A* Search Algorithm in Competitive Programming

Competitive Programming Club Meeting

Joshua Ganschow

March 25, 2025

Introduction

- An informed search algorithm that balances path cost with heuristic guidance.
- Widely applicable in grid problems, maze navigation, and puzzles commonly found in contests.
- Benefits: optimality, efficiency, and adaptability to different problem constraints.

A* Search Basics

- Maintains two primary functions:
  - $g(n)$: Cost from the start node to node n.
  - $h(n)$: Heuristic estimate from node n to the goal.
- Combines them into $f(n) = g(n) + h(n)$ to choose the next node.
- Uses a priority queue (open set) to explore promising paths first.

A* Algorithm: Pseudocode I

```
function A*(start, goal):
    openSet = priority_queue()
    openSet.push(start, f(start))
    gScore[start] = 0

    while not openSet.empty():
        current = openSet.pop()  # Node with lowest
            f(n)
        if current == goal:
            return reconstruct_path(current)

        for neighbor in current.neighbors:
            tentative_gScore = gScore[current] +
                cost(current, neighbor)
            if tentative_gScore <
                gScore.get(neighbor, infinity):
                neighbor.parent = current
```

A* Algorithm: Pseudocode II

```
                gScore[neighbor] = tentative_gScore
                fScore = tentative_gScore +
                    h(neighbor, goal)
                openSet.push(neighbor, fScore)

    return failure
```

Key Concepts

- **Admissible Heuristic:**
  - $h(n)$ never overestimates the true cost.
  - Ensures that A* finds an optimal solution.
- **Consistent Heuristic:**
  - For every node n and neighbor m:

$$h(n) \leq cost(n, m) + h(m)$$

  - Guarantees non-decreasing $f(n)$ values.

Designing Effective Heuristics

- Tailor heuristics to problem constraints.
- **Common Examples:**
  - *Manhattan Distance* for grid problems with only four directions.
  - *Euclidean Distance* when diagonal moves are allowed.
- Trade-off: More accurate heuristics might be costlier to compute.

Practical Applications in Competitive Programming

- Ideal for grid-based pathfinding, maze solving, and other puzzles.
- Adaptable to problems with varying constraints (e.g., minimizing cost, steps, or time).
- Can be integrated with other techniques for optimized contest solutions.

Applications in Deep Learning and AI

- **Robotics and Autonomous Navigation:**
  - Real-time path planning in dynamic environments.
  - Integrates with sensor data for obstacle avoidance.
- **Game AI:**
  - Powers non-player character (NPC) navigation.
  - Supports complex decision-making in large game worlds.
- **Reinforcement Learning and Planning:**
  - Used in model-based reinforcement learning for planning future actions.
  - Assists in simulating potential outcomes in dynamic systems.
- **Neural Architecture Search (NAS):**
  - Guides the search for efficient deep learning models.
  - Combines with deep networks to predict heuristic values for complex design spaces.

Python code

I have some example code for A* on a 2D grid with a cool visual.

Complexity and Considerations

- **Time Complexity:** Depends on the number of nodes and the heuristic's accuracy.
- **Space Complexity:** Storage for the open set and auxiliary data structures.
- **Trade-offs:**
  - More precise heuristics can significantly reduce search space but may increase computation.
- In contests, a well-chosen heuristic can be the key to fast and efficient solutions.

A* vs Other Search Algorithms

- **Dijkstra's Algorithm:**
  - No heuristic guidance; explores all paths equally.
- **Breadth-First Search (BFS):**
  - Suitable for unweighted graphs but inefficient in large search spaces.
- A* outperforms when a good heuristic is available.

Tips for Contest Use

- Start with simpler approaches (BFS, Dijkstra) and evolve to A*.
- Choose heuristics carefully ensure they are admissible and consistent.
- Test on edge cases to validate correctness and performance.
- Consider memory and time limits in contest settings.

Variants of A*

- **Weighted A***
  - Uses a weight $w > 1$ in the heuristic:

$$f(n) = g(n) + w \cdot h(n)$$

  - Trades optimality for faster search.
- **Iterative Deepening A* (IDA*)**
  - Combines A* with iterative deepening to reduce memory usage.
  - Increases threshold iteratively until the goal is found.
- **Real-Time A* (RTA*)**
  - Designed for environments with strict time constraints.
  - Makes decisions based on limited lookahead.
- **Other Variants**
  - *Memory-Bounded A* (MA*)*: Optimizes memory usage.
  - *Bidirectional A**: Searches from both start and goal simultaneously.

Conclusion

- A* is a great pathfinding algorithm, both easy to implement and efficient.
- Use it when you can!

Conclusion

- A* is a great pathfinding algorithm, both easy to implement and efficient.
- Use it when you can!

Practice and/or other materials:

- Leetcode - Sliding Puzzle
- Red Blob Games - A*