A-star (A*)

DCA3702

Ivanovitch Silva

A Formal Basis for the Heuristic Determination of Minimum Cost Paths

Publisher: IEEE Cite This PDF

Peter E. Hart ; Nils J. Nilsson ; Bertram Raphael **All Authors**

Abstract
Authors
References
Citations
Keywords
Metrics

Abstract:
Although the problem of determining the minimum cost path through a graph arises naturally in a number of interesting applications, there has been no underlying theory to guide the development of efficient search procedures. Moreover, there is no adequate conceptual framework within which the various ad hoc search strategies proposed to date can be compared. This paper describes how heuristic information from the problem domain can be incorporated into a formal mathematical theory of graph searching and demonstrates an optimality property of a class of search strategies.

Published in: IEEE Transactions on Systems Science and Cybernetics (Volume: 4 , Issue: 2, July 1968)

Page(s): 100 - 107

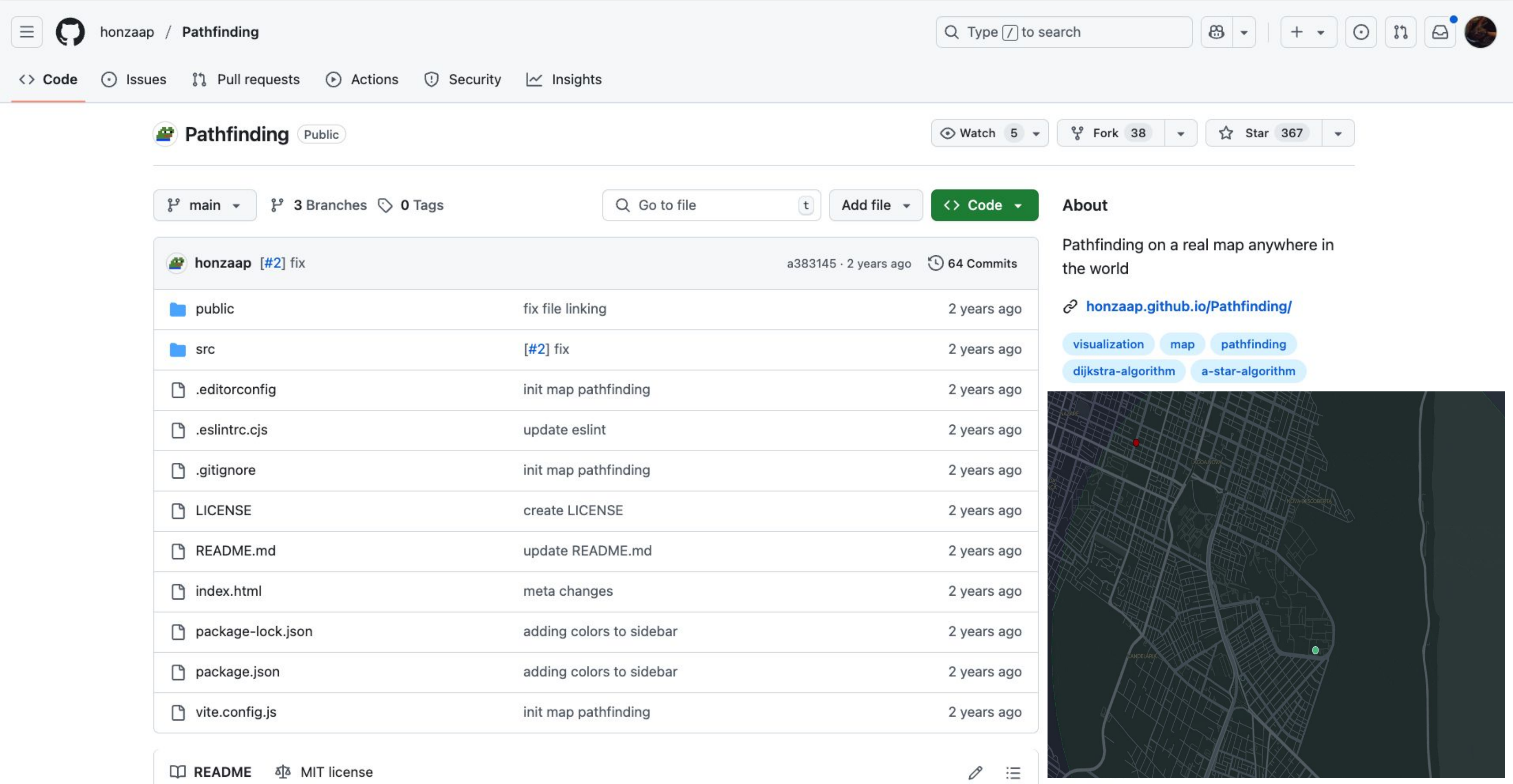
DOI: 10.1109/TSSC.1968.300136

Date of Publication: 31 July 1968

Publisher: IEEE

ISSN Information:

https://github.com/honzaap/Pathfinding/tree/main?tab=readme-ov-file



What is the A*?

Dijkstra's algorithm

Guarantees finding the cheapest known path so far.

Greedy search

Uses a **heuristic** to guide exploration towards the goal.

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

Heuristics

```
# Euclidean heuristic using OSMnx
heuristic_euclidean =
lambda u, v: ox.distance.euclidean(
    G.nodes[u]['y'], G.nodes[u]['x'],
    G.nodes[v]['y'], G.nodes[v]['x']
)
```

$$h(u,v) = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2}$$

Euclidean

For local maps, planar projections

```
# Great-circle heuristic using OSMnx
heuristic_great_circle =
lambda u, v: ox.distance.great_circle(
    G.nodes[u]['y'], G.nodes[u]['x'],
    G.nodes[v]['y'], G.nodes[v]['x']
)
```

$$a = \sin^2\left(\frac{\Delta \mathrm{Lat}}{2}\right) + \cos(\mathrm{Lat}_1)\cos(\mathrm{Lat}_2)\sin^2\left(\frac{\Delta \mathrm{Lng}}{2}\right)$$
$$c = 2 \cdot \arctan 2\left(\sqrt{a}, \sqrt{1-a}\right)$$
$$d = R \cdot c$$

Great Circle

For large geographical distances

```
# Manhattan heuristic approximation
heuristic_manhattan =
lambda u, v:
abs(G.nodes[u]['x'] - G.nodes[v]['x']) +
abs(G.nodes[u]['y'] - G.nodes[v]['y'])
```

$$h(u,v) = |x_u - x_v| + |y_u - y_v|$$

Manhattan

For grid-like street networks

Admissible Heuristics

A heuristic is admissible if it never overestimates the true cost to reach the goal.

$$h(n) \leq h^\star(n)$$

If $h(n) \leq h^*(n)$ for all n, then A* is guaranteed to find the optimal path.

What happens under the hood?

1. A* starts at the source node.

2. For each neighbor, it computes:

   g = actual cost to get there

   h= heuristic estimate to the goal

   f = g + h

3. It adds nodes to a **priority queue** (lowest f goes first).

4. It repeats this process until reaching the destination.

Start → [g + h] → expand node → update queue → repeat → Goal

```python
def A_star(graph, start, goal, heuristic):
    # Initialize the open set as a priority queue
    open_set = PriorityQueue()
    open_set.put(start, priority=0)

    # Initialize the cost from start to each node
    g_score = {node: float('inf') for node in graph.nodes}
    g_score[start] = 0

    # Initialize the estimated total cost (f = g + h)
    f_score = {node: float('inf') for node in graph.nodes}
    f_score[start] = heuristic(start, goal)

    # To reconstruct the path later
    came_from = {}

    ...
```

Algorithm Idea

```python
...

while not open_set.empty():
        # Select the node with the lowest f = g + h
        current = open_set.get()

        # If the goal is reached, reconstruct and return the path
        if current == goal:
            return reconstruct_path(came_from, current)

        # For each neighbor of the current node
        for neighbor in graph.neighbors(current):
            # Tentative g = current g + edge weight
            tentative_g = g_score[current] + graph.get_edge_weight(current, neighbor)

            # If this path is better than the previous best
            if tentative_g < g_score[neighbor]:
                # Record the best path so far
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g

                # Heuristic estimate to the goal
                h = heuristic(neighbor, goal)

                # Total estimated cost f = g + h
                f_score[neighbor] = tentative_g + h

                # Add neighbor to the open set with priority f
                open_set.put(neighbor, priority=f_score[neighbor])

    # If the goal was never reached
    return None
```

```python
def reconstruct_path(came_from, current):
    # Reconstructs the path from goal to start
    path = [current]
    while current in came_from:
        current = came_from[current]
        path.append(current)
    path.reverse()
    return path
```

A → B → C → D

```
came_from = {
      'B': 'A',
      'C': 'B',
      'D': 'C'
}
```

Algorithm Idea

Heuristic/Algorithm	Weight	Mean (ms)	Standard Deviation (ms)	Distance (m)	Travel Time (s)
euclidean	length	153.00	4.04	21697	1885
euclidean	travel_time	156.00	5.95	23830	1541
great_circle	**length**	**229.00**	**3.77**	**21697**	**1885**
great_circle	travel_time	19.80	3.72	24539	1940
manhattan (approx abs)	**length**	**146.00**	**1.03**	**21697**	**1885**
manhattan (approx abs)	travel_time	145.00	3.34	23830	1541
dijkstra	**length**	**149.00**	**6.61**	**21697**	**1885**
dijkstra	travel_time	135.00	4.31	23830	1541


A* (Length)
A* (Travel_Time)


Dijkstra (Length)
Dijsktra (Time)