Project Report: Design of The AI Expert System FOODIE

Authors: Shuting Hu, Qi Wen
Department of Electrical and Computer Engineering

As part of the requirements for ECE 579

I. Introduction

To design the AI expert system FOODIE, there are several aspects that we need to consider, such as the routing optimization of the robots, the expert module for bagging food items, the expert system module for selecting the right beverage for guests, and the robot arm's loading process. We will need to develop rule bases with rules and implement them accordingly.

II. FOODIE

A. FOODIE_Route

For route optimization, a common algorithm used is the Traveling Salesman Problem (TSP), which aims to find the shortest path to visit all the delivery locations. However, this algorithm can be computationally expensive for large sets of locations.

For optimizing the routes that the robots take based on the orders received, we can use a variant of the well-known TSP called the Vehicle Routing Problem (VRP). In VRP, we aim to minimize the total distance traveled by a fleet of vehicles (robots, in our case) to serve a set of customers (delivery locations).

One approach to solving VRP is using a metaheuristic algorithm called Ant Colony Optimization (ACO). In ACO, artificial ants are used to find good solutions to optimization problems by following a pheromone trail. The pheromone trail is a chemical left by the ants that helps guide the other ants to good solutions.

In our case, we can implement an ACO algorithm to optimize the routes taken by the robots.

Assumptions:

1. There are a fixed number of robots available for deliveries.
2. Each robot has a maximum capacity that cannot be exceeded.
3. All delivery locations have a fixed demand that cannot be split between multiple robots.
4. Delivery locations are known in advance and will not change during the day.
5. The distance between each pair of delivery locations is known and can be calculated.
6. The delivery time for each location is not considered in this optimization.

Algorithm:

Ant Colony Optimization (ACO)

Methods:

1. Initialization: Randomly generate a set of initial solutions (i.e. robot routes).
2. Ant behavior: Each ant chooses a delivery location to visit based on the pheromone trail and a heuristic function that takes into account the distance and demand of the location.
3. Pheromone update: After all ants have completed their routes, the pheromone trail is updated to reinforce better solutions.
4. Local search: After each iteration, a local search algorithm is applied to improve the best solution found so far.
5. Termination: The algorithm terminates after a fixed number of iterations or when the improvement in the best solution is below a certain threshold.

Rules:

1. Each robot can only visit locations that are within its maximum capacity.
2. Each location can only be visited by one robot.
3. The total demand for each robot cannot exceed its maximum capacity.
4. The pheromone trail evaporates over time to prevent convergence to suboptimal solutions.

B. FOODIE_BAGGER

The FOODIE_BAGGER module is responsible for bagging the food items into the robot's compartment. It follows a rule-based system to decide where each item should go. The system has three steps: bagging large items, bagging medium items, and bagging small items. Frozen items are put in freezer bags, and other items are put in paper bags. The size of the paper bags is equal for all items but is only capable of accommodating a finite number of items depending on their sizes.

Assumptions:

1. FOODIE receives an order containing a list of food items.
2. FOODIE has access to bags and freezer bags.
3. The items in the order have different sizes and fragility levels.
4. FOODIE needs to bag the items in the order based on their size, frozen state, and fragility level while ensuring that no item gets crushed in the bag.
5. Properties of items:
    - **Size**: “Large”, “Medium”, or “Small”
    - **Frozen**: “Yes”, or “No”; if yes, can only go into a freezer bag
    - **Fragile**: “Yes”, or “No”; if yes, no items allowed to be put on top of it
6. Item size chart:

Item	Large	Medium	Small
Size	4 units	2 units	1 unit

7. Bag and compartment sizes
    a. The size of each bag, including the freezer bag, is 10 units.
    b. The compartment of each robot can hold up to 5 bags.

Algorithm:

1. Create a list of bags with their sizes.
2. Iterate over the items in the order and add them to the appropriate bag based on their size, frozen state, and fragility level, following the rules defined below.
3. If an item cannot fit in the current bag, start a new bag.

Methods:

1. We will use a list to represent the bags.
2. We will check if an item is frozen.
3. We will check if an item is fragile.
4. We will define a set of rules to decide where to put each item in the order.

Rules:

1. If the item is large, put it in a bag.
2. If the item is medium and the current bag can contain this item, put it in the current bag. Otherwise, start a new bag and put the item in it.
3. If the item is small and the current bag can contain this item, put it in the current bag. Otherwise, start a new small bag and put the item in it.
4. If the item is frozen, put it in a freezer bag.
5. If the item is fragile, no more item is allowed in the current bag. Start a new bag.

C. FOODIE_Springs_to_Action

FOODIE_Springs_to_Action handles the delivery of the food order to the customer. The module works by dispatching a robot to the customer's address with the order. The robot uses GPS to navigate to the customer's address and delivers the food order to the customer.

Assumptions:

1. The expert system will only focus on selecting the right beverage for unexpected guests.
2. The available beverage options are water, juice, wine, beer, and liquor.

3. Each beverage type has specific brands associated with it.
4. The system will use backward chaining to infer the right beverage based on the given rules and facts.

Algorithm:

1. Backward chaining: starts with a goal and recursively searches the rule base to find rules that match the goal until all necessary facts are found.

Methods:

1. Inference engine: to perform backward chaining and match rules with facts.
2. Knowledge base: to store the rules and facts.

Rules:

Rule 1: If the guest is allergic to citrus, then choose carrot juice.
Rule 2: If the guest is a health nut, then choose carrot juice.
Rule 3: If the entrée is beef, then choose red wine.
Rule 4: If the entrée is chicken, then choose white wine.
Rule 5: If the entrée is seafood, then choose white wine or champagne.
Rule 6: If it's a casual party, then choose beer.
Rule 7: If it's a formal party, then choose wine or liquor.
Rule 8: If the guest is well-liked, then choose expensive liquor.
Rule 9: If the guest is not well-liked, then choose cheap liquor.
Rule 10: If it's a hot day, then choose cold water or juice.
Rule 11: If it's a cold day, then choose hot tea or coffee.
Rule 12: If it's a special occasion, then choose champagne.
Rule 13: If the guest is from Mexico, then choose Dos Equis beer.
Rule 14: If the guest is from Poland, then choose Polish vodka.
Rule 15: If the guest is from Italy, then choose Chianti wine.

D. FOODIE_Bagger_STRIPS

To show how the robot would use its arm to load the bags into its compartment, we can use a simple STRIPS-like planning system. We can represent the state of the system with a set of logical propositions, and actions with preconditions and effects that modify the state.

Define predicate describing the state

- Arm empty (AE)
- Arm holding (AH)
- Item ready to be put into a bag (IR)
- Bag has space (BHS)

- Item in the bag (IIB)
- Item Clear (IC) — item is in the bag and no items above it

Define rules

- Pick up (x)
    - Precondition & Delete: AE, IR(x)
    - Add: AH(x)
- Put into bag (x, bag_A)
    - Precondition & Delete: AH(x), BHS(bag_A)
    - Add: AE, IIB(x, bag_A), IC(x)
- Remove from bag (x, bag_A)
    - Precondition & Delete: AE, IIB(x, bag_A), IC(x)
    - Add: AH(x)
- Put the item back in line (x)
    - Precondition & Delete: AH(x)
    - Add: AE

Using the above STRIPS planning, we should be able to handle the order that needs to be modified by last-minute changes.

III. Conclusion

The project is related to optimizing routes for food delivery. The project involves using various algorithms and techniques to find the best possible route for food delivery based on different constraints and factors, such as distance, time, traffic, and delivery schedules. The project also involves using Python programming language.

Overall, the project involves solving a complex optimization problem that has real-world applications. It also requires a good understanding of algorithms and data structures, as well as proficiency in programming and data analysis. With proper implementation and testing, the project can potentially lead to significant improvements in food delivery efficiency and customer satisfaction.

IV. Appendix

A. FOODIE_Route

Sample runs for section A:

```
# Parameters
demands = np.array([0, 1, 1, 2, 4, 2, 4, 8, 8, 1])
n_robots = 4
robot_capacity = 10
```

n_ants = 10
alpha = 1
beta = 5
evaporation_rate = 0.1
n_iterations = 100

demands: An array representing the demand at each location. The demand is the amount of goods needed to be delivered to that location. The first element (with index 0) represents the depot, so its demand is set to 0.
n_robots: The number of robots available for deliveries. In this example, there are 4 robots.
robot_capacity: The maximum capacity that each robot can carry. In this example, each robot can carry up to 10 units of goods.
n_ants: The number of artificial ants used in the Ant Colony Optimization algorithm. A higher number of ants increases the exploration of different solutions, but may also increase the computation time.
alpha: A parameter that controls the importance of the pheromone trail in the Ant Colony Optimization algorithm. A higher value of alpha means the ants will be more influenced by the pheromone trail when selecting the next location.
beta: A parameter that controls the importance of the heuristic function (in this case, the inverse of the distance) in the Ant Colony Optimization algorithm. A higher value of beta means the ants will be more influenced by the heuristic function when selecting the next location.
evaporation_rate: The rate at which the pheromone trail evaporates in each iteration. This helps prevent the algorithm from converging too quickly to suboptimal solutions. The value should be between 0 (no evaporation) and 1 (complete evaporation).
n_iterations: The number of iterations the Ant Colony Optimization algorithm will run. A higher number of iterations allows the algorithm to explore more solutions, but may also increase computation time.
These parameters are used to initialize the Foodie_Route class and control various aspects of the Ant Colony Optimization algorithm. Adjusting these parameters can have an impact on the quality of the solution and the computation time.

Result:

Best solution: [[0, 1, 2, 3, 5, 6], [0, 9, 8], [0, 7], [0, 4]]
Best distance: 168.1373185347406

Explanation:

Robot 1 takes the route: depot (0) -> location 1 -> location 2 -> location 3 -> location 5 -> location 6 -> depot (0).
Robot 2 takes the route: depot (0) -> location 9 -> location 8 -> depot (0).
Robot 3 takes the route: depot (0) -> location 7 -> depot (0).
Robot 4 takes the route: depot (0) -> location 4 -> depot (0).
Best distance: The total distance traveled by all the robots for the best solution found.

In this example, the best distance is 168.1373185347406. This represents the total distance traveled by all three robots while completing their delivery routes.

Please note that the algorithm may produce different results each time it is run due to its stochastic nature. However, the solutions should generally be of similar quality.

B. FOODIE_BAGGER

Sample runs for section B:

```
Reading the item list as [item_name, size, frozen, fragile]...
['1-gallon water', 'large', '0', '0']
['1-gallon water', 'large', '0', '0']
['pint of ice cream', 'small', '1', '0']
['granola box', 'medium', '0', '0']
['loaf of bread', 'medium', '0', '1']
----------Bag large items----------
---
Current capacities of all bags are:[10 10 10 10 10]
Action: Put 1-gallon water into bag_0
---
Current capacities of all bags are:[ 6 10 10 10 10]
Action: Put 1-gallon water into bag_0
----------Bag medium items----------
---
Current capacities of all bags are:[ 2 10 10 10 10]
Action: Put granola box into bag_0
---
Current capacities of all bags are:[ 0 10 10 10 10]
Start a new bag: bag_current = 1
Action: Put loaf of bread into bag_1
loaf of bread is fragile, no more items into the current bag!
Start a new bag: bag_current = 2
----------Bag small items----------
---
Current capacities of all bags are:[ 0  0 10 10 10]
Next item is frozen! Start a new freezer bag or use an existing freezer bag with space left.
Action: Put pint of ice cream into (freezer) bag_4
-------------------------
Final capacities of all bags are:[ 0  0 10 10  9]
Bagging finished!
```