Data Collection in IoT Sensor Network Using Robots

For this project, three different algorithms were used and were mentioned in the supporting documents with various assumptions as suited for the proper implementation of the work.

Here in brief I will try to explain all the algorithms as given in the paper.

GREEDY1:

The suggested algorithm is a greedy one that seeks to maximize the awards gathered while determining a path from a source node to a destination node. What it does is as follows:

Sort all nodes according to reward value in ascending order.

This phase makes sure that visiting nodes with greater awards are given priority by the algorithm.
be conducted in rounds.

Set the current node and the budget to their initial values.

Verify if the currently visited nodes's feasible set contains the unvisited node with the biggest prize.

The nodes that can be reached from the current node within the budget are likely included in the feasible set.
Visit the node with the biggest award if it is feasible to do so.

Update the route and raise the amount of prizes earned.
Subtract the travel expenses to this node from the allotted budget.
Apply step 4 again while using the new current node.

Check to see if the unvisited node with the second-biggest prize is viable if the unvisited node with the largest award is not.

Continue doing this until there isn't enough money left over to visit any more unexplored nodes.

This algorithm's time complexity is given as O(|V|2), where |V| stands for the number of nodes. It means that the effectiveness of the method depends on how many nodes there are in the graph.

GREEDY2:

The greedy method known as algorithm 2 also finds a route from a source node to a destination node while maximizing the awards gathered. The prize cost ratio (pcr), a novel criterion introduced by this algorithm, is used to choose which node to visit next. The algorithm works in the following way:

be conducted in rounds.

During each round, determine if there are any feasible nodes that can still be visited from the current node using the money that is available.

The algorithm should end by traveling to the destination node if there are no more feasible nodes left, and it should then report the total prizes and costs of the path as well as the remaining budget.
Based on their prize cost ratio (pcr) to the current node order the unvisited viable nodes.

The ratio of the prize offered at node v to the edge weight between nodes 'u' and 'v' is used to calculate the pcr of an edge (u, v): pcr(u, v) = pv / w(u, v).
Locate the viable, unexplored node with the highest prize cost ratio.

In terms of maximizing the reward in relation to the expense, this node stands out as the best choice.
Update the current node and move to the node with the biggest pcr if one is found.

Based on the selected node, update the journey time and winnings.
By deleting the visited node, the list of unvisited nodes is updated.
Apply step 2 again using the updated current node .

The procedure should be stopped and the current results returned if there isn't any unvisited node that is both possible and has the biggest pcr within the available budget.

The method attempts to maximize the prizes received while taking into account the ratio of the prize to the expense of visiting each node by repeatedly choosing the unvisited and viable node with the largest pcr. When there are no more feasible nodes to visit within the budget allotted, the algorithm ends.

Greedy Algorithm 2's overall time complexity can be roughly calculated as O(|U|^2 log |U|), where |U| is the number of unvisited nodes.

MARL:

There are two stages of the MARL (Multi-Agent Reinforcement Learning) algorithm, also referred to as Algorithm 3. Let's examine each stage's time complexity in isolation:

Learning Stage: During the learning stage, m agents individually follow the action rule to travel to the next node, collect rewards, and update the Q-values of the relevant edges over the course of a predetermined number of episodes. The quantity of episodes and the decisions made within each episode determine how time-consuming the learning stage is.

Iterations over episodes: The overall number of iterations is determined by the predetermined number of episodes. This procedure requires O(n) time if there are n episodes.

Finding the route that has the most prize collected : The algorithm determines which of the m routes has the most rewards gathered, and it updates the reward value and Q-values of its associated edges. This step's implementation and the quantity of routes determine how time-consuming it is.

Execution Stage: During the execution stage, the traveling salesman moves from node s to node t, choosing the following node based on the highest Q-value. The amount of nodes and edges in the graph determines how time-consuming the execution phase will be.

The number of episodes, agents, and agents together with the size of the challenge (number of nodes and edges) will determine the overall time complexity of the MARL method.

Each episode has a maximum time for the first step of m |V |, where |V | is the total number of nodes, and a maximum time for the second step of m + |E|, where |E| is the total number of edges.
Algorithm 3's time complexity is therefore O(epi * m * |V |).

So I implemented all these three algorithms and have written python code for it. Briefly it can be explained as:

The code first imports the necessary libraries, after which it defines a number of functions:

Based on the coordinates of the two nodes, distance(node1, node2) calculates the distance between them.
Code uses a breadth-first search to determine whether a graph is connected.

Implements the first greedy method for resolving the sensor network puzzle, greedy1_algo(graph, source, destination, Budget_left).

The second greedy approach for resolving the sensor network issue is implemented by greedy2_algo(graph, source, destination, Budget_left).

The Multi-Agent Reinforcement Learning algorithm is implemented by MARL_algo(graph, source, destination, Budget_left) to address the sensor network problem.

Implements the exploitation action rule for node selection with the following parameters: Q_value, current_node, Unvisted_nodes, delta, and beta.

Implements the exploration action rule for node selection with the following parameters: Q_value, current_node, Unvisted_nodes, delta, and beta

main(): The primary process that executes the algorithms and accepts user input for the sensor network parameters.

The first algorithm put into place is called greedy_1, and it uses a greedy approach to determine the best path between the beginning node s and the target node t within the boundaries of the budget B that is provided. At each step until the target node is reached or the budget is used up, it chooses the unvisited node with the biggest prize that is attainable within the budget.

The second method used, greedy_2, is similar to greedy_1 but selects nodes in a different way. At each stage, it chooses the unvisited node with the highest prize-to-cost ratio that is feasible given the available funds.

The third algorithm used combines the learning and execution phases. It is called MARL (Multi-Agent Reinforcement Learning). It updates the Q-values during the learning stage after learning the Q-values for various state-action pairings based on the rewards. The next node is chosen during the execution stage based on the Q-values and the budget, and this

process continues until the target node is reached or there are no suitable nodes left to choose.

Overall, the code offers three distinct algorithms, each with its own methodology and approach, for resolving the sensor network issue.

Please find some of the example test cases below:

```
PS C:\Users\HP\Downloads\old-project-code> python .\project_set_2.py
Enter the following details:

Sensor network width: 100
Sensor network length: 100
Number of sensor nodes: 60
Transmission range of sensor nodes: 60
Number of data nodes: 55
Data nodes maximum capacity: 10
Robot's initial energy: 100000

Following are the data node and the data stored in the node:
Node = 43 , Data = 9
Node = 29 , Data = 9
Node = 6 , Data = 3
Node = 27 , Data = 8
Node = 38 , Data = 10
Node = 3 , Data = 8
Node = 59 , Data = 2
Node = 39 , Data = 4
Node = 57 , Data = 7
Node = 0 , Data = 2
Node = 40 , Data = 10
Node = 56 , Data = 7
Node = 57 , Data = 7
Node = 20 , Data = 3
Node = 12 , Data = 2
Node = 36 , Data = 2
Node = 21 , Data = 2
Node = 2 , Data = 3
Node = 3 , Data = 8
Node = 44 , Data = 2
Node = 37 , Data = 10
```

```
Results for Greedy1 Algorithm are as follows:
Route taken by the Robot:  [0, 54, 40, 48, 43, 11, 0]
Total Prize Collected by the Robot:  49
Total cost of the route taken bt Robot:  99984.38157455473
Energy left in the Robot:  15.618425445273587
Running time for this algorithm:  0.0

Results for Greedy2 Algorithm are as follows:
Route taken by the Robot:  [0, 22, 30, 29, 40, 48, 56, 57, 11, 51, 17, 49, 0]
Total Prize Collected by the Robot:  87
Total cost of the route taken bt Robot:  99945.80735941386
Energy left in the Robot:  54.19264058613393
Running time for this algorithm:  0.0

Results for MARL Algorithm are as follows:
Route taken by the Robot:  [0, 9, 2, 9, 2, 9, 2, 9, 2, 9, 2, 9, 2, 9, 2, 9, 2, 9, 2, 0]
Total Prize Collected by the Robot:  110
Total cost of the route taken bt Robot:  95044.41606685487
Energy left in the Robot:  4955.583933145139
Running time for this algorithm:  0.014983892440795898
PS C:\Users\HP\Downloads\old-project-code>
```

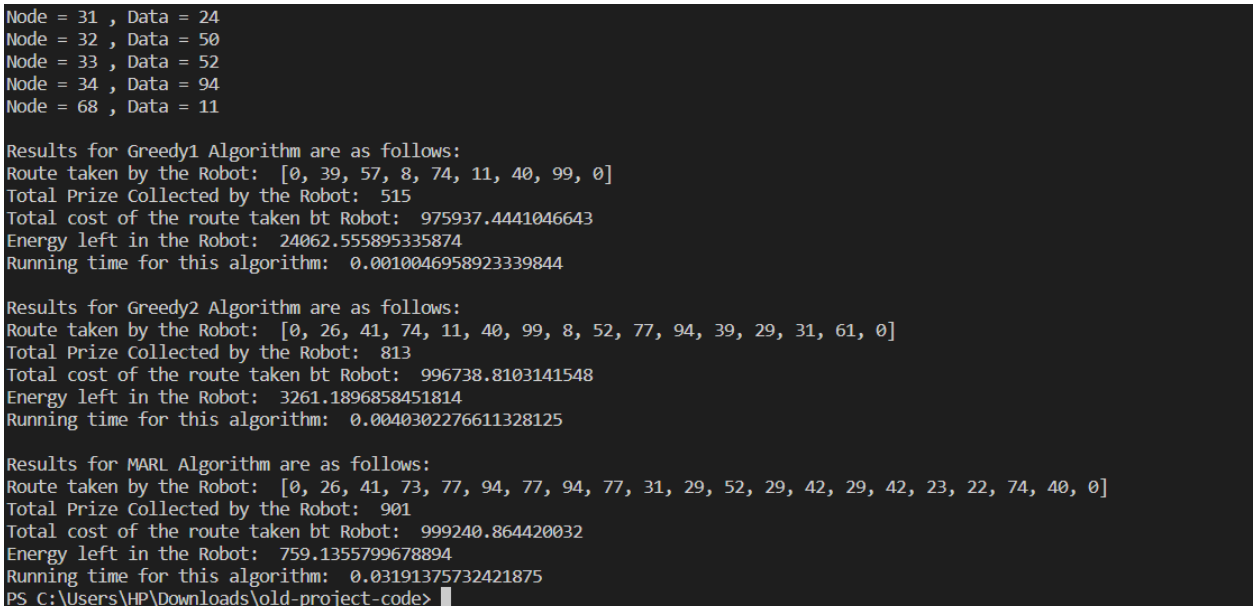
```
PS C:\Users\HP\Downloads\old-project-code> python .\project_set_2.py
Enter the following details:

Sensor network width: 100
Sensor network length: 100
Number of sensor nodes: 80
Transmission range of sensor nodes: 10
Number of data nodes: 20
Data nodes maximum capacity: 10
Robot's initial energy: 100000

The given graph network is not completely connected, please provide a different input.

Sensor network width: 1000
Sensor network length: 1000
Number of sensor nodes: 100
Transmission range of sensor nodes: 700
Number of data nodes: 80
Data nodes maximum capacity: 100
Robot's initial energy: 1000000

Following are the data node and the data stored in the node:
Node = 51 , Data = 29
Node = 29 , Data = 34
Node = 88 , Data = 29
Node = 32 , Data = 50
Node = 15 , Data = 78
Node = 99 , Data = 22
Node = 10 , Data = 45
Node = 27 , Data = 2
Node = 13 , Data = 14
```



Both the above cases show that the MARL algorithm produces better and optimized routes with higher total prize.

In Second, it can be observed that code prompts the user to enter inputs again as the graph is not connected with the given inputs.