Batch: B1	Roll No.: 16010121045
Experiment / assignment / tutorial No.	

Title: Implementation of Informed search algorithm(GBFS/A*)

Objective: Comparison and analysis of informed search algorithms

Expected Outcome of Experiment:

Course Outcome	After successful completion of the course students should be able to
CO2	Analyse and solve problems for goal based agent architecture (searching and planning 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:

The AI researchers have come up many algorithms those operate on state space tree to give the result. Goal based agent architectures solve problems through searching or planning. Depending on availability of more information other than the problem statement decides if the solution can be obtained with uninformed search or informed search.

Its fact that not all search algorithms end up in giving the optimal solution. So, it states the need to have a better and methodological approach which guarantees optimal solution.

New Concepts to be learned: Heuristic, Informed search, greedy best first search, A* search

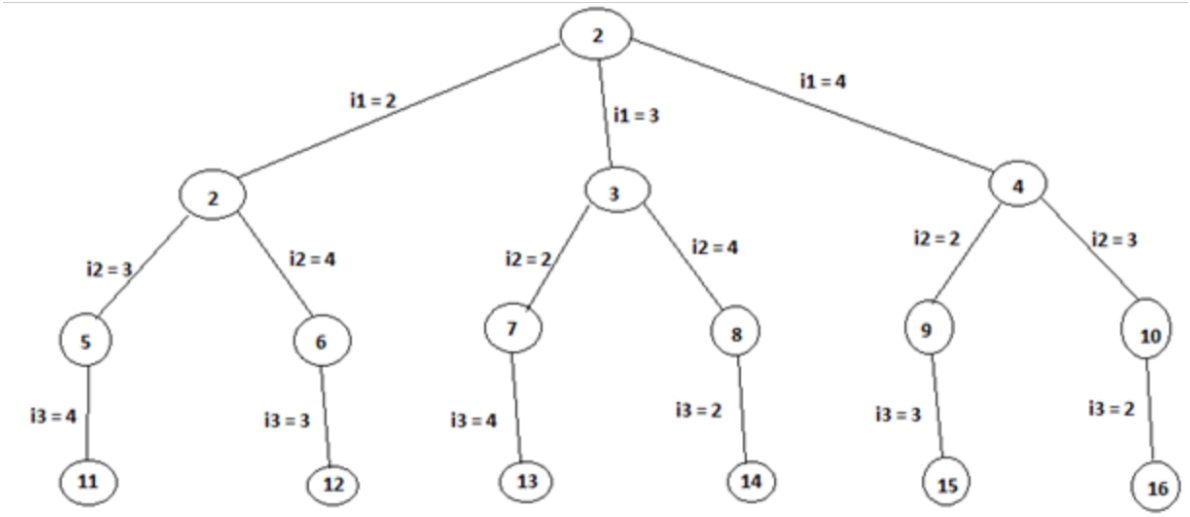
---

Informed searching techniques

- Greedy best first search
- A*

Chosen Problem statement: Traveling Salesman

State-space tree :



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):
        visited = [0]  # Start from city 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):
        start_node = (0, [0], 0)
```

```
        pq = [start_node]
        heapq.heapify(pq)

        while pq:
            total_distance, visited, current_city =
heapq.heappop(pq)

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

            for next_city in range(self.num_cities):
                if next_city not in visited:
                    next_visited = visited + [next_city]
                    next_distance = total_distance +
self.distances[current_city][next_city] +
self.distances[next_city][0]
                    heapq.heappush(pq, (next_distance,
next_visited, next_city))

num_cities = 4
distances = np.array([
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
])

tsp = TSP(num_cities, distances)

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

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

```
❯ python3 -u "/Users/pargatsinghdhanjal/Documents/College/Sem6/AI/Programs/tempCodeRunnerFile.py"
Greedy Best-First Search:
Route: [0, 1, 3, 2, 0]
Total Distance: 80

A* Algorithm:
Route: [0, 1, 3, 2, 0]
Total Distance: 125
```

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

Choose sidebar display

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.

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: e

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

Answer: c

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: e

Post Lab Subjective Questions:

1. **How best-first-search algorithm supports 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 the 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 shortcomings.

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.