Multi-Agent Reinforcement Learning for Resource Allocation in Large-Scale Robotic Warehouse Sortation Centers

Yi Shen[1], Benjamin McClosky[2], Joseph W. Durham[2], and Michael M. Zavlanos[1]

***Abstract*— Robotic sortation centers use mobile robots to sort packages by their destinations. The destination-to-sort-location (chute) mapping can significantly impact the volume of packages that can be sorted by the sortation floor. In this work, we propose a multi-agent reinforcement learning method to solve large-scale chute mapping problems with hundreds of agents (the destinations). To address the exponential growth of the state-action space, we decompose the joint action-value function as the sum of local action-value functions associated with the individual agents. To incorporate robot congestion effects on the rates at which packages are sorted, we couple the local action-value functions through the states of destinations mapped to nearby chutes on the sortation floor. We show that our proposed framework can solve large chute mapping problems and outperforms static or reactive policies that are commonly used in practice in robotic sortation facilities.**

I. Introduction

Modern warehouses often rely on mobile robots to transport and sort packages in their sortation hubs [1]–[4]. For example, in Amazon Robotics sortation hubs [5], packages arrive at induct stations, located on the perimeter of a sortation floor, where they are loaded onto mobile robots and transported based on their destinations to sort locations (eject chutes) arranged in a grid in the interior of the sortation floor; see Fig. 1. In these robotic sortation hubs, the volume of packages (throughput) that can be sorted by the robot fleet is partly determined by the availability of a sufficient number of eject chutes for each destination as well as by how the robots interfere with each other's motion as they navigate from the induct stations to their assigned eject chutes. As a result, the destination-to-chute mapping can significantly impact the throughput that the mobile robot fleet can deliver.

Robotic and automated warehouse systems as the one discussed above have been extensively studied in the literature. Generally, approaches to optimize their operation can be classified as simulation-based or analytical; see [2] for a detailed survey. Simulation-based methods rely on high-fidelity simulators to evaluate the performance of different operational decisions. However, designing good simulators is labor-intensive and their use for warehouse optimization is often hindered by prohibitively long simulation times for typical large-scale warehouses. In contrast, analytical methods focus on finding good approximate solutions fast, by solving optimization problems with fairly relaxed assumptions that capture the gist of reality. These approaches generally belong to two main categories: queuing networks (QNs) and mathematical programming models (e.g., mixed-integer programs (MIP)). QNs, e.g., open queuing networks [6] or semi-open queuing networks [7], rely on queueing theory to build models of sorting systems and characterize system performance in terms of, e.g., throughput and storage capacities. QNs typically focus on system-level design decisions, also called long-term decisions, such as design of facility layouts [8] or robot zoning strategies [9]; see [10] for a review of robotic sorting methods using QNs. In this paper, we instead focus on operational planning decisions, also called short-term decisions, and in particular, on the destination assignment problem (DAP) [11] assuming that facility layouts, workstation locations, robot path topologies, etc. are given. QNs are not suitable for solving operational-level DAPs; these problems can be solved using mathematical programming.

Fig. 1. A sortation floor in an Amazon Robotics sortation hub [3].

Mathematical programming has been long used for the optimization of warehouse systems, including for the solution of DAPs for sorting systems. For example, destination mapping for conveyor-based sorting systems is studied in [12] with the objective to minimize the total distance travelled by packages between inbound and outbound stations. Although distance travelled is only a proxy for system performance, the authors empirically show that the proposed method also improves package throughput. A different objective is considered in [13] that focuses on minimizing the worst-case flow imbalance across all work stations that process packages on the sortation floor. To capture package variability, a stochastic approach is developed that employs chance and robust constraints. In the case of robotic sorting systems, an integer programming method is developed in [14] to solve DAPs that minimize sortation effort and satisfy package

[1]Yi Shen and Michael M. Zavlanos are with Amazon Robotics and the Department of Mechanical Engineering and Materials Science, Duke University. {yishenn,miczavla}@amazon.com, {yi.shen478, michael.zavlanos}@duke.edu.

[2]Benjamin McClosky and Joseph W. Durham are with Amazon Robotics. {mcclosky, josepdur}@amazon.com

deadlines. A robust formulation of this problem is proposed in [15] that can handle uncertainty in the demands.

In this paper, we focus on DAPs for robotic sorting systems such as those described in Fig. 1. The solution to these DAPs is a chute mapping that determines how package destinations are distributed over the chutes on the sortation floor. This chute mapping directly impacts induct-to-eject mission lengths and congestion of robots on the sortation floor, which together determine the throughput of the sortation floor. Specifically, an inadequate number of chutes per destination can lead to large numbers of unsorted packages entering a overflow buffer as they wait their turn to be sorted, which can cause significant throughput drops. On the other hand, robot congestion on the sortation floor affects the rate at which robots deliver packages to their assigned chutes and, therefore, the rate at which packages are sorted for each destination. Our goal is to determine the optimal number of chutes per destination so that the number of unsorted packages is minimized. While we do not explicitly model robot congestion, we incorporate congestion effects in the chute mapping decisions through their impact on the rate at which packages are sorted for each destination.

An important challenge in designing effective chute maps is the ability to capture uncertainty in the incoming packages that can substantially affect the required number of chutes per destination. Since accurate statistical models of this uncertainty are hard to obtain in practice, conventional mathematical programming methods cannot be used to obtain effective solutions to chute mapping problems. To address this challenge, in this paper we employ model-free reinforcement learning (RL) to dynamically determine the optimal number of chutes per destination over the course of a day.[1] Specifically, we formulate the chute mapping problem as a multi-agent Markov game where the agents correspond to destinations and the actions control the number of chutes assigned to each destination at each time step. To address the exponential growth of the state-action space, we decompose the joint action-value function as the sum of local action-value functions associated with the individual destinations, that are coupled through the states of other destinations mapped to nearby chutes on the sortation floor. This way, we incorporate robot congestion effects on the rates at which packages are sorted for destinations mapped to nearby chutes on the sortation floor. Then, using the local action-value functions, we formulate an integer program to optimally allocate the available chutes to the agents. We show that our proposed framework can solve large chute mapping problems and outperforms static or reactive policies that are commonly used in robotic sortation floors.

We note that RL has been used before to solve resource allocation problems. For example, a deep deterministic policy gradient method is developed in [16] to solve cloud computing

[1]We assume that a chute placing policy is given, that assigns specific chutes on the floor to every destination after the number of chutes per destination is determined. Learning a complete chute map that determines both the number and locations of chutes for every destination is a more challenging problem due to the exponential growth of possible chute assignments.

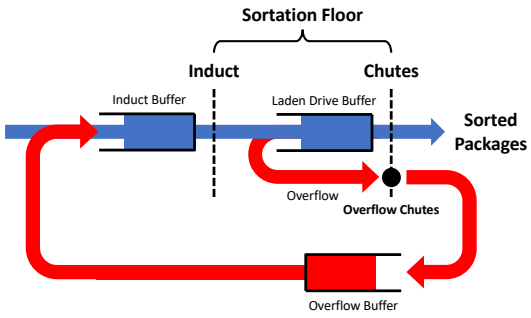

Fig. 2. Flow of packages on the sortation floor, from induct stations to eject chutes.

problems, which can be traced back to the seminal work by Tesauro et al. [17], [18]. Yet, the method in [16] is a single-agent method that cannot handle the exponential growth of the state-action space akin to large-scale resource allocation problems as the one considered here. Most relevant to the method proposed here is perhaps the work in [19], that proposes a multi-agent RL framework for ocean transportation networks. Specifically, in [19], a multi-agent Q-learning algorithm is developed where the local Q-networks depend on the joint states (including the limited shared resources) and the joint actions. However, since the joint state-action space grows exponentially with the number of agents, the local Q-networks are hard to learn and this method does not scale well in practice. Instead, here the local Q-networks are only loosely coupled, in a way that improves scalability yet allows to model robot congestion effects on the rates at which packages are sorted on the sortation floor. Moreover, compared to [19], our method proposed here models resources explicitly as actions and respects budget constraints when taking joint actions.

The rest of the paper is organized as follows. In section II, we provide preliminaries on Markov games and deep Q learning. In section III, we develop the proposed multi-agent reinforcement learning framework that can handle large numbers of agents and constrained actions. Finally, in Section IV, we validate the effectiveness of our proposed method and benchmark against static or reactive policies that are commonly used in robotic sortation floors.

II. Problem Definition

In this paper we consider the destination assignment problem (DAP), also called chute mapping problem, for robotic sortation systems, as those discussed in Section I and Fig. 1. We assume that packages arrive randomly at induct stations located at the perimeter of the sortation floor and are placed onto mobile robots, also called drives, that transport them based on their destinations to eject chutes in the interior of the sortation floor. Moreover, we assume that each destination can be serviced by multiple chutes and each chute can service a single destination.

The flow of packages on the sortation floor can be modeled using two buffers, the *laden drive buffer* and the *overflow*

buffer, as seen in Fig. 2. The laden drive buffer models the volume of packages on robot drives as they are being transported to chutes on the sortation floor. To reduce robot congestion, the number of robot drives that can concurrently transport packages to any one chute on the sortation floor is capped. Therefore, the laden drive buffer has limited capacity for packages per destination. If the number of chutes allocated to any one destination is insufficient, the sortation floor will not be able to sort the volume of incoming packages for that destination.[2] In this case, packages at the induct stations that cannot be absorbed by the laden drive buffer are picked up by robot drives and are routed to designated *overflow chutes* on the sortation floor, from where they enter an *overflow buffer* and subsequently reenter the sortation system for future processing; see Fig. 2. Our goal in this paper is to determine the optimal number of chutes per destination so that the number of unsorted packages in the overflow buffer is minimized and, therefore, the throughput of the sortation floor is maximized.

III. Method

A. Multi-Agent Reinforcement Learning Formulation

Since the rates at which packages for each destination arrive at the induct stations are unknown and can fluctuate significantly during the course of a day, we formulate the chute mapping problem described in Section II as a sequential decision making problem, specifically, a multi-agent reinforcement learning problem (MARL), that determines an optimal sequence of chute allocations that minimizes the number of packages in the overflow buffer at each time step (e.g., every 1 hour). To do so, we define a Markov game over N agents (the destinations) by a tuple $\big(N, \mathcal{S}, \{\mathcal{O}^i\}_{i=1}^N, \{\mathcal{A}^i\}_{i=1}^N, P, \{r^i\}_{i=1}^N, \gamma, \rho_0\big)$, where $\mathcal{S}$ is the joint state space, $\mathcal{O}^i \subset \mathcal{S}$ and $\mathcal{A}^i$ are the local observation and action spaces of agent i, $\gamma \in (0,1)$ is the discount factor, and ρ_0 is the initial state distribution. The local observation of each agent i at each time step is defined as the number of packages associated with this agent that are currently in the induct buffer and will be pushed into the sortation system in the next 30 minutes as well as the number of packages associated with this agent that are currently in the overflow buffer; see Fig. 2. Moreover, the local action of each agent i at each time step is the number of chutes assigned to that agent at that time step. The joint action space of all agents is defined by $\mathcal{A} = \Pi_{i=1}^N \mathcal{A}^i$. Note that all the state and action variables are discrete.

In the Markov game described above, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ denotes the state transition probability, that is the probability that packages will be sorted by the chutes or alternatively sent to the overflow buffer. To obtain the state transition probability we construct an approximate input-output model of the system described in Fig. 2, specifically, the simulation environment described in Section IV-A, where the rate at which packages enter the overflow buffer is equal to the rate at which packages arrive at the induct (new packages plus packages from overflow) minus the rate at which packages are processed by the chutes. Specifically, the rate at which packages for each destination are processed by the chutes is determined by the number of chutes assigned to each destination and the rate at which these chutes can process packages. We assume that the chutes can generally process packages at their maximum rate which, however, can be lower due to robot congestion on the sortation floor, which affects mission times and, therefore, the rates at which robots can transport packages to their assigned chutes. While we do not explicitly model robot congestion on the sortation floor, we assume that the processing rates of destinations mapped to nearby chutes on the sortation floor are coupled due to such congestion effects; see Section III-B.1 for details on how we model dependencies between destinations. Finally, $r^i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function of agent i. The reward function at each time step captures the number of chutes allocated to each destination as well as the number of packages for each destination in the overflow buffer. Large numbers of chutes per destination and packages in the overflow buffer are penalized.

At each time step t, every agent i selects an action according to a local policy $\pi^i : \mathcal{O}^i \times \mathcal{A}^i \to [0,1]$ that represents the probability of assigning a different number of chutes to destination i when observing o^i. Then, the goal of every agent i is to learn the optimal local policy $\pi^{i,*}$ that maximizes its expected future return $\mathbb{E}[R_t^i] = \mathbb{E}[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}^i]$, where $r_{t'}^i$ is the reward received at time step t' and the expectation is taken over the randomness in the stochastic local policy and the transition probability. Since the agent actions are generally coupled, to find the optimal local policies $\pi^{i,*}$ we utilize the joint action-value function $Q^\pi(s,a) = \mathbb{E}[\sum_{i=1}^N R_t^i | s_t = s, a_t = a]$ under the joint policy π, which captures the expected return that can be achieved by taking a joint action $a = (a^1, \cdots, a^N)$ at state s and following the joint policy π afterwards. In what follows, we assume that all agents take actions independently so that the policy space does not grow exponentially, i.e., $\pi = \Pi_{i=1}^N \pi^i$. Then, to find the optimal policy π^*, we employ Deep Q Network (DQN) learning [20]. Specifically, we approximate the optimal action-value function $Q^{\pi^*}(s,a)$ by a deep neural network $Q(s,a;\theta)$ with parameter θ and learn the optimal action-value function corresponding to the optimal policy by minimizing the loss function

$$L(\theta) = \mathbb{E}_{s,a,r,s'}[(Q(s,a;\theta) - y)^2], \tag{1}$$

where $y = r + \gamma \max_{a'} \bar{Q}(s', a'; \bar{\theta})$ approximates the optimal target values $r + \gamma \max_{a'} Q^{\pi^*}(s', a')$. $\bar{Q}$ is a separate target Q network that is periodically updated using the most recent values for the parameter θ, which helps to stabilize the learning process. To further stabilize learning, we also utilize a replay buffer $\mathcal{D}$ that contains multiple transition

[2]Note that the required number of chutes per destination depends not only on the corresponding volume of incoming packages but also on the ability of the laden drive buffer to empty itself. Generally, the longer packages remain in the laden drive buffer, the more chutes are needed to avoid package overflows. This is the case, e.g., in the presence of robot congestion that increases the mission times of robot drives on the sortation floor and, therefore, the time packages spend in the laden drive buffer.

pairs (s, a, r, s') to compute the expectation in (1). Then, the optimal policy can be found as $\pi^*(s, a; \theta) = \frac{1}{|\mathcal{A}(s)|}$ if $a \in \mathcal{A}(s)$ and $\pi^*(s, a; \theta) = 0$ if $a \notin \mathcal{A}(s)$, where $\mathcal{A}(s) = \arg\max_a Q(s, a; \theta^*)$ and $\theta^* = \arg\min L(\theta)$.

B. Networked Value Decomposition Network (NVDN)

There are two main challenges in using (1) to learn the optimal action-value function for the chute mapping problem under consideration. First, the state-action space grows exponentially with the number of agents. A simple remedy to this problem is learning separate Q networks independently for each agent. Nevertheless, since the agents update their policies independently as they learn, the environment is no longer stable from the perspective of each agent and thus the Markov assumption is violated. As a result, individual Q network approaches may diverge [21]. The second challenge is ensuring that the data in the replay buffer used to calculate the expectation in (1) are feasible. This requirement is particularly relevant in the chute mapping problem considered here, since the agents share limited actions (numbers of available chutes). In what follows, we propose a Networked Value Decomposition Network (VDN) method to address these challenges and solve the proposed multi-agent RL problem.

1) Dimension Reduction of the State-Action Space: To limit the dimension of the state-action space, we assume that the joint Q network that captures the expected return of the joint chute mapping action (chute assignment for all agents) can be decomposed as the sum of local Q networks that capture the expected return of local chute mapping actions (chute assignment to individual agents). Specifically, let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ denote a directed graph that measures the dependencies among agents, where $\mathcal{N}$ is the set of agents and $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ is the set of edges, and let $G^i = \{j \in \mathcal{N} | (j, i) \in \mathcal{E}\}$ denote the set of agents that can influence the observations and/or rewards of agent i. Then, we assume that the joint Q network can be written as

$$Q(s, a, \{\theta^i\}_{i=1}^N) = \sum\nolimits_{i=1}^{N} Q^i(o^i, a^i, o^{G^i}; \theta^i), \qquad (2)$$

where $o^{G^i} = \{o^j\}_{j \in G^i}$ is the collection of local observations of all agents in the set G^i. As a result, the input space only grows linearly with the number of agents.

We note that the construction in (2) generalizes the notion of value decomposition networks (VDN) in [22], where the local Q^i networks only depend on local observations and actions as $Q(s, a) = \sum_{i=1}^N Q^i(o^i, a^i)$. A limitation of VDN is that if, e.g., the rewards of the local agents also depend on the observations of their neighbors (along with their own observations), then the local Q^i networks cannot learn the correct state-action values due to the missing observations. We call the method proposed here networked VDN (NVDN).

We also note that, in the chute mapping problem considered here, the proposed networked VDN method allows to model practical dependencies between the agents. Specifically, the directed graph $\mathcal{G}$ can model the effect of the joint chute mapping decisions on the rates at which the chutes can process the packages of their assigned destinations. For example, the assignment of destinations to chutes on the sortation floor can give rise to high-traffic areas (robot congestion) that can affect the rates at which chutes in these areas can process packages for their assigned destinations; see the simulation environment described in Section IV-A.[3]

2) Feasibility of Joint Actions: When the actions are unconstrained, the agents will select the actions that maximize their individual Q networks; the joint action is a collection of all agents' actions. However, in the chute mapping problem considered here, agents share limited actions (available chutes). Thus, coordination among agents is required to assign more resources to agents with higher state-action values. Specifically, given a limited budget M on the joint actions, we require that the joint action satisfies $\sum_{i=1}^N a_i \leq M$. Then, to select the best joint action that maximizes the joint Q network while satisfying the required budget constraints, we solve the following integer program for any given state s:

$$\begin{aligned} \max_{a^1, \cdots, a^N} \ & \sum\nolimits_{i=1}^{N} Q^i(o^i, a^i, o^{G^i}; \theta^i), \\ \text{s.t.} \ & \sum\nolimits_{i=1}^{N} a^i \leq M, \ a^i \in \mathbb{N}. \end{aligned} \qquad (3)$$

Note that Q^i is a vector with the same size as the action space a^i for any given state. Moreover, this integer program does not involve any Q learning steps and can be solved very efficiently using dynamic programming or commercial solvers, e.g., FICO Xpress [24]. The actions obtained by the solution of problem (3) are used to generate feasible data in the replay buffer to compute the expectation in (1) during each learning step. Also, the solution of (3) provides the optimal actions after learning has converged. The full algorithm is presented in Algorithm 1.

Note that the proposed method can be easily extended to account for additional operational constrains including, e.g., limits on the agents' consecutive action variations by adding the constraints $|a_t^i - a_{t-1}^i| \leq \delta$ for some $\delta \in \mathbb{N}$.

IV. Numerical Experiments

A. Simulator Setup

To validate Algorithm 1 we developed a simplified warehouse simulator that at a high level captures the main operations that take place in an Amazon sortation center, as summarized in Fig. 2. Specifically, we assume that every hour (a time step) a total of 20,000 new packages arrive at the induct stations (induct buffer data) for $N = 100$ different destinations. Every 6 hours this number is reduced to 15,000 to capture package variations due to upstream tasks related to human activity or logistic operations, e.g., breaks between shifts. Such variations are common in production data. The hourly number of packages per destination is sampled from a generative model that captures similar patterns as those observed in real induct buffer data, e.g., patterns on the number of packages during peak time. Fig. 3 shows the hourly

[3]Note that in many RL applications, dependencies between agents cannot be easily described by a graph using prior knowledge. An example is collaboration between teams in games such as StarCraft [23]. The chute mapping problem in this sense is a special case of multi-agent RL problems.

Algorithm 1 Networked VDN with Budget Constraints

Input: Number of agents N, budget on actions M

Initialize replay buffer $\mathcal{D}^i$, action-value function Q^i, and target action-value function $\bar{Q}^i$, for all agent $i = 1, \cdots, N$
for episode $= 1, \ldots, m$ **do**
  Observe an initial observation o_0^i for all agents i
  for step $t = 0, \cdots, T$ **do**
    With probability ϵ select a random action a_t such that $\sum_{i=1}^{N} a_t^i = M$; otherwise, select a_t according to (3)
    Execute action a_t and observe r_t^i and o_{t+1}^i for all agents
    for each agent $i = 1, ..., N$ **do**
      Collect local observations $(o_t^{G_i}, o_{t+1}^{G_i})$ from neighbors and store local transition $(o_t^i, o_t^{G_i}, a_t^i, r_t^i, o_{t+1}^i, o_{t+1}^{G_i})$ in the replay buffer $\mathcal{D}^i$
      Sample a minibatch of transitions $(o_t^i, o_t^{G_i}, a_t^i, r_t^i, o_{t+1}^i, o_{t+1}^{G_i})$ from $\mathcal{D}^i$
      Set $y_t^i = r_t^i$ if episode terminates at $t+1$ and otherwise set $y_t^i = r_t^i + \gamma \max_{a'} \bar{Q}(o_{t+1}^i, o_{t+1}^{G_i}, a'; \bar{\theta}^i)$
      Run a gradient descent step on the loss $(y_t^i - Q(o_t^i, o_t^{G_i}, a_t^i; \theta^i))^2$ with respect to the local policy parameters θ^i
      Set $\bar{\theta}^i = \theta^i$ every c steps

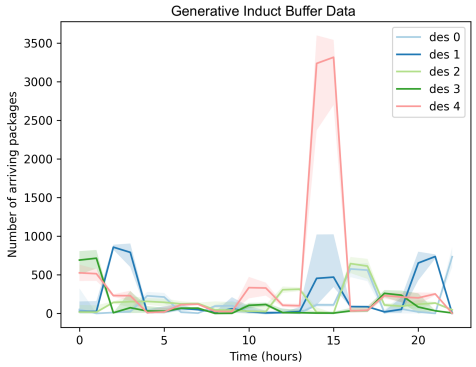

Fig. 3. Hourly numbers of incoming packages for the first five destinations calculated from 30 samples (days). The solid lines represent averages and the shades represent $\pm$ standard deviations.

average number of packages for the first five destinations sampled from the above generative model.

We assume there are 440 destination-free chutes on the sortation floor, each one of which can be assigned to only one destination and each one of which can process at most 100 packages per hour. To reduce the computational cost of the proposed RL algorithm, we partition the total 440 chutes into 340 static chutes and $M = 100$ dynamic chutes, so that the assignment of destinations to static chutes remains unchanged over the whole simulation and only the assignment of destinations to dynamic chutes changes using Algorithm 1.[4] Specifically, we define the static chute map as

$$a_{\text{static}}^i = \lceil (\alpha \cdot \texttt{mean}(\{n^i\}) + \beta \cdot \texttt{std}(\{n^i\})) \rceil,$$

where $\{n^i\}$ denotes the samples obtained by the generative model for each destination i and $\lceil \cdot \rceil$ is the ceiling function. We appropriately select α and β so that $\sum_i a_{\text{static}}^i = 340$.

[4]The idea is that static chutes absorb average volumes and dynamic chutes are used to absorb large deviations from the steady state.

Note that the static chute map, in general, may not be able to service destinations during peak hours. For example, in Fig. 3, over 3,000 packages arrive around time step 15 for destination 4. The static map assigns only 11 chutes to destination 4 that can collectively process up to 1,100 packages per hour, which is much lower than 3,000.

As discussed in Section III, we assume that the processing rates of the chutes can be negatively affected by robot congestion on the sortation floor. Since we assume that the packages for each destination arrive uniformly at random at the induct stations on the perimeter of the sortation floor, we assign destinations to the static chutes so that the sum of pairwise Manhattan distances between all chutes assigned to every individual destination is maximized. Spreading the chutes assigned to individual destinations over the sortation floor reduces the chances of robot congestion due to volume surges for specific destinations. Fig. 4 shows the floor layout with 340 static chutes assigned to 100 destinations. In this static map, the *neighbors* of any destination i are all destinations j with chutes located directly North, South, East, and West of any chute assigned to destination i. Since this neighbor set can be very large (for many destinations) we remove from the neighbor set of destination i any destinations j that have only few chutes adjacent to chutes assigned to destination i. Effectively, two destinations are neighbors if their assigned chutes are next to each other frequently enough. Therefore, we can define the graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ in Section III-B that measures the dependencies among agents. Using this dependency graph, the processing rate of each chute assigned to destination i is defined as

$$t_i = 100 - \lceil (\sum\nolimits_{j \in G_i} x_j)/\gamma \rceil,$$

where x_j denotes the packages arriving at the induct stations for destination j during the next hour, $\gamma > 0$ is a parameter selected so that the rates t_i obtain values in a range that is typical during operation of the sortation floor (here $\gamma = 200$), and $G^i = \{j \in \mathcal{N} | (j, i) \in \mathcal{E}\}$ is the neighbor set of agent i defined in Section III-B.1. Note that the rate t_i is not the true rate at which the individual chutes assigned to destination i process packages. The rate p_i is an approximate average processing rate for destination i that can be different from the true processing rates of the individual chutes.

During training of the RL policy, we assume the processing rate of each destination is only affected by its neighbors in the static map. Otherwise, the graph $\mathcal{G}$ will change and become a state variable in the RL problem that will significantly increase the dimension of the state space and affect the complexity of learning. However, during testing of the learned RL policies, the processing rates depend on both the static and dynamic maps. Therefore, there exists a shift between the training and testing environments as a result of the different processing rates used for training and testing. Fig. 5 shows the average processing rate for the first two destinations. In what follows, we show that the policies trained on the static map are robust

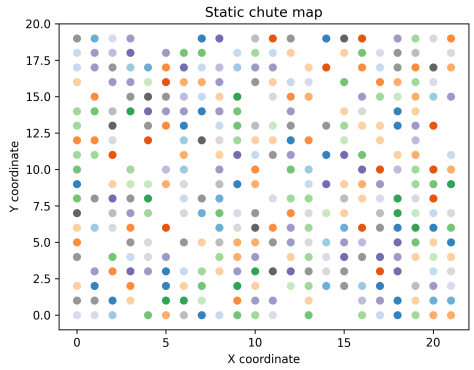

Fig. 4. The sortation floor layout. The colored dots represent the static chute map, where each unique color corresponds to a unique destination. Empty positions are reserved for dynamic chutes.

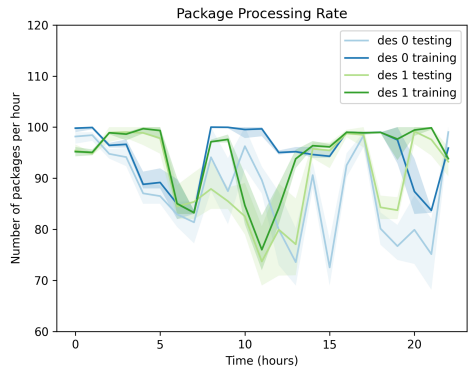

Fig. 5. Processing rates of the first two destinations calculated from 30 samples.The solid lines represent averages and the shades represent $\pm$ standard deviations.

to this shift and transfer well to the dynamic map too.[5]

As discussed in Section III, packages that exceed the chutes' capacities are sent to the overflow buffer and added to the sequence of new packages of the next time step. One episode (day) consists of 24 time steps (hours), after which the environment is reset. The observation of each agent consists of the sum of the first 10,000 packages from induct buffer data o^i, i.e., partial observation, and the number of packages in the overflow buffer b^i. The individual reward is defined by $r^i = -b^i - a^i$, where a^i is the number of dynamic chutes being used.

B. Results

We first compare our RL policy to a static policy, commonly used in warehouse sortation facilities, that assigns the dynamic chutes to destinations in a similar way as the static policy a^i_{static} defined above. In Fig. 6, we observe that this static policy maintains over 25,000 unsorted packages per hour, since it cannot adapt to varying demands. As a result, unsorted packages accumulate over time burdening the overflow buffer and decreasing throughput. This is expected as the static policy does not use data from induct or the overflow buffer.

[5]We place the dynamic chutes on the sortation floor in a similar way, so that the sum of pairwise Manhattan distances between the chutes assigned to each destination is maximized.

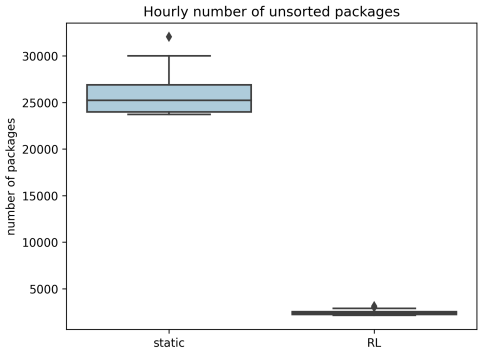

Fig. 6. Comparison between the static policy and the RL policy. (Statistics are obtained from 30 runs.)

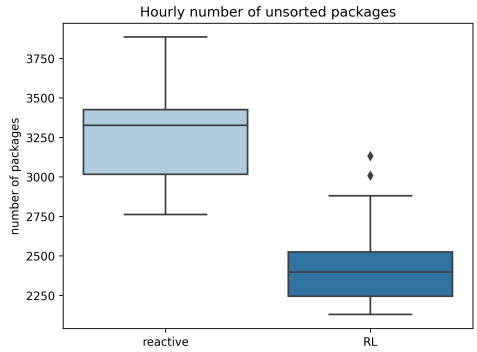

Fig. 7. Comparison between the reactive policy and the RL policy. (Statistics are obtained from 30 runs.)

Alternatively, we also compare to a reactive policy that assigns chutes in proportion to the number of packages in the overflow buffer b^i and the number of incoming packages o^i obtained from partial induct buffer data (same as the RL policy). Specifically, we define the probability of assigning chute j to agent i as

$$p(i) = \frac{\max\{\frac{\lambda(b^i+o^i)}{100} - a^i_{\text{static}}, 0\}}{\sum_{k=1}^{N} \max\{\frac{\lambda(b^k+o^k)}{100} - a^k_{\text{static}}, 0\}}.$$

Therefore, we can obtain a reactive chute map by sampling from this distribution up to the allowable budget $M = 100$ of dynamic chutes. Note that destinations with a larger number of packages will be assigned more chutes in expectation. In the experiments, we selected the value of λ that returned the reactive policy with the best performance. A comparison with the proposed RL policy is shown in Fig. 7. We observe that the RL policy outperforms the reactive policy, which is expected since the RL policy is optimized over a larger functional space compared to the linear space that defines the reactive policy. In addition, the RL policy is learned by maximizing the expected accumulated future rewards while the reactive policy is a pre-determined policy that does not predict the number of packages in the future. More importantly, computation of the RL policy is fast, similar to computation of the static and reactive policies, as it only requires evaluation of a trained neural network and solution of a simple integer program as in (3).

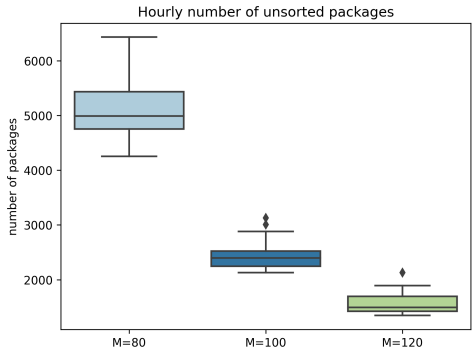

Fig. 8. Comparison between RL policies with different budgets. Statistics are obtained over 30 runs.

Finally, we show that the proposed RL policy can be transferred to environments with different budget constraints $M = \{80, 120\}$ without retraining (zero shot transfer) from the training environment $M = 100$. This owes to the nature of the proposed algorithm where the chutes are assigned to destinations independently via the solution of the integer program in (3), as opposed to directly from the trained local Q networks. The results are shown in Fig. 8. Note that the dynamic chute map with a smaller budget is not as capable as the one with a larger budget due to fewer available resources. As a result, the number of unsorted packages increases as the budget M decreases.

V. Conclusion

In this work, we proposed a multi-agent reinforcement learning method to solve chute mapping problems in warehouse sortation centers. Specifically, we formulated a networked decomposition network framework where the joint action-value function is the sum of all local agents' action-value functions. We showed that our proposed framework can solve large chute mapping problems and outperforms static or reactive policies that are commonly used in practice in robotic sortation facilities. It is also transferable to different sortation environments.

References

[1] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI magazine*, vol. 29, no. 1, pp. 9–9, 2008.

[2] K. Azadeh, R. De Koster, and D. Roy, "Robotized and automated warehouse systems: Review and recent developments," *Transportation Science*, vol. 53, no. 4, pp. 917–945, 2019.

[3] Amazon, "How amazon robots navigate congestion," https://www.amazon.science/latest-news/how-amazon-robots-navigate-congestion, 2022.

[4] ——, "10 years of amazon robotics: how robots help sort packages, move product, and improve safety," https://www.aboutamazon.com/news/operations/10-years-of-amazon-robotics-how-robots-help-sort-packages-move-product-and-improve-safety, 2022, accessed: 2023-01-23.

[5] ——, "Tour an amazon fulfillment center," https://www.aboutamazon.com/workplace/tours, 2023, accessed: 2023-01-23.

[6] S. S. Heragu, X. Cai, A. Krishnamurthy, and C. J. Malmborg, "Analytical models for analysis of automated warehouse material handling systems," *International Journal of Production Research*, vol. 49, no. 22, pp. 6833–6861, 2011.

[7] D. Roy, "Semi-open queuing networks: a review of stochastic models, solution methods and new research areas," *International Journal of Production Research*, vol. 54, no. 6, pp. 1735–1752, 2016.

[8] T. Lamballais, D. Roy, and M. De Koster, "Estimating performance in a robotic mobile fulfillment system," *European Journal of Operational Research*, vol. 256, no. 3, pp. 976–990, 2017.

[9] M. Yu and R. B. De Koster, "The impact of order batching and picking area zoning on order picking system performance," *European Journal of Operational Research*, vol. 198, no. 2, pp. 480–490, 2009.

[10] B. Zou, R. De Koster, Y. Gong, X. Xu, and G. Shen, "Robotic sorting systems: Performance estimation and operating policies analysis," *Transportation Science*, vol. 55, no. 6, pp. 1430–1455, 2021.

[11] N. Boysen and M. Fliedner, "Cross dock scheduling: Classification, literature review and research agenda," *Omega*, vol. 38, no. 6, pp. 413–422, 2010.

[12] S. Fedtke and N. Boysen, "Layout planning of sortation conveyors in parcel distribution centers," *Transportation Science*, vol. 51, no. 1, pp. 3–18, 2017.

[13] L. J. Novoa, A. I. Jarrah, and D. P. Morton, "Flow balancing with uncertain demand for automated package sorting centers," *Transportation Science*, vol. 52, no. 1, pp. 210–227, 2018.

[14] R. Khir, A. Erera, and A. Toriello, "Two-stage sort planning for express parcel delivery," *IISE Transactions*, vol. 53, no. 12, pp. 1353–1368, 2021.

[15] ——, "Robust planning of sorting operations in express delivery systems," *European Journal of Operational Research*, 2022.

[16] B. Du, C. Wu, and Z. Huang, "Learning resource allocation and pricing for cloud profit maximization," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 7570–7577.

[17] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White, "A multi-agent systems approach to autonomic computing," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 2004, pp. 464–471.

[18] G. Tesauro *et al.*, "Online resource allocation using decompositional reinforcement learning," in *AAAI*, vol. 5, 2005, pp. 886–891.

[19] X. Li, J. Zhang, J. Bian, Y. Tong, and T.-Y. Liu, "A cooperative multi-agent reinforcement learning framework for resource balancing in complex logistics network," *arXiv preprint arXiv:1903.00714*, 2019.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[21] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[22] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-decomposition networks for cooperative multi-agent learning," *arXiv preprint arXiv:1706.05296*, 2017.

[23] M. Samvelyan, T. Rashid, C. S. De Witt, G. Farquhar, N. Nardelli, T. G. Rudner, C.-M. Hung, P. H. Torr, J. Foerster, and S. Whiteson, "The starcraft multi-agent challenge," *arXiv preprint arXiv:1902.04043*, 2019.

[24] FICO, "Xpress optimizer," https://www.fico.com/en/products/fico-xpress-optimization, 2023, accessed: 2023-01-23.