Dispatching of Autonomous Cargo Transport Rovers with Multi-Agent Reinforcement Learning on a Graph Network

Problem:

How to efficiently dispatch a fixed number of autonomous rovers on a graph network to complete stochastic cargo delivery tasks.

Scenario:

In this scenario, we are operating with a fixed number of agents on a known graph network representing the world the agents are able to navigate within. Throughout the day, cargo delivery tasks will randomly arrive at certain nodes which, to be completed, will require an agent to navigate to the current location of the cargo, then navigate to the desired location of the cargo. Different nodes within the graph possess different probabilities of being a cargo origin, we can think of this as a kind of hub which would be an advantageous location for agents to cluster around. The agents will also have to maintain their battery charge throughout the day by visiting charging stations on the graph network. All of these factors together pave the way for an incredibly intelligent dispatching system to minimize time required to complete delivery tasks over the course of a day.

Applications:

Applications of this scenario exist in warehouse settings, large campus settings, autonomous food delivery settings, and large metropolitan areas for things like people movers and delivery vehicles. These applications make this task incredibly relevant in the real world, and vital to the field of robotics.

Prior Work:

Efficient Ridesharing Order Dispatching with Mean Field Multi-Agent Reinforcement Learning is the primary inspiration for this problem. In their paper they tackle the ridesharing dispatching problem using reinforcement learning. They discuss the need for reinforcement learning to tackle the complex supply/demand relation brought about by the nature of ridesharing. The dispatching decisions in the ridesharing context have huge effects on customer and driver satisfaction.

The primary complexity of the ridesharing problem is brought about by the difference in quantity of rideshare orders in different locations. If you have a driver complete a ride that pulls them away from an order dense location they will make less money which should be avoided.

We can set up a related problem in the autonomous cargo transport industry by considering high density locations to be cargo hubs such as a mailroom in the campus application or a strip of restaurants in a city food delivery application. We do not have to worry about the desires of the driver since we are using autonomous vehicles but we add the complexity of having to charge the vehicles while they are completing orders. This adds an interesting element to the optimization by incentivising the routing of low battery robots to destinations near charging stations. Additionally, rovers can be instructed to travel to other locations without the need for an order to be at that location.

Our application would create the possibility of a fully closed loop system where humans do not have to intervene in the work of the rovers.

Simulations:

The paper above uses two different simulations, a grid-based simulation and a coordinate-based simulation. In the grid-based simulation, the map plane is hexagonally tiled

which means that there is no distance between the driver and any new orders which arrive in that cell. The coordinate-based simulation places each driver on a 2D plane with the corresponding coordinate value which means that drivers will be required to travel a distance to pick up rides. This complicates their optimization problem greatly.

Proposed Simulation:

I am proposing a graph based simulation which consists of nodes which can have one of many times as well as any number of agents currently inhabiting them. As well as edges which take a fixed amount of time to cover.

Here is how my simulation will function:

Manifest:

The simulation user will be able to tune lots of parameters in the simulation using a manifest file including the number of agents, charging rate, agents range, task frequency, and many more things.

Map:

- The map will be a simple graph with nodes and edges.
- The map can be created manually by the user or generated randomly with certain parameters.
- There will be charging nodes and standard nodes, both nodes can hold any number of agents.
- Standard nodes will have a fixed probability of a new cargo task beginning at that node.
- Edges will have a fixed time required to traverse, no decisions can be made while traversing an edge.

Tasks:

- Tasks are represented by an origin and a destination.
- Agents can only complete a single task at a time.
- Agents cannot deviate from the shortest path while completing a task.
- Tasks will have a given percentage chance of spawning at any given node on the graph at any particular time step.

Decisions:

- At each time step the user makes a decision for each agent which is currently available.
- The agent can complete a given task, go to a charging station, go to any other available node on the network, or stay put.
- Agents will not be allowed to go to any node, or begin any task, which would leave them in a state where they will not be able to make it to a charging station.
- The user will have access to all global information such as the specifications of the map as well as the state of each agent.

Scoring:

Scoring will be purely based on the amount of time tasks spend waiting to be picked up.

Reinforcement Learning

I propose testing any machine learning solution to this problem against three competitors. I will play the game a few times to see my score, I will write a deterministic algorithm which selects random inputs, and I will write a greedy deterministic algorithm and compare each of these to my hopefully improved reinforcement learning based approach.

The reinforcement learning algorithm I will start with is the Q-learning algorithm. This requires the continual calling of the Q function. The Q function changes value based on the

current state of the world. Each rover knows the overall state of itself and the world. It knows the cost or “discount” of each move (in the short term), and it will be able to calculate the reward of that individual move.

I propose we set the reward for charging to be proportional to the percent of battery left and then we decrease the value of the Q function based on how long tasks sit while not being completed as a cost.

During training, we must provide all of the relevant data to the neural network including distance of each agent to a charger, battery percentage, current tasks available, current standard nodes available, distance to each task, ending location and the probability of new tasks at that location, how stale each order is, and so on. This global image will be necessary to make informed decisions on an agent by agent basis.

I would also like to test some of the reinforcement learning setups as described in this paper. There is a ton of expansion in the reinforcement learning field and Q-learning in particular. I imagine I will attempt to use most of these over the course of my work.