Project Report

on

Route Finding using Beam Search Algorithm

Submitted by

ADITYA PARMAR

25IMC13007

Under the guidance of

Mr. Kishan Ahuja

partial fulfilment for the award of the degree of MASTER'S IN ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Chandigarh University

October 2025

Certificate

This is to certify that **Aditya Parmar**, a student of **Master in Artificial Intelligence & Machin Learning (AI & ML)**, has successfully completed the **Minor Project** titled **"Route Finding using Beam Search Algorithm"** under the esteemed guidance of **Mr. Kishan Ahuja, Assistant Professor, University Institute of Computing (UIC), Chandigarh University**.

This project was undertaken as a part of the academic curriculum and is submitted in **partial fulfilment of the requirements** for the MAM program. The work presented in this project is a result of **group research, diligent effort, and dedication**, demonstrating the student's ability to apply theoretical knowledge to practical problem-solving.

The project **"Route Finding using Beam Search Algorithm"** is a web-based application designed to determine the most efficient route between two points using the Beam Search pathfinding algorithm. This project combines principles of **Advanced Internet Programming**, **full-stack web development**, and **real-time communication** technologies to deliver a responsive and interactive user experience.

I hereby confirm that this project is an **original work** carried out by the student and has **not been submitted elsewhere** for the award of any other degree, diploma, or certification.

Project Guide:
Mr. Kishan Ahuja
Assistant Professor
University Institute of Computing
Chandigarh University

Acknowledgement

I would like to express my sincere gratitude to **Chandigarh University** and the **University Institute of Computing (UIC)** for providing me with the opportunity to undertake this project, **"Route Finding using Beam Search Algorithm"**.

I extend my heartfelt appreciation to my esteemed mentor, **Mr. Kishan Ahuja, Assistant Professor**, for his invaluable guidance, continuous support, and insightful feedback throughout the project. His expertise in **Advanced Programming Language** played a crucial role in the successful completion of this project.

I am also grateful to my friends and peers for their encouragement and discussions, which helped refine my approach. Lastly, I thank my family for their unwavering support and motivation during this project.

This project has been an incredible learning experience, and I hope it serves as a foundation for further exploration in **Advanced Programming Language.**

Aditya Parmar

MCA – (AI & ML)

Chandigarh University

Table of Contents:-

Abstract

This mini-project presents an implementation and visualization of the **Beam Search Algorithm** for route finding using **Python and Pygame**. Beam Search is a heuristic-based optimization of the Breadth-First Search that reduces computational complexity by expanding only a limited number of promising nodes at each level, defined by a fixed beam width.

The primary objective of this project is to demonstrate how Beam Search can effectively balance *search efficiency* and *solution quality* in scenarios with large or complex search spaces. The algorithm uses heuristic values to evaluate the closeness of nodes to the goal, ensuring that only the most promising paths are explored further.

The project includes an interactive graphical interface built using Pygame, where users can visually select starting and goal nodes, observe the node expansion process, and view the final path discovered by the algorithm. Each step of the search process is animated to depict how Beam Search prioritizes nodes and prunes suboptimal paths.

This visual and interactive approach not only enhances the understanding of heuristic search techniques but also demonstrates how Artificial Intelligence concepts can be applied to real-world problems such as route planning, robot navigation, and decision-making systems. The project concludes that Beam Search, though approximate in nature, provides an efficient and practical solution for pathfinding tasks when computational resources or time constraints make exhaustive searches infeasible.

Introduction

Route finding is one of the most important applications of Artificial Intelligence (AI), where the goal is to determine an efficient and optimal path between two locations. Traditional search algorithms such as **Breadth-First Search (BFS)** or **Depth-First Search (DFS)** often become inefficient when dealing with large or complex search spaces, as they explore many unnecessary paths. To address this limitation, **heuristic-based algorithms** are used to make the search process faster and more focused.

Beam Search is one such heuristic search technique that combines the advantages of both informed search and limited exploration. Instead of exploring all possible nodes at each level, it keeps only a fixed number of the most promising nodes — known as the **beam width** — based on their heuristic values. This makes the algorithm much more efficient in terms of both time and memory.

In this mini-project, **Beam Search** has been implemented using **Python and Pygame** to visually demonstrate the process of route finding. The graphical interface allows users to select start and goal nodes, view node connections, and observe how the algorithm expands and evaluates paths. The visualization helps in understanding how Beam Search intelligently chooses routes based on heuristic guidance rather than exhaustive search.

This project not only shows the working of an informed search algorithm but also highlights its importance in real-world applications such as navigation systems, robotics, and game development, where fast and efficient decision-making is required.

Tools and Technologies Used

To develop and visualize the Beam Search route-finding project effectively, several programming tools and technologies were utilized. Each component played a vital role in ensuring smooth implementation, visualization, and user interaction.

Tool / Technoloy	Description / Purpose
Python 3.x	The core programming language used to implement the Beam Search algorithm and control the logic of the application. Python's simplicity and extensive libraries make it ideal for AI-based projects.
Pygame	A Python library used to create graphical interfaces and animations. It was used to visualize the nodes, edges, heuristic evaluations, and the search process dynamically.
Heapq Module	A built-in Python module used for efficient implementation of priority queues, which helps in maintaining the best possible paths during Beam Search.
Math Module	Provides mathematical functions (like distance calculations) to detect mouse clicks near node positions and manage graphical interactions.
Time Module	Used to control animation speed by adding small delays, making the visualization of node expansion smoother and more understandable.
Sys Module	Handles system-level operations like program termination and event management within the Pygame window.
Computer System (Hardware)	A basic computer with Python installed is sufficient to run the program. Recommended specifications include at least 4 GB RAM and a standard graphics processor for smooth Pygame rendering.

Source Code

```
import pygame, sys, math, time, heapq

# Initialize pygame
pygame.init()
WIDTH, HEIGHT = 700, 500
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Route Finding using Beam Search – Aditya Parmar")
font = pygame.font.SysFont("arial", 20, bold=True)
clock = pygame.time.Clock()

# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GREEN = (0, 200, 0)
RED = (220, 60, 60)
BLUE = (60, 130, 250)
GREY = (200, 200, 200)
YELLOW = (250, 230, 90)

# Graph positions (for drawing)
positions = {
'me': (100, 250),
'B': (250, 150),
'C': (250, 350),
'D': (400, 100),
'E': (400, 300),
'F': (550, 350),
'you': (600, 200)
}

# Graph connections with costs
graph = {
'me': [('B', 3), ('C', 6)],
'B': [('D', 4), ('E', 5)],
'C': [('F', 9)],
'D': [('you', 7)],
'E': [('you', 2)],
'F': [],
'you': []
}

# Heuristic values
heuristic = {
'me': 10, 'B': 8, 'C': 9, 'D': 5,
'E': 3, 'F': 7, 'you': 0
}

# Draw graph nodes and edges
def draw_graph(path=[], explored=set(), start=None, goal=None):
screen.fill(WHITE)

# Draw edges
```

- for node, edges in graph.items():
- for neighbor, cost in edges:
    - pygame.draw.line(screen, GREY, positions[node], positions[neighbor], 2)
    - midx = (positions[node][0] + positions[neighbor][0]) // 2
    - midy = (positions[node][1] + positions[neighbor][1]) // 2
    - text = font.render(str(cost), True, BLACK)
    - screen.blit(text, (midx, midy))

- # Draw current explored nodes
- for node in explored:
- pygame.draw.circle(screen, RED, positions[node], 25)

- # Draw final path
- if len(path) > 1:
- for i in range(len(path) - 1):
    - pygame.draw.line(screen, BLUE, positions[path[i]], positions[path[i + 1]], 4)

- # Draw all nodes (last so they appear on top)
- for node, pos in positions.items():
- if node in path:
    - color = GREEN
- elif node == start:
    - color = BLUE
- elif node == goal:
    - color = RED
- else:
    - color = YELLOW

- pygame.draw.circle(screen, color, pos, 25)
- pygame.draw.circle(screen, BLACK, pos, 25, 2)
- label = font.render(node, True, BLACK)
- screen.blit(label, (pos[0] - 8, pos[1] - 10))

- pygame.display.flip()

- # Beam Search algorithm (with animation)
- def beam_search(start, goal, beam_width):
- queue = [(heuristic[start], [start])]
- explored = set()

- while queue:
- new_paths = []

- # Visualize current queue (exploration)
- for _, path in queue:
    - draw_graph(path, explored, start, goal)
    - time.sleep(0.6)

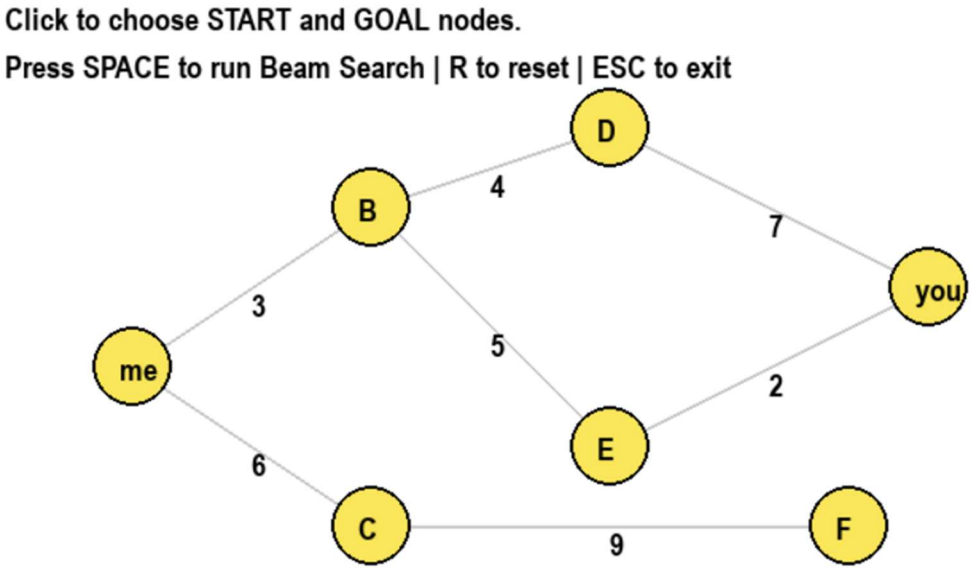
- for _, path in queue:
    - node = path[-1]
    - explored.add(node)
    - if node == goal:
    - return path

    - for neighbor, _ in graph[node]:
    - new_path = path + [neighbor]
    - h = heuristic[neighbor]
    - new_paths.append((h, new_path))

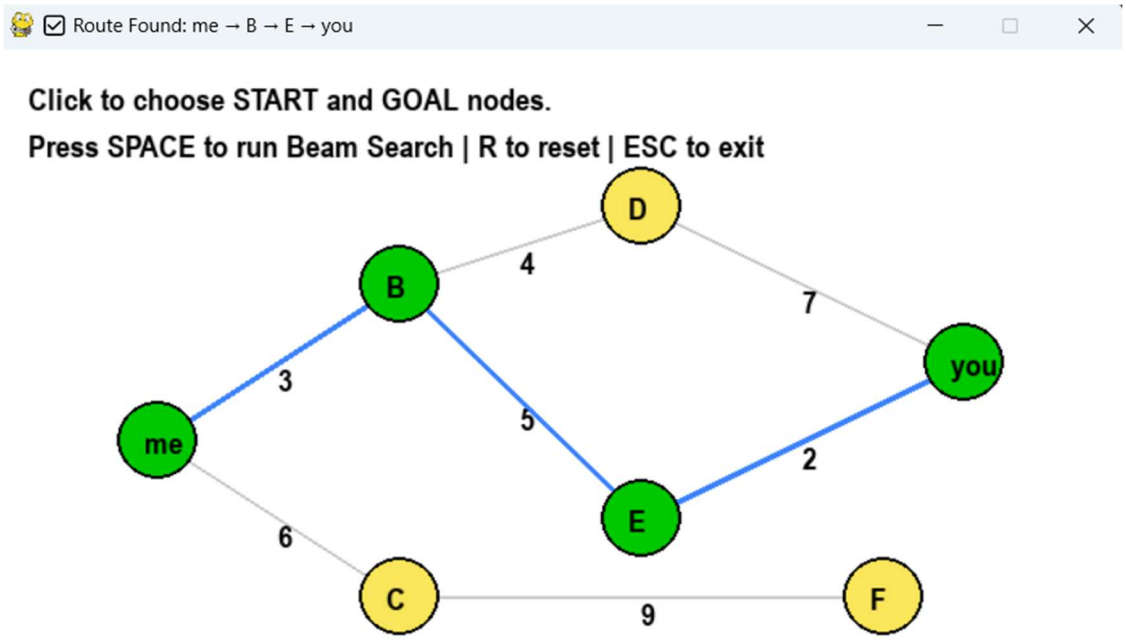
```
# Keep only best "beam_width" paths
queue = heapq.nsmallest(beam_width, new_paths, key=lambda x: x[0])

return None

# Main game loop
def main():
start, goal = None, None
route = []
beam_width = 2
running = True

while running:
draw_graph(route, start=start, goal=goal)

# Display info
info1 = font.render("Click to choose START and GOAL nodes.", True, BLACK)
info2 = font.render("Press SPACE to run Beam Search | R to reset | ESC to exit", True, BLACK)
screen.blit(info1, (20, 20))
screen.blit(info2, (20, 50))
pygame.display.flip()

for event in pygame.event.get():
    if event.type == pygame.QUIT:
    pygame.quit()
    sys.exit()

    if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_ESCAPE:
            pygame.quit()
            sys.exit()
    elif event.key == pygame.K_r:
            start, goal, route = None, None, []
            pygame.display.set_caption("Route Finding using Beam Search – Aditya Parmar")
    elif event.key == pygame.K_SPACE and start and goal:
            pygame.display.set_caption("Running Beam Search...")
            route = beam_search(start, goal, beam_width)
            if route:
            pygame.display.set_caption(f"✅ Route Found: {' → '.join(route)}")
            else:
            pygame.display.set_caption("❌ No Route Found")

    if event.type == pygame.MOUSEBUTTONDOWN:
    mx, my = pygame.mouse.get_pos()
    for node, pos in positions.items():
            if math.hypot(mx - pos[0], my - pos[1]) < 25:
            if not start:
            start = node
            pygame.display.set_caption(f"Start Node Selected: {start}")
            elif not goal:
            goal = node
            pygame.display.set_caption(f"Goal Node Selected: {goal}")

clock.tick(30)

# Run the game
if __name__ == "__main__":
main()
```

Result

i. First window.



ii. Second window show shortest path found from point “me” to “you”



Implementation Steps

The development of the "Route finding using Beam Search Algorithm" was carried out in the following essential steps:

Step 1: Initialize Pygame Environment

- Created a display window and defined colors, fonts, and screen dimensions.
- Used a dictionary to store node positions for visualization.

Step 2: Define the Graph

- The graph is represented using adjacency lists.
- Each node has edges connecting to neighbors with associated path costs.

Step 3: Define Heuristic Function

- Heuristic values estimate the cost from each node to the goal.
- These values guide the algorithm toward promising routes.

Step 4: Beam Search Algorithm

- Start from the initial node and generate all possible paths.
- Sort paths based on heuristic values and retain only the top *k* (beam width).
- Expand nodes level by level until the goal node is reached.
- Visual feedback shows node exploration and path selection.

Step 5: Visualization

- Nodes, edges, and path costs are drawn using Pygame.
- The interface allows users to select start and goal nodes by mouse clicks.
- Pressing the *SPACEBAR* triggers the search process.
- Pressing *R* resets the graph, and *ESC* exits the program.

Step 6: Display Result

- Once the route is found, the complete path is displayed with connecting lines.
- The title bar updates dynamically to show the found path or an error message.

Conclusion

This mini-project successfully demonstrates the working and effectiveness of the **Beam Search algorithm** in solving route-finding problems. By integrating heuristic-based search logic with an interactive **Pygame** interface, the project provides both practical functionality and visual understanding of how informed search algorithms operate.

The algorithm efficiently balances exploration and computation by expanding only the most promising nodes based on their heuristic values, rather than exploring the entire search space. This makes Beam Search suitable for large and complex problems where exhaustive algorithms like BFS or DFS would be inefficient.

Through visualization, users can clearly observe how the algorithm selects, evaluates, and prunes paths to reach the destination. The project highlights the real-world applicability of AI search techniques in areas such as **navigation systems, robotics, and game development**.

In conclusion, the project not only achieves its goal of implementing a route-finding system using Beam Search but also serves as an excellent educational tool to understand heuristic-based AI algorithms in an intuitive and engaging way.

References

- GeeksforGeeks. (n.d.). “Beam Search Algorithm in AI.” Retrieved from https://www.geeksforgeeks.org/beam-search-algorithm/
- Pygame Documentation: https://www.pygame.org/docs/
- Medium Article: “Understanding Beam Search Algorithm in Artificial Intelligence.
- TutorialsPoint. (n.d.). “Informed Search Algorithms in AI.”
- Stack Overflow discussions on implementing heuristic-based search algorithms.