Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Artificial Intelligence (DJS22DSC502)

AY: 2024-25

Experiment 9

DHRUV SHAH 60009220132 D1-1

(Planning)

Aim: Implement a plan using AO*.

Theory:

The Depth-first search and Breadth-first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph. The main difference lies in the way termination conditions are determined since all goals following an AND node must be realized; whereas a single goal node following an OR node will do. So for this purpose, we are using AO* algorithm. Like A* algorithm here we will use two arrays and one heuristic function.

OPEN: It contains the nodes that have been traversed but yet not been marked solvable or unsolvable.

CLOSE: It contains the nodes that have already been processed.

AO* Search Algorithm

Step 1: Place the starting node into OPEN.

Step 2: Compute the most promising solution tree say T0.

Step 3: Select a node n that is both on OPEN and a member of T0. Remove it from OPEN and place it in CLOSE

Step 4: If n is the terminal goal node then leveled n as solved and leveled all the ancestors of n as solved. If the starting node is marked as solved then success and exit.

Step 5: If n is not a solvable node, then mark n as unsolvable. If starting node is marked as unsolvable, then return failure and exit.

Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

Step 8: Exit.

Lab Assignment to do:

Consider the use case of a plan to travel from Mumbai to Goa to attend a wedding at Taj Aguada. The plan needs to be decided based on the cost. You can either travel by train or bus or flight and stay in a hotel near or far to the wedding venue. The three options of the venues are Westin, Kennel Worth and Maria Rica hotels. You can choose between a two days package for stay and meal together or separately. Other option for your travel and stay will be a vanity van. There you need to decide if you want to cook or eat outside.
Implement AO* to find the most suitable plan in terms of cost.

```
def table():
    travel_cost = {
        "bus": 2000,
        "train": 1000,
        "flight": 12000
    }
    stay_cost = {
        "westin": 15000,
        "kennel worth": 10000,
        "maria rica": 12000
    }
    meal_cost = {
        "with food": 2500,
        "without food": 4500
    }
    vanity_van = {
        "cost": 29000,
        "distance": 1000,
        "meal_options": {
            "cooked": 2000,
            "eat outside": 4500
        }
    }

    distance_costs = {
        "westin": 2000,
        "kennel worth": 3000,
        "maria rica": 7000
    }

    return {
        "Travel Costs": travel_cost,
        "Stay Costs": stay_cost,
        "Meal Costs": meal_cost,
        "Vanity Van": vanity_van,
        "Distance Costs": distance_costs
    }

print(table())
```

```
{'Travel Costs': {'bus': 2000, 'train': 1000, 'flight': 12000}, 'Stay Costs': {'westin': 15000, 'kennel worth': 10000, 'maria rica'
```

```
def ao_star():
    table_data = table()

    # Extracting table data
    travel_costs = table_data["Travel Costs"]
    stay_costs = table_data["Stay Costs"]
    meal_costs = table_data["Meal Costs"]
    vanity_van = table_data["Vanity Van"]
    distance_costs = table_data["Distance Costs"]

    paths = []

    # Generate all possible paths with travel, stay, and meal combinations
    for travel, travel_cost in travel_costs.items():
        for stay, stay_cost in stay_costs.items():
            for meal_option, meal_cost in meal_costs.items():
                total_cost = travel_cost + stay_cost + distance_costs[stay] + meal_cost
                paths.append((travel, stay, meal_option, total_cost))

    # Add paths for Vanity Van options
    for meal_option, meal_cost in vanity_van["meal_options"].items():
        total_cost = vanity_van["cost"] + vanity_van["distance"] + meal_cost
        paths.append(("Vanity Van", "Vanity Van", meal_option, total_cost))

    # Display table of options
    print(f"{'Travel Option':<15} {'Stay Option':<15} {'Meal Option':<15} {'Total Cost':<10}")
    print("=" * 55)
    for travel, stay, meal_option, total_cost in paths:
        print(f"{travel:<15} {stay:<15} {meal_option:<15} {total_cost:<10}")

    # Find the path with the minimum cost
    best_path = min(paths, key=lambda x: x[3])
    min_cost = best_path[3]

    return min_cost, best_path
```

```
print(ao_star())
```

```
Travel Option   Stay Option     Meal Option     Total Cost
=======================================================
bus             westin          with food       21500
bus             westin          without food    23500
bus             kennel worth    with food       17500
bus             kennel worth    without food    19500
bus             maria rica      with food       23500
bus             maria rica      without food    25500
train           westin          with food       20500
train           westin          without food    22500
train           kennel worth    with food       16500
train           kennel worth    without food    18500
train           maria rica      with food       22500
train           maria rica      without food    24500
flight          westin          with food       31500
flight          westin          without food    33500
flight          kennel worth    with food       27500
flight          kennel worth    without food    29500
flight          maria rica      with food       33500
flight          maria rica      without food    35500
Vanity Van      Vanity Van      cooked          32000
Vanity Van      Vanity Van      eat outside     34500
(16500, ('train', 'kennel worth', 'with food', 16500))
```

Start coding or generate with AI.