

|  |
| --- |
| **RAJALAKSHMI INSTITUTE OF TECHNOLOGY** |
| (An Autonomous Institution, Affiliated to Anna University, Chennai) |

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**ACADEMIC YEAR 2025 - 2026**

**SEMESTER III**

**ARTIFICIAL INTELLIGENCE LABORATORY**

**MINI PROJECT REPORT**

|  |  |
| --- | --- |
| **REGISTER NUMBER** | 2117240070039 |
| **NAME** | BHAVATHARINI.S |
| **PROJECT TITLE** | IMPLEMENTATION AND ANALYSIS OF A SEARCH ALGORITHM FOR ROUTE OPTIMIZATION |
| **DATE OF SUBMISSION** | 30.9.2025 |
| **FACULTY IN-CHARGE** | **Mrs. M. Divya** |

**Signature of Faculty In-charge**

**INTRODUCTION**

* Artificial Intelligence (AI) enables machines to think, learn, and make decisions like humans.
* It applies data-driven algorithms to solve real-world problems efficiently.
* AI is widely used in automation, robotics, and intelligent navigation systems.
* Project Focus: Implement and analyze the A\* Search Algorithm for route optimization

**PROBLEM STATEMENT**

* Finding the shortest and most efficient route between two nodes while minimizing computation time.

**GOAL**

* To develop a system that uses heuristic-based search to achieve faster and optimal pathfinding compared to traditional methods like dijkstra algorithm

**THEORETICAL BACKGROUND**

* + The **A\*** Search Algorithm is a heuristic-based search technique that finds the least-cost path from a start node to a goal node. It uses the evaluation function:  
       
    where,

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

* + **g(n)** is the actual cost from the start node to the current node
  + **h(n)** is the estimated cost from the current node to the goal (heuristic).
  + Other algorithms like **Dijkstra’s algorithm** and **Greedy Best-First Search** are also used for route optimization, but A\* outperforms them by combining both path cost and heuristic intelligence.
  + **Justification:** A\* is chosen because it provides optimal and efficient solutions when using a good heuristic, making it ideal for route optimization and

**ALGORITHM EXPLANATION WITH EXAMPLE**

* **Explanation –**

**A\*** finds the **shortest path** using the formula f(n) = g(n) + h(n).

g(n) = actual cost so far, h(n) = estimated cost to the goal.

The node with the **lowest f(n)** is chosen at each step until the goal is reached.

It ensures an **optimal and efficient path** by combining cost and heuristic.

* **Example –**

Suppose we need to go from **A (Start)** to **D (Goal)**.

The paths and costs are:

* + A → B = 1, A → C = 3
  + B → D = 4, C → D = 2

Heuristic (h): B = 2, C = 1, D = 0

For each node:

* + B → f(B) = 1 + 2 = 3
  + C → f(C) = 3 + 1 = 4

A\* chooses **B first** (lowest f = 3), then goes to **D**.  
**Shortest Path:** A → B → D (Total Cost = 5)

**IMPLEMENTATION AND CODE**

import random

import math

import time

from queue import PriorityQueue

from collections import defaultdict

import matplotlib.pyplot as plt

N = 200 # number of nodes (increase for stronger effect)

k = 6 # connect each node to its k nearest neighbors

SEED = 42 # reproducible

random.seed(SEED)

# ---------- CREATE RANDOM 2D NODES ----------

nodes = [f"n{i}" for i in range(N)]

coords = {nodes[i]: (random.random()\*100, random.random()\*100) for i in range(N)}

def euclid(a, b):

(x1, y1), (x2, y2) = coords[a], coords[b]

return math.hypot(x1 - x2, y1 - y2)

# ---------- BUILD GRAPH: connect to k nearest neighbors ----------

graph = {n: {} for n in nodes}

for a in nodes:

dists = [(euclid(a, b), b) for b in nodes if a != b]

dists.sort(key=lambda x: x[0])

for dist, b in dists[:k]:

graph[a][b] = dist

graph[b][a] = dist # undirecte

def a\_star(graph, start, goal, heuristic):

open\_pq = PriorityQueue()

open\_pq.put((heuristic[start], start))

g = {n: float("inf") for n in graph}

g[start] = 0

came\_from = {}

closed = set()

expanded = 0

while not open\_pq.empty():

\_, current = open\_pq.get()

if current in closed:

continue

closed.add(current)

expanded += 1

if current == goal:

path = []

node = goal

while node in came\_from:

path.append(node)

node = came\_from[node]

path.append(start)

return list(reversed(path)), g[goal], expanded

for nbr, w in graph[current].items():

tentative = g[current] + w

if tentative < g[nbr]:

g[nbr] = tentative

came\_from[nbr] = current

f = tentative + heuristic[nbr]

open\_pq.put((f, nbr))

return None, float('inf'), expanded

def dijkstra(graph, start, goal):

pq = PriorityQueue()

pq.put((0, start))

g = {n: float("inf") for n in graph}

g[start] = 0.0 # Ensure this is a float

came\_from = {}

visited = set()

expanded = 0

while not pq.empty():

dist, current = pq.get()

if current in visited:

continue

visited.add(current)

expanded += 1

if current == goal:

# reconstruct path

path = []

node = goal

while node in came\_from:

path.append(node)

node = came\_from[node]

path.append(start)

return list(reversed(path)), g[goal], expanded

for nbr, w in graph[current].items():

tentative = g[current] + float(w) # ✅ make sure w is a float

if tentative < g[nbr]:

g[nbr] = tentative

came\_from[nbr] = current

pq.put((tentative, nbr))

return None, float('inf'), expanded

print("\nAvailable nodes:")

print(", ".join(nodes[:20]), "...") # Show first 20 nodes for preview

print("Total nodes:", len(nodes))

start = input("\nEnter START node (e.g., n0): ").strip()

goal = input("Enter GOAL node (e.g., n199): ").strip()

if start not in graph or goal not in graph:

print("\n⚠️ Invalid node name! Please choose valid node IDs like 'n0' to 'n199'.")

exit()

# HEURISTIC

heuristic = {n: euclid(n, goal) for n in nodes}

t0 = time.time()

path\_a, cost\_a, exp\_a = a\_star(graph, start, goal, heuristic)

t\_a = time.time() - t0

t0 = time.time()

path\_d, cost\_d, exp\_d = dijkstra(graph, start, goal)

t\_d = time.time() - t0

print("\n--- Algorithm Performance Comparison ---")

print(f"{'Algorithm':<12}{'Cost':<10}{'Time (ms)':<12}{'Nodes Expanded':<15}{'Path length'}")

print("-"\*70)

print(f"{'A\*':<12}{cost\_a:<10.3f}{t\_a\*1000:<12.3f}{exp\_a:<15}{len(path\_a) if path\_a else 'N/A'}")

print(f"{'Dijkstra':<12}{cost\_d:<10.3f}{t\_d\*1000:<12.3f}{exp\_d:<15}{len(path\_d) if path\_d else 'N/A'}")

if t\_a < t\_d:

print("\n✅ A\* was faster in time.")

else:

print("\n⚙️ Dijkstra was faster in time (try increasing N or k).")

if exp\_a < exp\_d:

print("✅ A\* expanded fewer nodes (better heuristic guidance).")

else:

print("⚙️ A\* expanded more nodes (weaker heuristic or small graph).")

# ---------- VISUALIZATION ----------

def draw\_graph\_and\_path(coords, graph, path, start\_node, goal\_node, title="A\* Path Visualization"):

plt.figure(figsize=(12, 8))

for u, neighbors in graph.items():

x1, y1 = coords[u]

for v in neighbors:

x2, y2 = coords[v]

plt.plot([x1, x2], [y1, y2], 'k-', alpha=0.1, linewidth=0.5)

all\_x = [coords[n][0] for n in coords]

all\_y = [coords[n][1] for n in coords]

plt.scatter(all\_x, all\_y, s=20, c='blue', label='All Nodes', zorder=2)

if path:

path\_x = [coords[n][0] for n in path]

path\_y = [coords[n][1] for n in path]

plt.plot(path\_x, path\_y, 'r-', linewidth=3, label='A\* Path', zorder=3)

start\_x, start\_y = coords[start\_node]

goal\_x, goal\_y = coords[goal\_node]

plt.scatter([start\_x], [start\_y], s=100, c='lime', edgecolors='black', label='Start', zorder=4)

plt.scatter([goal\_x], [goal\_y], s=100, marker='s', c='red', edgecolors='black', label='Goal', zorder=4)

plt.title(title + f"\nNodes: {N}, k: {k}, Cost: {cost\_a:.2f}")

plt.xlabel('X Coordinate')

plt.ylabel('Y Coordinate')

plt.legend()

plt.grid(True, linestyle=':', alpha=0.6)

plt.show()

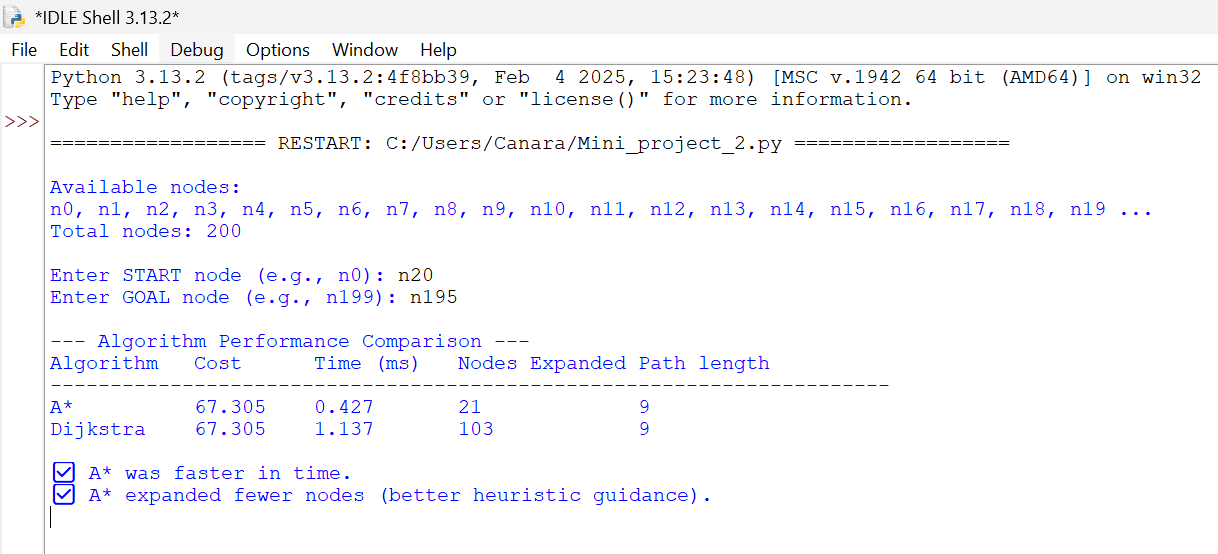
if path\_a:

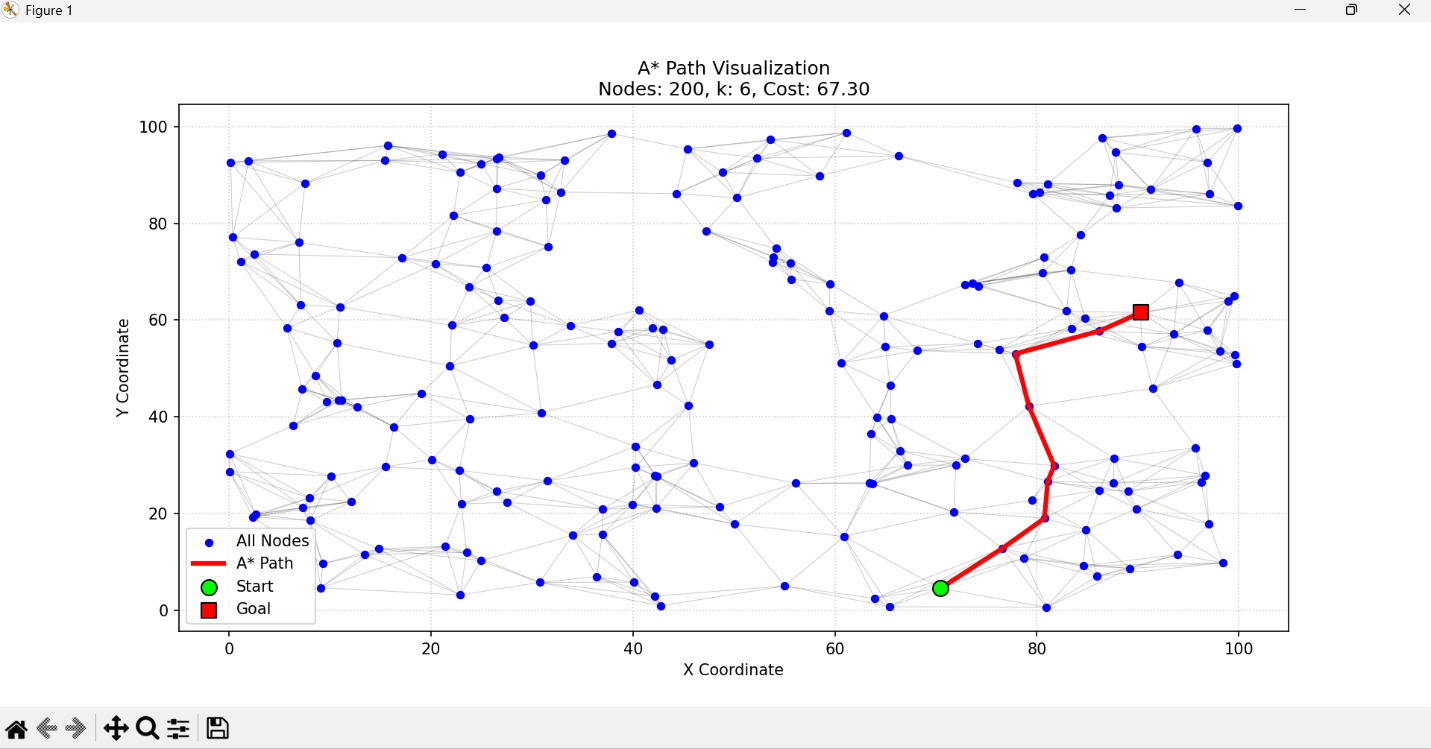
draw\_graph\_and\_path(coords, graph, path\_a, start, goal)

else:

print("\nCannot visualize: A\* failed to find a path.")

**OUTPUT**





**RESULTS AND FUTURE ENHANCEMENT**

* **Result-**

Implemented the A\* Search algorithm to find the most efficient route between selected nodes and compared its performance with Dijkstra’s algorithm

The output displays the optimal path, total cost, execution time , and visualization proving that A\* performs faster and expands fewer nodes due to its heuristic based search

* **Future Enhancement-**

Integrate real world map data (GPS/Google Maps API) for practical route optimization

Extend the model to handle dynamic traffic conditions and real time path updates.

|  |  |
| --- | --- |
| **Git Hub Link of the project and report** | [**https://github.com/sbhavatharini0110-web/Artificial-Intelligence-Mini-Project/blob/0da6b031034dbad2d388a002db8d7942689794e8/A\*\_alg\_code**](https://github.com/sbhavatharini0110-web/Artificial-Intelligence-Mini-Project/blob/0da6b031034dbad2d388a002db8d7942689794e8/A*_alg_code) |

**REFERENCES**

* Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th Ed.).
* GeeksforGeeks – A\* Search Algorithm Explanation.
* TutorialsPoint – Artificial Intelligence Search Techniques.
* Gate smashers – A\* search algorithm in AI (Youtube video)
* Towards Data Science - *Understanding A* Pathfinding and its Implementation in Python.