A

# Artificial Intelligence and Machine Learning

# Lab Project

on

# Implementation of AIML Algorithms

Submitted to Manipal University, Jaipur

Towards the partial fulfillment of the Award of the Degree of

**BACHELOR OF TECHNOLOGY**

In Information Technology

2024-2025

By

Lakshya Pawar
229302177
IT-C

MANIPAL UNIVERSITY JAIPUR

Under the guidance of

Dr. Venkatesh Gauri Shankar

**Department of Information Technology**
**School of Computer Science and Engineering**
**Manipal University Jaipur**
**Jaipur, Rajasthan**

# 1. Introduction

Artificial Intelligence (AI) and Machine Learning (ML) are pivotal in solving complex real-world problems, ranging from pathfinding to decision-making systems. This project implements five AI algorithms: A* Search for the Water Jug and 8-Puzzle problems, Find-S for concept learning, BFS and DFS for the Traveling Salesman Problem (TSP), and a Tic-Tac-Toe game with an AI opponent. These implementations demonstrate how AI can optimize solutions in logistics, gaming, and learning systems, addressing challenges like resource allocation, strategic planning, and pattern recognition. The scope includes developing interactive applications using Streamlit for four projects and Tkinter for Tic-Tac-Toe, providing visual and user-friendly interfaces to showcase algorithmic efficiency in real-time scenarios.

# 2. Problem Definition and Methodology as Proposed Solution

## Problem Definition

The project addresses the following problems:

1. **Water Jug Problem (A* Search)**: Given two jugs of capacities (m) and (n) liters, achieve a specific amount (d) liters in one jug using operations like fill, empty, or pour.
2. **8-Puzzle Problem (A* Search)**: Rearrange a 3x3 grid from an initial configuration to a goal configuration by sliding tiles, with one empty space.
3. **Find-S Algorithm**: Learn a maximally specific hypothesis from positive and negative training examples for concept generalization.
4. **Traveling Salesman Problem (BFS and DFS)**: Find the shortest path visiting all cities exactly once and returning to the starting city.
5. **Tic-Tac-Toe (Minimax-like AI)**: Develop an interactive game where a player competes against a computer that makes strategic moves.

## Proposed Solution and Methodology

- **A* Search Water Jug and 8-Puzzle*:*** Uses a heuristic-based search to find the optimal path. For the Water Jug, the heuristic is the minimum difference between the current jug amounts and the goal. For the 8-Puzzle, the Manhattan distance guides tile movements. Steps include initializing the state, generating successors, and selecting the path with the lowest cost (f = g + h).
- **Find-S Algorithm**: Iteratively refines a hypothesis by generalizing attributes from positive examples, replacing conflicting attributes with '?'.

- **BFS and DFS for TSP**: BFS explores all possible paths level-by-level to find the shortest cycle, while DFS uses backtracking to explore paths depth-first. Both return the minimum-cost Hamiltonian cycle.
- **Tic-Tac-Toe AI**: The computer evaluates the board to win, block, or make random moves, ensuring a challenging opponent.

---

# 3. Implementation

### File Name: a_star_water_jug.py

```
import streamlit as st
import heapq
import time
import matplotlib.pyplot as plt

class WaterJugSolver:
    def __init__(self, jug1, jug2, goal):
        self.jug1 = jug1
        self.jug2 = jug2
        self.goal = goal
        self.visited = set()

    def is_goal(self, state):
        return self.goal in state

    def get_heuristic(self, state):
        return min(abs(state[0] - self.goal), abs(state[1] -
self.goal))

    def get_successors(self, state):
        successors = []
        a, b = state
        j1, j2 = self.jug1, self.jug2

        successors.append(((j1, b), "Fill Jug 1"))
        successors.append(((a, j2), "Fill Jug 2"))
        successors.append(((0, b), "Empty Jug 1"))
        successors.append(((a, 0), "Empty Jug 2"))
        transfer = min(a, j2 - b)
        successors.append(((a - transfer, b + transfer), "Pour Jug 1 →
Jug 2"))
        transfer = min(b, j1 - a)
```

```
        successors.append(((a + transfer, b - transfer), "Pour Jug 2 →
Jug 1"))

        return successors

    def solve(self):
        heap = []
        heapq.heappush(heap, (0, 0, (0, 0), []))
        while heap:
            f, g, current, path = heapq.heappop(heap)
            if self.is_goal(current):
                return path + [(current, "Goal Reached")]
            if current in self.visited:
                continue
            self.visited.add(current)
            for neighbor, action in self.get_successors(current):
                if neighbor not in self.visited:
                    new_path = path + [(current, action)]
                    new_g = g + 1
                    new_f = new_g + self.get_heuristic(neighbor)
                    heapq.heappush(heap, (new_f, new_g, neighbor,
new_path))
        return None

# Visualize the water levels
def draw_jugs(jug1_val, jug2_val, jug1_cap, jug2_cap):
    fig, ax = plt.subplots(figsize=(2, 3))
    ax.set_xlim(0, 3)
    ax.set_ylim(0, max(jug1_cap, jug2_cap) + 1)

    # Jug 1
    ax.add_patch(plt.Rectangle((0.5, 0), 0.8, jug1_cap, fill=False,
edgecolor="black", linewidth=1))
    ax.add_patch(plt.Rectangle((0.5, 0), 0.8, jug1_val,
color="#00aaff"))
    ax.text(0.9, jug1_cap + 0.5, "Jug 1", ha="center", fontsize=12)
    ax.text(0.9, -1, f"{jug1_val}/{jug1_cap}", ha="center",
fontsize=10, color="blue")

    # Jug 2
    ax.add_patch(plt.Rectangle((1.7, 0), 0.8, jug2_cap, fill=False,
edgecolor="black", linewidth=1))
```

```
    ax.add_patch(plt.Rectangle((1.7, 0), 0.8, jug2_val,
color="#00aaff"))
    ax.text(2.1, jug2_cap + 0.5, "Jug 2", ha="center", fontsize=12)
    ax.text(2.1, -1, f"{jug2_val}/{jug2_cap}", ha="center",
fontsize=10, color="blue")

    ax.axis("off")
    st.pyplot(fig)

# Streamlit UI
st.set_page_config(page_title="Water Jug A* Visualizer",
layout="centered")
st.title("Water Jug Problem - A* Algorithm")

with st.form("jug_form"):
    jug1 = st.number_input("Enter Jug 1 Capacity:", min_value=1,
value=4)
    jug2 = st.number_input("Enter Jug 2 Capacity:", min_value=1,
value=3)
    goal = st.number_input("Enter Goal Amount:", min_value=1, value=2)
    start = st.form_submit_button("Solve & Visualize")

if start:
    solver = WaterJugSolver(jug1, jug2, goal)
    solution = solver.solve()

    if not solution:
        st.error("❌ No solution found.")
    else:
        st.success("✅ Solution found!")
        placeholder = st.empty()
        for i, (state, action) in enumerate(solution):
            with placeholder.container():
                st.subheader(f"Step {i+1}: {action}")
                draw_jugs(state[0], state[1], jug1, jug2)
                st.markdown("---")
            time.sleep(1.2)
```

**File Name: a_star_8puzzle.py**

```
import streamlit as st
import heapq
import time
```

```python
SIZE = 3

def manhattan(board, goal):
    distance = 0
    for i in range(SIZE):
        for j in range(SIZE):
            val = board[i][j]
            if val != 0:
                # Find the position of val in the goal state
                for gi in range(SIZE):
                    for gj in range(SIZE):
                        if goal[gi][gj] == val:
                            distance += abs(i - gi) + abs(j - gj)
                            break
    return distance

def board_to_tuple(board):
    return tuple(tuple(row) for row in board)

def find_blank(board):
    for i in range(SIZE):
        for j in range(SIZE):
            if board[i][j] == 0:
                return i, j

def valid_moves(i, j):
    moves = []
    if i > 0: moves.append(('U', i - 1, j))
    if i < SIZE - 1: moves.append(('D', i + 1, j))
    if j > 0: moves.append(('L', i, j - 1))
    if j < SIZE - 1: moves.append(('R', i, j + 1))
    return moves

def apply_move(board, move):
    i, j = find_blank(board)
    direction, new_i, new_j = move
    new_board = [row[:] for row in board]
    new_board[i][j], new_board[new_i][new_j] = new_board[new_i][new_j],
new_board[i][j]
    return new_board

def a_star(start, goal):
    visited = set()
```

```
    heap = []
    heapq.heappush(heap, (manhattan(start, goal), 0, start, []))
    while heap:
        f, g, current, path = heapq.heappop(heap)
        if current == goal:
            return path + [current]
        visited.add(board_to_tuple(current))
        i, j = find_blank(current)
        for move in valid_moves(i, j):
            new_board = apply_move(current, move)
            if board_to_tuple(new_board) not in visited:
                heapq.heappush(heap, (g + 1 + manhattan(new_board,
goal), g + 1, new_board, path + [current]))
    return []

def draw_board(board):
    return "\n".join([" ".join([str(cell) if cell != 0 else ' ' for
cell in row]) for row in board])

def eight_puzzle_ui():
    st.set_page_config(page_title="8-Puzzle Solver", layout="centered")
    st.title("🧩 8 Puzzle Solver - A* Algorithm")
    st.markdown("Enter the initial and final board configurations (0 =
blank):")

    default_initial = "1 2 0\n3 4 5\n6 7 8"
    input_initial = st.text_area("Initial Puzzle Configuration",
default_initial, height=100)

    default_goal = "1 2 3\n4 5 6\n7 8 0"
    input_goal = st.text_area("Final (Goal) Puzzle Configuration",
default_goal, height=100)

    if st.button("Solve"):
        try:
            # Check initial state
            start = [[int(x) for x in row.strip().split()] for row in
input_initial.strip().split("\n")]
            if len(start) != SIZE or any(len(row) != SIZE for row in
start):
                raise ValueError("Initial state must be a 3x3 grid.")

            # Check goal state
```

```
            goal = [[int(x) for x in row.strip().split()] for row in
input_goal.strip().split("\n")]
            if len(goal) != SIZE or any(len(row) != SIZE for row in
goal):
                raise ValueError("Goal state must be a 3x3 grid.")

            # Check that both states contain numbers 0-8 exactly once
            start_nums = sorted([num for row in start for num in row])
            goal_nums = sorted([num for row in goal for num in row])
            if start_nums != [0, 1, 2, 3, 4, 5, 6, 7, 8] or goal_nums
!= [0, 1, 2, 3, 4, 5, 6, 7, 8]:
                raise ValueError("Both initial and goal states must
contain exactly one of each number 0-8.")

            st.write("Solving...")
            solution_path = a_star(start, goal)
            if not solution_path:
                st.error("No solution found!")
            else:
                st.success(f"Solution found in {len(solution_path)-1}
moves!")
                placeholder = st.empty()
                for step in solution_path:
                    placeholder.code(draw_board(step))
                    time.sleep(0.5)
        except Exception as e:
            st.error(f"Invalid input! Please use a 3x3 grid with
numbers 0-8.\n\nError: {e}")

if __name__ == "__main__":
    eight_puzzle_ui()
```

### File Name: find_s.py

```
import streamlit as st

def find_s(examples):
    hypothesis = ['0'] * len(examples[0][0])
    for example, label in examples:
        if label == 1:
            for i in range(len(hypothesis)):
                if hypothesis[i] == '0':
                    hypothesis[i] = example[i]
                elif hypothesis[i] != example[i]:
```

```
                    hypothesis[i] = '?'
    return hypothesis

# Streamlit UI
st.title("🧠 Find-S Algorithm")

num_attributes = st.number_input("Number of attributes:", min_value=1,
step=1)
attribute_names = [st.text_input(f"Attribute {i+1}", value=f"Attribute
{i+1}") for i in range(num_attributes)]

st.subheader("Add Examples")
if 'examples' not in st.session_state:
    st.session_state.examples = []

with st.form("example_form"):
    example_values = [st.text_input(f"{attribute_names[i]}",
key=f"val_{i}") for i in range(num_attributes)]
    label = st.selectbox("Label", [1, 0], format_func=lambda x:
"Positive" if x == 1 else "Negative")
    if st.form_submit_button("Add Example"):
        if all(value.strip() for value in example_values):
            st.session_state.examples.append((example_values, label))
            st.success("Example added!")
        else:
            st.error("Fill all fields.")

if st.session_state.examples:
    st.subheader("Examples")
    for i, (example, label) in enumerate(st.session_state.examples):
        st.write(f"Example {i+1}: {example}, Label: {'Positive' if
label == 1 else 'Negative'}")

if st.button("Compute Hypothesis"):
    if st.session_state.examples:
        hypothesis = find_s(st.session_state.examples)
        st.subheader("Hypothesis")
        st.write({attribute_names[i]: hypothesis[i] for i in
range(len(hypothesis))})
    else:
        st.warning("Add at least one example.")
```

**File Name: bfs_dfs_tsp.py**

```
import streamlit as st
from collections import deque

def tsp_dfs(graph, start):
    n = len(graph)
    visited = [False] * n
    min_cost = float('inf')
    best_path = []

    def dfs(curr, count, cost, path):
        nonlocal min_cost, best_path
        if count == n and graph[curr][start]:
            total_cost = cost + graph[curr][start]
            if total_cost < min_cost:
                min_cost = total_cost
                best_path = path + [start]
            return

        for i in range(n):
            if not visited[i] and graph[curr][i]:
                visited[i] = True
                dfs(i, count + 1, cost + graph[curr][i], path + [i])
                visited[i] = False

    visited[start] = True
    dfs(start, 1, 0, [start])
    return min_cost, best_path

def tsp_bfs(graph, start):
    n = len(graph)
    queue = deque()
    min_cost = float('inf')
    best_path = []

    queue.append((start, [start], 0))

    while queue:
        node, path, cost = queue.popleft()

        if len(path) == n and graph[node][start] != 0:
            total_cost = cost + graph[node][start]
            if total_cost < min_cost:
```

```
                min_cost = total_cost
                best_path = path + [start]
            continue

        for i in range(n):
            if i not in path and graph[node][i] != 0:
                queue.append((i, path + [i], cost + graph[node][i]))

    return min_cost, best_path

# ---------- Streamlit UI -----------

st.title("🧭 TSP using BFS and DFS")
algo = st.radio("Choose Algorithm", ["DFS", "BFS"])
n = 4

prefill = [
    "0 10 15 20",
    "10 0 35 25",
    "15 35 0 30",
    "20 25 30 0"
]

st.write("### Distance Matrix:")
matrix_input = []
valid_input = True

for i in range(n):
    row = st.text_input(f"Row {i+1}:", value=prefill[i])
    try:
        values = list(map(int, row.strip().split()))
        if len(values) == n:
            matrix_input.append(values)
        else:
            valid_input = False
            st.error(f"Row {i+1} must have exactly {n} numbers.")
    except ValueError:
        valid_input = False
        st.error(f"Invalid values in Row {i+1}.")

if valid_input and st.button("Solve"):
    if algo == "DFS":
        cost, path = tsp_dfs(matrix_input, 0)
```

```
    else:
        cost, path = tsp_bfs(matrix_input, 0)

    if cost == float('inf'):
        st.error("No valid path found.")
    else:
        st.success(f"✅ Minimum Cost: {cost}")
        st.write("📍 Optimal Path:", " → ".join(map(str, path)))
```

### File Name: tic_tac_toe.py

```
import tkinter as tk
from tkinter import messagebox
import random

# --- Config ---
root = tk.Tk()
root.title("🎮 Tic Tac Toe - You vs Computer")
root.resizable(False, False)
root.configure(bg="#ffe6f0")

COLORS = {
    "font": ("Comic Sans MS", 22, "bold"),
    "x": "#0077b6", # Player
    "o": "#d62828", # Computer
    "button_bg": "#fff8dc",
    "active": "#fddde6",
    "title": "#ff6f61",
    "restart": "#8ecae6",
}

board = [["" for _ in range(3)] for _ in range(3)]
buttons = [[None]*3 for _ in range(3)]

def check_win(p):
    return any(
        all(board[i][j] == p for j in range(3)) or
        all(board[j][i] == p for j in range(3)) for i in range(3)
    ) or all(board[i][i] == p for i in range(3)) or all(board[i][2 - i]
== p for i in range(3))

def check_draw():
    return all(cell for row in board for cell in row)
```

```
def end_game(msg):
    messagebox.showinfo("Game Over", msg)
    for row in buttons:
        for btn in row:
            btn.config(state="disabled")

def computer_move():

    for player in ["O", "X"]:
        for i in range(3):
            for j in range(3):
                if board[i][j] == "":
                    board[i][j] = player
                    if check_win(player):
                        board[i][j] = "O"
                        buttons[i][j].config(text="O", fg=COLORS["o"],
state="disabled")
                        return
                    board[i][j] = ""

    empty = [(i, j) for i in range(3) for j in range(3) if board[i][j]
== ""]
    if empty:
        i, j = random.choice(empty)
        board[i][j] = "O"
        buttons[i][j].config(text="O", fg=COLORS["o"],
state="disabled")

def on_click(i, j):
    if board[i][j] == "":
        board[i][j] = "X"
        buttons[i][j].config(text="X", fg=COLORS["x"],
state="disabled")
        if check_win("X"):
            end_game("🎉 You win!")
        elif check_draw():
            end_game("It's a draw!")
        else:
            root.after(300, computer_turn)

def computer_turn():
    computer_move()
    if check_win("O"):
```

```
        end_game("💻 Computer wins!")
    elif check_draw():
        end_game("It's a draw!")

def restart():
    for i in range(3):
        for j in range(3):
            board[i][j] = ""
            buttons[i][j].config(text="", state="normal",
bg=COLORS["button_bg"],
                                 activebackground=COLORS["active"])

# --- UI Setup ---
tk.Label(root, text="Tic Tac Toe", font=("Comic Sans MS", 26, "bold"),
         fg=COLORS["title"], bg="#ffe6f0").grid(row=0, column=0,
columnspan=3, pady=15)

for i in range(3):
    for j in range(3):
        btn = tk.Button(root, font=COLORS["font"], width=5, height=2,
                        bg=COLORS["button_bg"],
activebackground=COLORS["active"],
                        command=lambda i=i, j=j: on_click(i, j))
        btn.grid(row=i+1, column=j, padx=8, pady=8)
        buttons[i][j] = btn

tk.Button(root, text="🔁 Restart", font=("Arial", 14, "bold"),
          bg=COLORS["restart"], fg="white", activebackground="#219ebc",
          command=restart).grid(row=4, column=0, columnspan=3, pady=12)

root.mainloop()
```

---

# 4. Results

## Commands Used for Execution

**Streamlit Projects** (a_star_water_jug.py, a_star_8puzzle.py, find_s.py, bfs_dfs_tsp.py):

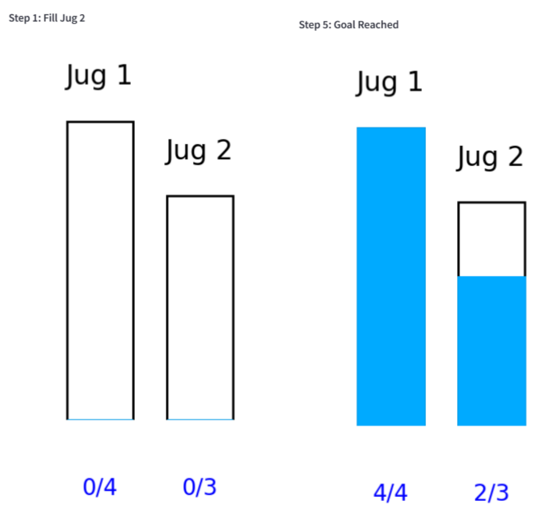
streamlit run <filename>.py

**Tkinter Project** (tic_tac_toe.py):

python tic_tac_toe.py

### Case 1: Water Jug Problem (A* Search)

**Description**: Solve for 2 liters in either jug with capacities of 4L and 3L, for example.

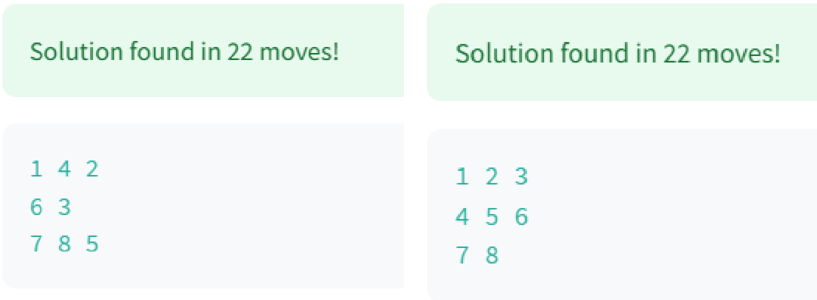




(*final state*)

*Shows steps like Fill Jug 1, Pour Jug 1 → Jug 2, achieving (2,0).*

### Case 2: 8-Puzzle Problem (A* Search)

**Description**: Move from initial state [1,2,0,3,4,5,6,7,8] to goal [1,2,3,4,5,6,7,8,0], for example.

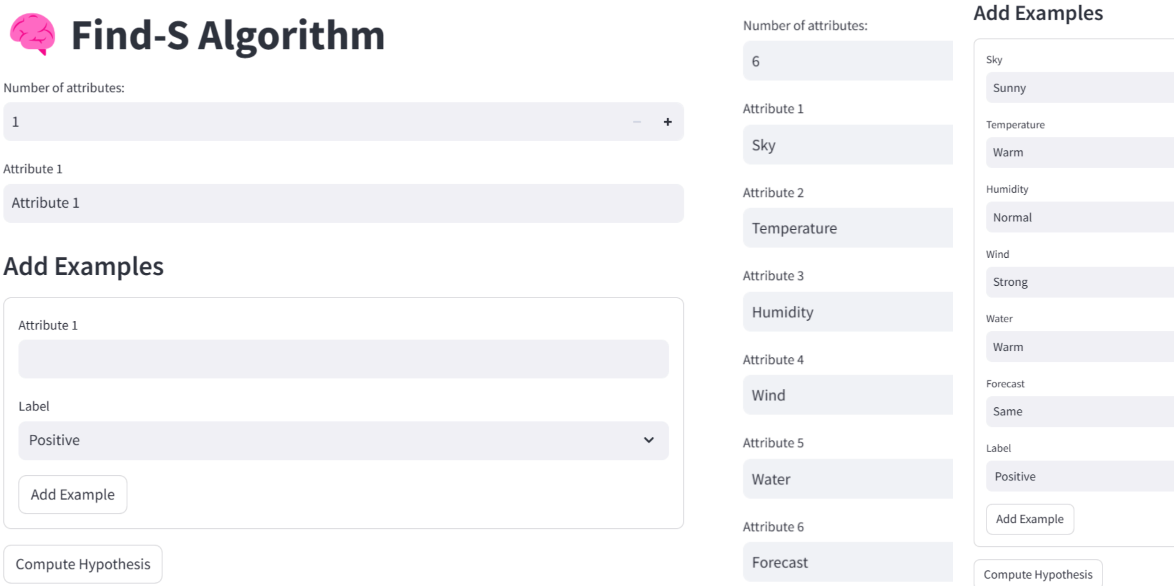


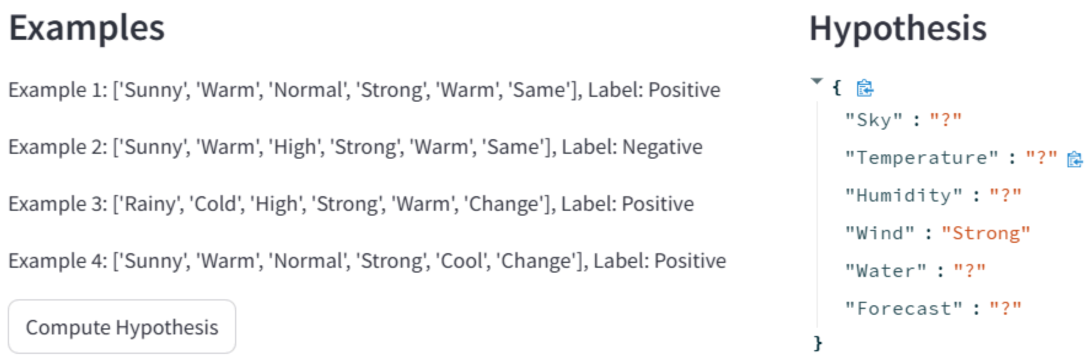

(*final state*)

*Displays tile movements to reach the goal configuration.*

## Case 3: Find-S Algorithm

**Description**: Learn hypothesis from examples (e.g., Sunny, Warm, Positive).

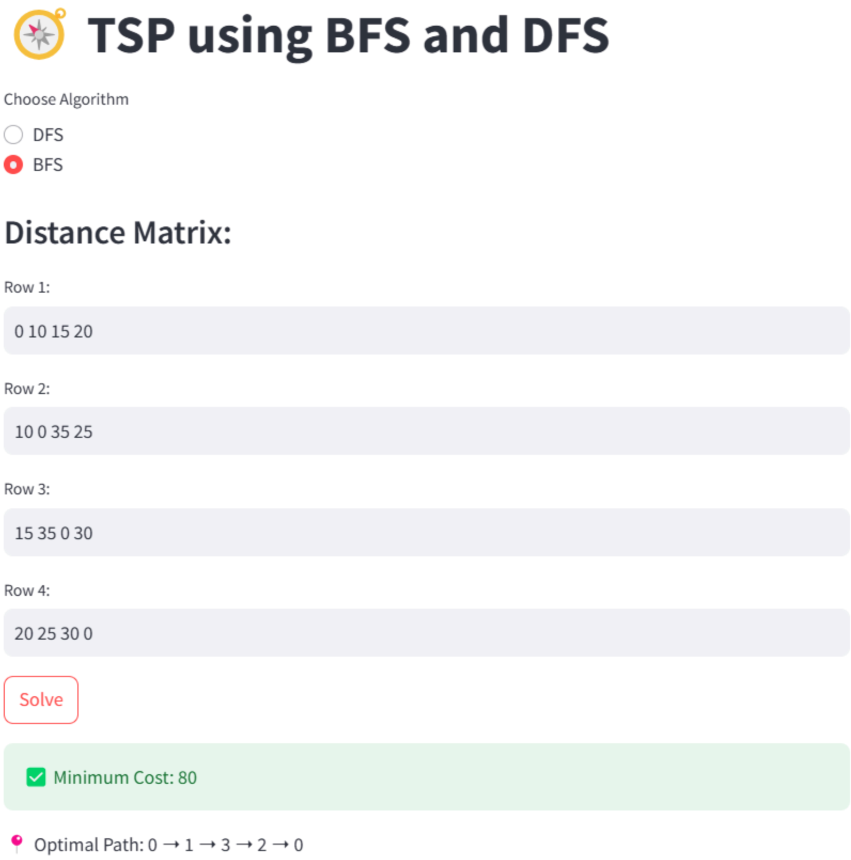



*Shows hypothesis like {Weather: Sunny, Temp: ?}.*

## Case 4: TSP (BFS/DFS)

**Description**: Find the shortest path for a 4-city graph with a given distance matrix, for example.




*Displays path (0→1→3→2→0) with cost 80.*

### Case 5: Tic-Tac-Toe

**Description**: Player wins by aligning three 'X's diagonally.

## 5. Conclusion and Future Work

This project successfully implemented five AI algorithms, demonstrating their applications in solving diverse problems like pathfinding, learning, and gaming. The A* algorithm efficiently solved the Water Jug and 8-Puzzle problems, Find-S generalized concepts, BFS/DFS optimized TSP routes, and the Tic-Tac-Toe AI provided a competitive experience. Streamlit and Tkinter enhanced user interaction, making the solutions accessible and visually engaging. Future work could include integrating more advanced heuristics for A*, implementing Minimax with alpha-beta pruning for Tic-Tac-Toe, and scaling TSP solutions for larger graphs using approximation algorithms.

## Bibliography/References


1. GeeksforGeeks: https://www.geeksforgeeks.org/
2. CodeChef: https://www.codechef.com/
3. Python Documentation: https://docs.python.org/
4. Corey Schafer YouTube Channel: https://www.youtube.com/@coreyms
5. Streamlit Documentation: https://docs.streamlit.io/
6. Tkinter Documentation: https://docs.python.org/3/library/tkinter.html
7. Stack Overflow: https://stackoverflow.com/
8. Grok, AI Assistant by xAI: https://x.ai/ (Guidance and Explanations for AIML Algorithms Project.)