Drone Delivery Problem?

- Considers the **actual travel cost** between delivery locations.
- Uses a **heuristic function** to guide the search toward promising routes.
- Reduces unnecessary exploration compared to DFS and BFS.
- Can guarantee an **optimal solution** when an admissible heuristic is used.
- Helps minimize **delivery distance, time, and energy consumption**.
- Provides a good balance between **solution quality** and **computational efficiency**.

How Does A* Work?

A* works by **systematically exploring paths** while always choosing the most promising option based on both the **actual cost so far** and an **estimated remaining cost**. At each step, A* evaluates nodes using the formula: $f(n) = g(n) + h(n)$

Step-by-step process:

1. **Start node initialization**
 - The start node is placed in a **priority queue**.
 - Its cost so far $g(n)$ is 0.
 - Its heuristic $h(n)$ estimates the remaining distance to the goal.
2. **Node selection**
 - A* always selects the node with the **lowest f(n)** value from the queue.
 - This node is considered the most promising to lead to an optimal solution.
3. **Node expansion**
 - The selected node is expanded by generating its neighboring nodes.
 - For each neighbor:
 - Update the actual cost $g(n)$.
 - Compute the heuristic $h(n)$.
 - Calculate the total score $f(n)$.
4. **Queue update**
 - All newly generated nodes are added to the priority queue.
 - Nodes with lower $f(n)$ values are explored first.
5. **Goal check**
 - If the goal node is reached (or all required nodes are visited), A* stops.
 - The path with the lowest total cost is returned

Python Implementation

```
● ● ●

import heapq

def heuristic(node, unvisited, graph):
    if not unvisited:
        return 0
    return min(graph[node][u] for u in unvisited)

def a_star(graph, start):
    n = len(graph)
    pq = []
    heapq.heappush(pq, (0, 0, start, [start], {start}))

    best_cost = float('inf')
    best_path = []

    while pq:
        f, g, node, path, visited = heapq.heappop(pq)

        if len(visited) == n:
            if g < best_cost:
                best_cost = g
                best_path = path
            continue

        unvisited = set(range(n)) - visited

        for nxt in unvisited:
            new_g = g + graph[node][nxt]
            h = heuristic(nxt, unvisited - {nxt}, graph)
            f = new_g + h
            heapq.heappush(pq, (f, new_g, nxt, path + [nxt], visited | {nxt}))
    return best_path, best_cost

graph = [
    [0, 2, 9, 10, 7],
    [1, 0, 6, 4, 3],
    [15, 7, 0, 8, 9],
    [6, 3, 12, 0, 5],
    [10, 4, 8, 6, 0]
]

best_path, best_cost = a_star(graph, 0)

nodes = ['A', 'B', 'C', 'D', 'E']
readable_path = [nodes[i] for i in best_path]

print("A* Optimal Path:", readable_path)
print("Total Cost:", best_cost)
```

Advantages

- Finds the **shortest / least-cost path**
 - Uses a **heuristic** to guide the search efficiently
 - Expands fewer nodes than DFS and BFS
 - Guarantees optimality with an **admissible heuristic**
 - Suitable for **weighted graphs** (real distances)
-

Disadvantages

- Requires **more memory** than DFS
 - Performance depends heavily on heuristic quality
 - Can become slow for very large search spaces
 - Designing a good heuristic may be difficult
-

1. Execution Time

A* is generally faster than DFS and BFS because it explores only the most promising routes. However, with a weak heuristic, its performance may degrade

O(b^d)

2. Memory Usage

A* uses **high memory** because it stores:

- Priority Queue
- Multiple partial paths
- Cost information for each node

O(b^d)

3. Success Rate

A* always finds a solution **if one exists**, provided the graph is finite and connected.

4. Solution Optimality

A* produces an **optimal solution** when using an **admissible heuristic**.

It guarantees the shortest or least-cost route.

5. Scalability

A* scales **better than DFS and BFS**,
but for very large problems (many delivery points),
its memory usage can become a limitation.