Efficient Dispatching of Autonomous Cargo Vehicles

August Soderberg
asoderberg@brandeis.edu
Brandeis University

ABSTRACT

The world is becoming more and more dependent on autonomous vehicles for moving cargo in many applications such as warehouses, food delivery, college campuses, trucking, and rail. This shift away from human involvement presents a unique opportunity to create accurate simulation environments and highly optimized dispatching algorithms to improve efficiency of operation and reduce costs to the user and the operator. In this paper, a low-level, graph-based simulation environment is proposed to accurately replicate many autonomous cargo vehicle scenarios to aid in the development and testing of dispatching algorithms. Additionally, multiple dispatching algorithms are shown to optimize the dispatching of vehicles in simulation. A reinforcement learning model efficiently learns unknown world information such as the locations and rate at which new cargo tasks are generated. Also, brute-force optimization is used to minimize the time to complete deliveries in scenarios where all world information is known to the dispatcher.

KEYWORDS

autonomous robotics, dispatching, reinforcement learning, optimization

1 INTRODUCTION

The development and use of autonomous vehicles to move cargo is skyrocketing. Warehouses are steadily removing the number of employees involved in their processes. Amazon's internal development of their autonomous warehouse vehicles through Amazon Robotics has drastically changed the way their massive fulfillment centers operate with thousands of their vehicles performing tasks simultaneously [1]. Many high density urban areas and college campuses have turned to small autonomous vehicles for their food delivery services with companies like Starship Technologies leading the way [7]. Trucking is seeing a boom in autonomous technologies with companies like TuSimple [12]. The Oodi Library in Helsinki, Finland employs autonomous robots to help reshelve books [11].

Each of the scenarios above can be seen as analogues of a general autonomous cargo delivery scenario. We can consider the environment in which the vehicles operate as an undirected graph network where nodes are locations of interest (cargo pick-up or drop-off areas, vehicle charging station, intersections, etc.) and edges are paths vehicles take to travel between nodes in the network. Vehicles are required to travel around the graph network to complete tasks, namely, driving to a node to grab cargo and then moving the cargo from one node in the network to another. The vehicles have a certain range they are able to travel based on battery or fuel capacity which decreases as more distance is traveled. This abstraction to a general autonomous cargo delivery scenario begets the utility of a general simulation environment and dispatching heuristics.

The rest of this paper is organized as follows. §2 reviews the related work on dispatching in ride sharing and autonomous mobile robots. §3 describes the proposed simulation for the general autonomous cargo delivery scenario. §4 presents the reinforcement learning solution and results. §5 presents the formulation and solution to the optimization solution. §6 summarizes the research.

2 RELATED WORK

Efficient dispatching of vehicles is a well researched field in the domain of autonomous and non-autonomous ridesharing and robotics applications. Many vehicle dispatching solutions rely on centralized or decentralized reinforcement learning (RL) to solve the task-assignment problem [5, 8]. These RL models require the creation of a simulation environment in which the models can learn and develop their policy.

The remainder of this section will be as follows: §2.1 will be focused on the simulation environments developed by Li et al. for their mean-field multi-agent reinforcement learning model. §2.2.1 will discuss the simulation developed by Malus et al. for their autonomous mobile robots dispatching problem solved by a multi-agent reinforcement learning learning model analyzed in §2.2.2. §2.3 briefly mentions other dispatching problems.

2.1 Ridesharing Order Dispatching

Li et al. developed a novel multi-agent reinforcement learning model for solving the ridesharing order dispatching problem. This problem consists of multiple agents operating in a stochastic order system where agents can fulfil multiple orders simultaneously. An order o_j consists of an initial location $ori_{i,j}$ meaning the user j's initial location is in the neighborhood of loc_i where the driver currently is and a desired destination of location $des_{i,j}$. Let us consider agent a, locations x, y, z and orders o_1, o_2 where:

$$ori_{i,1}, ori_{i,2} \in x$$

$$des_{i,1} \in y$$

$$des_{i,2} \in z$$

If considered desirable, agent a in the neighborhood of $loc_i = x$ could begin o_1 and then begin o_2 by traveling to locations $ori_{i,1}$ and $ori_{i,2}$ before traveling to their respective destinations [5].

Two simulations, one grid-based and one coordinate-based, were developed to solve the Ridesharing Order Dispatching problem.

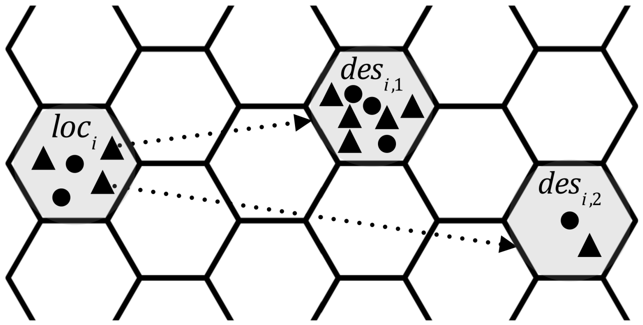

Figure 1: Hexagonal map tiling outlined by Li et al. in their grid-based environment

2.1.1 Grid-based environment. In the grid-based environment (Figure 1), the map is tiled by hexagons where the inter-hexagonal distance is approximately 1.2 kilometers. For this reason, the authors have restricted the dispatching algorithm such that if O is the set of all active orders and agent a is in the neighborhood of location loc_i then

$$O_a = \{o_j | o_j \in O, ori_{i,j} = loc_i\}$$

where O_a is the set of all active orders that can be completed by a. Thus, the grid-based environment does not consider any travel time between agents and the orders they choose to pick up [5].

Additionally, to emulate the real world application, agents will transition from active to inactive as decided by a stochastic process [5].

2.1.2 Coordinate-based environment. In the coordinate-based environment, each agent's precise location is defined by a coordinate pair and each order is defined by two coordinate pairs representing the origin and destination of the order. In this environment, with time between the dispatching of an agent and the arrival at the order's location being taken into account, another emulation of the real world application has been implemented. Orders may become unavailable after an agent has been dispatched but before arrival at the origin location based on a random process correlated with the distance between the agent and the order [5].

2.2 Dispatching Autonomous Mobile Robots (AMRs)

Malus et al. focused on the dispatching of AMRs which is a distinction from the dispatching of autonomous guided vehicles (AGVs) which follow strict paths to navigate the world through a simple line following algorithm often relying on magnetic tape on the ground. AMRs on the other hand will move freely without predefined routes through the use of complex sensors which enable localization and safe navigation [8].

2.2.1 AMR simulation environment. The environment in which the AMRs operated consisted of an inaccurate Python simulation for data generation and learning and a highly accurate physics based simulation built in Robot Operating System (ROS) utilizing the simulator Gazebo. The ROS and Gazebo simulation was exclusively used for validation of the policy learned by the models trained utilizing the Python simulation. The world created in Gazebo for validation consisted of five agents in a rectangular room divided the majority of the way down the middle by a wall. Note that for each scenario tested, a new world must be created in the Gazebo simulator to match the scenario on which the model learned in Python [8].

2.2.2 AMR Dispatching Algorithm. Malus et al. propose a new distributed RL model based on the actor-critic model. The actor-critic model is adept at handing continuous action and state spaces since it utilizes two neural-network models, one to learn the policy and one to learn the reward function which is not well-defined for complex continuous action spaces. The proposed Twin Delayed Deep Deterministic (TD3) policy gradient algorithm is an actor-critic algorithm which approximates two Q-functions and a policy. The algorithm is distributed across all agents. When a new order spawns, the algorithm iterates as follows:

(1) Send new world state to all agents
(2) Agents place bids between 0 and 1 for the new order
(3) The agent with the highest bid is dispatched to the order

The inverse of the time required to complete an order is recorded each time an order is completed. When a new order spawns, each agent is given the accumulated inverses of the task waiting times as a reward. This means that all agents are given identical rewards to ensure that the behavior of a single agent supports the goal of the collective [8].

2.3 Other Dispatching Problems

There are several other peripherally related dispatching problems such as virtual coupling [2], taxi dispatching [6], energy distribution [4], and network dispatching for internet services [3]. These applications are not as relevant to the general autonomous cargo delivery scenario but the researching of them is of note to the field.

3 PROPOSED SIMULATION

Developing useful dispatching algorithm heuristics to solve the general autonomous cargo delivery scenario requires a simulation environment which accurately captures the specifics of a real world application without overfitting to the nuances of a specific application; this is the simulation proposed.

3.1 Simulation Implementation

The simulation is composed of the following parts.

3.1.1 Graph Network. The simulation is based on a connected, undirected graph network also referred to as the map (Figure 2). The map is composed of nodes and edges. Nodes will spawn tasks (§3.1.4) for the vehicles to complete. The nodes may also be chargers which will increase the range of the vehicles (§3.1.3) while the vehicles rest at the charging node. Nodes do not have any maximum capacity, as such any number of vehicles can occupy a node at a given time. Edges are undirected and given a length which determines, proportionally, the time required for a vehicle to traverse the edge. Edges also do not have any maximum capacity and can thus be traversed in either direction by any number of vehicles at a time.

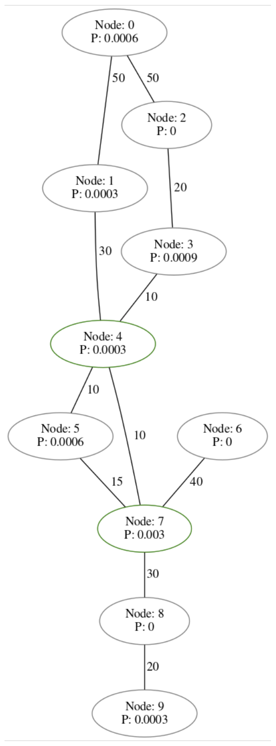

Figure 2: An example scenario map with 8 regular nodes, 2 charging nodes, and random spawning probabilities P

3.1.2 Navigation. Vehicles will navigate through the map following only the shortest path between the current location and the destination. Upon creation of the graph G

$$G = \{N, E\}$$

where N is the set of nodes and E is the set of edges, a mapping r is created, using Dijkstra's algorithm

$$r : P \longrightarrow T$$

where

$$P := \{(u, v) | u, v \in N\}$$

and

$$r(u, v) = t \in T \iff t \text{ is the minimum travel time between } u \text{ and } v$$

For example, if a vehicle is at node a, and is dispatched to pick up a task a node b and deliver it to node c, the time required for that journey is precisely equal to

$$r(a, b) + r(b, c)$$

3.1.3 Vehicles. Vehicles in the simulation are considered electric and as such have a maximum range and current range. The current range of each vehicle decreases linearly with the distance traveled. Vehicles are only able to carry a single unit of cargo at a time and must complete a task in the quickest fashion possible. Vehicles are forced to remain in a safe state, this means that vehicles will reject any dispatching request which leaves them with range less than the distance to the nearest charger.

3.1.4 Tasks. Tasks are defined by spawn time, spawn node, and a completion node. Each node has a fixed probability of spawning in a new task at any given time. The completion node for each task is determined uniformly across all nodes. For a task to be completed, a vehicle must drive to the spawn node, then drive to the completion node.

3.1.5 Known and unknown information. The simulation is designed such that any quantity of information from the underlying configuration and current state of the simulation can be given to the dispatcher to aid in its decision. In §4 the task spawning probability is learned by the reinforcement learning model and is not provided to the model. In §5, the spawning probabilities are provided to allow for optimization.

3.1.6 Design for optimization. For every task spawned during a complete run of the simulation, the time between the spawning of the task and the picking up of the task by the vehicle is tracked. This value is most directly affected by the efficiency of the dispatching algorithm which is why it is chosen as the performance metric of the simulation. At the end of a run, the total time spent by tasks waiting to be picked up is reported. The minimization of this value is the goal of a dispatching algorithm.

3.2 Modularity

The specifications of the simulation are all determined in an editable manifest. This allows the number of vehicles, vehicle speed, battery capacity, and charging speed to be adjusted as appropriate as well as the location of charging nodes, relative spawning probabilities of nodes in the graph, the structure of the graph network including number of nodes and edges, and edge lengths. Each of these values can be adjusted to ensure the simulation is able to mimic any particular real world scenario.

4 REINFORCEMENT LEARNING DISPATCHER

Reinforcement learning is utilized in our scenario to learn the unknown world features to achieve efficient dispatching. In the case where the dispatcher is unaware of certain simulation specifications a strictly optimized solution is infeasible; for this reason, RL is used to learn a dispatching policy by learning and leveraging the known world information.

To test this case, the relative task spawning probabilities of each node are removed from the state information input to the RL model. This scenario would mimic a real world application where task spawning rates per location are difficult to determine but could be modeled by an unknown function f. The goal of the dispatching algorithm is to learn the function f and utilize that information in its dispatching decisions.

4.1 The Models

The dispatching algorithm is learned by two independent agents: The pre-task and post-task dispatchers. The pre-task dispatcher is responsible for deciding which vehicle to dispatch to a newly spawned task. The post-task dispatcher is responsible for deciding to which node to send a vehicle immediately after it has completed a task. Proximal policy optimization (PPO) reinforcement learning models were used for the pre- and post-task dispatchers [10]. The reward provided to each model is identical to promote coordinated learning and cooperation.

4.1.1 The pre-task dispatcher. The continuous state matrix for the pre-task dispatcher is defined as

$$\mathcal{S}_{pre} = \begin{bmatrix} a_1 & d_1 & r_1 & c_1 \\ a_2 & d_2 & r_2 & c_2 \\ \dots & \dots & \dots & \dots \\ a_n & d_n & r_n & c_n \end{bmatrix}$$

where n is the number of vehicles in the scenario and for the i^{th} vehicle, $a_i \in \{0,1\}$ indicates whether it is able to complete the given task, $d_i \in \mathbb{R}^+$ equals the distance from the vehicle to the task, $r_i \in \mathbb{R}^+$ equals the remaining range the vehicle would have after completing the task, and $c_i \in \{0,1\}$ indicates whether the vehicle is currently charging.

The output of the pre-task model is a vector W of length n. The dispatcher performs action masking by selecting vehicle i for the task if

$$w_i = \max(\{w_i | w_i \in W \text{ and } a_i = 1\})$$

4.1.2 The post-task dispatcher. The continuous state matrix for the post-task dispatcher is defined as

$$\mathcal{S}_{post} = \begin{bmatrix} a_1 & r_1 & c_1 & o_1 \\ a_2 & r_2 & c_2 & o_2 \\ \dots & \dots & \dots & \dots \\ a_n & r_n & c_n & o_n \end{bmatrix}$$

where n is the number of nodes in the network and for the i^{th} node, $a_i \in \{0,1\}$ indicates whether the vehicle can travel to the node, $r_i \in \mathbb{R}$ equals the remaining range the vehicle would have after arriving at the node, $c_i \in \{0,1\}$ indicates whether the node is a charging node, and $o_i \in \{0,1\}$ indicates whether there is already a vehicle at the node. Keep in mind that multiple vehicles can occupy nodes at the same time. The output of the pre-task model is a vector W of length n. A similar action masking procedure follows at this point where the vehicle is sent to node i if

$$w_i = \max(\{w_i | w_i \in W \text{ and } a_i = 1\})$$

4.2 Reward Function

Both dispatching agents will make many decisions throughout a single run of the simulation without updating the internal weights of their neural-networks. For each decision, the state of the internals, the state of the environment, and the action vector are saved. Once the a run of the simulation is complete, the total waiting time of all tasks during the run t is recorded and a retroactive reward value r can be defined as

$$r = \frac{-t}{n_d}$$

where n_d is the number of dispatching decisions made. This reward value is then applied to each decision made by the model and the model is able to observe the internal state, environment state, action vector, and reward value in order to improve its performance. This is a single epoch of the training paradigm.

4.3 Results

The reinforcement learning models were able to learn and utilize the relative spawning probabilities of the nodes in the network. To illustrate this finding, a comparison is made between the results of the reinforcement learning model, a greedy dispatcher, and a random dispatcher.

4.3.1 Greedy dispatcher. The greedy pre-task dispatcher simply selects the closest vehicle to a new task to complete it while the post-task dispatcher leaves all vehicles with remaining range above a minimum-range threshold where they are after completing tasks and sends those with remaining range below the minimum-range threshold to the nearest charger. This is advantageous for three reasons: First, while the task spawning locations are determined by highly variant probabilities, the task final location is uniformly random across all nodes; this uniformity means that an advantageous strategy that doesn't utilize the spawning probabilities such as attempting to spread vehicles out to maximize map coverage is equivalent to allowing vehicles to stay were they are. Second, vehicles are unable to be dispatched to tasks if they are busy traveling between other nodes. The lack of excess travel time means the maximum number of vehicles are ready to take on new tasks which reduces total task waiting time. Third, this strategy minimizes energy use which means vehicles are less likely to be unable to be dispatched for tasks because of low range.

4.3.2 Random dispatcher. The random pre-task dispatcher selects a vehicle at random to complete a new task while the random post-task dispatcher selects a node at random to send the vehicle to after they have finished a task.

The *RL dispatcher* significantly reduces task waiting time compared to the other two dispatching algorithms. (Figure 4, Figure 5) The comparison between the *greedy dispatcher* and the *RL dispatcher* illustrates the drastic reduction of task waiting time achievable through the learning of each node's task spawning probability. While neither has access to the true task spawning probability, the *RL dispatcher* is able to learn the probabilities to improve overall performance.

5 OPTIMIZED DISPATCHER

Optimization can be utilized to achieve efficient dispatching in the scenario where all world features are known to the dispatcher. To illustrate more clearly the effect of optimization on our dispatching problem, we will solely consider optimization of the post-task dispatcher problem. For the remainder of this section, the pre-task dispatcher in our optimized solution will function identically to that of the *greedy dispatcher* while we attempt to optimize the post-task dispatcher.

5.1 Philosophy

If we consider the human decision making process for our post-task dispatching scenario we see that certain aspects of the state of the world affect, positively and negatively, the likelihood for a node to be selected as the destination for a vehicle. The intuition of the human will determine how much the distance to the node, the spawning probability of the node, whether or not it is a charging

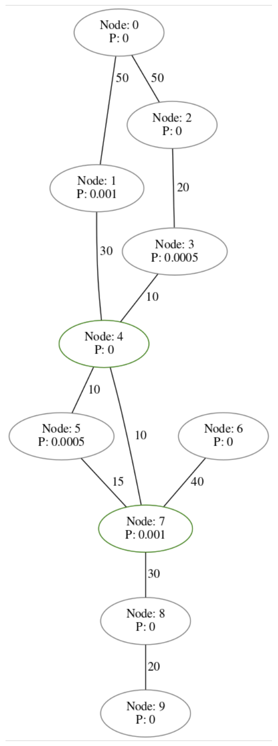

Figure 3: The map used for training and testing of the RL Dispatcher

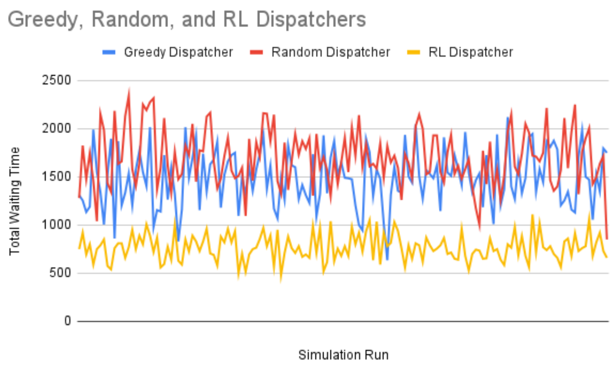

Figure 4: A comparison of the total waiting time for tasks across 150 simulation runs of the *greedy dispatcher*, *random dispatcher*, and *reinforcement learning dispatcher*

node, and whether there is already an vehicle at the node, affect the decision whether or not to dispatch the vehicle to the node. The human will weigh each of these factors for each node in the world to determine which one they perceive as the most advantages destination for the vehicle. If the human were able to adjust their perceived importance of each of these factors over hundreds of test runs, they would discover the optimal importance that each aspect of the world state should carry.

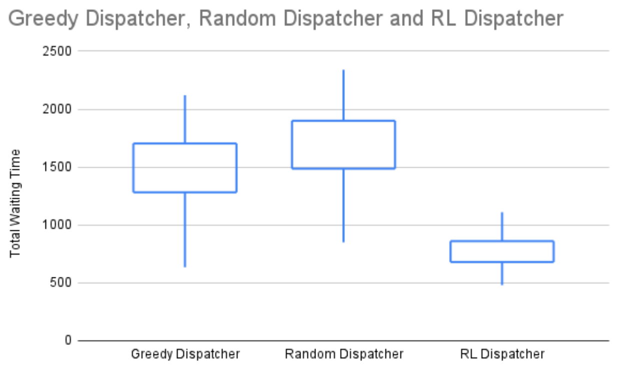

Figure 5: Total waiting time distribution comparison between the dispatching algorithms

5.2 Post-task Dispatching Algorithm

The post-task dispatching algorithm will function as a bidding war between each node to be the recipient of the vehicle to be dispatched. Each node n_i which can be safely reached by the vehicle will be given a state vector s_i where

$$s_i = (r_i, p_i, c_i, o_i)$$

where $r_i \in \mathbb{R}^+$ is the remaining range of the vehicle after driving to node i, $p_i \in \mathbb{R}^+$ is the spawning probability of node i, $c_i \in \{0, 1\}$ indicates whether node i is a charging node, and $o_i \in \{0, 1\}$ indicates whether another vehicle is already occupying node i.

The bid for each node i will be determined as a dot product of s_i and a weights vector w (to be optimized in §5.3) where each element of $w_i \in [-1, 1]$. The node with the highest bid will then be selected and the vehicle will be dispatched to that node.

5.3 Optimization and Results

Brute-force optimization can be used to find a well-performing value of w for the post-task dispatcher. Since the values of r_i, p_i, and c_i should increase the value of the bid, while o_i should decrease the value of the bid, the searchable area for each variable are as follows:

$$w_r, w_p, w_c \in [0, 1]$$

$$w_o \in [-1, 0]$$

where

$$w = (w_r, w_p, w_c, w_o)$$

We will test ten values for each element of w uniformly distributed over its respective range. The function which our brute-force optimization attempts to minimize is the mean of ten complete simulation runs utilizing the *post-task dispatching algorithm* from §5.2. Once the minimum value in our searching space is determined, a Nelder-Mead simplex algorithm is used to further optimize the value of w [9].

5.3.1 Hopeful Dispatcher. For comparison with the optimized solution, a *hopeful dispatcher* is developed to attempt to solve the post-task optimization problem more efficiently. The *hopeful dispatcher* utilizes the same pre-task dispatching algorithm as the *greedy dispatcher* and our *optimized dispatcher*. For the post-task dispatching algorithm, the *hopeful dispatcher* will send the vehicle to the node with the highest task spawning probability which is not already occupied by another vehicle. This method of post-task dispatching is attempting to reduce the total task waiting time by maximizing the number of zero waiting time task pick-ups.

The particular scenario used for the optimization and testing is a twenty-node scenario with fixed spawning probabilities, fixed charger locations, and eight vehicles. (Figure 6)

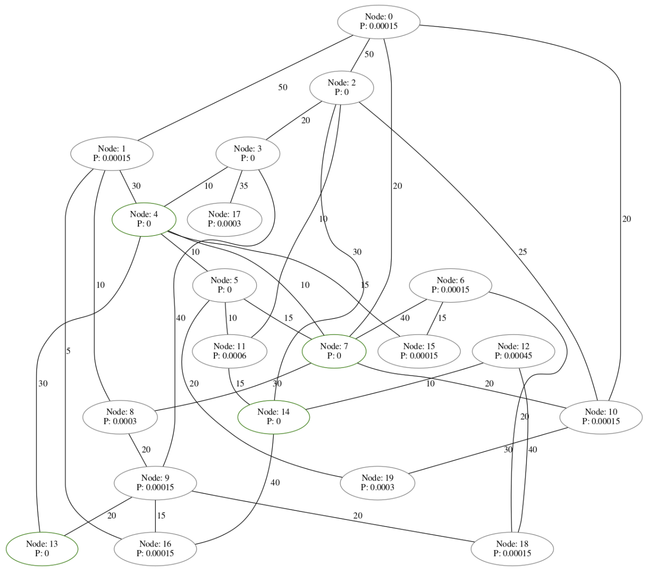

Figure 6: A twenty-node scenario used for testing with the *optimized dispatcher* and the *hopeful dispatcher*

Once the optimized values of w were found, 150 complete simulation runs were recorded with both *hopeful dispatcher* and the *optimized dispatcher*. (Figure 7, Figure 8)

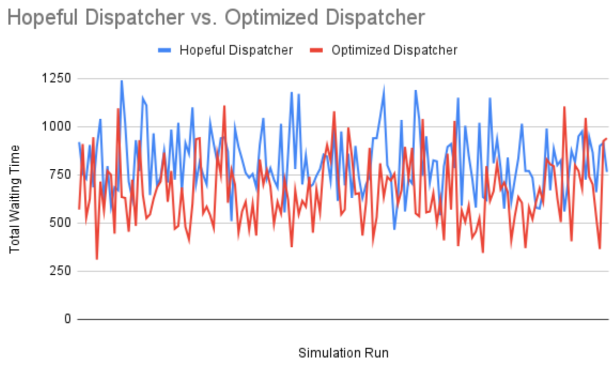

Figure 7: The total waiting time for tasks for each of the 150 runs of the simulation separated by the dispatching algorithm

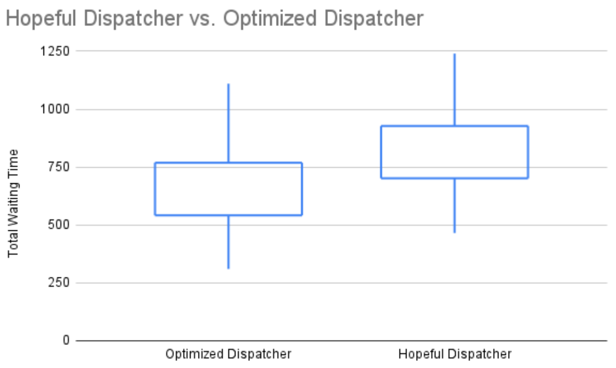

Figure 8: Total waiting time distribution comparison across all 150 runs of the simulation.

The p-value of these distributions is less than 0.00001 meaning the *optimized dispatcher* has significantly outperformed the *hopeful dispatcher*. However, it is incorrect to conclude that the *optimized dispatcher* has achieved the most efficient dispatching performance for this particular scenario. It has simply located a local minimum for the performance of this specific algorithm.

6 CONCLUSION

The utilization and development of autonomous cargo vehicles is continually increasing as companies find more and more applications for them. With that increasing development comes the search for efficient dispatching algorithms to reduce costs to users, operators, and the world. The need for simulation environments which accurately capture the nature of these systems and allow for simpler state of the art development is at an all time high. A novel, graph-based simulation environment is proposed to replicate real world scenarios for the simple development of dispatching algorithms. Additionally, a reinforcement learning solution and a brute-force optimization solution are presented as heuristics for efficient dispatching algorithms.

The graph-based simulation can be modified to represent most autonomous cargo delivery scenarios. Vehicles are dispatched around the graph network, while maintaining sufficient range by utilizing charging nodes, to pick-up and drop-off cargo at the locations designated by the tasks spawned into the world. The simulation environment can report any and all state information to the dispatcher before finally reporting the total task waiting time as a performance metric.

The reinforcement learning dispatching algorithm learns unknown world information such as task spawning probabilities and utilizes that knowledge to improve its performance. It reduced the mean total task waiting time by 48.3% and 54.7% for the *greedy dispatcher* and *random dispatcher* algorithms respectively.

The brute-force optimization dispatching algorithm searched for an optimal solution to the dispatching problem by leveraging its knowledge of all world information. It reduced the mean total task waiting time from the *hopeful dispatcher* by 19.4%.

More accurate and useful simulation environments bring about more state of the art research focused on sophisticated, impressive, and versatile dispatching algorithms which can drastically improve the function of those systems in everyday life.

7 ACKNOWLEDGEMENTS

This research was made possible by the continued advising, support, and mentoring of Professor R. Pito Salas as well as the valuable consultation of Professor Pengyu Hong.

REFERENCES

[1] Robert Bogue. 2016. Growth in e-commerce boosts innovation in the warehouse robot market. *Industrial Robot* 43, 6 (2016). https://www.emerald.com/insight/content/doi/10.1108/IR-07-2016-0194/full/html

[2] Francesco Flammini, Stefano Marrone, Roberto Nardone, Alberto Petrillo, Stefania Santini, and Valeria Vittorini. 2018. Towards Railway Virtual Coupling. https://doi.org/10.1109/ESARS-ITEC.2018.8607523

[3] Guerney D.H. Hunt, Germán S. Goldszmidt, Richard P. King, and Rajat Mukherjee. 1998. Network Dispatcher: a connection router for scalable Internet services. *Computer Networks and ISDN Systems* 30, 1 (1998), 347–357. https://doi.org/10.1016/S0169-7552(98)00088-9 Proceedings of the Seventh International World Wide Web Conference.

[4] Hossein Karami, Mohammad Javad Sanjari, Seyed Hossein Hosseinian, and G. B. Gharehpetian. 2014. An Optimal Dispatch Algorithm for Managing Residential Distributed Energy Resources. *IEEE Transactions on Smart Grid* 5, 5 (2014), 2360–2367. https://doi.org/10.1109/TSG.2014.2325912

[5] Minne Li, Zhiwei, Qin, Yan Jiao, Yaodong Yang, Zhichen Gong, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. 2019. Efficient Ridesharing Order Dispatching with Mean Field Multi-Agent Reinforcement Learning. https://doi.org/10.48550/ARXIV.1901.11454

[6] Zhidan Liu, Jiangzhou Li, and Kaishun Wu. 2022. Context-Aware Taxi Dispatching at City-Scale Using Deep Reinforcement Learning. *IEEE Transactions on Intelligent Transportation Systems* 23, 3 (2022), 1996–2009. https://doi.org/10.1109/TITS.2020.3030252

[7] Ingrid Lunden. 2022. Starship Technologies raises another $42M to fuel the growth of its fleet of self-driving delivery robots. *TechCrunch* (2022). https://techcrunch.com/2022/03/01/starship-technologies-raises-another-42m-to-fuel-the-growth-of-its-fleet-of-self-driving-delivery-robots/

[8] Andreja Malus, Dominik Kozjek, and Rok Vrabič. 2020. Real-time order dispatching for a fleet of autonomous mobile robots using multi-agent reinforcement learning. *CIRP Annals* 69, 1 (2020), 397–400. https://doi.org/10.1016/j.cirp.2020.04.001

[9] Donald M. Olsson and Lloyd S. Nelson. 1975. The Nelder-Mead Simplex Procedure for Function Minimization. *Technometrics* 17, 1 (1975), 45–51. https://doi.org/10.1080/00401706.1975.10489269 arXiv:https://www.tandfonline.com/doi/pdf/10.1080/00401706.1975.10489269

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. https://doi.org/10.48550/ARXIV.1707.06347

[11] Tom Scott. 2019. Why Helsinki's Library Robots Aren't Important. https://www.youtube.com/watch?v=dPb9o3uDF_Q

[12] TuSimple. 2022. Setting the stage for the world's first driver out autonomous truck runs. *Transport Dive* (2022). https://www.transportdive.com/spons/setting-the-stage-for-the-worlds-first-driver-out-autonomous-truck-runs/619958/