Multi-Agent Reinforcement Learning with Hierarchical Coordination for Emergency Responder Stationing

Amutheezan Sivagnanam [1] **Ava Pettet** [2] **Hunter Lee** [2] **Ayan Mukhopadhyay** [2] **Abhishek Dubey** [2] **Aron Laszka** [1]

Abstract

An emergency responder management (ERM) system dispatches responders, such as ambulances, when it receives requests for medical aid. ERM systems can also proactively reposition responders between predesignated waiting locations to cover any gaps that arise due to the prior dispatch of responders or significant changes in the distribution of anticipated requests. Optimal repositioning is computationally challenging due to the exponential number of ways to allocate responders between locations and the uncertainty in future requests. The state-of-the-art approach in proactive repositioning is a hierarchical approach based on spatial decomposition and online Monte Carlo tree search, which may require minutes of computation for each decision in a domain where seconds can save lives. We address the issue of long decision times by introducing a novel reinforcement learning (RL) approach, based on the same hierarchical decomposition, but replacing online search with learning. To address the computational challenges posed by large, variable-dimensional, and discrete state and action spaces, we propose: (1) actor-critic based agents that incorporate transformers to handle variable-dimensional states and actions, (2) projections to fixed-dimensional observations to handle complex states, and (3) combinatorial techniques to map continuous actions to discrete allocations. We evaluate our approach using real-world data from two U.S. cities, Nashville, TN and Seattle, WA. Our experiments show that compared to the state of the art, our approach reduces computation time per decision by three orders of magnitude, while also slightly reducing average ambulance response time by 5 seconds.

[1]Pennsylvania State University, University Park, PA, USA [2]Vanderbilt University, Nashville, TN, USA. Correspondence to: Amutheezan Sivagnanam <aqs7319@psu.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

1. Introduction

Dynamically repositioning resources under uncertainty is an important problem in societal-scale cyber-physical systems (Pettet et al., 2022), such as bike repositioning (Li et al., 2018), ride-hailing and public transit (Jin et al., 2019; Xi et al., 2021; Talusan et al., 2024), and emergency response management (ERM) (Mukhopadhyay et al., 2020). In such problems, a decision-maker must sequentially optimize the allocation of resources in space and time to respond to uncertain demand (e.g., calls for service). We focus specifically on ERM, a critical problem faced by urban communities across the globe. Indeed, 240 million emergency medical service calls are made in the U.S. alone each year (Mukhopadhyay et al., 2020). Whenever a request for medical aid is reported to an ERM, a responder is dispatched to the scene of the incident. Responders administer critical services on the scene (e.g., basic life support), transfer the patient to the nearest hospital, and head back to their assigned waiting locations (called *depots*) to wait until their next dispatch. Due to the critical nature of emergency medical aid, dispatching decisions are typically constrained to greedy policies that send the nearest available responder (Mukhopadhyay et al., 2020). However, it is possible to proactively optimize the waiting locations of the responders so that expected response times for future incidents are minimized (Pettet et al., 2022; 2020).

The problem of proactive repositioning is computationally challenging due to the uncertainty in future demand and the combinatorial state-action space of the problem—the number of possible responder assignments grows exponentially with the number of responders. For example, in one of our experimental settings, Nashville, TN, the number of possible allocations at each decision epoch is on the order of 10^{33}. Prior works use *centralized*, *decentralized*, and *hierarchical* approaches to solve the ERM reallocation problem. (Ji et al., 2019) propose a learning-based approach for centralized reallocation, which can reallocate a responder each time it finishes serving a request. Unfortunately, such centralized solutions with a monolithic state-action space (Mukhopadhyay et al., 2018; Ji et al., 2019) do not scale to large ERM systems. Decentralized approaches (Pettet et al., 2020), on the other hand, split the state-action space such that each responder makes its own decisions, sacrificing coordination between the responders to achieve scalability, which can lead to sub-optimal decisions.

The state-of-the-art approach (Pettet et al., 2022) applies hierarchical planning to the responder allocation problem, which in principle lies midway between the centralized and decentralized approaches. This approach first partitions the spatial area under consideration into smaller and more manageable areas called *regions*. A *low-level planner* uses Monte-Carlo tree search (MCTS), an online technique, to solve each region-specific problem independently. A *high-level planner* allocates responders between the regions based on expected demand. (Pettet et al., 2022)'s hierarchical approach alleviates scalability concerns while maintaining coordination between nearby responders, and *comprehensively outperforms other methods in this domain*. However, the framework's use of MCTS for low-level planning requires running extensive simulations at decision time, which need significant computation time for decision-making (between 2–4 minutes). This is infeasible in practice since delays in reallocation during a coverage gap (when an area does not have any available responders) could be catastrophic in ERM. This raises an interesting research problem: *How can we reduce decision time to avoid delays in making reallocation decisions without sacrificing solution quality compared to the current state of the art?*

We address this problem by introducing a novel learning-based approach that can make reallocation decisions in a fraction of a second. However, applying a learning-based approach directly to the problem formulation of (Pettet et al., 2022) is challenging since: **(1)** even with the spatial decomposition, both *state and action spaces are high-dimensional and discrete*; **(2)** low-level planners must be able to handle *variable-dimensional state and action spaces* as the number of responders in a region may vary over time; and **(3)** *rewards are very noisy*, especially for the high-level planner, since response times can vary widely depending on the locations and times of incidents. We tackle these challenges systematically in this paper.

Specifically, we make the following contributions. **(1)** We introduce a multi-agent reinforcement-learning with hierarchical coordination for responder repositioning by replacing the high- and low-level planners of (Pettet et al., 2022)'s framework with *actor-critic based agents that can handle complex, high-dimensional action spaces*. **(2)** To handle complex, high-dimensional states, we introduce *projections from states to low-dimensional features*, which capture relevant state information. **(3)** We incorporate *transformers into low-level agents to handle variable-dimensional states and actions*. **(4)** To facilitate gradient-based actor training, we introduce *efficient combinatorial optimizations that take continuous actions from an actor and map them to discrete allocations*. **(5)** To reduce noise in the high-level agent's rewards, we *estimate its rewards using the low-level agents' critics*. **(6)** We evaluate our approach using real-world data from Nashville and Seattle, two cities in the U.S., and show that our approach not only *reduces the computational time per decision by multiple orders of magnitude* but also slightly reduces ambulance response times compared to the state of the art.

2. Problem Formulation

We begin by introducing the assumptions of the ERM reallocation problem and modeling it as a continuous-time Markov decision process. Then, in Section 2.2, we describe the hierarchical decision-making framework that enables tractable decision-making. This problem formulation and hierarchical framework are based on state-of-the-art prior work (Pettet et al., 2022; Mukhopadhyay et al., 2020). We introduce our novel learning-based framework in Section 3.

2.1. Model

An ERM system manages a set of resonders to serve requests[1] for medical aid that are distributed over space and time and are unknown in advance. The ERM's spatial area of operation is divided into a grid of equally sized square cells ($\mathcal{C}$). Requests for medical aid follow a spatio-temporal probability distribution that can be modeled using independent Poisson distributions for each cell. For a specific cell $c \in \mathcal{C}$, we denote the expected rate of incident occurrence at time t for some duration (e.g., over the next hour) by λ_t^c.

The ERM system consists of the following components: a set of responders ($\mathcal{V}$) to serve requests, a fixed set of spatially-located depots ($\mathcal{D}$) where responders idle between serving requests, and a fixed set of hospitals ($\mathcal{H}$) where patients are taken after being picked up at a request's location. When a request is reported, the ERM system assigns the nearest available responder to service it (this is governed by the practical constraints that ERM operators face). If no responder is available, the incident enters a waiting queue. Once a responder arrives at the request's location, it treats the patient on-scene for a fixed time (t_{serve}), after which it transports the patient to the nearest hospital. The movement of responders between cells follows a time-varying travel model ($\mathcal{M}$): the time taken to move from cell $c_i \in \mathcal{C}$ to cell $c_j \in \mathcal{C}$ at time t is given by $\mathcal{M}(c_i, c_j, t)$. After servicing a request, the responder is either immediately dispatched to the request at the top of the waiting queue or returns to an empty depot to wait if the queue is empty.

We model the ERM's stochastic decision-making problem of dynamically reallocating responders ($\mathcal{V}$) to depots ($\mathcal{D}$) in anticipation of future incidents as a continuous-time Markov

[1]We use *requests* and *incidents* synonymously.

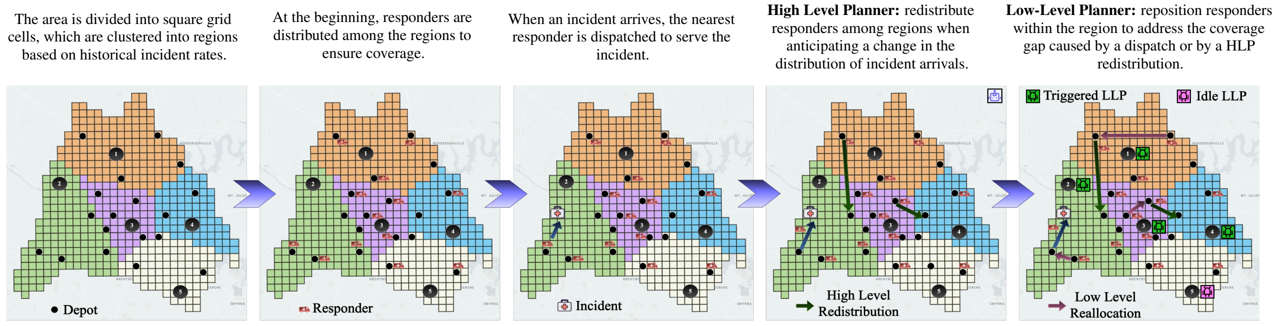

The area is divided into square grid cells, which are clustered into regions based on historical incident rates.
At the beginning, responders are distributed among the regions to ensure coverage.
When an incident arrives, the nearest responder is dispatched to serve the incident.
High Level Planner: redistribute responders among regions when anticipating a change in the distribution of incident arrivals.
Low-Level Planner: reposition responders within the region to address the coverage gap caused by a dispatch or by a HLP redistribution.
Depot
Responder
Incident
High Level Redistribution
Triggered LLP
Idle LLP
Low Level Reallocation

Figure 1: High-level overview of state-of-the-art hierarchical framework (Pettet et al., 2022), described in Section 2.2.

decision process (MDP). The objective of the MDP is to reduce the system's expected incident response time (i.e., cumulative time taken to reach the scene of each incident after it is reported). We describe the MDP below.

State At time t, the system state is denoted by s_t and consists of the set of incidents waiting for service $\mathcal{I}_t$, the incident rates $\lambda_t^c, \forall c \in \mathcal{C}$ of the cells, and the state $\mathcal{P}_t$ of the responders. For each responder $v \in \mathcal{V}$, the state $p_t^v \in \mathcal{P}_t$ is a tuple $\langle d^v, i^v, h^v, c^v, t^v_{avail} \rangle$, where $d^v \in \mathcal{C}$ is the depot where responder v is assigned to wait when it is not serving an incident; $i^v \in \mathcal{C}$ is the location of the incident to which responder v is currently assigned; $h^v \in \mathcal{C}$ is the location of the hospital to which responder v is currently taking a patient; $c^v \in \mathcal{C}$ is responder v's current location; and $t^v_{avail} \in \mathbb{R}^+$ is the point in time at which responder v will become available, i.e., when it will drop off the patient at the assigned hospital. Variables i^v, h^v, and t^v_{avail} are empty if responder v is not assigned to an incident at time t.

Transition The environment has two types of events: incident occurrences and incident rate changes. We define two types of decision epochs based on these events: (1) when a responder is dispatched to serve an incident and (2) when changes in incident rates are detected. Between two decision epochs, the state of the system evolves as responders serve their incidents and move to their assigned depots according to the travel model $\mathcal{M}$. We assume the assignment of responders to depots does not change between consecutive decision epochs since no new information becomes available.

Action An action is the reallocation of some responders $v \in \mathcal{V}$ from their currently assigned depot d_t^v to a different depot d_{t+1}^v ($d_t^v \neq d_{t+1}^v$) at decision epoch t.

Reward If the next decision epoch is due to the arrival of an incident, then the reward for the last action is the response time for the incident. On the other hand, if the next decision epoch is due to a change in the incident rates, in which case there is no incident to serve, the reward is 0. Note that in the ERM setting, the goal is to find an allocation that minimizes the expected incident response times. Therefore, despite using the standard term "reward," the objective is to minimize the expected discounted rewards.

2.2. Hierarchical Decision Framework

A key challenge to solving the MDP defined in Section 2.1 is the combinatorial nature of the state and action spaces. We address this scalability challenge by using the hierarchical decision-making framework introduced by (Pettet et al., 2022). We assume that the ERM's spatial area has been divided into a set of smaller, more manageable regions $\mathcal{G}$ based on the historical incident rates (using the same approach as (Pettet et al., 2022)), with $\mathcal{C}^g$ denoting the cells assigned to region $g \in \mathcal{G}$. Decision-making for these regions is decomposed into two stages: high-level and low-level decision making. First, the high-level decision agent distributes the available responders between the regions. We denote the region to which responder v is assigned as $g^v \in \mathcal{G}$. Then, low-level agents optimize the responder assignments independently within each region. This significantly reduces the complexity of each region's assignment subproblem, as the low-level agents need to consider only the interactions between responders and depots within a single region.

In this hierarchical decision-making framework, the following general procedure is followed each time an incident is reported: (1) The nearest available responder is dispatched. (2) The high-level planner decides if the allocation of responders to regions is unbalanced, and if so, actuates an appropriate redistribution of responders among the regions. (3) The low-level planner is invoked for every region if the high-level planner changed the region distribution; otherwise, it is invoked only for the region from which the responder was dispatched (in the first step) to address any coverage gaps that arose due to the dispatch. Figure 1 shows a high-level overview of the hierarchical framework.

3. Solution Approach

Now, we introduce our novel learning-based approach for proactive repositioning in ERM. We utilize the hierarchical framework from the state-of-the-art approach (Pettet et al., 2022), but we replace both the *high-level planner* (HLP) and the *low-level planners* (LLPs) with deep reinforcement learning agents to overcome the long decision-making time of online search. Due to the critical nature of ERM, any additional time spent on planning can have a negative impact on serving future incidents (Jaldell et al., 2014). However, the application of reinforcement learning faces several computational obstacles. Even after the hierarchical decomposition, action spaces remain vast, which poses challenges for finding optimal actions. We propose actor-critic-based agents for both the HLP and the LLPs, specifically, the DDPG algorithm (Lillicrap et al., 2015), since a trained actor can choose an action at a very low computational cost. In Appendix G, we explain the rationale behind our choice of applying DDPG in more detail. However, this leads to another challenge since such agents are ill-suited for discrete action spaces; we address this by letting actors choose continuous actions, which we map to discrete allocations using efficient combinatorial optimization: minimum-cost flow for HLP and maximum-weight matching for LLPs.

Similarly, state spaces remain vast even after decomposition; hence, we map states to low-dimensional feature vectors, which capture relevant information, for both LLPs and the HLP. LLPs also face the challenge of variable-dimensional state and action spaces due to the varying number of responders in a region; we tackle this by incorporating transformers into the actors. Finally, the HLP faces the challenge of noisy rewards that are weakly correlated to its actions since each response time depends on only one of many regions (i.e., the region where the incident occurred); we address this by estimating HLP rewards using LLP critics.

3.1. Low-Level Decision Agent: Reallocating Responders within a Region

We first introduce an MDP formulation of the problem of repositioning responders *within* a region by a low-level planner. Then, we explain how we build on the DDPG algorithm (Lillicrap et al., 2015) to train an agent. Figure 2 provides an overview of the architecture of our low-level agent.

We formulate each region's reallocation problem as an MDP as follows: **State:** The state s_t^g of region g consists of the incident rates of the cells in region g ($\lambda_t^c, \forall c \in \mathcal{C}^g$) and the states of responders currently assigned to g ($\mathcal{P}_t^g = \{p_t^v \mid p_t^v \in \mathcal{P}_t \wedge v \in \mathcal{V}^g\}$). **Transition:** We consider two decision epochs: (1) when a responder assigned to region g is dispatched to a request, and (2) when the set of responders assigned to g is changed by the high-level agent (described in detail in Section 3.2). **Action:** An action $\mathcal{A}[g]$ is a repositioning of responders between depots in the region g. **Reward:** We consider the same reward as the original MDP, i.e., response time, but calculated only for incidents served by the responders in region g.

Actor Input To implement low-level planning, we apply actor-critic based RL to the MDP above. To tackle the challenge of high-dimensional state space, we transform the region state s_t^g into the following two sets of features:

- *Arrival time* $\phi_t[d, v]$: total time that each responder $v \in \mathcal{V}^g$ would take to reach each depot $d \in \mathcal{D}^g$ if responder $v \in \mathcal{V}^g$ were assigned to idle at depot d after completing its current task. We let $\phi_t[d, v] = 0$ if responder v is currently at depot d. Intuitively, $\phi_t[d, v]$ captures how soon responder v would be ready to serve incidents nearby depot d. Arrival time $\phi_t[d, v]$ is computed as:
$$\phi_t[d, v] = \begin{cases} \mathcal{M}(c^v, d, t) & \text{if } t^v_{avail} < t \\ t^v_{avail} - t + \mathcal{M}(h^v, d, t^v_{avail}) & \text{otherwise.} \end{cases}$$
- *Nearby incident rate* λ_t^d: sum of the incident rates λ_t^c at time t for cells $c \in \mathcal{C}^g$ that are *near* depot d. Specifically, $\lambda_t^d = \sum_{c \,\in\, \mathbf{NearCells}(d,t)} \lambda_t^c$, where $\mathbf{NearCells}(d, t)$ is the set of cells for which d is the closest depot, and is computed as $\{c | c \in \mathcal{C}, \text{argmin}_{\hat{d} \in \mathcal{D}} \mathcal{M}(c, \hat{d}, t) = d\}$. Intuitively, λ_t^d estimates the future demand for which depot d is likely to be "responsible."

Actor Network We describe the key elements of the actor network here; see Appendix B.1 for a more detailed description. We feed a sequence of feature vectors $\langle \phi_t[d, v], \lambda_t^d \mid \forall d \in \mathcal{D}^g \rangle$, one feature vector for each responder $v \in \mathcal{V}^g$, into the actor network to obtain a sequence of likelihood vectors $\mathbf{a}_t^g[v]$, one likelihood vector for each responder $v \in \mathcal{V}^g$. The actor network is based on Transformer-XL (Dai et al., 2019), consisting of N sequential TrXL layers, which enable "coordination" between responders based on their input features. We apply *softmax* activation after the TrXL layers to output likelihood values $\mathbf{a}_t^g[v]$, where $\mathbf{a}_t^g[v] \cdot 1 = 1$, $\mathbf{a}_t^g[v] \geq 0$, which assign responder v to depots in the region. We combine the likelihood vectors of all the responders in the region to obtain the continuous action $\mathbf{a}_t^g$.

Discrete Action Since the output $\mathbf{a}_t^g$ of the actor network is continuous, it is necessary to discretize actions for actual allocation. We compute a discrete assignment of responders to depots ($\mathcal{A}[g]$) by finding a maximum weight matching (Duan & Pettie, 2014; Kuhn, 1955) for matrix $\mathbf{a}_t^g$, which maximizes the linear sum of likelihood values for responders and their assigned depots (see Appendix B.2 for more details). This enables us to efficiently compute a discrete assignment that is most similar to the continuous actor output.

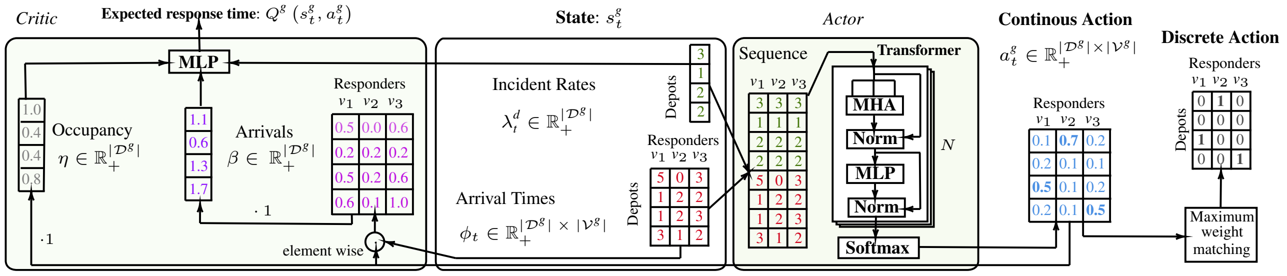

Critic
Expected response time: $Q^g(s_t^g, a_t^g)$
MLP
Occupancy $\eta \in \mathbb{R}_+^{|\mathcal{D}^g|}$
Arrivals $\beta \in \mathbb{R}_+^{|\mathcal{D}^g|}$
Responders v_1 v_2 v_3
element wise
State: s_t^g
Incident Rates $\lambda_t^d \in \mathbb{R}_+^{|\mathcal{D}^g|}$
Arrival Times $\phi_t \in \mathbb{R}_+^{|\mathcal{D}^g| \times |\mathcal{V}^g|}$
Depots
Actor
Sequence
Transformer
MHA
Norm
MLP
Norm
N
Softmax
Continous Action $a_t^g \in \mathbb{R}_+^{|\mathcal{D}^g| \times |\mathcal{V}^g|}$
Discrete Action
Maximum weight matching

Figure 2: Overview of the low-level RL agent training process using DDPG for a region $g \in \mathcal{G}$. First, we map the complex, variable-dimensional state (s_t^g) to a sequence of feature vectors, which we feed to the *actor* to obtain a continuous action ($\mathbf{a}_t^g$). Next, we discretize the continuous action using *maximum weight matching* to allocate responders within the region g. Finally, we use the *critic* to judge the performance of the *actor* by feeding the state and action as fixed-sized vectors to the *critic* and perform learning against response time to serve the incident.

Critic To handle complex state and action spaces, the critic relies on the following three sets of features, which are computed from s_t and $\mathbf{a}_t^g$ for each depot $d \in \mathcal{D}^g$:

- *Depot occupancy* $\eta[d]$: overall likelihood of some responder being assigned to depot d. Occupancy $\eta[d]$ is computed by summing the corresponding likelihood values in $\mathbf{a}_t^g$ and truncating the sum to be between 0 and 1, i.e., $\eta[d] = \text{Clip}(\sum_v^{\mathcal{V}^g} \mathbf{a}_t^g[d,v], 0, 1)$. Intuitively, $\eta[d]$ is the heuristic chance that at least one responder is assigned to depot $d \in \mathcal{D}^g$.
- *Likely available time* $\beta[d]$: weighted sum of the arrival times of responders to depot d. Available time $\beta[d]$ is computed as: $\beta[d] = \sum_v^{\mathcal{V}^g} \phi_t[d,v] \cdot \mathbf{a}_t^g[d,v] \, \forall d \in \mathcal{D}^g$. Intuitively, in combination with $\eta[d]$, time $\beta[d]$ indicates how soon a responder is expected to arrive at the depot.
- *Nearby incident rate* λ_t^d: same as λ_t^d in actor input.

We feed these feature vectors into the critic network, which is a multi-layer perceptron, to obtain the estimated average response time $Q_t^g(s_t^g, a_t^g)$ for action a_t^g in state s_t^g:

$$Q_t^g(s_t^g, a_t^g) = \mathbf{MLP}(\langle \eta[d], \beta[d], \lambda_t^d \mid \forall d \in \mathcal{D}^g \rangle)$$

3.2. High-Level Decision Agent: Reallocating Responders between Regions

We first introduce an MDP formulation of the problem of reallocating responders *among* regions by a high-level planner. Then, we explain how we apply DDPG, mapping states to feature vectors, discretizing actions, and estimating HLP rewards from LLP critics. Figure 3 provides an overview of the architecture of the high-level agent.

We formulate the high-level redistribution problem as an MDP as follows: **State:** We consider the same state representation as the original MDP (s_t), which consists of the incident rates of all cells ($\lambda_t^c, \forall c \in \mathcal{C}$) and the state of all responders ($\mathcal{P}_t$). **Transition:** Transitions occur whenever incident rates change. During a transition, responders are redistributed among the regions based on the actions described below. **Action:** An action specifies which responders to move from one region to another.

To find optimal actions given the MDP for the high-level planner, we use a similar approach as the low-level planner. First, we leverage an abstraction of input features that represent the complex state space. Then, we use DDPG to train the high-level using our novel reward estimation. We discuss the reward estimation in detail later in this section.

Actor We overcome the complexity of the state space by mapping the state to the following features for each region $g \in \mathcal{G}$:

- *Region incident rates* λ_t^g: sum of incident rates for all cells in region $g \in \mathcal{G}$ at time t ($\lambda_t^g = \sum_c^{\mathcal{C}^g} \lambda_t^c$).
- *Allocation* $\mathbf{A}_{t-1}[g]$: number of responder currently assigned to the region $g \in \mathcal{G}$.

Actor Network We feed the input feature vectors into the actor network, which is a multi-layer perceptron, to obtain the action $\mathbf{a}_t^h$, which is a non-negative continuous vector of dimension $|\mathcal{G}| - 1$:

$$\mathbf{a}_t^h = \mathbf{MLP}(\langle \lambda_t^g, \mathbf{A}_{t-1}[g] \mid \forall g \in \mathcal{G} \rangle)$$

We let the actor network compute values for first $|\mathcal{G}| - 1$ regions that provides the ratio of responders with respect to the last region. We obtain the distribution of responders to regions ($\mathbf{a}_t^H$) by appending a constant value of 1 to the vector $\mathbf{a}_t^h$ and perform normalization $\mathbf{a}_t^H = \frac{\mathbf{a}_t^h \frown 1}{|\mathbf{a}_t^h \frown 1|}$.

Discrete Action Based on the normalized action $\mathbf{a}_t^H$, we finally compute the number of responders $\mathbf{A}_t$ assigned to

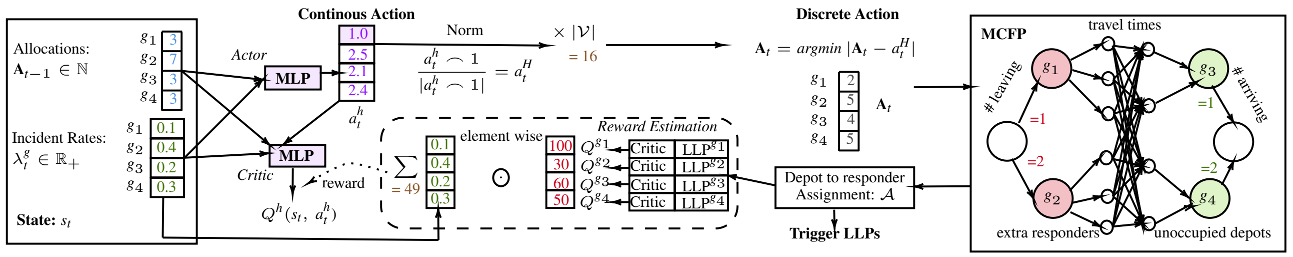


Figure 3: Overview of the training process of the high-level RL agent using DDPG. First, we map the state to a fixed-size feature vector and feed it into the MLP-based *actor* to generate the *continuous action* ($\mathbf{a}_t^h$). Next, we discretize the continuous action and feed it into the *minimum-cost flow problem* to generate the assignment of responders to depots ($\mathcal{A}$). After that, we trigger those LLPs whose regions were affected by the high-level reallocation. Finally, we use the *critic* to judge the performance of the *actor* by training the critic with *rewards estimated by the LLP critics*.

each region g, where $\mathbf{A}_t = \operatorname{argmin}|\mathbf{A}_t - \mathbf{a}_t^H|$, satisfying the following constraints: number of responders assigned to any region $g \in \mathcal{G}$ should not exceeds the number of depots available in the region ($\mathbf{A}_t[g] \leq |\mathcal{D}^g|,\ \forall g \in \mathcal{G}$), and all the responders must be assigned to one region ($\sum_g^{\mathcal{G}} \mathbf{A}_t[g] = |\mathcal{V}|$). Since this computation is trivial, we provide a detailed description in the technical appendix (see Appendix C.1).

Assignment of Responders to Depots Once we discretize the action and obtain the number of responders allocated to each region, we must choose the specific responders to reallocate and which depots they should be assigned to in their new regions. We tackle this problem by transforming it into a standard minimum-cost flow problem (MCFP) (Ahuja et al., 1993), and solve it by minimizing the total expected travel time for the responders to reach their newly assigned depots (for details, see Appendix C.2). Thereby, we obtain the assignment $\mathcal{A}$, which allocates responders $\mathcal{V}_{leaves}$ to depots $\mathcal{D}_{unoccupied}$.

Reward Estimation It is non-trivial to estimate the rewards for the high-level planner – the redistribution of responders is *rewarded* only when responders are dispatched to incidents efficiently, which in turn, depends on the allocation chosen by the low level planner. Hence, the rewards from the regions are noisy. Further, events happen in regions without synchronization, and each LLP transitions independently. To tackle these issues, we introduce a novel reward estimation technique for the high-level RL agent: we use the low-level RL agents' critics to estimate the value of each region's allocation. We hypothesize that these low-level critic values estimate how well a specific distribution of responders across regions helps reduce response times. In addition, since incident rates can vary across decision-making epochs, we also incorporate each region's incident rate to prioritize regions based on forecasted demand. Accordingly, we estimate the reward for the high-level RL agent as a weighted sum of critic values ($\sum_g^{\mathcal{G}} \lambda_t^g \cdot Q_g^*(s_t^g, \mathbf{a}_t^g|\theta^{Q_g^*})$) from the low-level agents corresponding to each region $g \in \mathcal{G}$.

Critic We feed the feature vectors for the critic network (i.e., the feature vectors for the actor network to represent the state + action) and compute the value ($Q_t(s_t, \mathbf{a}_t^h)$) of performing the action $\mathbf{a}_t^h$ at the state s_t as follows:

$$Q_t(s_t, \mathbf{a}_t^h) = \mathbf{MLP}(\langle \lambda_t^g, \mathbf{A}_{t-1}[g] \mid \forall g \in \mathcal{G}, \mathbf{a}_t^h \rangle)$$

4. Numerical Results

4.1. Dataset and Experiment Setup

We evaluate our approach using real-world data from two U.S. cities. First, we apply our approach to emergency response data from Nashville, TN, which was published by (Pettet et al., 2022). This data includes processed incident chains, incident rates, depot locations, hospital locations, operational data provided by the Nashville fire department, and dynamic travel times (i.e., travel times that vary by time of the day and by day of the week) generated using contraction hierarchies (Geisberger et al., 2008) and OSRM (Huber & Rust, 2016). A one-by-one mile square grid is applied to the city, which aligns with the configurations followed by local authorities. The city has 36 depots, 9 general hospitals, and 26 responders to assist emergency response management. Further, the data contains 60 incident chains sampled from the historical incident data distribution and corresponding rate data. Each chain spans 11 days and contains between 1818 and 2025 incidents, averaging 1907 incidents per chain. The data also includes several region segmentations $\mathcal{G}$ of the city's cells, obtained by applying the k-means clustering algorithm based on historical incident rates in cells $\mathcal{C}$ and geographic proximity. We also evaluate our algorithm using publicly available data from another U.S. city, Seattle, WA (City of Seattle, 2022). Due to limited space, we present the Seattle results in Appendix D.3. Our implementation and data are available as part of the supplementary material.

We assume that depots can accommodate at most one re-

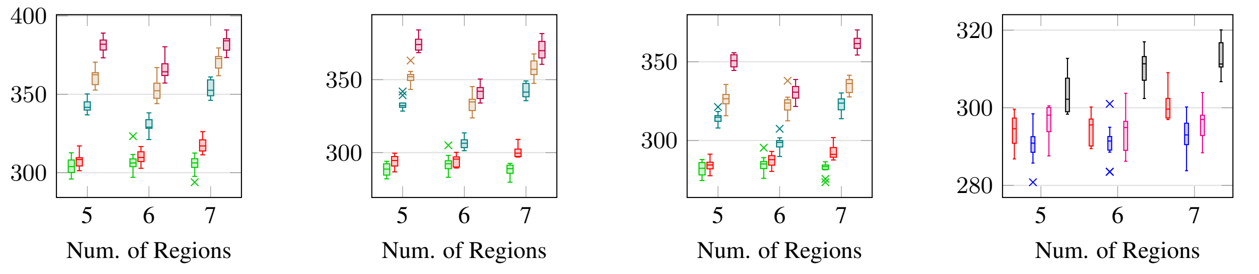


Figure 4: Distribution of average response times (lower is better) with our approach (🟩), MCTS (🟥), p-median with α = 1.0 (🟦), greedy policy (🟫), and static policy, i.e., no proactive repositioning (🟪) for 10 different sample incident chains with (a) 24 responders, (b) 26 responders, and (c) 28 responders. (d) distribution of average response times using MCTS (🟥) and various architectures as the actor for the low-level agent (TrXL (🟦), GTrXL (🟪), and LSTM (⬛)), trained and evaluated with the HLP from prior work (Pettet et al., 2022) for 10 different sample incident chains with 26 responders (Nashville).

sponder at a time (i.e., $\mathbf{DC}(d) = 1, \forall d \in \mathcal{D}$), same as (Pettet et al., 2022). In practice, we could create extra single-capacity depots to represent depots that can accommodate more than one responder, and apply the same solution approach. We assume that the time taken to serve an incident (t_{serve}) is constant 20 minutes (same as (Pettet et al., 2022)).

4.2. Baselines

We compare our approach against the state-of-the-art MCTS approach (Pettet et al., 2022) and three other baselines. For the other baselines, we leverage the high-level planner from (Pettet et al., 2022). We set the initial state of the environment by distributing the responders among the region based on the high-level planner of (Pettet et al., 2022), and sequentially assign responders to depots in each region. Specifically, we use the following baselines in addition to (Pettet et al., 2022): (1) *p-median-based policy*: we use a modified p-median formulation suggested by (Vazirizade et al., 2021), which incorporates a *balancing* term to account for account for responders becoming unavailable when attending incidents; (2) *greedy policy*: we make reallocation decisions based on the expected incident rates and expected travel times to reach the depots; and (3) *static policy*: a baseline where the allocation is never changed from the initial one. Note that the static policy closely resembles real-world strategies followed by first-responders in practice (Pettet et al., 2022). We describe the baselines in the technical appendix (Appendix D.1).

For the baseline experiments, we trigger both the high- and low-level agents each time an incident occurs or when there have been no incidents in the last 60 minutes, same as suggested in the original work by (Pettet et al., 2022). However, when evaluating our approach, we trigger the high-level agent each time the predicted incident rates change, with at least a 60-minute interval after the previous trigger; and trigger the low-level agent for a region whenever responders enter or leave the region, or serve an incident. For MCTS baseline, we use the same hyperparameters as (Pettet et al., 2022): iteration limit is 1000, discount factor is 0.99995, UCT trade-off parameter is 1.44, and number of generated samples is 50.

4.3. Training

We train both the high- and low-level agents using 50 sampled incident chains and evaluate the trained models using the remaining 10 chains. We use the Adam optimizer (Kingma & Ba, 2014) and a learning rate of 10^{-3}. We set the reward decay rate (γ) for low-level agents to 0.5. We run all algorithms on an Intel Xeon E5-2680 28-core CPU with 128GB of RAM. We detail the agent-specific training below.

Low-Level RL Agent We train a low-level agent for each region g individually using our approach and using the DRLSN baseline (Ji et al., 2019). We vary the number of responders assigned to the region during training randomly based on a binomial distribution with values ranging from 1 to $|\mathcal{D}^g|$ and with probability $\frac{|\mathcal{V}|}{|\mathcal{D}|}$. After the arrival of an incident (from the sampled incident chain) or after 1 hour without any incidents, we select a reallocation action using the RL policy based on the current state of the environment (one specific region). We apply the reallocation action to the environment and capture the response time of serving the next incident as a negative reward for the reallocation action. We use a fixed-size experience-replay buffer, where the experiences are stored based on a First-In-First-Out (FIFO) policy. Accordingly, we add the experience tuple (i.e., state, action, reward, next state) for each transition to the buffer. During the learning process, we randomly sample a fixed-size batch of experiences from the buffer, and we train the actor and critic networks using the DDPG algorithm. After each transition step, we update the actor and critic networks until they converge to the optimal RL policy.

We perform an architecture search based on the experiences gathered from online RL to obtain the best hyper-parameters for our approach's TrXL-based actor network (described in detail in Appendix D.4). For our critic network, we use a feed-forward network with one hidden layer with 64 neurons and ReLU activation, followed by a dropout layer with a dropout rate of 0.1 and an output layer with one neuron and linear activation. For our baseline DRLSN, we consider an MLP with three hidden layers containing 256, 128, and 64 neurons respectively and use ReLU activation in all the layers.

High-Level RL Agent The high-level agent is trained after completing the low-level agent training. At the beginning of each episode, we provide an initial allocation of responders to regions that is proportional to the region-level incident rates (i.e., sum of the incident arrival rates of all cells in the region), and the low-level agent of each region is invoked to perform an initial allocation of responders to depots within its region. Whenever the distribution of incident rates changes at the regional level, we select a high-level redistribution action using the RL policy. We apply the redistribution action to the environment, invoke low-level agents as needed, and capture the weighted sum of the low-level critics as a reward for the redistribution action. We use a fixed-size replay buffer with a FIFO policy to store experiences (i.e., tuples of state, action, reward, and next state). During the learning process, we randomly sample batches of experience from the replay buffer, and we train the actor and critic networks using DDPG. After each transition step, we update the actor and critic networks until convergence. We vary the number of responders between 26 ± 3 in each episode during training.

For the actor, we use a feed-forward network with two hidden layers of 256 and 64 neurons, respectively, with ReLU activation, followed by a dropout layer with a dropout rate of 0.1 and an output layer with one neuron and linear activation. For the critic, we use a feed-forward network with one hidden layer of 64 neurons with ReLU activation, followed by a dropout layer with a dropout rate of 0.1 and an output layer with one neuron and linear activation.

Training Time Our low-level agent using TrXL takes around 1 day ($\approx$ 100 episodes) to converge to an optimal policy for regions with 8 or less depots. For regions with 9 or more depots, it takes up to 14 days after obtaining the best architecture. Our high-level RL agent takes around 2 days ($\approx$ 40 episodes) to converge.

4.4. Evaluation

Computation Times On average, our DDPG-based approach takes around 0.22 seconds to make a single decision, considerably less than the MCTS baseline (Pettet et al., 2022), which takes 3 minutes. While a 3-minute delay might not necessarily seem significant, it is actually substantial in the context of this problem setting: the average travel time between depots is around 15 minutes in Nashville, and the average response time is around 4.8 minutes. In a domain where seconds can save lives, an additional 3-minute delay is very significant. While other baselines may take less time (DRLSN: 0.045 seconds; p-median: 0.20 seconds; greedy: 0.10 seconds), we point out that a latency of 0.22 seconds is negligible in practice for ERM.

Response Times Figures 4a to 4c show average response times for our DDPG-based approach compared against baseline approaches (MCTS, p-median-based policy, greedy policy, and static policy) using 10 sampled incident chains. On average, we outperform the state-of-the-art MCTS baseline by 5 seconds in 5-region segmentation and by 13 seconds in 7-region segmentation, which are significant savings in the ERM domain (Pettet et al., 2022; Mayer, 1979). We observe that p-median, greedy, and static policy always perform poorly compared to the state-of-the-art approach. We include a comparison with the DRLSN baseline in the technical appendix as both MCTS and the proposed approach outperform it by a significant margin (see Appendix D.2). In addition, we also train our LLP agent considering the entire city to be a single region, and evaluate it using the same incident chains. We find that this centralized variant performs 66 seconds worse than our hierarchical approach.

Ablation Study: Low-Level Agents with Different Architectures Figure 4d shows the distribution of average response times using different low-level actor architectures—TrXL, Gated TrXL (GTrXL) (Parisotto et al., 2020; Parisotto & Salakhutdinov, 2021), and LSTM—trained and evaluated with the HLP from prior work (Pettet et al., 2022), alongside the MCTS baseline. We observe that TrXL-based low-level agents perform better than other architectures. TrXL-based low-level agents reduce time to serve an incident by 5 seconds on average compared to GTrXL and by 18 seconds on average compared to LSTM. Accordingly, we train the high-level RL agent with TrXL-based low-level agents.

5. Related Work

We provide a brief discussion of the most closely related prior work here; for a broader discussion, please see Appendix F.

Zhang et al. (2021) classify Multi-Agent Reinforcement Learning (MARL) based on the type of agents (i.e., homogeneous or heterogeneous), objective (i.e., cooperative, competitive, or combination of both cooperative and competitive (Lowe et al., 2017)), control mechanism (i.e., centralized or decentralized), and learning paradigm (i.e., singular or hierarchical). In our approach, we consider cooperative heterogeneous agents with hierarchical coordination and the shared goal of minimizing response times for future incidents. Most cooperative MARLs maximize a shared reward when a centralized controller controls it or maximize an average reward in a decentralized setting (Oroojlooy & Hajinezhad, 2023; Zhang et al., 2018; Kar et al., 2012). In our approach, we train the low-level agents to maximize rewards independently (Eghtesad et al., 2024). Then, we introduce a reward-estimation mechanism for the high-level agent (i.e., convex combination of low-level critic values) to coordinate the independent execution of low-level agents.

In the domain of resource reallocation, Jin et al. (2019) apply a Multi-Agent Hierarchical Reinforcement Learning (MA-HRL) approach, based on Feudal RL (Dayan & Hinton, 1992; Vezhnevets et al., 2017). To make each agent aware of other agents at the same hierarchy level, Jin et al. (2019) introduce attention between the agents; whereas in our approach, we let the low-level agents act independently of each other; instead, we utilize the high-level agent for coordination.

6. Conclusion

We introduce a novel multi-agent RL-based approach by replacing the high- and low-level planners of Pettet et al.'s framework with deep reinforcement learning, addressing the computational challenges faced by learning. We show using real-world data that our approach reduces computation time per decision by *three orders of magnitude* compared to the state of the art. We also confirm the general advantage of hierarchical approaches over centralized ones as a centralized variant of our learning-based approach performs poorly in comparison. Finally, we demonstrate that redistributing responders when incident rates change is better than redistributing after every incident.

Software and Data

Software code and data are available online (Sivagnanam et al., 2024). Within the ZIP file `MARL-HC-ERS-ICML24.zip`, we provide our complete code base as well as implementations for all baselines, i.e., DRLSN baseline (Ji et al., 2019), MCTS baseline (Pettet et al., 2022), p-median baseline (Vazirizade et al., 2021), greedy policy, and static policy. For instructions on training and evaluation, please see `README.txt` (available in the root folder of the extracted ZIP file).

Acknowledgments

This material is based upon work sponsored by the National Science Foundation under Grants CNS-1952011 and CNS-2238815 and by the Department of Energy under Award Number DE-EE0009212. The authors acknowledge the computing support from NSF Chameleon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or of the Department of Energy. The authors would like to thank the anonymous reviewers of ICML'24 for their insightful feedback and suggestions.

Impact Statement

With our approach, emergency response management systems can make proactive reallocation decisions in a fraction of a second, ensuring better services for all incoming requests. Making reallocation decisions in a fraction of a second is extremely beneficial in ERM, a domain where seconds may mean the difference between life and death. Our approach also slightly reduces the average response time compared to the current state-of-the-art approach, which requires extensive computation for each decision. Overall, these improvements have the potential to shorten emergency response times, leading to a positive societal impact. Further, our research did not involve any personally identifiable or sensitive information.

References

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. *Network flows: Theory, algorithms, and applications*. Prentice-Hall, 1993.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

City of Seattle. Collisions all years. `https://data.seattle.gov/dataset/Collisions-All-Years/pfun-q57u`, 2022.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019)*, pp. 2978–2988, 2019.

Dayan, P. and Hinton, G. E. Feudal reinforcement learning. *Advances in Neural Information Processing Systems (NIPS 1992)*, 5, 1992.

Duan, R. and Pettie, S. Linear-time approximation for maximum weight matching. *Journal of the ACM*, 61(1): 1–23, 2014.

Eghtesad, T., Li, S., Vorobeychik, Y., and Laszka, A. Multi-agent reinforcement learning for assessing false-data injection attacks on transportation networks. In *23rd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, May 2024.

Geisberger, R., Sanders, P., Schultes, D., and Delling, D. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *International workshop on experimental and efficient algorithms*, pp. 319–333. Springer, 2008.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Huber, S. and Rust, C. Calculate travel time and distance with openstreetmap data using the open source routing machine (osrm). *The Stata Journal*, 16(2):416–423, 2016.

Hutsebaut-Buysse, M., Mets, K., and Latré, S. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4 (1):172–221, 2022.

Jaldell, H., Lebnak, P., and Amornpetchsathaporn, A. Time is money, but how much? the monetary value of response time for thai ambulance emergency services. *Value in health*, 17(5):555–560, 2014.

Ji, S., Zheng, Y., Wang, Z., and Li, T. A deep reinforcement learning-enabled dynamic redeployment system for mobile ambulances. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 3 (1):1–20, 2019.

Jin, J., Zhou, M., Zhang, W., Li, M., Guo, Z., Qin, Z., Jiao, Y., Tang, X., Wang, C., Wang, J., et al. Coride: joint order dispatching and fleet management for multi-scale ride-hailing platforms. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM 2019)*, pp. 1983–1992, 2019.

Kar, S., Moura, J. M., and Poor, H. V. Qd-learning: A collaborative distributed strategy for multi-agent reinforcement learning through consensus. *arXiv preprint arXiv:1205.0047*, 2012.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kuhn, H. W. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2): 83–97, 1955.

Li, M., Qin, Z., Jiao, Y., Yang, Y., Wang, J., Wang, C., Wu, G., and Ye, J. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *The world wide web conference*, pp. 983–994, 2019.

Li, Y., Zheng, Y., and Yang, Q. Dynamic bike reposition: A spatio-temporal reinforcement learning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2018)*, pp. 1724–1733, 2018.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Lobel, S., Rammohan, S., He, B., Yu, S., and Konidaris, G. Q-functionals for value-based continuous control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 8932–8939, 2023.

Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Pieter Abbeel, O., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Mayer, J. D. Emergency medical service: delays, response time and survival. *Medical Care*, 17(8):818–827, 1979.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Mukhopadhyay, A., Wang, Z., and Vorobeychik, Y. A decision theoretic framework for emergency responder dispatch. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2018)*, pp. 588–596, 2018.

Mukhopadhyay, A., Pettet, G., Vazirizade, S., Vorobeychik, Y., Kochenderfer, M., and Dubey, A. A review of emergency incident prediction, resource allocation and dispatch models. *arXiv preprint arXiv:2006.04200*, 2020.

Oroojlooy, A. and Hajinezhad, D. A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence*, 53(11):13677–13722, 2023.

Parisotto, E. and Salakhutdinov, R. Efficient transformers in reinforcement learning using actor-learner distillation. *arXiv preprint arXiv:2104.01655*, 2021.

Parisotto, E., Song, F., Rae, J., Pascanu, R., Gulcehre, C., Jayakumar, S., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., et al. Stabilizing transformers for reinforcement learning. In *Proceedings of the 37th International*

Conference on Machine Learning, volume 119, pp. 7487–7498. PMLR, 2020.

Pateria, S., Subagdja, B., Tan, A.-h., and Quek, C. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

Pettet, G., Mukhopadhyay, A., Kochenderfer, M., Vorobeychik, Y., and Dubey, A. On algorithmic decision procedures in emergency response systems in smart and connected communities. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2020)*, pp. 1046–1054, 2020.

Pettet, G., Mukhopadhyay, A., Kochenderfer, M. J., and Dubey, A. Hierarchical planning for dynamic resource allocation in smart and connected communities. *ACM Transactions on Cyber-Physical Systems*, 6(4), November 2022. ISSN 2378-962X. doi: 10.1145/3502869. URL https://doi.org/10.1145/3502869.

Schjølberg, M. E., Bekkevold, N. P., Sánchez-Díaz, X., and Mengshoel, O. J. Comparing metaheuristic optimization algorithms for ambulance allocation: An experimental simulation study. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '23, pp. 1454–1463, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701191. doi: 10.1145/3583131.3590345. URL https://doi.org/10.1145/3583131.3590345.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sivagnanam, A., Pettet, A., Lee, H., Mukhopadhyay, A., Dubey, A., and Laszka, A. Multi-agent reinforcement learning with hierarchical coordination for emergency responder stationing (ICML-2024) (Code and data), May 2024. URL https://doi.org/10.6084/m9.figshare.25872640.

Talusan, J. P., Han, C., Mukhopadhyay, A., Laszka, A., Freudberg, D., and Dubey, A. An online approach to solving public transit stationing and dispatch problem. In *15th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, May 2024.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems (NIPS 2017)*, 30, 2017.

Vazirizade, S. M., Mukhopadhyay, A., Pettet, G., El Said, S., Baroud, H., and Dubey, A. Learning incident prediction models over large geographical areas for emergency response. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 424–429. IEEE, 2021.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, volume 70, pp. 3540–3549. PMLR, 2017.

Xi, J., Zhu, F., Chen, Y., Lv, Y., Tan, C., and Wang, F. Ddrl: A decentralized deep reinforcement learning method for vehicle repositioning. In *Proceedings of the 2021 IEEE International Intelligent Transportation Systems Conference (ITSC 2021)*, pp. 3984–3989. IEEE, 2021.

Xu, Z., Bai, Y., Zhang, B., Li, D., and Fan, G. Haven: Hierarchical cooperative multi-agent reinforcement learning with dual coordination mechanism. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 11735–11743, 2023.

Zhang, K., Yang, Z., Liu, H., Zhang, T., and Basar, T. Fully decentralized multi-agent reinforcement learning with networked agents. In *International Conference on Machine Learning*, pp. 5872–5881. PMLR, 2018.

Zhang, K., Yang, Z., and Başar, T. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pp. 321–384, 2021.

Zhou, M., Jin, J., Zhang, W., Qin, Z., Jiao, Y., Wang, C., Wu, G., Yu, Y., and Ye, J. Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 2645–2653, 2019.

Table 1: List of Symbols

Symbol	Description
	Constants
$\mathcal{G}$	Regions
$\mathcal{D}$	Depots
$\mathcal{H}$	Hospitals
$\mathcal{C}$	Cells
$\mathcal{M}$	Travel model
$\mathcal{D}^g$	Depots in region $g \in \mathcal{G}$
$\mathcal{V}$	Responders
	Variables
$\mathcal{V}^g$	Responders assigned to region $g \in \mathcal{G}$
$\mathbf{a}_t^h$	High-level RL agent action
$\mathbf{a}_t^g$	Low-level RL agent action
$\mathbf{A}_t$	Redistribution of responders to regions
$\mathbf{A}_t[g]$	Number of responders assigned to region $g \in \mathcal{G}$
$\mathcal{A}$	Assignment of responders to depots
$\mathcal{A}[g]$	Assignment of responders to depots in region $g \in \mathcal{G}$
λ_t^c	Incident rate in cell $c \in \mathcal{C}$ at time t
$\mathcal{I}_t$	Incidents waiting for service at time t
$\mathcal{P}_t$	State of responders at time t
g^v	Region assigned to responder v
d^v	Depot assigned to responder v
i^v	Incident assigned to responder v
h^v	Hospital assigned to responder v
c^v	Current position of responder v
t_{avail}^v	Time at which responder v will be available
$\phi_t[d, v]$	Total time responder v takes to reach depot d after completing its current task at time t

A. Notation

Table 1 provides a summary of the most important notation used throughout our paper.

B. Low-Level Decision Agent

B.1. Architecture of TrXL-based Actor

We build our actor network with TrXL layers (Dai et al., 2019) as its fundamental units. A TrXL layer consists of two main components: multi-head attention (MHA) and multilayer perceptron (MLP). After each main component is a normalization layer (Norm-MHA and Norm-MLP). Our actor-network contains N layers of TrXL followed by a *Softmax* layer at the end. We use $Input_X^L$ to denote the input of component $X \in \{\text{MHA, Norm-MHA, MLP, Norm-MLP}\}$ at layer $L \in \{1, 2, 3, \dots, N\}$, and $Output_X^L$ to denote the output of component X at layer L. The first layer takes as input a sequence of feature vectors $\langle \phi_t[d, v], \lambda_t^d \rangle_{d \in \mathcal{D}^g}$ corresponding to each responder $v \in \mathcal{V}^g$. After the first layer, the input of each subsequent layer is the output of the preceding layer. Accordingly, we can express $Input_{\text{MHA}}^L$ formally as follows:

$$Input_{\text{MHA}}^L = \begin{cases} \{\langle \phi_t[d, v], \lambda_t^d \mid \forall d \in \mathcal{D}^g \rangle \mid \forall v \in \mathcal{V}^g\} & \text{if } L = 1 \\ Output_{\text{Norm-MLP}}^{L-1} & \text{otherwise.} \end{cases}$$

We feed these values as inputs to MHA (i.e., *query*, *key*, *value*):

$$Output_{\text{MHA}}^L = \mathbf{MHA}(Input_{\text{MHA}}^L) \tag{1}$$

where MHA is a multi-head attention layer, same as (Vaswani et al., 2017), where we set the dimension of key $|key| = |\mathcal{D}^g|$ (i.e., number of depots in the region $g \in \mathcal{G}$).

Then, we add the output $Output_{\text{MHA}}^L$ of the MHA with its input $Input_{\text{MHA}}^L$ and apply the normalization layer from (Ba et al., 2016):

$$Output_{\text{Norm-MHA}}^L = \text{Norm}(\text{Add}(Input_{\text{MHA}}^L, Output_{\text{MHA}}^L)) \tag{2}$$

Next, we feed the output of the normalization into an multilayer perceptron (MLP):

$$Output_{\text{MLP}}^L = \mathbf{MLP}(Output_{\text{Norm-MHA}}^L) \tag{3}$$

Then, the output $Output_{\text{MLP}}^L$ of the MLP is added with the output $Output_{\text{Norm-MHA}}^L$ of the normalization after MHA, followed by another normalization layer (Ba et al., 2016):

$$Output_{\text{Norm-MLP}}^L = \text{Norm}(\text{Add}(Output_{\text{Norm-MHA}}^L, Output_{\text{MLP}}^L)) \tag{4}$$

The steps indicated by Equations (1) to (4) are repeated N times sequentially, and then we feed the output of the N^{th} layer into a softmax layer. We apply softmax separately to each responder $v \in \mathcal{V}^g$ (i.e., we apply it separately to each element of the sequence of vectors, consisting of one vector for each responder $v \in \mathcal{V}^g$) to obtain the likelihood of assigning responder v to each depot in the region. Accordingly, we obtain $\mathbf{a}_t^g[v]$ as the output after applying softmax for each responder v:

$$\mathbf{a}_t^g[v] = Softmax(Output_{\text{Norm-MLP}}^N[v])$$

ensuring that $\mathbf{a}_t^g[v] \cdot 1 = 1$ and $\mathbf{a}_t^g[v] \geq 0$ for each $v \in \mathcal{V}^g$.

B.2. Discretizing the Continuous Action

We use maximum weight matching (MWM) in a weighted bipartite graph to efficiently compute a discrete assignment of responders to depots $\mathcal{A}[g]$ that is similar to the continuous actor output $\mathbf{a}_t^g$. There is a set of graph nodes representing responders $\mathcal{V}^g$ and a set of nodes representing depots $\mathcal{D}^g$; each responder $v \in \mathcal{V}^g$ is connected to each depot $d \in \mathcal{D}^g$ by an edge of weight $\mathbf{a}_t^g[v][d]$ (the low-level agent's preference for assigning v to d). Maximizing with respect to the weights provides a discrete assignment close to the continuous action.

We formally define the maximum weight matching to discretize a continuous action as follows.

First, we define the binary decision variable $x_{v,d} \in \{0,1\}$ for $\forall v \in \mathcal{V}^g$ and $\forall d \in \mathcal{D}^g$, where

$$x_{v,d} = \begin{cases} 1 & \text{if responder } v \text{ is assigned to depot } d \\ 0 & \text{otherwise.} \end{cases}$$

Next, a *matching* in the graph assigns each responder to at most one depot. a *maximum matching* assigns each responder to exactly one depot. We can *matching* using following two constraints:

First, each responder $v \in \mathcal{V}^g$ in the region needs to be assigned to one depot within the region:

$$\sum_{d}^{\mathcal{D}^g} x_{v,d} = 1$$

Second, at most one responder can be assigned to each depot $d \in \mathcal{D}^g$ in the region:

$$\sum_{v}^{\mathcal{V}^g} x_{v,d} \leq 1$$

Finally, we formulate the discrete assignment problem with the objective of finding feasible $\langle x_{v,d} \rangle$ that assign vehicles to depots with high $\mathbf{a}_t^g[v][d]$ (i.e., assign each vehicle v to a depot d for which the actor output a high likelihood $\mathbf{a}_t^g[v][d]$ of assignment). Formally, the discrete assignment problem maximizes the following objective:

$$\text{Objective}_{\text{MWM}} = \sum_{d}^{\mathcal{D}^g} \sum_{v}^{\mathcal{V}^g} x_{v,d} \cdot \mathbf{a}_t^g[v][d]$$

where $\mathbf{a}_t^g[v][d]$ is the likelihood output by the actor for assigning responder $v \in \mathcal{V}^g$ to depot $d \in \mathcal{D}^g$.

The above problem is a *maximum weight matching problem*, which is computationally trivial to solve using standard approaches (e.g., Edmonds' algorithm).

C. High-Level Decision Agent

C.1. Discretizing the Continuous Action

Algorithm 1 GreedyAlgorithm

```
Input: $\mathbf{a}_t^H, \mathcal{G}, \mathcal{D}, \mathcal{V}$
Output: $\mathbf{A}_t$
 1: $V_{avail} \leftarrow |\mathcal{V}|$
 2: for $g \in \mathcal{G}$ do
 3:     $\mathbf{A}_t[g] \leftarrow 0$
 4: end for
 5: while $\sum_g^{\mathcal{G}} \mathbf{A}_t[g] < V_{avail}$ do
 6:     for $g \in \mathcal{G}$ do
 7:         $\mathbf{A}_t[g] \leftarrow \left\lfloor \frac{\mathbf{a}_t^H[g]}{\sum_{\hat{g}}^{\mathcal{G}} \mathbf{a}_t^H[\hat{g}]} \cdot V_{avail} \right\rfloor$
 8:     end for
 9:     $V_{remain} \leftarrow V_{avail} - \sum_g^{\mathcal{G}} \mathbf{A}_t[g]$
10:     while $V_{remain} > 0$ do
11:         $\hat{g} \leftarrow \text{argmax}_{g \in \mathcal{G}}(\mathbf{a}_t^H[g] \cdot V_{avail} - \mathbf{A}_t[g])$
12:         $\mathbf{A}_t[\hat{g}] \leftarrow \mathbf{A}_t[\hat{g}] + 1$
13:         $V_{remain} \leftarrow V_{remain} - 1$
14:     end while
15:     for $g \in \mathcal{G}$ do
16:         if $\mathbf{A}_t[g] > |\mathcal{D}^g|$ then
17:             $\mathbf{A}_t[g] \leftarrow |\mathcal{D}^g|$
18:             $\mathcal{G} \leftarrow \mathcal{G} \setminus \{g\}$
19:             $V_{avail} \leftarrow V_{avail} - |\mathcal{D}^g|$
20:         end if
21:     end for
22: end while
```

In Section 3.2, we provide a brief description of generating the discrete number of responders allocated to each region $\mathbf{A}_t$, where $\mathbf{A}_t = \text{argmin}\, |\mathbf{A}_t - \mathbf{a}_t^H|$, from a normalized continuous action $\mathbf{a}_t^H$. Here, we explain the greedy algorithm (see Algorithm 1) that we use to compute a feasible discrete allocation $\mathbf{A}_t$ that is similar to the continuous allocation $\mathbf{a}_t^H$. Please note that this algorithm is trivial (finding a vector of integer values subject to upper bounds, minimizing the difference to a vector of desired continuous values); we provide a description for the sake of completeness.

We initialize the greedy algorithm with no vehicles allocated to any region. Then, we follow an iterative process, which includes three steps. First, we determine the allocation for every region based on the available responders V_{avail} and the ratio between the action value $\mathbf{a}_t^h[g]$ corresponding to the region and the sum of all the action values $\sum_{\hat{g}}^{\mathcal{G}} \mathbf{a}_t^h[\hat{g}]$. Then, we compute the set of remaining responders (V_{remain}) as the difference between all available responders V_{avail} and the responders allocated in previous steps $\sum_g^{\mathcal{G}} \mathbf{A}_t[g]$. If there are any responders left awaiting allocation, we follow another iterative process. In each iteration, we choose the region with the highest difference between expected and allocated responders using the previous step (i.e., $\mathbf{a}_t^H[g] \cdot V_{avail} - \mathbf{A}_t[g]$). Then, we add one more responder to the chosen region. Finally, we check if the allocation to any region exceeds the number of depots in the region; in case of such a situation, the allocation is fixed at the number of depots (to avoid allocating more responders than what is feasible), and the region is removed from future iterations.

C.2. Assignment of Responders to Depots

Next, we discuss how the high-level agent decides which responders should leave each region (for regions whose allocations have been reduced) and to which depots these responders should be assigned in their new regions (which are regions whose allocations have been increased). To minimize the gap in coverage while responders drive to their new regions, we formulate this assignment as a minimum cost flow problem (MCFP) that minimizes the total travel time of all the reallocated responders.

Formulation of Minimum Cost Flow Problem To formulate the assignment problem as an MCFP, we define a graph $\mathbf{G}(\mathcal{Y}, \mathcal{E})$ with nodes $\mathcal{Y}$ and edges $\mathcal{E}$. Each node $y \in \mathcal{Y}$ represents one of the following:

1.) abstract source $\mathbf{Y}_{source}$ and sink $\mathbf{Y}_{sink}$ nodes for the flow

2.) regions $\mathcal{G}_{leaves}$ from which responders will leave: $\mathcal{G}_{leaves} = \{g \mid g \in \mathcal{G}, \mathbf{A}_{t-1}[g] > \mathbf{A}_t[g]\}$

3.) regions $\mathcal{G}_{arrive}$ where responders will arrive: $\mathcal{G}_{arrive} = \{g \mid g \in \mathcal{G}, \mathbf{A}_{t-1}[g] < \mathbf{A}_t[g]\}$

4.) all the responders $\mathcal{V}_{leaves}$ in regions $g \in \mathcal{G}_{leaves}$ from which responders will leave: $\mathcal{V}_{leaves} = \bigcup_g^{\mathcal{G}_{leaves}} \mathcal{V}^g$

5.) unoccupied depots $\mathcal{D}_{unoccupied}$ in regions $g \in \mathcal{G}_{arrive}$ where responders will arrive: $\mathcal{D}_{unoccupied} = \bigcup_g^{\mathcal{G}_{arrive}} \{d \mid d \in \mathcal{D}^g \wedge d \notin \{d^v \mid v \in \mathcal{V}^g\}\}$

A pair of two nodes $y_1, y_2 \in \mathcal{Y}$ are connected by a directed edge $(y_1, y_2) \in \mathcal{E}$ if and only if one of the following conditions met:

1.) $y_1 = \mathbf{Y}_{source} \wedge y_2 \in \mathcal{G}_{leaves}$: the source node is connected to all nodes representing the regions from which responders will leave;

2.) $y_1 = g^{y_2} \wedge y_2 \in \mathcal{V}_{leaves}$: each node representing a region from which responders will leave is connected to all nodes that represent the responders in that region;

3.) $y_1 \in \mathcal{V}_{leaves} \wedge y_2 \in \mathcal{D}_{unoccupied}$: each node representing a responder from a region from which responders will leave is connected to all nodes representing unoccupied depots in regions where responders will arrive;

4.) $y_1 \in \mathcal{D}_{unoccupied} \wedge y_2 \in \mathcal{G}_{leaves} \wedge y_1 \in \mathcal{D}^{y_2}$: each node that represents an unoccupied depot in a region where responders will arrive is connected to the node representing the region of the depot;

5.) $y_1 \in \mathcal{G}_{leaves} \wedge y_2 = \mathbf{Y}_{sink}$: all nodes representing regions where responders will arrive are connected to the sink node (see Figure 3 for an illustration).

Each edge $(y_1, y_2) \in \mathcal{E}$ has a cost $\mathbf{Cost}(y_1, y_2)$ and a capacity $c(y_1, y_2)$. We let the cost between nodes representing $v \in \mathcal{V}_{leaves}$ and $d \in \mathcal{D}_{unoccupied}$ be the total time $\phi_t[d, v]$ that it would take responder v to move to depot d; and for all other edges, we let the cost be zero. We let the capacity $c(y_1, y_2)$ of directed edges $(y_1, y_2) \in \mathcal{E}$ be the following:

$$c(y_1, y_2) = \begin{cases} \mathbf{A}_{t-1}[g] - \mathbf{A}_t[g] & \text{if } y_1 \in \mathbf{Y}_{source} \wedge y_2 \in \mathcal{G}_{leaves} \\ \mathbf{A}_t[g] - \mathbf{A}_{t-1}[g] & \text{if } y_1 \in \mathcal{G}_{arrives} \wedge y_2 \in \mathbf{Y}_{sink} \\ 1 & \text{otherwise.} \end{cases}$$

Finally, we require the total amount of flow from source $\mathbf{Y}_{source}$ to sink $\mathbf{Y}_{sink}$ to be $\sum_g^{\mathcal{G}_{leaves}} (\mathbf{A}_{t-1}[g] - \mathbf{A}_t[g])$.

Minimum Cost Flow as a Responder Assignment The integer solution of the above minimum cost flow problem is an assignment that minimizes the total travel time of the reallocated responders. First, for each responder v, an integer solution of the above problem assigns a positive flow for at most one depot d. This assignment is feasible: for each region from which responders will leave, the right number of responders will have a positive flow; and for each region where responders will arrive, the right number of depots will have a positive flow. If each responder v drives to the assigned depot d, then their total travel time will be minimal. Finding an integer solution to a minimum cost flow problem is computationally easy.

Standard Capacity and Conservation Constraints The above formulation is subject to the standard constraints of flow conservation and edge capacity, and its objective is standard minimization of flow cost. For the sake of completeness, we provide a formal specification of these constraints and the objective. Let integer variable $x(y_1, y_2)$ indicate whether there is flow from node $y_1 \in \mathcal{Y}$ to node $y_2 \in \mathcal{Y}$ (i.e., if $x(y_1, y_2) = 0$ then there is no flow, meaning possible responder movement, otherwise there is flow). In addition, the formulation needs to ensure the following flow constraints. First, the net flow in all nodes, except the source and sink nodes, must be equal to zero:

$$\sum_{j \in \mathcal{Y}} x(j, i) - \sum_{k \in \mathcal{Y}} x(i, k) = 0 \quad \forall i \in \mathcal{Y} \setminus \{\mathbf{Y}_{source}, \mathbf{Y}_{sink}\}$$

Second, responders that leave their current region must arrive at a new region:

$$\sum_{i \in \mathcal{Y}} x(\mathbf{Y}_{source}, i) = \sum_{j \in \mathcal{Y}} x(j, \mathbf{Y}_{sink}) = \sum_{g}^{\mathcal{G}_{leaves}} (\mathbf{A}_{t-1}[g] - \mathbf{A}_t[g])$$

Third, each edge $(y_1, y_2) \in \mathcal{E}$ can only accommodate responders up to the capacity of the edge $x(y_1, y_2) \leq c(y_1, y_2)$. Since the cost is zero for edges that do not connect a responder at one end with a depot at the other end (i.e., $\textbf{Cost}(y_1, y_2) = 0$, if $y_1 \notin \mathcal{V}_{leaves} \vee y_2 \notin \mathcal{D}_{unoccupied}$), we express the objective of the formulation based on the following expression:

$$\text{Objective}_{\text{MCFP}} = \min \sum_{v}^{\mathcal{V}_{leaves}} \sum_{d}^{\mathcal{D}_{unoccupied}} \phi_t[d, v] \cdot x(v, d)$$

D. Additional Numerical Results

D.1. Description of Baselines

Due to limited space, we described some baselines briefly in the main text. Here, we provide detailed descriptions.

p-Median-Based Policy For each region, the low-level responder reallocation problem is mapped to a *p-median problem*, which assigns the responders to depots in the region so that the average demand-weighted distance between cells and the nearest depots is minimized. However, the p-median formulation does not account for responders being unavailable while serving incidents. (Vazirizade et al., 2021) modify the standard formulation by including a balancing term α in the objective, which penalizes responders that cover areas with disproportionately more incidents compared to other responders. If $\alpha = 0$, then the problem is the standard p-median problem; and $\alpha > 0$ penalizes responders that cover areas with disproportionally higher incident rates.

Greedy Policy The greedy policy is a simple heuristic approach that reallocates responders in a region to depots based the incident rates and the expected travel times to the depots, similar to the features that we use in our learning-based approach. Comparison to this heuristic approach as a baseline demonstrates the need for a more complex, learning-based approach instead of simple heuristics based on basic features, such as incident rates around depots and expected travel times. At each decision epoch, the greedy algorithm first chooses the $|\mathcal{V}^g|$ depots in region g that have the highest nearby incident rates (λ_t^d). Then, the greedy algorithm reallocates the responders to these depots using minimum-weight perfect matching based on the expected travel time between depot locations and the current positions of the responders as weights (i.e., minimizing total travel times for reallocation).

D.2. Additional Numerical Results for Nashville

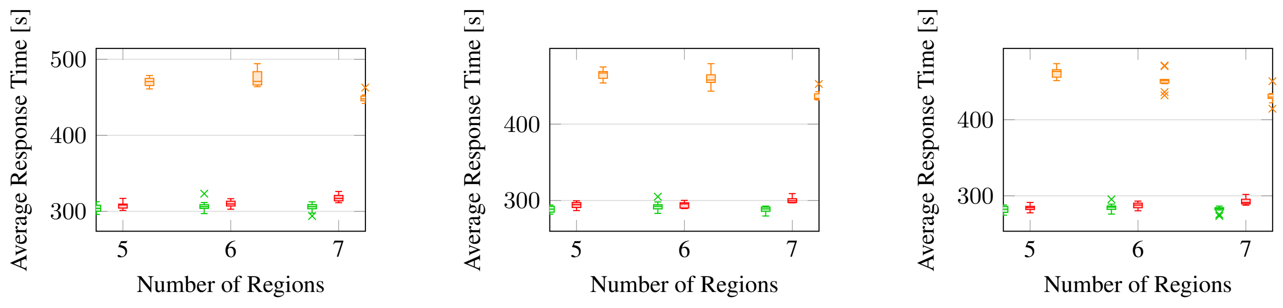
For the DRLSN baseline (Ji et al., 2019), we adapt the centralized approach to a hierarchical one similar to (Pettet et al., 2022) and use the same high-level planner as (Pettet et al., 2022) (same approach that we follow for the other baselines, such as p-median based policy, greedy policy, and static policy). The rationale behind this extension is to provide all approaches with the benefit of hierarchical planning; otherwise, DRLSN would inherently be at a disadvantage. For our experiments with DRLSN, we consider triggering both the high-level planner and the low-level planners every time an incident arrives or when there have been no incidents in the past 60 minutes, same as in (Pettet et al., 2022).

Figure 5 shows the average response times for our approach, MCTS, and DRLSN based on the same sample chains that we used in the response-time analysis in the numerical results of the main text (see Section 4.4). We observe that in all cases, DRLSN suffers from response times higher than around 450, which is 50% worse than our approach. Note that we use the same data here as in Figures 4a to 4c (in the main text); the difference in the scaling of the vertical axis between the two sets of figures is due to the inclusion of DRSLN as a baseline (which performs significantly worse than MCTS and the proposed DDPG-based approach, thereby changing the scale of the vertical axis).

Learning Curves Figures 6a to 6c show the evolution of the low-level policy μ_g for regions 1, 6 and 7 of the 7-region decomposition in Nashville. The vertical axis indicates the average response times, and the horizontal axis indicates the number of training episodes. The light green area (🟩) represents the 10th to 90th percentiles of average response times when evaluated over 5 different sample chains using 10 policies. The light gray area (⬜) represents the 10th to 90th percentiles of average response times over 15 different random policies. We observe that the trained policy performs considerably better than a random policy, even after a relatively low number of episodes.

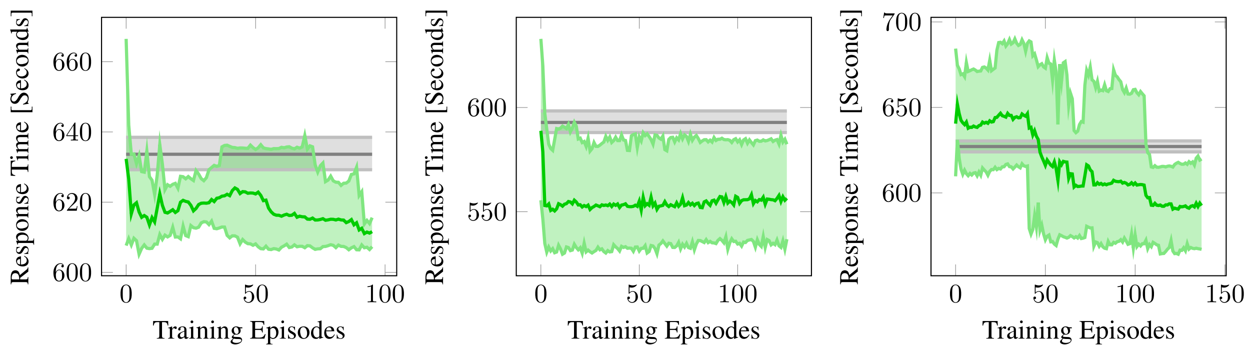
Figure 7 shows the performance of the policy μ_h for the high-level decision agent of the 7-region decomposition in Nashville. The vertical axis indicates the average response times, and the horizontal axis indicates the number of training episodes. The light green area represents the 10th to 90th percentiles of the average response times obtained at each training episode for 5 different policies. Based on the results, our high-level policy training helps to reduce the response time compared to the state-of-the-art approach.

Noisy Observations We perform experiments to verify whether our policies are resilient to noisy observations during evaluation, which can model uncertainty in predicting incident rates and travel times. As the observation values are non-negative by definition, we employ a multiplicative noise drawn from a log-normal distribution with zero mean and standard deviation ranging from 0.1 to 0.3 (note these



(a) 24 Responders (b) 26 Responders (c) 28 Responders

Figure 5: Distribution of average response times (lower is better) with our approach (🟩), MCTS (🟥), and DRLSN (🟧) for 10 different sample incident chains (**Nashville**). In this figure, we plot the same data for our approach and MCTS as in Figures 4a to 4c; the only difference is the inclusion of DRLSN, which changes the scaling of the vertical axis.



(a) Region #1 (b) Region #6 (c) Region #7

Figure 6: Evolution of the performance of the low-level policy μ_g throughout the training process for regions 1, 6, 7 of the 7-region decomposition in **Nashville**, measured as the average response time. The dark green line (🟩) indicates the average of 10 different policies (trained on the given number of episodes), which are evaluated on 5 different sample chains with the number of responders ranging from 1 to $|\mathcal{D}^g|$. The light green area (🟩) indicates the 10th to 90th percentiles of average response times over 10 different policies. The dark gray line (⬛) indicates the mean of average response time over the same set of samples when using a random policy, and the light gray area (⬜) indicates the 10th to 90th percentiles of average response times over 15 different random policies.

are the mean and std. dev. values for the normal distribution). These standard deviations roughly correspond to noise levels ranging from $\pm 7\%$ to $\pm 20\%$.

We add noise to both low-level agent observations (Arrival time and Nearby incident rate) and high-level agent observations (Region incident rates, i.e., sum of incident rates for all the cells in each region).

In Figure 8, each square shows the average response time for 10 sample chains (same as the ones used in the experimental section of our paper) for three different responder values (24, 26, and 28) with 7-region decomposition. The horizontal and vertical axes show the standard deviations of the long-normal noise for travel times and for incident rates, respectively.

We observe that the increase in average response times is not significant even with $\pm 20\%$ noise added to both observed incident rates and travel times, demonstrating that our policies are robust to uncertain predictions of incident rates and travel times.

D.3. Numerical Results for Seattle

We also evaluate our algorithm using publicly available data from the U.S. city of Seattle, WA (City of Seattle, 2022). We apply the same one-by-one mile square grid to the city to ensure the same experimental setup for the two geographical areas that we consider (i.e., for Nashville and Seattle). The city has 34 depots and 14 general hospitals to assist emergency response management. Figure 9 shows the region segmentation for Seattle data (similar figures for Nashville data can be found in (Pettet et al., 2022)). We

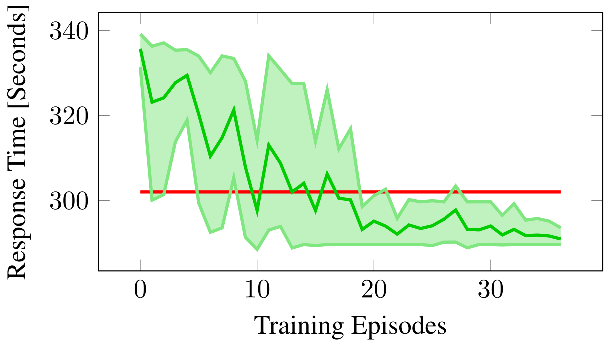


Figure 7: Evolution of the performance of the policy μ_h throughout the training process for the high-level decision agent of the 7-region decomposition in **Nashville**, measured as the average response time. The dark green line (🟩) indicates the average of 5 different policies (trained on the given number of episodes), which are evaluated on 5 different sample chains with the number of responders in the entire city ranging from 24 to 28. The light green area (🟩) indicates the 10th to 90th percentiles of average response times over 5 different policies. The red line (🟥) indicates the average response times obtained when using the state-of-the-art MCTS approach on the same 5 sample chains with the number of responders in the entire city ranging from 24 to 28.

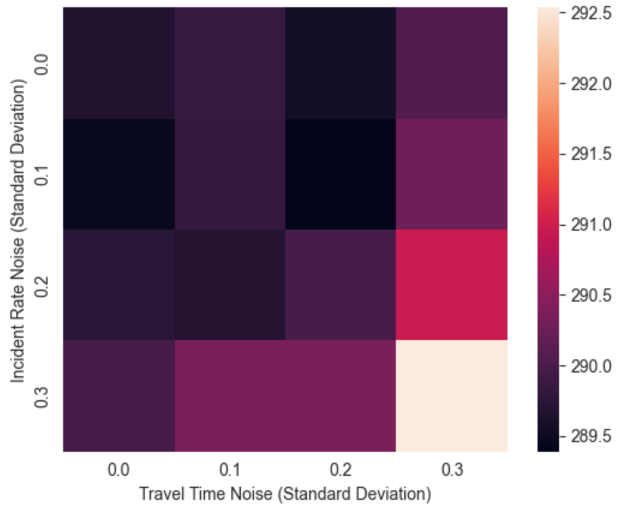


Figure 8: Caption

use 60 incident chains sampled from historical incident data distribution. Each chain is sampled over 9 days and contains between 238 and 274 incidents, averaging 240 incidents per chain. As there is no public information about how many responders are operating in Seattle, we choose the typical number of responders to be 25 (roughly consistent with the ratio of responders to depots from Nashville) and perform experiments with 23, 25, and 27 responders. Finally, we train the high-level RL agent with 25 $\pm$ 3 responders.

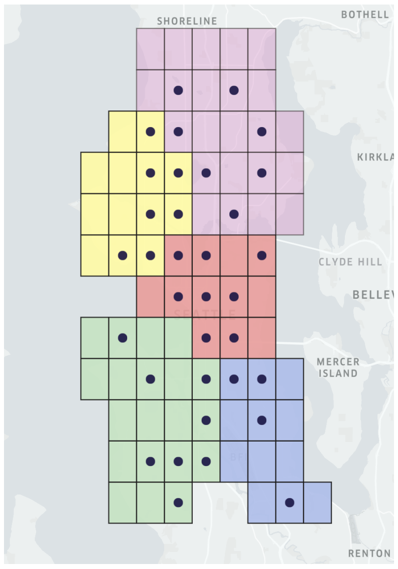
Response Times Figure 10 shows the average response times for our DDPG-based approach compared against baseline approaches (MCTS, p-median-based policy, greedy policy, and static policy), based on 10 incident chains used as the evaluation set. On average, the proposed approach outperforms MCTS by 10 seconds in all cases. We observe that p-median, greedy, and static policy always perform poorly compared to our DDPG-based approach, and they perform mostly poorly compared to the state-of-the-art approach (except for 3 out of 9 scenarios based on the number of regions and number of responders). Figure 11 shows the average response times for our approach, MCTS, and DRLSN based on the sample chains used in the previous analysis (again, we show results with DRLSN separately as both other approaches outperform it by a large margin, thereby making the difference between DDPG and MCTS difficult to observe when shown together). We observe that DRLSN always has response times higher than 150-200 seconds, which is at least 50% worse than our approach.

Learning Curves Figures 12a to 12c show the performance of the low-level policy μ_g for regions 1, 2, and 3 of the 7-region decomposition in Seattle. The vertical axis indicates the average response times, and the horizontal axis indicates the number of training episodes. The light green area (🟩) represents the 10th to 90th percentiles of average response times over 10 different policies. The light gray area (⬜) represents the 10th to 90th percentiles of the average response times over 15 different random policies. We observe that our trained policy is significantly better than the random policy and learns fast.

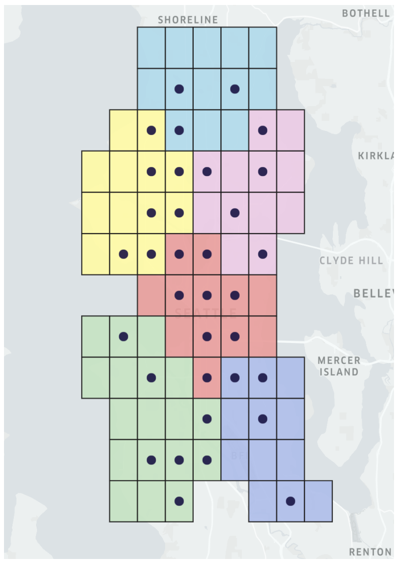
Figure 13 shows the performance of the policy μ_h throughout the training process for the high-level decision agent of the 7-region decomposition in Seattle, measured as the average response time. The light green area represents the 10th to 90th percentiles of the average response times over 10 different policies. The red line indicates the average response times obtained using the state-of-the-art MCTS approach on the same 5 sample chains with the number of responders ranging from 23 to 27. We observe that our high-level policy training helps to reduce the response time compared to the state-of-the-art approach.

D.4. Architecture Search for Low-Level RL Agent

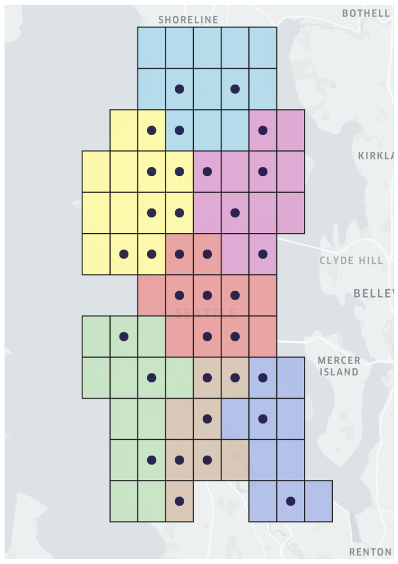
We experiment with architectures such as LSTM, TrXL (Vaswani et al., 2017; Dai et al., 2019; Parisotto et al., 2020), and Gated Transformer-XL (GTrXL) (Parisotto et al., 2020; Parisotto & Salakhutdinov, 2021) as neural-network architecture choices for the actor in the low-level agent. We



(a) 5 Regions

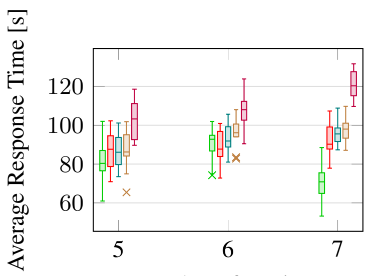


(b) 6 Regions

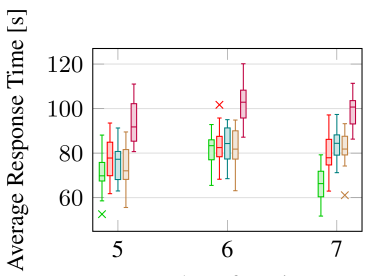


(c) 7 Regions

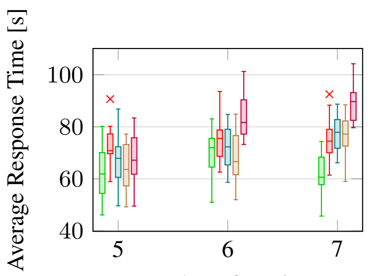
Figure 9: Segmentation of Seattle into 5, 6, and 7 regions. Dots on the map indicate depots where responders can wait.



(a) 23 Responders

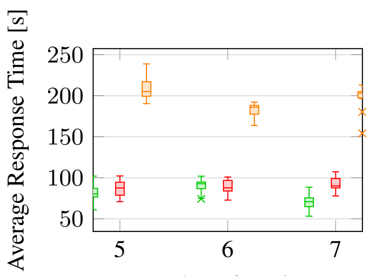


(b) 25 Responders

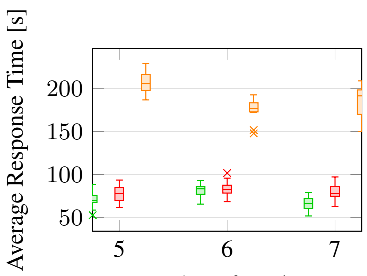


(c) 27 Responders

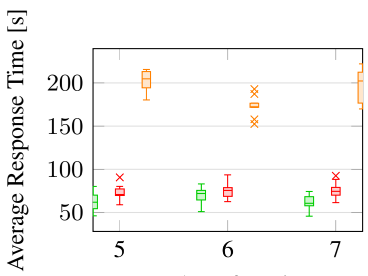
Figure 10: Distribution of average response times (lower is better) with our approach (🟩), MCTS (🟥), p-median with $\alpha = 1.0$ (🟦), greedy policy (🟫), and static policy, i.e., no proactive repositioning (🟪) for 10 different sample incident chains (**Seattle**).



(a) 23 Responders

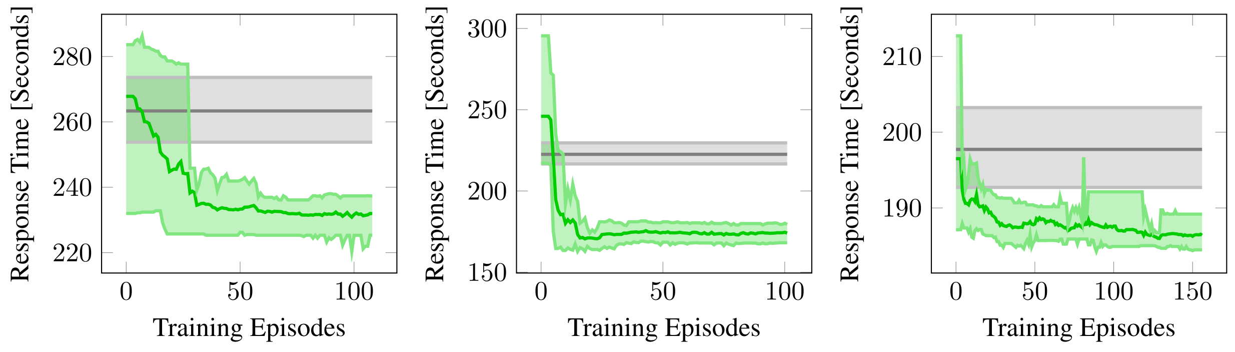


(b) 25 Responders



(c) 27 Responders

Figure 11: Distribution of average response times (lower is better) with our approach (🟩), MCTS (🟥), and DRLSN (🟧) for 10 different sample incident chains **(Seattle)**. In this figure, we plot the same data for our approach and MCTS as in Figure 10; the only difference is the inclusion of DRLSN, which changes the scaling of the vertical axis.



(a) Region #1 (b) Region #2 (c) Region #3

Figure 12: Evolution of the performance of the low-level policy μ_g throughout the training process for regions 1, 2, and 3 of the 7-region decomposition in **Seattle**, measured as the average response time. The dark green line (🟩) indicates the average of 10 different policies (trained on the given number of episodes), which are evaluated on 5 different sample chains with the number of responders ranging from 1 to $|\mathcal{D}^g|$. The light green area (🟩) indicates the 10th to 90th percentiles of average response times over 10 different policies. The dark gray line (⬛) indicates the mean of average response time over the same set of samples when using a random policy, and the light gray area (⬜) indicates the 10th to 90th percentiles of average response times over 15 different random policies.

perform a random architecture search to obtain the best hyperparameters for each architecture (i.e., LSTM, TrXL, GTrXL). Accordingly, we tune the following hyperparameters in the actor-network using TrXL:

- Hidden layers in MLP: 1 / 2 / 3 layers
- Number of neurons per hidden layers in MLP: 32 / 64 / 128 / 256 neurons per layer
- Dropout rate in MLP (after every hidden layer): 0.0 / 0.0125 / 0.025 / 0.05 / 0.1
- Number of attention heads: 1 / 2 / 3 / 4 / 5 heads
- Number of layers (N): 1 / 2 / 3 layers

We perform the architecture search for each region in the segmentations (i.e., 5, 6, and 7 regions). We terminate the architecture search after obtaining an actor-network that can outperform the competitive baseline using MCTS. Tables 2 and 3 show the best hyperparameters for the TrXL based low-level agent for each region (with 5, 6, and 7 region segmentations) for Nashville and Seattle data, respectively.

E. Statistical Tests of Significance

In this section, we present the results of statistical tests on the significance of our numerical results. Specifically, we present paired two-sample permutation tests comparing the average response times of our proposed approach to the average response times of each baseline approach based on a sample of 10 incident chains in each case. We perform these tests for various numbers of regions (5, 6, and 7) and various numbers of responders for both Nashville and Seattle. Since our goal is to compare the average response times, we establish the following null and alternative hypotheses:

- *Null Hypothesis* (H_0): mean of the response times obtained using our DDPG-based approach is the same as the mean of the response times obtained using the baseline (i.e., MCTS, p-median-based policy, greedy policy, static policy, or DRLSN);
- *Alternate Hypothesis* (H_1): mean of the response times obtained using our DDPG-based approach is significantly different from the mean of the response times obtained using the baseline (i.e., MCTS, p-median-based policy, greedy policy, static policy, or DRLSN).

We use the difference between the means as our test statistic.

Tables 4 and 5 shows the p-values to reject the null hypothesis based on samples of 10 incidents chains for Nashville and Seattle, respectively. We observe that for Nashville, *the average response times obtained using our DDPG-based approach are significantly better than those of the baselines* (considering the significance threshold for the p-value to be 5%), except in two cases: 6 region decomposition with 24 and 26 responders with the state-of-the-art MCTS approach. Note that even in these cases, the p-value is low, suggesting that the null hypothesis is likely false, and the key advantage of our approach over MCTS is not in lowering response time but in lowering running time by several orders of magnitude. Similarly, for Seattle, we observe that the average response times obtained using our DDPG-based approach are significantly better than those of the baselines in most cases, except for a few scenarios. Again, we observe that MCTS is close in two cases, but the main advantage of our approach is significantly lowering running time in all cases. We also observe similarity in a few other cases. Note that we performed the tests with relatively small samples (10

Table 2: Best Hyperparameters for Low-Level Agent Actor using TrXL (**Nashville**)

Number of regions	**5**				
Region identifier	0	1	2	3	4
Number of layers (N)	1	3	3	2	3
Number of heads in MHA	5	3	3	1	5
MLP	256, 128	128, 64	128, 64	256, 128	64
Dropout rate	0.1	0.1	0.1	0.0	0.05

Number of regions	**6**					
Region identifier	0	1	2	3	4	5
Number of layers (N)	2	3	3	3	1	2
Number of heads in MHA	1	2	5	5	2	4
MLP	256, 128	64	32	64	128, 64	256, 128
Dropout rate	0.05	0.0	0.1	0.1	0.0	0.0

Number of regions	**7**						
Region identifier	0	1	2	3	4	5	6
Number of layers (N)	2	3	1	1	2	2	1
Number of heads in MHA	3	1	3	5	4	3	5
MLP	256, 128, 64	256, 128, 64	128, 64	64	32	256, 128	64
Dropout rate	0.1	0.1	0.1	0.05	0.0	0.05	0.0

Table 3: Best Hyperparameters for Low-Level Agent Actor using TrXL (**Seattle**)

Number of regions	**5**				
Region identifier	0	1	2	3	4
Number of layers (N)	3	3	1	3	2
Number of heads in MHA	1	3	3	1	5
MLP	256	256	32	256	32
Dropout rate	0.0125	0.0	0.1	0.0125	0.0125

Number of regions	**6**					
Region identifier	0	1	2	3	4	5
Number of layers (N)	1	1	1	1	2	1
Number of heads in MHA	5	3	5	5	4	5
MLP	128	128, 64	256	128, 64	256	128
Dropout rate	0.1	0.0125	0.1	0.025	0.0125	0.1

Number of regions	**7**						
Region identifier	0	1	2	3	4	5	6
Number of layers (N)	1	3	1	2	2	2	2
Number of heads in MHA	1	1	2	5	4	3	2
MLP	64	256	64	256, 128, 64	64	128, 64	256, 128, 64
Dropout rate	0.025	0.1	0.05	0.0	0.0	0.05	0.0125

Table 4: p-Values to Reject the Null Hypothesis (**Nashville**)

Number of regions	**5**			**6**			**7**		
Number of responders	24	26	28	24	26	28	24	26	28
MCTS	0.01	0.01	0.02	**0.14**	**0.20**	0.02	0.00	0.00	0.00
p-median-based policy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
greedy policy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
static policy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
DRLSN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 5: p-Values to Reject the Null Hypothesis (**Seattle**)

Number of regions	**5**			**6**			**7**		
Number of responders	23	25	27	23	25	27	23	25	27
MCTS	0.04	0.01	0.00	**0.37**	**0.36**	0.04	0.00	0.00	0.00
p-median-based policy	0.03	0.02	0.02	**0.07**	**0.07**	**0.16**	0.00	0.00	0.00
greedy policy	**0.05**	**0.09**	0.02	0.00	**0.55**	**0.19**	0.00	0.00	0.00
static policy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
DRLSN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

chains, i.e., 10 values); more extensive experiments could lead to lower p-values.

F. Extended Related Work

In Section 5, we provided a concise summary of the most relevant prior work due to the limited space. In this section, we provide a broader discussion of related work.

F.1. Emergency Response Management

Prior works use *centralized* (Schjølberg et al., 2023; Ji et al., 2019), *decentralized* (Pettet et al., 2020), and *hierarchical* (Pettet et al., 2022) approaches to solve the ERM reallocation problem. Schjølberg et al. (2023) use a genetic algorithm-based solution to the responder redistribution problem. However, Schjølberg et al. (2023) reposition responders only at shift changes (e.g., from day shift to night shift) throughout the day. In contrast, our approach performs proactive reallocation whenever a new incident arrives or when the arrival rates of incidents change. Ji et al. (2019) reallocate each responder once it finishes serving its current incident assignment. In contrast, in our approach, we allow complete reallocation at each incident arrival and at each change in the rate of future incidents. In Ji et al. (2019)'s approach, during each reallocation step, the trained RL agent outputs a score for all available depots (based on features such as nearby incident rates for the depot and the expected time for the nearest-k responders to reach the depot) and chooses to assign the responder to the depot with the highest score. In our approach, we consider similar features when performing the reallocation: we consider features such as nearby incident rates for each depot and the expected time for a responder to reach a depot.

F.2. Dispatching Problem

There are two key differences between order-dispatching approaches, such as the works (Zhou et al., 2019; Li et al., 2019) and proactive allocation for emergency response. First, order-dispatching approaches focus on responding to requests by optimizing the dispatch of vehicles when new requests arrive. In emergency-response management (ERM), dispatch decisions cannot be optimized as ERM systems are typically mandated to always dispatch the nearest responder (Pettet et al., 2022; Ji et al., 2019; Mukhopadhyay et al., 2020). Therefore, similar to prior work, we focus on optimizing the allocation of responders in anticipation of the arrival of future requests. Second, more importantly, both works (Zhou et al., 2019; Li et al., 2019) assume high-level states and actions, defined in terms of numbers of vehicles in given areas. In contrast, we consider fine-grained states and actions, defined in terms of allocating specific vehicles to specific locations (depots). High-level states and actions are appropriate for managing large-scale ride-sharing services with hundreds (or thousands) of vehicles (Zhou et al., 2019; Li et al., 2019); however, emergency response requires fine-grained management of individual responders as every second counts. Our work focuses on the computational challenges that arise from the combinatorial nature of these fine-grained states and actions.

F.3. Transformers

Dai et al. (2019) introduce a transformer variant that can work with variable-length inputs. Parisotto et al. (2020)

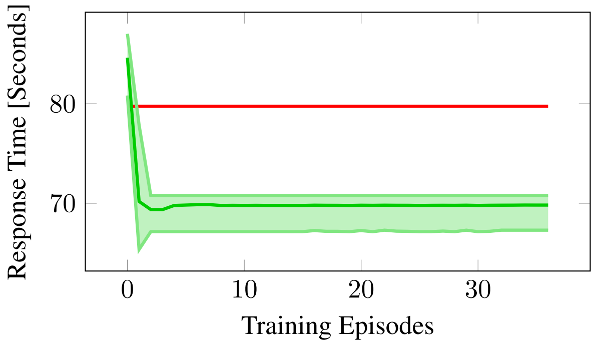


Figure 13: Evolution of the performance of the policy μ_h throughout the training process for the high-level decision agent of the 7-region decomposition in **Seattle**, measured as the average response time. The dark green line (🟩) indicates the average of 10 different policies (trained on the given number of episodes), which are evaluated on 5 different sample chains with the number of responders in the entire city ranging from 23 to 27. The light green area (🟩) indicates the 10th to 90th percentiles of average response times over 5 different policies. The red line (🟥) indicates the average response times obtained when using the state-of-the-art MCTS approach on the same 5 sample chains with the number of responders in the entire city ranging from 23 to 27.

introduce stabilization over TrXL, via performing normalization before MHA and MLP; further, they concatenate the residual connections using a Gated Layer, and the complete architecture is often called Gated TrXL (GTrXL). To train our low-level RL agents, we try both TrXL and GTrXL variants. However, in contrast to the results of Parisotto et al. (2020), we find that TrXL can perform better than GTrXL in our problem setting. Accordingly, we use TrXL as our neural network architecture for low-level RL agents.

F.4. Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning (HRL) solves complex tasks by training agents to make decisions over multiple levels of temporal abstraction (Pateria et al., 2021; Hutsebaut-Buysse et al., 2022; Xu et al., 2023). Pateria et al. (2021) categorize HRL based on the number of agents in the system (i.e., single agent or multi-agent), nature of the tasks (i.e., heterogeneous or homogeneous) and whether or not the policy has to discover the sub-tasks. Xu et al. (2023); Jin et al. (2019) apply HRL in MARL that operates cooperatively. In contrast to HRL, in our work, we consider a MARL system with heterogeneous agents (i.e., a set of low-level agents with the task of repositioning responders within their designated regions and a high-level agent with the task of redistributing responders among regions) that acts cooperatively to achieve the goal of minimizing the response time for serving incidents.

G. Rationale behind Application of DDPG

Our approach uses the well-known DDPG algorithm to train both low-level and high-level agents. However, the DDPG actor outputs a continuous action. In contrast, our environment expects a discrete action that represents the assignment of responders to depots or the redistribution of responders to regions. To discretize the continuous actor action, we use combinatorial optimization techniques.

One seemingly trivial solution to the problem above is using an RL algorithm that works well with discrete action spaces, such as Deep Q-Learning Network (DQN) (Mnih et al., 2015) or Soft-Actor Critic (SAC) (Haarnoja et al., 2018). While these RL algorithms are often straightforward in terms of computing the state-action value, they run into scalability issues when the space of possible actions grows prohibitively large. For example, consider a region with 10 depots and 10 responders. There are $10! \approx 3 \times 10^6$ possible ways to reposition responders between depots. Even if we can infer the value of a single repositioning action at the speed of 0.01 seconds, we still require 10^4 seconds (around 3 hours) to compute the values of all possible allocations. In contrast, using DDPG can make a single decision in a fraction of a second (0.22 seconds).

Another way to tackle the discretization problem is to apply an RL approach that assumes a parametric distribution over the action space, whose parameters are to be estimated by the actor (e.g., Proximal Policy Optimization (PPO) (Schulman et al., 2017), SAC (Haarnoja et al., 2018), Q-functionals (Lobel et al., 2023)). In this case, the actor output would be similar to that of our DDPG actor (i.e., compact marginal of a larger distribution), losing the advantages of these RL algorithms while having to deal with additional challenges, such as sampling from a combinatorial discrete action space.