**K. J. Somaiya College of Engineering, Mumbai-77**

| | |
|---|---|
| **Batch: B2** | **Roll No.: 16010122221** |
| **Experiment / assignment / tutorial No. 5** | |
| **Grade: AA / AB / BB / BC / CC / CD /DD** | |
| **Signature of the Staff In-charge with date** | |

**Title:** Implementation of informed search algorithm(Greedy Best First search/A*)

**Objective:** Examine the efficiency and performance of informed search algorithms in solving various problems

**Expected Outcome of Experiment:**

| Course Dutcome | er successful completion of the course students should be able to |
|---|---|
| 2 | alyse and solve problems for goal based agent architecture (searching and nning algorithms) |

**Books/ Journals/ Websites referred:**


1. **"Artificial Intelligence: a Modern Approach" by Russell and Norving, Pearson education Publications**
2. **"Artificial Intelligence" By Rich and knight, Tata Mcgraw Hill Publications**
3. **http://people.cs.pitt.edu/~milos/courses/cs2710/lectures/Class4.pdf**
4. **http://cs.williams.edu/~andrea/cs108/Lectures/InfSearch/infSearch.html**
5. **http://www.cs.mcgill.ca/~dprecup/courses/AI/Lectures/ai-lecture02.pdf http://homepage.cs.uiowa.edu/~hzhang/c145/notes/04a-search.pdf**
6. **http://wiki.answers.com/Q/Informed_search_techniques_and_uninformed_search_techniques**


**Pre Lab/ Prior Concepts:**

Problem solving, state-space trees, problem formulation, goal based agent architecture

**Historical Profile:**

**Problem-Solving Agent**

A problem-solving agent is designed to find solutions to well-defined problems. This agent typically follows these steps:

1. **Formulate the Problem**: Define the initial state, goal state, and possible actions.

2. **Search for a Solution**: Use an appropriate search strategy to explore the problem space.

3. **Execute the Solution**: Apply the sequence of actions derived from the search.

---

**New Concepts to be learned:**
Informed search.

---

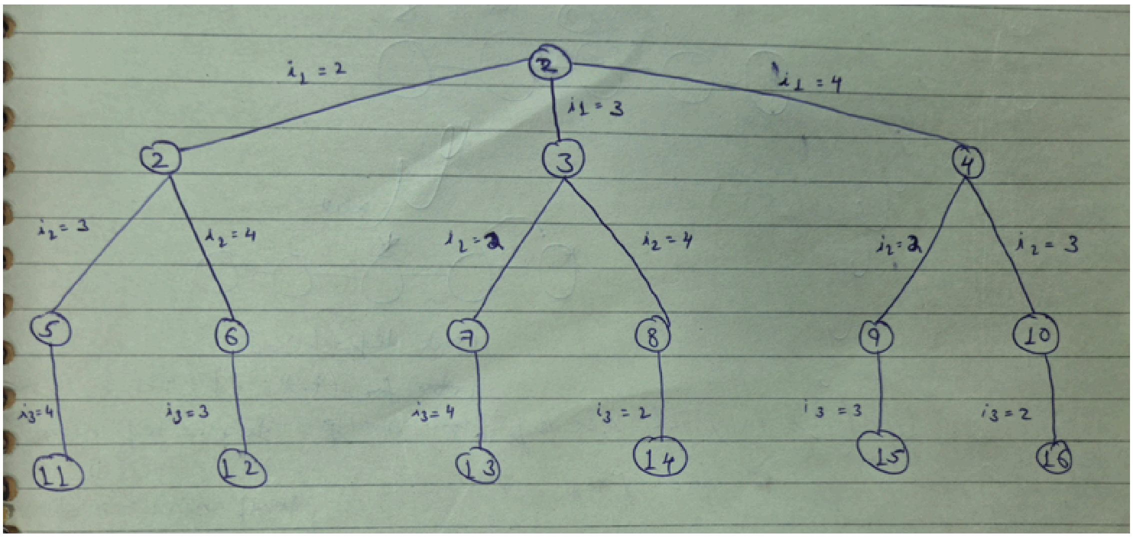
**Informed Searching Technique**
Greedy Best-First Search:

- Chooses the path that appears closest to the goal using a heuristic.
- Quick but might not find the optimal path.

***A** *** **Search:**

- Combines path cost and heuristic to find the best path.
- Ensures the optimal path if the heuristic is accurate.

**State-space tree :**



**Source code:**

```
import numpy as np

import heapq


class TSP:

    def __init__(self, num_cities, distances):

        self.num_cities = num_cities

        self.distances = distances


    def greedy_best_first_search(self):

        """

        Greedy Best-First Search implementation for the TSP.

        At each step, chooses the nearest unvisited city.

        """

        visited = [0]

        total_distance = 0


        while len(visited) < self.num_cities:

            current_city = visited[-1]

            min_distance = float('inf')

            next_city = None

```

```
            for city in range(self.num_cities):
                if city not in visited:
                                        if  self.distances[current_city][city]  <
min_distance:
                                                                min_distance  =
self.distances[current_city][city]
                        next_city = city

            visited.append(next_city)
            total_distance += min_distance

        total_distance += self.distances[visited[-1]][visited[0]]
        visited.append(visited[0])
        return visited, total_distance

    def a_star(self):
        """
        A* algorithm implementation for the TSP.
        Uses a priority queue based on f_score = g_score + heuristic
        where g_score is the actual distance traveled so far,
        and heuristic estimates remaining distance.
        """
        start_node = (0, [0], 0, 0)
        pq = [start_node]
```

```
        heapq.heapify(pq)


        best_solution = None

        best_distance = float('inf')


        while pq:

            f_score, path, current_city, g_score = heapq.heappop(pq)


            if len(path) == self.num_cities:

                total_distance = g_score + self.distances[current_city][path[0]]


                if total_distance < best_distance:

                    best_distance = total_distance

                    best_solution = path + [path[0]]


                return best_solution, best_distance


            for next_city in range(self.num_cities):

                if next_city not in path:

                    new_g_score = g_score + self.distances[current_city][next_city]


                    remaining_cities = [c for c in range(self.num_cities) if c not in path and c != next_city]
```

```
                    if remaining_cities:
                        min_remaining = min(self.distances[next_city][c] for c in remaining_cities)

                        min_return = self.distances[next_city][path[0]]
                        heuristic = min_remaining + min_return
                    else:
                        heuristic = self.distances[next_city][path[0]]

                    new_f_score = new_g_score + heuristic

                    # Add to priority queue
                    heapq.heappush(pq, (new_f_score, path + [next_city], next_city, new_g_score))

        return best_solution, best_distance

num_cities = 5
distances = np.array([
    [0,  10, 15, 20, 30],
    [10,  0, 35, 25, 40],
    [15, 35,  0, 30, 50],
    [20, 25, 30,  0, 15],
```

```
    [30, 40, 50, 15,  0]
])


tsp = TSP(num_cities, distances)


# For Greedy
route_greedy, distance_greedy = tsp.greedy_best_first_search()
print("Greedy Best-First Search:")
print("Route:", route_greedy)
print("Total Distance:", distance_greedy)


# For A*
route_a_star, distance_a_star = tsp.a_star()
print("\nA* Algorithm:")
print("Route:", route_a_star)
print("Total Distance:", distance_a_star)
```

**output screenshots:**

```
(base) PS C:\Users\aksha\Desktop\Sem-6\AI\AI_codes> python -u "c:\Users\
Greedy Best-First Search:
Route: [0, 1, 3, 4, 2, 0]
Total Distance: 115

A* Algorithm:
Route: [0, 1, 4, 3, 2, 0]
Total Distance: 110
(base) PS C:\Users\aksha\Desktop\Sem-6\AI\AI_codes>
```

**Comparison of performance of Greedy and A* Algorithm:**

| **Basis** | **Greedy** | **A*** |
|---|---|---|
| Completeness | No - might get stuck in loops. | Yes (unless there are infinitely many nodes with f≤f(G)) |
| Time Complexity | O(bm), but a good heuristic can give dramatic improvement. | Exponential |
| Space Complexity | O(bm) - keeps all nodes in memory. | Keeps all nodes in memory. |
| Optimality | May not be optimal. | Optimal |

**Properties of A* algorithm:**

1. Completeness: A* finds a solution if one exists in a finite search space.

2. Optimality: A* guarantees the shortest path if the heuristic is admissible and consistent.

3. Memory Efficiency: A* uses a priority queue to explore promising paths first.

4. Time Complexity: A* depends on heuristic quality but typically behaves like O(bd).

5. Heuristic Function: Guides search by estimating the cost to reach the goal.

6. Adaptability: Customizable for different problem domains.

7. Node Expansion Order: Expands nodes with lowest total cost (f-value).

8. Potential Pitfalls: Inefficiency or failure if heuristic isn't admissible or consistent, and high memory usage in large search spaces.

**Conclusion:**

The comparison between Greedy Best-First Search and A* highlights their distinct strengths and limitations in solving informed search problems. Greedy search, though faster with lower time complexity, sacrifices optimality and completeness, often getting stuck in loops due to its heuristic-driven approach. A*, on the other hand, ensures optimal solutions by combining both cost-so-far and heuristic evaluation but requires significant memory and computational resources, leading to exponential time complexity. This trade-off emphasizes the importance of choosing the appropriate algorithm based on problem constraints and requirements. For scenarios demanding accuracy, A* excels, while Greedy is ideal for rapid, suboptimal solutions in constrained environments.

**Post lab Objective questions**

1. **A heuristic is a way of trying**
   a. To discover something or an idea embedded in a program
   b. To search and measure how far a node in a search tree seems to be from a goal
   c. To compare two nodes in a search tree to see if one is better than the other
   **d.** Only (a) and (b)
   **e.** Only (a), (b) and (c).

**Answer:**

0. **A* algorithm is based on**
   a. Breadth-First-Search
   b. Depth-First –Search
   c. Best-First-Search
   **d.** Hill climbing.
   **e.** Bulkworld Problem.

**Answer:**

**3. What is a heuristic function?**

**a.** A function to solve mathematical problems
**b.** A function which takes parameters of type string and returns an integer value
**c.** A function whose return type is nothing
**d.** A function which returns an object
**e.** A function that maps from problem state descriptions to measures of desirability.

**Answer:**

**Post Lab Subjective Questions:**

1. **How does the Greedy Best-First Search algorithm use a heuristic evaluation function?**

   Best-first search algorithm supports heuristic evaluation function by using it to determine the "best" node to expand next. Instead of exploring all possible paths, it
   selects the node that is most likely to lead to the goal based on the heuristic evaluation
   function. This function provides an estimate of the cost from the current node to the
   goal, guiding the search towards the most promising paths.

2. **Find a good heuristic function for following**

   a. Monkey and Banana problem
   For the Monkey and Banana problem, a good heuristic function could be the Manhattan distance between the monkey and the banana. This heuristic estimates
   the number of moves required for the monkey to reach the banana by only considering the straight-line distance in each direction, ignoring any obstacles or
   walls.
   b. Traveling Salesman problem
   For the Traveling Salesman problem, a good heuristic function could be the minimum spanning tree (MST) heuristic. This heuristic calculates the minimum

spanning tree of the graph representing the cities and then adds the minimum edge
weight not in the MST to the total cost. Although not always optimal, it provides a
good estimate of the shortest tour length.

**3. Define the heuristic search. Discuss benefits and short comings.**

Heuristic search is a search algorithm that uses a heuristic evaluation function to guide the search towards the most promising paths. It estimates the cost from the current state
to the goal state and selects the next state to explore based on this estimation.

Benefits:

• Heuristic search can significantly reduce the search space by focusing on promising paths, leading to faster search times.

• It can be applied to large problem spaces where exhaustive search is not feasible.

• Heuristic search algorithms are often relatively simple to implement.

Shortcomings:

• The quality of the solution depends heavily on the quality of the heuristic function. A poorly designed heuristic can lead to suboptimal solutions or even failure to find a solution.

• Heuristic search algorithms are not guaranteed to find the optimal solution, as they may get stuck in local optima.

• It may be challenging to design a heuristic function that accurately estimates the true cost to reach the goal, especially in complex problem domains.