Principled Algorithms for Real-time Sequential Decision Making for Large Scale Cyber-Physical Systems

By

Geoffrey Pettet

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

December 17, 2022

Nashville, Tennessee

Approved:

Acknowledgements

This research was supported in part by the Tennessee Department of Transportation, the Laboratory Directed Research and Development program at Pacific Northwest National Laboratory, the U.S. Department of Energy under awards DE-AC05-76RL01830 and DE-EE0009212, and the National Science Foundation under award numbers CNS-1640624, IIS-1814958, IIS-1905558, and CNS-1952011.

I am grateful to my advisor Dr. Abhishek Dubey for his mentorship and support during my graduate studies. He has helped me become a better academic without letting me forget that I'm also a software engineer. I am also thankful to my committee members Dr. Janos Sztipanovits, Dr. Gautam Biswas, Dr. Jules White, Dr. Dan Work and, Dr. Hiba Baroud for their support, guidance, and insights into this work.

I am also grateful to Dr. Malini Ghosal, Dr. Shant Mahserejian, and Dr. Siddharth Sridhar at Pacific Northwest National Laboratory for their mentorship and for aiding me in applying my research to new and exciting domains.

I am also grateful to Colleen Herndon for her guidance and for facilitating our collaboration with the Metro Nashville Information Technology Services and Metro Nashville Fire Department.

I would like to give a special thank you to Dr. Ayan Mukhopadhyay for his close collaboration and guidance over the course of this work. Thank you to Dr. Douglas Schmidt for his mentorship during my undergraduate studies and for helping me prepare for graduate school. Thank you to Dr. Scott Eisele for his support over the past several years. And thank you to all of my other collaborators and mentors through the years, including Dr. Yevgeniy Vorobeychik, Dr. Aron Laszka, Dr. Mykel Kochenderfer, Dr. Sayyed Mohsen Vazirizade, Dr. Kyle Wray, Chinmaya Samal, and all of my friends at the SCOPE-Lab.

Finally, I am especially grateful to my family: to my mother Lori Pettet, father Mark Pettet, and brother John Pettet for their love and support over all these years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Cyber-Physical Systems (CPS) are composed of devices that integrate software, networking, and physical processes. Advances in embedded computing, networking, and cloud architectures have allowed scalable data collection and processing, which has enabled the breath and scale of CPS applications to increase. As such, we are now at a point where CPS are being scaled to applications that can effect entire cities and societies, which we refer to as *Societal-Scale Cyber-Physical Systems* (SCPS), including energy distribution [1], transportation [2, 3, 4], and emergency response services [5, 6, 7]. These systems leverage the staggering quantity of data that can be collected in urban environments to learn how to better manage resources, services, and infrastructure. Managing such a system is a complex decision-making problem that involves *forecasting* how the environment is likely to evolve in the future and *planning* to make decisions that maximize the long-term utility of the system.

Let us consider three important SCPS applications to contextualize the technical challenges in realizing such large scale CPS. First is the problem of emergency response management (ERM). Emergency incidents occur frequently throughout urban areas, and must be serviced by emergency response vehicles such as ambulances and fire trucks. ERM is the problem of managing the locations and optimizing the dispatch of these response vehicles to incidents such that response times are minimized. Second, consider the problem of managing the charging schedules of electrified fleets of vehicles – electrified vehicles (such as public transit or delivery vehicles) must be charged throughout the day. Chargers are spatially distributed around the city, so vehicle routes must be integrated with a charging schedule to balance cost and strain placed on the power grid while maintaining required levels of service. Last, consider shared on-demand transit (i.e. ridepool). Recent research has shown that on-demand transit services can be made more cost and time efficient by allowing vehicles to service multiple requests simultaneously [4]. However, allowing shared vehicles increases the combinatorial complexity of the request to vehicle matching problem significantly; this complexity must be managed by ridepool applications.

Each of these applications are varients of the SpatioTemporal Resource Management (STRM) problem, which often manifests in SCPS. *Resources* are physical or logical entities that have some spatiotemporal properties. Often, these resources are mobile – ambulances in ERM, electric vehicles, and rideshare taxis are all examples of mobile resources. These resources often have to service some *demand*, which is application specific – ambulances must respond to emergency incidents, electric vehicles must charge at spatially located chargers when they run low on energy, and taxis must pickup and drop off customers. This demand is often heterogeneously distributed around the city, and is not known in advance. STRM is the problem of managing the spatial distribution of these resources, as well as how they respond to demand such that some utility is optimized. We refer to optimizing the spatial distribution of the resources when not servicing demand as *allocation* – this is how the system prepares for demand. We refer to the actions taken in response to demand as *dispatching*. Effective STRM is important – for some systems such as ridepooling, inefficient deployments can result in increased costs and unhappy users. For others it is safety critical, such as emergency response management. STRM is used throughout this dissertation to illustrate the challenges associated with designing and deploying SCPS.

It is important to consider two key properties of the environments in which SCPS operate. First, they are spatially heterogeneous. No two communities are exactly alike in their layouts, and within a city there is much variation in the distribution of road networks, population density, and demographics. Second, large geographic areas are highly non-stationary - in the short term, features such as traffic and weather change throughout a day, and impactful events like football games can cause significant disruption in normal behavioural patterns. Communities also evolve over long time scales; infrastructure such as roadways

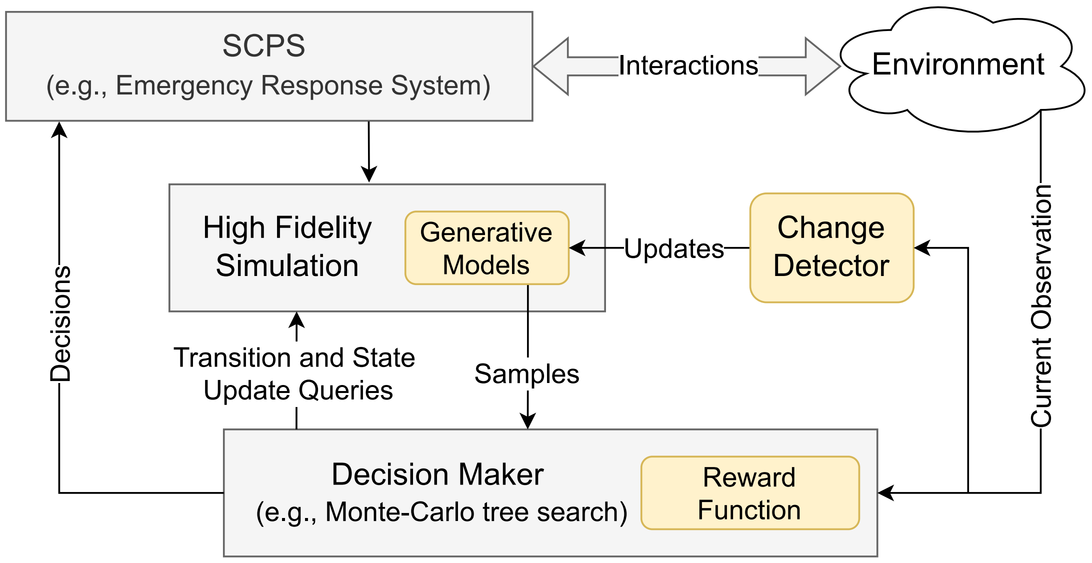

Figure 1.1: Overview of the proposed decision support framework for online planning in SCPS. The decision-maker queries a simulation of the SCPS and its environment to estimate the impact of potential action trajectories on how the environment might evolve, and the actions' corresponding values using a reward function. The simulation is built upon generative models of the environment, which are updated as changes in the environment are observed.

are constantly being built, population demographics shift, and governing bodies change. These properties make SCPS applications difficult to manage, since they must adapt to their diverse, heterogeneous environment as it evolves.

The objective of this dissertation is to study what is required for *proactive* SCPS approaches that foresee how the system will evolve, capture the long-term value of actions, and make decisions that optimize the expected long-term utility of the SCPS. Proactive management requires an integrated decision support framework that includes forecasting and planning components, as shown in Fig. 1.1. In this proposed framework, a decision-maker uses a simulation of the SCPS and its environment to query how the system is likely to evolve given potential action trajectories, and the actions' corresponding values using a reward function. The simulation is built upon generative models of the environment, which are updated as changes in the environment are observed.

Each of these components serve an important role in the operation of a resource management pipeline, and come with challenging technical problems when scaled to SCPS and when applied to non-stationary environments. The next two sections will now give an overview of these challenges and potential solution methods for the two components of interest in this dissertation: (1) scalable and adaptive planning methods (section 1.1) and high-resolution generative forecasting models (section 1.2).

1.1 Planning

A proactive SCPS pipeline must include a planning algorithm that makes decisions to maximize the long-term utility of the system. STRM dispatching and allocation decision making, for example, should incorporate the uncertain output of demand detection and forecasting models to determine actions that will serve current demand while foreseeing the decision's effect on future states of the system.

A popular strategy used to plan under uncertainty is to learn a *policy*, which is a general mapping from states of the environment to actions that should be taken. Reinforcement Learning (RL) and Approximate Dynamic Programming both fall into this category, and have recently been applied to STRM problems [6, 8, 9, 10]. To use these methods, practitioners first create a detailed model of the environment, and then perform computation offline to learn a policy. Unfortunately, these methods take a long time to converge to a policy when applied to large scale, practical problems. Since urban environments are highly non-stationary, this means that a policy might be out of date by the time it has been learned. They are also not resilient to failures, since they are dependent on the environment that was defined during learning being a good approximation of the real environment – equipment failures and large environmental shifts, such as those caused by disasters, invalidate the policies.

Rather than aiming to find a policy for the entire state-space offline, an alternative approach is to perform online planning and focus on finding a good action for the current state of the world. Given a generative model of the environment, heuristic search approaches

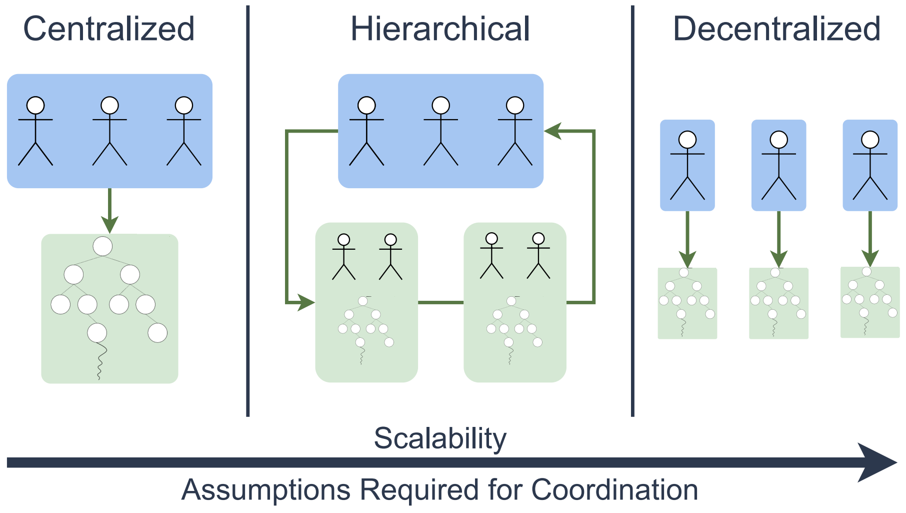

Figure 1.2: The spectrum of divide and conquer approaches for decision-making in multi-agent SCPS. A completely centralized approach uses a monolithic state representation. In a completely decentralized approach, each agent simulates what other agents do and performs its own action estimation. A hierarchical approach segments the planning problem into sub-problems to improve scalability without agents estimating other agents' actions.

such as Monte-Carlo tree search (MCTS) can be used to find promising actions for the environment as it is at the time of decision-making. An advantage of using MCTS is that the Markovian assumption can be relaxed, and high-fidelity simulators can be used to estimate utilities from different actions. These simulators can be updated to reflect changes in the environment, making MCTS adaptive to environmental shifts. Such an approach has been applied to real-time control problems in domains such as the smart grid [11], game playing [12, 13], and autonomous driving [14, 15]. However, state of the art MCTS approaches have difficulty converging in a reasonable amount of time for practical SCPS problems with large state-action spaces [16].

One approach to scale online planning approaches to complex environments is the established "divide and conquer" strategy, which is to break down the large problem into manageable sub-problems, and then combine these solutions. Such strategies exist on a spectrum for multi-agent SCPS, as shown in figure 1.2. On one side of the spectrum is fully centralized decision-making, which represents the entire environment using a monolithic state-space. A centralized approach has the advantage of making no assumptions regarding the interactions between agents, and most accurately represents the environment. Unfortunately, such an approach quickly becomes intractable as the scale of the problem increases and the interactions between agents explode the state-action space.

On the other side of the spectrum is a completely decentralized method, where each agent performs local decision-making to determine its own course of action. A decentralized approach is extremely scalable, since each agent can ignore any interactions that are not directly relevant to its own decisions. The agents' independence also makes this approach robust to communication failures. However, there are several challenges to implementing a decentralized decision-maker. As the agents cooperate to achieve a single goal, they must estimate what other agents will do in the future as they optimize their actions. Note that high fidelity models for estimating agents' actions limit scalability, but inaccurate models can lead to sub-optimal decisions. It is important that agent's find an appropriate trade-off between these two competing issues.

Finally, there is the middle ground of hierarchical planning [17], which focuses on learning local policies, known as *macros*, over manageable subsets of the state space by leveraging the spatial structure in the environment. It is motivated by the concept of *jurisdictions* or *action-areas*, which create different zones to partition and better manage infrastructure. These subproblems encompass multiple agents, allowing nearby agents that are most likely to interact to coordinate. This facilitates high scalability while retaining the coordination between the most relevant agents. Implementing such an approach requires overcoming several challenges, including how to split the full environment into subproblems that will lead to the best inter-agent coordination, and how these sub-problems should evolve as the environment changes. A decision-maker must also define the coordination procedure between the subproblems, as some actions will likely require agents to cross the subset boundaries.

Each of these methods for scalable online planning require knowledge and assumptions of the environment and its structure. A different method is to consider a hybrid decision-making approach that combines the strengths of RL and online planning while mitigating some of their weaknesses. Such an approach is used by the state-of-the-art AlphaZero algorithm, which uses MCTS as a policy improvement procedure while learning an RL policy for stationary environments [18]. The intuition behind using a hybrid approach in non-stationary settings is that if the environment has not changed too much between when an optimal policy was learned and when a decision needs to be made, the policy could still provide useful information for decision-making. The policy's estimates could be used to guide the search, even if the estimates are stale. The challenges with this approach are to determine precisely how to integrate the learned policy with the online search, and what (if any) guarantees can be made about the optimality and robustness to non-stationarity of the resulting algorithm.

This dissertation explores each of these approaches, and presents techniques to overcome their unique challenges. The advantages and limitations of each are presented, such that practitioners can evaluate which is most applicable to a given domain.

1.2 Forecasting

In order to proactively plan, one must have an understanding of how the SCPS and it's environment evolves over time. This encompasses several aspects of the environment. In ERM, for example, the features of interest include traffic congestion and travel times, weather features such as precipitation and visibility, large events such as sports games or festivals, and demand such as traffic and medical incidents. Many of these are co-dependent — for example, one must model where resources will likely be in the future using a travel model, which is dependent on knowing how traffic will evolve throughout each vehicle's route. Accurately modeling such features is key to estimating the long-term value of actions that are taken by the decision maker.

Modeling SCPS environments is difficult due to the challenges of heterogeneity and non-stationarity. Take motor vehicle accidents (MVA), for example. These incidents are known to be inherently random [19], and are dependant on many environmental factors such as weather, traffic, and even other incidents (incidents can 'cascade', causing more incidents). Techniques such as the negative binomial distribution [20], artificial neural networks [21], and hierarchical analysis [22] have been used to great effect when attempting to predict incident frequency for specific areas, and have helped determine features of roadways that affect incident occurrence. Prediction methods such as the negative binomial regression [23] and random effect probit models [19] have also been used to analyze feature effects on accident frequency, and generating predictive models for specific areas. Unfortunately, these studies generally make assumptions about the locations that they are analyzing. For example, they study a specific length of freeway, or look at only intersections and their features in a specific city. These approaches do not easily scale to the large, heterogeneous environments of SCPS.

To be useful for proactive decision-making, these models must also have high spatio-temporal resolution, so that a decision algorithm can have a precise understanding of the locations and times where events are likely to occur. Unfortunately, increasing the resolution of models introduces *sparsity*, meaning that there are significant class imbalances between positive and negative samples in the data, since each spatiotemporal window is less likely to see any significant demand. Therefore, adaptive forecasting frameworks are needed that can model the high-dimensional feature spaces of SCPS environments at high spatio-temporal resolutions.

1.3 Motivating Research Questions

To create a principled framework for decision-making in SCPS, this dissertation addresses the following research questions:

1. **Online planning approaches require a foundation of high resolution generative models and simulators. How can we construct an integrated decision support framework that combines these models and procedures to update them with on-**

line planners? SCPS environments are often highly non-stationary. Online planning approaches can adapt to changes in the environment, as they perform their computation at decision time using high-fidelity models and simulations to determine promising action trajectories. These models can be updated as soon as environmental changes are detected, and such changes can be immediately incorporated in decision-making. However, an integrated decision support framework is needed to efficiently integrate these models with the online planner and update them as changes occur. This dissertation presents such a framework, and demonstrates that it facilitates adaptive decision-making procedures.

2. **Centralized online planning approaches cannot scale to the complex state-action spaces of SCPS. How can these problems be split into manageable subproblems?** As the number of agents increases in a SCPS, the complexity of the state-action space explodes due to their interactions. Standard, centralized planning approaches such as Monte-Carlo tree search take a significant amount of time to converge in these large environments, as they must evaluate many different potential action trajectories at decision time. One way to address this is a hierarchical approach that splits the large SCPS into smaller subproblems that can be solved tractably using MCTS. This dissertation presents a hierarchical online planning approach that includes a high-level planner to split the problem into subproblems and coordinate between them, and a low-level planner that performs planning for each subproblem.

3. **Centralized and hierarchical planning rely on network infrastructure to coordinate between agents. How can we create adaptive planning algorithms that are robust to failures in communication networks?** Many SCPS, such as emergency response services, serve safety-critical functions for a society. Such systems must be robust to failures in their communication networks, which centralized and hierarchical planning approaches rely on to coordinate between agents. This dis-

sertation presents a decentralized planning approach that allows for each agent to independently determine its own course of action with limited communication. We investigate methods for agents to estimate the behavior of other agents during planning, and how to ensure system constraints are satisfied despite agents performing local planning.

4. **Policies that are learned offline likely encode useful information even if the environment has changed since training time. How can these stale policies be used to improve online planning algorithms?** The policies learned by learning approaches such as reinforcement learning (RL) become stale when the environment changes. However, we hypothesize that if the changes are relatively minor, the policy still encodes information that is relevant to the updated environment. How can we utilize this knowledge to improve the convergence of online search approaches? In this dissertation we present a hybrid approach that is more adaptive to environmental changes than RL approaches while converging significantly faster than online planning in isolation.

5. **Understanding the long-term impact of decisions requires environmental forecasting models with high spatiotemporal resolution that generalize to heterogeneous areas. How can we learn such models despite high data sparsity?** The foundation of online planning algorithms are the models they use to forecast how the environment will evolve and estimate the long-term value of actions. SCPS often encompass large and heterogeneous geographic areas, and are dependent on a diverse set of environmental features. These models must have a high spatial-temporal resolution to be useful for precise planning, which often results in high data sparsity. This dissertation presents a method for clustering sparse events using heterogeneous features across a region. We show that these groups have similar arrival distributions, which makes forecasting for each group more accurate. These groups are then

mapped to spatial locations to gain models with high spatial-temporal resolution.

1.4 Outline

The remainder of this dissertation describes the work performed to answer these research questions posed to develop principled decision-making frameworks for SCPS. The key challenges, the innovations presented to address them, and the related publications are summarized in Table 1.1.

Chapter 2 details the Emergency Response Management (ERM) domain, which is used to demonstrate the forecasting and planning approaches presented throughout this dissertation. ERM dispatch and allocation are safety critical problems faced by communities across the globe, and have properties such as tight time constraints on decision making and large state-action spaces that illustrate the challenges that often arise when managing a SCPS. Focusing on one well defined SCPS problem allows us to discuss various technical approaches in greater detail; however, the approaches presented in this dissertation are generalizable to other SCPS problems, which is demonstrated in Chapter 8.

Chapter 3 presents a framework for centralized online planning in SCPS. It breaks down the problem into three atomic sub-components: generative environmental models, a simulation of the environment, and the decision-making agent. It presents an online method for updating the generative models. It then composes these components into an integrated decision support framework. The framework is applied to the problem of Emergency responder dispatching in a simulation of Nashville, TN, and we find that the proposed approach can decrease the response time for *specific subsets of incidents* by up to **39 seconds** on average in resource constrained situations when compared to existing responder dispatching strategies. However, this approach is not scalable to more complex SCPS problems, such as ERM responder allocation.

Chapter 4 presents a hierarchical planning framework which focuses on learning local policies, known as *macros*, over subsets of an SCPS state space by leveraging the spatial

Table 1.1: Summary of the key research challenges, the innovations presented in this dissertation to address them, and the related publications.

Research Question	Outcomes	Innovations	Publications
(Q1) Constructing an integrated framework for SCPS decision-making.	A flexible template for SCPS decision support systems.	A novel template for software frameworks that combines generative models of an SCPS environment, procedures to update these models, a high-fidelity simulation of the SCPS, and a decision agent. This template is generalizable to many SCPS domains.	[24, 25] [26, 27]
(Q2) Scalable online planning for non-stationary SCPS.	Hierarchical adaptive planning procedures.	A hierarchical planning framework that uses a high-level planner to (1) split up the system into tractable sub-problems, and (2) coordinate between the sub-problems. Two novel coordination procedures are developed: an algorithm based on queuing theory, and an algorithm using random forests A low-level planner uses Monte-Carlo tree search to independently make decisions for each sub-problem.	[28, 16]
(Q3) Planning that is robust to communication network failures.	Decentralized adaptive planning procedures.	A decentralized planning framework where each agent independently determines its own course of action with limited communication. Inexpensive models of agent behavior allow an agent to forecast what other agents will do to avoid conflicts. A filter ensures that system-wide constraints are satisfied.	[29]
(Q4) Incorporating learning-based models with online planning.	Policy-augmented Monte-Carlo tree search (PA-MCTS).	A novel definition of non-stationary Markov decision processes for SCPS. A hybrid decision-making approach that uses policies learned offline on out-of-date versions of the environment to improve the convergence speed of online planning in non-stationary settings. Several properties of the algorithm are proven, such as when it will choose a better action than either of its constituent policies.	[30]
(Q5) Creating high -resolution forecasting models for sparse data.	Clustering based spatial-temporal forecasting framework	A forecasting framework that uses clustering to aggregate similar events to address sparsity. These clusters are mapped to spatial locations to achieve high spatial-temporal resolution.	[31, 32]

structure in the problem. It incorporates a principled algorithmic approach that partitions the spatial area under consideration into smaller areas, and then treats each sub-area as individual planning problems which are smaller than the original problem by construction, ensuring the scalabilty of the approach. To handle any situations that require coordination across regions, a top-level planner detects states where this "interaction" is necessary and finds actions such that the system's overall utility can be maximized. The approach is applied to ERM responder allocation in a simulated version of Nashville, and improves the response times to *all* incidents by up to **23 seconds** on average under normal conditions when compared to procedures used in practice. The adaptability of the approach to non-stationary conditions is also demonstrated by simulating simultaneous vehicle failures, and the hierarchical framework reduces average response times by **82 seconds** compared to existing methods when there are 3 simultaneous failures.

Both of these centralized and hierarchical approaches require communication between agents to coordinate within a sub-problem. Chapter 5 presents a decentralized planning framework for multi-agent SCPS that enables each agent to independently determine its own course of action, allowing the system to function with limited inter-agent communication. Techniques are presented to ensure that system-level constraints are satisfied. This framework is applied to the ERM resource allocation problem and is shown to be extremely scalable as the number of agents increases while improving the mean and variance of the the response time distribution compared to classical optimization approaches.

Chapter 6 presents an alternative approach to decision-making in non-stationary environments – a hybrid approach that combines a learning-based policy with online planning, called *Policy Augmented Monte Carlo tree search* (PA-MCTS). The intuition behind this approach is that if the environment has not changed too much between when an optimal policy was learned and when a decision needs to be made, the policy can still provide useful information for decision-making. Combining an *old* policy (i.e., a policy learned on an environment that has changed since) with a *current* online search (i.e., search using the current environmental parameters) results in a two-fold advantage. First, given a specific computational budget, the framework converges to significantly better decisions than standard MCTS. Second, the online search makes the approach significantly more robust to environmental changes than standard state-of-the-art learning-based approaches. To show the potential of PA-MCTS for non-stationary systems, it is applied to the classical inverted pendulum control problem under several environmental changes. PA-MCTS is found to converge to the optimal return with less than **10%** of the required computation compared to pure MCTS under several environmental changes, while still being able to converge when subjected to over **twice the magnitude** of change compared to a pure learning-based method.

The decision-making approaches and frameworks described above rely on a foundation of generative models to forecast how the environment will evolve. Chapter 7 presents a generative modeling approach that enables forecasting over large, heterogeneous geographic areas using mixed-typed features (i.e., categorical and numeric features) using the Similarity Based Agglomerative Clustering (SBAC) algorithm. The framework was applied to the incident forecasting problem in Nashville, and the predicted distribtuion of incidents was found to have a normalized root mean squared error of **1.656425** when compared to a validation set, showing that the predicted results match the sparse real world data.

After demonstrating these approaches for decision-making in SCPS using the ERM problem (as described in Chapter 2), Chapter 8 wraps up the dissertation by applying the framework to two other SCPS domains – electric fleet charging and dynamic vehicle routing – to demonstrate the generalizability of the decision support framework. When used to schedule electric bus charging in a simulation of Richland, WA's bus network, the proposed methods are shown to save over **$100k** per year in operating costs compared to a greedy strategy. When the framework is applied to dynamically route paratransit vehicles in a simulation of Chattanooga, TN, it saves **$145k** per year in energy costs to operate the system, and reduces CO_2 emissions by **576 metric tons** per year.

Finally, Chapter 9 concludes with a summary of the dissertation's accomplishments and proposes possible future work in decision-making for SCPS.

Chapter 2

Illustrative Domain: Emergency Resource Management

The challenges that arise when performing decision-making for SCPS are contextualized throughout this dissertation using the domain of Emergency Response Management (ERM), which is a challenge faced by communities across the globe. First responders need to respond to a variety of incidents such as fires, traffic accidents, and medical emergencies. They must respond quickly to incidents to minimize the risk to human life [33, 34]. Consequently, considerable attention in the last several decades has been devoted to studying emergency incidents and response. A category of incidents that is of particular concern for communities are Motor Vehicle Accidents (MVA) due to their frequency and the extent of their damage – Globally, about 3,200 people die every day from road accidents alone, leading to a total of 1.25 million deaths annually [35]. In fact, it is noted that MVAs are on the rise due to rapid urbanization and increasing traffic volume, and are set to be the fifth largest cause of death worldwide by 2030 without appropriate measures [36].

This dissertation uses the specific problem of responding to MVAs as an illustrative example, as there is a rich body of prior work on traffic accidents we can draw inferences from [37, 20, 38, 39, 40, 8, 41] . Focusing on one well defined SCPS problem allows us to discuss various technical approaches in greater detail, but the the core aspects of the problem (forecasting and decision making) are generalizable – other emergency response problems (e.g. wildfire management and medical incident response) and most SCPS applications in general must deal with the heterogenous, dynamic urban environments that make incident prediction difficult, and many require the allocation and dispatch of resources similar to ambulances. This is demonstrated in chapter 8, where the decision-support framework that is presented by this dissertation is applied to other SCPS problems: electric fleet charge scheduling and the dynamic vehicle routing problem.

2.1 The Emergency Response Problem

Traffic accident ERM is the problem of optimally responding to motor vehicle incidents in urban areas. Incidents are reported to central emergency response agencies, which have streamlined mechanisms for processing the request. For example, in the United States, emergency helpline calls are placed by dialing 911. People in need of assistance or other people who might have observed an incident can report it to the concerned authorities. Such a report is typically referred to as a "call" for emergency services. After being reported to 911, the call is then transferred to the agency concerned with traffic incidents (such as the fire department) by a computerized mechanism. The agency then uses its computer-aided dispatch (CAD) system to dispatch a responder to the scene. This set of events defines an ERM system, and it governs the pipeline of incident response, including detecting and reporting incidents, monitoring and controlling a fleet of response vehicles, and finally dispatching responders when incidents occur. In many cases there are multiple organizations governing this pipeline for an urban area; for example, ambulances and police cars might be dispatched from different departments.

Agents[1] that respond to traffic incidents include ambulances, police vehicles, and fire trucks (among others) and are referred to as *responders*. Responders are typically equipped with devices that facilitate communication to and from central control stations. In many cases, especially in the United States, responders are also equipped with computational devices like laptops as well. Once an incident is reported, responders are dispatched by a human agent to the scene of the incident (guided by some algorithmic approach like a CAD system). This process typically takes a few seconds,[2] but can be longer if dispatchers are busy. If no responder is available, the incident typically enters a waiting queue and is attended once a responder becomes free. Each responder is located in a specific *depot* (fire-station, for example), which are situated at various points in the spatial area under

[1] We use the term "agents" as is common in multi-agent systems community.

[2] This is based on our communication with fire departments in the United States [42]; time taken to dispatch responders presumably varies across the globe.

consideration. Once a responder has finished servicing an incident, it is directed back to the depot and becomes available to be re-dispatched by the dispatcher while en-route. An aspect that plays a key role in dispatch algorithms is that if there are any free responders available when an incident is reported, one must be dispatched to attend to the incident. This constraint is a direct consequence of the bounds within which emergency responders operate, as well as the critical nature of the incidents.

2.1.1 Problem Definition

To provide common context, we define a broad mathematical formulation for incident forecasting and planning problems. Given a spatial area of interest S the decision-maker observes a set of samples (possibly noisy) drawn from an incident arrival distribution. These samples are denoted by $\{(s_1,t_1,k_1,w_1),(s_2,t_2,k_2,w_2),\ldots,(s_n,t_n,k_n,w_n)\}$, where s_i, t_i and k_i denote the location, time of occurrence, and reported severity of the ith incident, respectively, and $w_i \in \mathscr{R}^m$ represents a vector of features associated with the incident. We refer to this tuple of vectors as D, which denotes the input data that the decision-maker has access to. The vector w can contain spatial, temporal, or spatio-temporal features and it captures covariates that potentially affect incident occurrence. For example, w can include features such as weather, traffic volume, and time of day [39, 43, 44, 45].

The most general form of an incident prediction model can be defined as the following:

Definition 2.1.1. *An incident forecasting model is defined as a function $f(X \mid w, \theta)$ where X is a random variable representing a measure of incident occurrence, w represents a set of feature vectors associated with each incident, and θ represents the models' parameters.*

The random variable X can represent various measures of incident occurrence, such as a *count* of incidents (the number of incidents in S during a specific time period) or *time* between successive incidents. The goal of a decision-maker is to find the *optimal* parameters θ^* that best describe D. This can be formulated as a maximum likelihood estimation (MLE) problem or an equivalent empirical risk minimization (ERM) problem.

Chapter 7 presents an approach to model the function $f(X \mid w, \theta)$ that enables forecasting over large, heterogeneous geographic areas using mixed-typed features such as precipitation, roadway type, time and date, and the proximity to an intersection.

Another important step in an emergency response pipeline is planning in anticipation of incidents. This involves stationing responders strategically and dispatching them as incidents occur. This process can be broadly defined as the following optimization problem:

Definition 2.1.2. *ERM planning can be represented by the optimization problem* $\max_y G(y \mid f)$*, where* y *represents the decision variable (which typically denotes the location of emergency responders in space),* G *is a reward function chosen by the decision-maker, and* f *is the model of incident occurrence.*

The specific reward used depends on what is being optimized; for example G might measure the total coverage (spatial spread) of the responders, or the expected response time to incidents. Therefore, given f, the decision-maker seeks to maximize the function G.

Chapter 3 presents a decision support framework to support online planning for ERM, and presents a centralized solution approach for the ambulance dispatching problem. Chapters 4 and 5 instead tackle the more complex dynamic ambulance allocation problem, which requires the more scalable hierarchical and decentralized planning approaches.

Chapter 3

A Decision-Theoretic Framework for SCPS Planning

3.1 Overview

A key challenge for decision making agents in SCPS are that the environments these systems operate in are often non-stationary. For example, consider Emergency Response Management (ERM) systems, which direct emergency vehicles such as ambulances to service incidents that occur throughout an urban area. The decisions made by an ERM agent depend on many factors of the environment, including traffic congestion, travel times, weather, and the likely locations of future incidents. In the long term, new infrastructure, demographic shifts, and climate change will shift these distributions steadily over time. In the short term, events such as sporting games, inclement weather, and road closures can create sudden, unexpected changes.

Popular learning-based decision-making approaches such as reinforcement learning (RL) learn a policy offline. Unfortunately, such policies can become stale as the environment shifts and lead to sub-optimal decisions. An alternative approach is online planning, which uses generative models of the environment to simulate into the future, evaluating possible action trajectories to determine the best action at the current time. Since this computation is done at decision time, as long as the generative models are kept up to date, these approaches can adapt to the environment as it change.

Implementing online planning for SCPS requires an integrated framework of components to learn generative models of the environment, update these models as the environment changes, and use the models to make decisions. This chapter presents a modular, integrated decision support framework that combines these components and applies them to the SCPS of Emergency Response Management (ERM) problem of ambulance dispatch to emergency incidents. It uses Monte-Carlo tree search (MCTS) to perform centralized planning. While this is applicable to the relatively small state-action space of vehicle dispatch, this centralized approach will not scale to more complex SCPS problem such as resource allocation. However, the decision support framework presented in this chapter forms the foundation for the scalable planning approaches discussed in later chapters.

The work comprising this chapter has been published in the Proceedings of the 10TH IEEE International Conference on Cyber-Physical Systems (ICCPS) [46].

- A. Mukhopadhyay[1], G. Pettet[1], C. Samal, A. Dubey, and Y. Vorobeychik (2019). "An Online Decision-Theoretic Pipeline for Responder Dispatch," in *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2019, Montreal, QC, Canada,* pp. 185-196.

3.2 Introduction

Emerging Trends and Challenges: Smart and connected communities are Human-in-the-Loop Cyber-Physical systems (H-CPS), with interactions between humans, the outside environment, and computational tools that assist in decision-making processes [47]. Analysis and optimization of H-CPS's is challenging primarily due to the inherent complexity and the sheer number of agents involved. Making accurate models is difficult, and simple rule based strategies often fail to capture the dynamics of the problem space.

Consider the classical problem of emergency response. The goal of responders is to minimize the variance in the operational delay between the time incidents are reported and when responders arrive on the scene. However, solving this problem requires not just sending the nearest emergency responder, but sometimes being proactive placing emergency vehicles in regions with higher incident likelihood. Sending the nearest available responder by euclidean distance ignores road networks and their congestion, as well as where the

[1]These authors have contributed equally

resources are stationed. Greedily assigning resources to incidents can lead to resources being pulled away from their stations, increasing response times if an incident occurs in the future in the area where responder should be positioned.

Data-driven approaches have been shown to produce more informed solutions to such problems [48] – examples include predicting crime and traffic accidents in urban areas [49, 50], and building architectures for smart city ecosystems [51]. In this paper, we leverage the potential of data-driven approaches and utilize real-world incident data for making informed decisions about effective stationing and dispatch of Emergency Medical Services (EMS) resources in a large urban community (Nashville, TN).

Contributions: We break down the problem of responder dispatch into three atomic sub-components: incident prediction, environment simulation, and the dispatching approach. Our contributions are as follows:

- Incident Prediction: Online Survival Analysis - We define a novel online approach to incident prediction that predicts incidents in time and space. Previous work in this domain has treated this as a batch learning problem [52, 50, 49], in which incident prediction models are learned once, and are subsequently used to aid response decisions. This fails to capture the changing dynamics of urban systems in which emergency responders operate, and we bridge this gap by creating an online incident prediction algorithm.

- Dispatch Algorithm - We formulate the problem of dispatching responders to incidents as a Semi Markov Decision Process (SMDP). Such an approach has recently been shown to work exceptionally well in this domain [53]. However, such systems have enormous computational load that limit their deployment in practice. We highlight this issue through the course of the paper and design an efficient solution that is fast, scalable and can work in a dynamic environment.

- Decision Theoretic Framework - We compose the Incident Prediction, Environment

Table 3.1: Notation Table

Symbol	Meaning
G	Set of equally sized grids
R	Set of Responders
t	Arrival-time between incidents
w	Features that affect incident arrival
f	A distribution over t, conditional on w
M_s	Responder Dispatch SMDP
M_d	Responder Dispatch Discrete-Time MDP (DTMDP)
h	Horizon of the Monte-Carlo Search Tree
D	Historical Dataset of incidents
$D^{'}$	Stream Dataset of incidents
L	Log-Likelihood of Incidents
Θ	A Generative Model of the Urban Area

Simulation, and Dispatching components into a framework that makes real time dispatching decisions based on traffic congestion and predicted incident distributions. Each component is modular, so improvements are easy to integrate into the framework.

Outline: We present and evaluate each of the components separately, as well as the entire system that combines them into an online pipeline and show that it results in better performance, and a remarkable decrease in computational run-time. We begin by presenting a high-level system model and problem description in Section 3.3, and present our solution in Section 3.4. We show our empirical evaluation in Section 3.5, go over a summary of prior work in the field in Section 3.6 and summarize the paper in Section 3.7. Table 3.1 describes the symbols used.

3.3 Problem Description

The problem deals with an urban area, in which incidents like traffic accidents, fires, distress calls and crimes happen in space and time. Such incidents are reported to a central emergency response system, which then dispatches responders like police vehicles and ambulances. This system governs the entire pipeline of incident response, including detecting and reporting incidents, monitoring and placing a fleet of response vehicles, and finally dispatching responders when incidents occur. Such responders are equipped with devices that facilitate communication to and from central control stations. They are then dispatched by a human (guided by some algorithmic approach), a process which typically takes seconds, but can be longer if dispatchers are busy [42].

For simplicity we discuss our approach with a single responder type and a single type of incident, but such *homogeneity* is not required for this approach.

Formally, we consider that the entire urban area is divided into as set of grids G. Incidents happen in these grids with an inter-arrival temporal distribution f, conditional on a set of features w. Such incidents need to be responded to by a set of responders R. Each responder is allocated to a specific *depot*, which are immobile stations located in a particular grid. Once a responder has finished servicing an incident, it is directed back to its depot and becomes available to be re-dispatched while in route. We also assume that if there are any free responders when an incident is reported, then some responder must be dispatched to attend to the incident. This is a direct consequence of the legal bounds within which emergency responders operate, as well as of the critical nature of the incidents. If an incident happens and there are no free responders available, then the incident enters a non-priority waiting queue and is attended to when responders become free.

Dispatch Systems in use today by major metropolitan areas such as Nashville work as follows: when an incident is reported, the system dispatches the closest available responder using the euclidean distance (i.e. ”as the crow flies”) between the incident and the responder’s current position [42]. This method has several disadvantages: 1) The euclidean distance between two locations is not necessarily representative of the actual time to travel between them since travel time depends on the road network and its current congestion. 2) By using the responder’s current location, it ignores where the responder should be stationed. Nashville‘s incident response data shows that this can lead to responders being pulled away from their depots, causing future incidents around those depots to have longer response times. 3) This method ignores the likely future incident distribution. Although dispatching the closest responder is greedily optimal, it may not be the best choice given the distribution of future incidents.

This motivates us to model responder dispatch formally. We begin by introducing the problem formulation of the online dispatch system in this section. We denote by $\tau \sim f$, a random variable that represents time between incidents in the urban area.

Given such a model of incident arrival, we now look at the model for responder dispatch. We formally model the problem of dynamic incident response as a semi-Markov decision process (SMDP) [54, 53], and refer to this process as M_s. An SMDP is described by the following tuple,

$$\{S, A, p_{ij}(a), t(i, j, a), \rho(i, a), \alpha\} \tag{3.1}$$

where S is a finite state space, A is the set of actions, $p_{ij}(a)$ is the probability with which the process transitions from state i to state j when action a is taken, $t(i, j, a)$ is a distribution over the time spent during the transition from state i to state j under action a, $\rho(i, a)$ is the reward received when action a is taken in state s_i, and α is the discount factor for future rewards.

States: At any point in time t, the state of the problem consists of the a tuple $\{I^t, R^t, E^t\}$, where I^t is a collection of grid indices that are waiting to be serviced, ordered according their times of occurrence. R^t represents a collection of vectors, where $r_i^t \in R^t$ captures all relevant information about the i^{th} responder such as it's current position and status. The state variable E^t captures relevant environmental factors that affect dynamic dispatch of responders at time t. Such factors are problem specific, so we leave the choice of such features to the designer of the responder system, but describe the specific features used in our system later.

Actions: Actions in our world correspond to directing responders either to incidents or back to their depots, when the process encounters a decision-making state. We denote the set of all actions by A and refer to a generic action by a. We use $A(s^i)$ to denote the set of actions that are available in state $s^i \in S$, and impose a constraint that whenever at least one responder is available and an incident occurs, we immediately dispatch *some responder*. Since the entire process evolves in continuous time, one can consider the existence of a single decision-making state at any instant, which leads to a single action being needed.

Transitions: The SMDP model evolves as a result of incidents that happen in space and time, and actions that are taken. The state transition probabilities are represented by the random variable $p_{ij}(a)$, which for any state s_i, represents the probability over the system transitioning to state s_j when action a is taken. Also, the time taken for the transition is represented by the random variable $t(i, j, a)$. We collectively refer to the state transition probabilities and time transition probabilities as *transition probabilities* throughout the rest of the paper.

Rewards: Since the broader goal of this problem is to minimize response times for emergency responders, we choose to look at *costs* instead of *rewards*. For each action a that is taken in state s_i, the system incurs a cost $\rho(s_i, a)$, and we seek to find actions for each state that minimizes the expected sum of costs.

Policy: A policy for a decision-making problem specifies an action for each state of the system. The goal of solving the SMDP is to find a policy that maximizes the sum of expected rewards that the decision process generates as a result of following the said policy. Our goal is to approximate the optimal policy π^* which, starting from for an arbitrary state s_i, minimizes the sum of expected discounted costs.

3.4 Our Solution

Before introducing the technical details of our solution approach, we provide a broad overview of the algorithmic approach we take and the associated technical challenges. We point out that two characteristics are fundamental to the operation of an emergency response system - first, it must be equipped with the ability to perform real-time computing in order to process the continuous stream of data received from responders and calls, and secondly, it must be equipped with principled algorithmic approaches to dispatch responders as and when incidents happen. We clarify that real-time computation essentially refers to a soft real-time problem - once incidents happen, the entire pipeline can afford a *short* latency to update existing models and calculate dispatch decisions. With these characteristics in mind, we start by looking at incident prediction algorithms. The canonical way to predict incidents in space and time is using historical data to learn a predictive model and then simulate incidents [55, 56]. However, since accidents often cascade, it is imperative that the model is updated as and when incidents happen. The primary technical challenge here is that re-training the entire model each time an incident happens (or periodically after some pre-defined number of incidents) is computationally slow and puts a heavy toll on the responder-dispatch framework that can only afford a low latency. This calls for the need to design an online mechanism to predict incidents that can be updated as incidents happen. We introduce such a model and explain the algorithmic details involved in section 3.4.1.

Having looked at the problem of incident prediction, we now look at dispatching responders given an incident prediction model. The SMDP formulation introduced in section 3.3 is difficult to solve since the state transition probabilities are unknown and cannot be computed in closed-form. One way to tackle this problem is to access a generative model to learn the state-transition probabilities while learning a policy. This has recently been shown to work well on the responder-dispatch problem [53]. However, we point out that such an approach has two major limitations. First, for any urban system, the state space is practically intractable even without environment variables. Even on fairly powerful computing systems, it would take weeks to train the policy [53]. The inclusion of environment variables would be computationally infeasible. Secondly, and partly as a consequence of the first issue, prior approaches are simply not suited for dynamic environments: if a single responder breaks down, traffic conditions change, or incident models evolve, existing approaches [53, 57] prescribe re-learning from scratch, which takes time that is incompatible with the latency constraints on the system. In order to alleviate this concern, we take

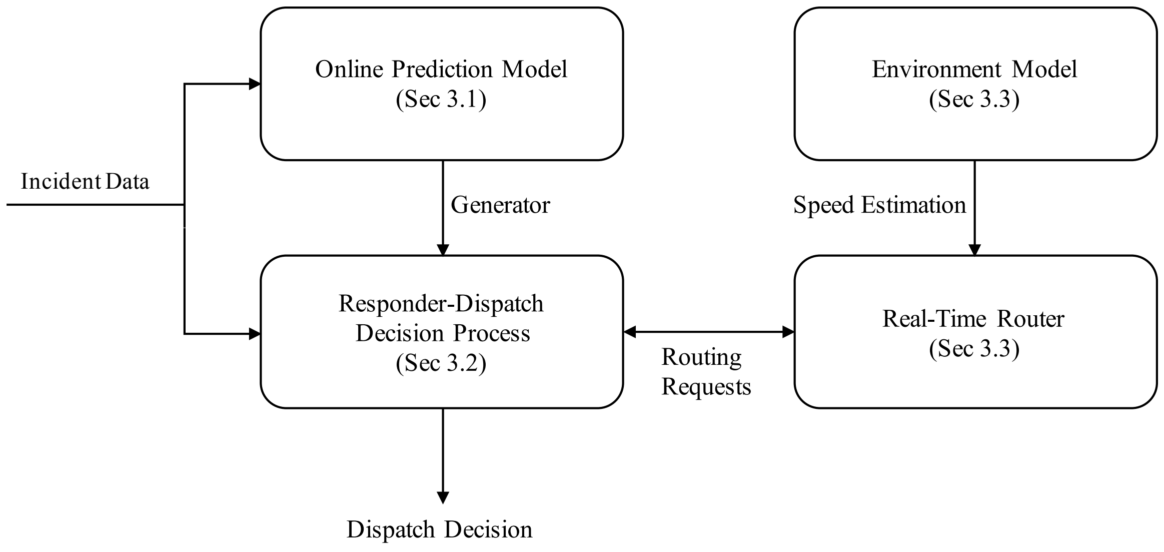

Figure 3.1: System Overview

the SMDP formulation and design an algorithmic approach that bypasses the need to learn the transition probabilities. This saves vital computation time and lets us design an online algorithm that is updated in real-time as the environment evolves (see Section 3.4.2).

In order to consider the effect of environmental factors on responder dispatch, it is essential to understand how such factors evolve. This motivates us to create predictive models for the environment. One factor of interest in the context of emergency responder-dispatch is traffic conditions in urban areas, since they directly affect travel times of responders. We take this into account by describing a model that enables us to learn the evolution of traffic in an urban area from prior data, and predict traffic conditions on the fly while attempting to solve the MDP (see section 3.4.3).

The high-level process flow that ties the three problems into one complete pipeline of responder dispatch is shown in figure 3.1. The online prediction model consumes actual incident data and at any point in time, provides a simulator that captures the latest trends in incident arrival. Also, the model to learn the evolution of environmental variables is used to find optimal vehicular routes between any two points in the urban area. These two atomic pieces are fed into the decision process for responder-dispatch, that given any state of the

SMDP, outputs a decision that governs which responder should be sent to respond to an incident.

3.4.1 Real Time Incident Prediction

We now present the technical details and formalize our methodology, and begin by looking at a principled incident prediction algorithm. Formally, we want to learn a probability distribution over incident arrival in space and time. In order to do so, we leverage our prior work in which we have shown how survival models prove to be extremely effective in predicting incidents like crimes and traffic accidents [55, 50, 53]. Survival Analysis is a class of methods used to analyze data comprising of time between incidents of interest [58]. Survival models can be parametric or non-parametric in nature, with parametric models assuming that *survival time* follows a known distribution. Based on our prior work, we choose a parametric model over incident arrival, and represent the survival model as $f(\tau|\gamma(w))$, where f is the probability distribution for a continuous random variable τ representing the inter-arrival time, which typically depends on covariates w via the function γ. The model parameters can be estimated by the principled procedure of Maximum Likelihood Estimation (MLE). The spatial granularity at which such models are learned can be specified exogenously - a system designer can choose to learn a separate f for each discretized spatial entity (grids in our case), learn one single model for all the grids or learn the spatial granularity from data itself. This choice is orthogonal to the approach described in this paper and we refer interested readers to our prior work [50] for a discussion about such models.

We shift our focus directly to survival models that are used to learn $f(\tau|\gamma(w))$. Intuitively, given an incident and a set of features, we want to predict when the next incident might happen. Before proceeding, we introduce an added piece of notation - we assume the availability of a dataset $D = \{(x_1,w_1),(x_2,w_2),..,(x_n,w_n)\}$, where x_i represents the time of occurrence of the i^{th} incident and w_i represents a vector of arbitrary features associated with the said incident. A realization of the random variable τ, used to measure the inter-arrival time between incidents, can be represented as $\tau_i = x_{i+1} - x_i$. The function γ is usually logarithmic and the relationship of the random variable τ with the covariates can be captured by a log-linear model. Formally, for a time-interval τ_i and associated feature vector w_i, this relationship is represented as

$$log(\tau_i) = \beta_1 w_{i1} + \beta_2 w_{i2} + \ldots + \beta_m w_{im} + y \tag{3.2}$$

where, $\beta \in \mathbb{R}^m$ represents the regression coefficients and y is the error term, distributed according to the distribution h. The specific distribution of f is decided by how the error y is modeled. We choose to model τ by an exponential distribution (for the sake of brevity, we refer interested readers to prior work [53] for more details on why an exponential distribution is particularly useful in such models). It turns out that when y follows the extreme value distribution, then τ is distributed exponentially. Thus, in our incident prediction model, we assume that h takes the following form

$$h_Y(y) = e^{y - e^y}$$

Using equation 3.2, for a given set of incidents, the log-likelihood of the observed data under the specific model can be expressed as

$$L = \sum_{i=1}^{n} \log h(\tau_i - w_i^T \beta) \tag{3.3}$$

The standard way to estimate the parameters of the model is to use a gradient-based iterative approach like the Newton-Raphson algorithm, yielding a set of coefficients β^* that maximize the likelihood expression. Now, the model over inter-arrival times is generative, it can be used to simulate chains of incidents, which is particularly helpful in building a simulator, the purpose of which we explain in the next section.

As pointed out before, such an approach is *offline*. However, it is imperative to capture the latest trends in incident arrival to accurately predict future incidents, which motivates us to design an online approach for learning and predicting incidents. We introduce some added notation before describing the algorithmic approach. First, we reiterate that β^* is used to refer to coefficients already learned from dataset D. Further, we assume that a new set of incidents $D^{'} = \{(x^{'}_1, w_1), (x^{'}_2, w_2), .., (x^{'}_k, w_k)\}$ is available that consists of incidents that have happened after (in time) the original set of incidents. We aim to update the regression coefficients β using $D^{'}$, assuming that the model already has access to β^*.

In order to address this problem, we use stochastic gradient descent to update the distribution f in an online fashion. Formally, we start with the known coefficients β^* and, at any iteration p of the process, we use the following update rule

$$\beta^{p+1} = \beta^p + \alpha \nabla L(\beta^p, D^{'})$$

where $\nabla(L(\beta^*, D^{'})$ is the gradient of the log-likelihood function calculated using $D^{'}$ at β^p and α is the standard step-size parameter for gradient based algorithms. Using equation 3.3, likelihood of the incidents in the dataset $D^{'}$ can be represented by

$$L = \sum_{i=1}^{k} \log e^{(\log \tau_i - \beta^* w) - e^{(\log \tau_i - \beta^* w)}}$$

and subsequently,

$$\frac{\partial L}{\partial \beta_j} = \sum_{i=1}^{k} -w_{ij} + w_{ij} \{ e^{(\log \tau_i - \beta^* w_i)} \}$$

The update step is repeated until improvements in the likelihood of the available data. Having already summarized the important steps in the algorithm in this section, we present it formally in Algorithm 1.

This mechanism enables us to update the incident prediction model in an online manner, saving vital computation time for the responder dispatch system. Also, this implicitly betters the dispatch algorithm by generating incident chains that capture the latest trend in

```
Algorithm 1: Streaming Survival Analysis
1 INPUT: Regression Coefficients $\beta^*$, Dataset $D^{'}$, Tolerance $\alpha$, Likelihood Function $L$,
    Maximum Iterations $MAX\_ITER$ ;
2 for $p = 1..MAX\_ITER$ do
3     $\beta^{p+1} = \beta^p + \alpha \nabla L(\beta^p, D^{'})$ ;
4     if $L(\beta^{p+1}, D^{'}) < L(\beta^p, D^{'})$ then
5         Return $\beta^p$;
6 Return $\beta^p$
```

incident occurrence.

3.4.2 Dispatch Algorithm

We begin the discussion on our dispatch algorithm by first explaining how the SMDP problem in formulation 3.1 can be solved by canonical policy iteration. A principled algorithmic approach [53] to solve the responder-dispatch SMDP is to first convert the SMDP to a discrete-time MDP M_d, which can be represented as

$$\{S, A, \bar{p}_{ij}, \rho, V_\beta, \beta_\alpha\}$$

where $\bar{p}_{ij}(a) = \beta_\alpha^{-1}\beta_\alpha(i,a,j)p_{ij}(a)$ is the scaled probability state transition function and β_α is the updated discount factor. The transformed MDP is equivalent to the original MDP according to the total rewards criterion [53, 54], and hence it suffices to learn a policy for M_d. Given such a conversion, the approach to solving the MDP involves accessing a simulator to learn the state transition probabilities for M_d [53]. The algorithm, *SimTrans*, an acronym for *Simulate and Transform*, uses canonical Policy Iteration on the transformed MDP M_d, with an added computation. It tracks the states and actions encountered by the simulator and gradually builds statistically confident estimates of the transition probabilities.

This process, however, is extremely slow and fails to work in dynamic environments since any change in the problem definition (the number of responders, or the position of a depot) renders the learned policy stale. In order to tackle this problem, we first highlight an important observation - one need not find an optimal action for each state as part of the solution approach since at any point in time, only one decision-making state might arise that requires an optimal action. This difference is crucial, as it lets us bypass the need to learn an optimal policy for the entire MDP. Instead, we describe a principled approach that evaluates different actions at a given state, and selects one that is sufficiently close to the optimal action. We do this using sparse sampling, which creates a sub-MDP around the neighborhood of the given state and then searches that neighborhood for an action. In order to actualize this, we use Monte-Carlo Tree Search (MCTS).

Another important observation is that the incident prediction model discussed in section 3.4.1 is generative and independent of dispatch decisions, which lets us simulate incidents independently. Note that since models of travel (discussed in section 3.4.3) as well as service times for responders can also be learned from data, the entire urban area can therefore be simulated. We denote such a simulator by Θ, which can *generate* samples of how the urban area evolves, even though the exact state-transition probabilities are unknown. This observation lets us simulate future states from a given state, leading to the creation of a state-action tree as shown in Fig. 3.2. We use this to design an algorithmic approach called *Real-Time SMDP Approximation*, and explain it next. Through the course of this discussion, we assume that the simulator can access a modular (and possibly exogenously specified) model to predict the environment at any point in time.

Algorithms 2, 3, 4, and 5 describe the various functions of our MCTS approach. We start our discussion with Algorithm 2, which is the highest level procedure that is invoked when presented with a decision-making state. First, b incident chains are sampled using the generative model Θ (refer to step 3 in Algorithm 2), where each chain is a time ordered list of sampled incidents. We create multiple chains in order to limit the impact of variance in the generative model. Next, the algorithm starts building the MCTS tree. We use the function $node(s, \eta, d, t)$ to refer to the creation of a node in the tree, which tracks the

Algorithm 2: Real-Time SMDP Approximation Main Procedure

```
INPUT: State $s$, Current Environment $E$, Horizon $h$, Stochastic Horizon $h^s$, Simulation Budget $b$, Generative Model $\Theta$ ;
Set current depth $d \leftarrow 0$;
$C \leftarrow b$ incident chains generated by $\Theta(E)$ ;
Set Scores $U \leftarrow \varnothing$ ;
$\bar{A}$ = SelectCandidateActions $(s, d, h^s)$ ;
foreach incident chain $c \in C$ do
    $u \leftarrow$ ChainEvaluation $(c, s, d, \bar{A}, h^s, h)$ ;
    foreach candidate action $a \in \bar{A}$ do
        $U[a] \leftarrow U[a] + u[a]$ ;
Return $\text{argmin}_{a \in \bar{A}}(U[a])$ ;
```

Algorithm 3: Select Candidate Actions for Given State

```
Function SelectCandidateActions (State s, Depth d, Stochastic Horizon $h^s$)
    $A^s \leftarrow$ set of available actions in state $s$ ;
    $a^* = \text{argmin}_{a \in A^s}(\rho(s,a))$ ;
    if depth $d \geq h^s$ then
        Return $a^*$ ;
    else
        $\bar{A}^s = \{a | a \in A^s \text{ and } \rho(s,a) \leq \varepsilon * \rho(s,a^*)\}$ ;
        Return $\bar{A}^s$ ;
```

current state of the system (s), the cost of the path from the root to the node (η), the depth of the tree (d) and the total time elapsed (t). Also, we use *UpdateState(s,a,c)* to retrieve the next state of the system, given the current state s, action a and chain c. For any state, we start by finding a set of candidate actions for the given incident (refer to step 5 in Algorithm 2), which takes the algorithmic flow to Algorithm 3. The candidate actions are chosen according to the current depth of the MCTS tree - if the tree is within the stochastic horizon h^s, the candidate actions include all actions with a cost that is at most ε times the cost of the myopically optimal action a^*. The parameter ε can be varied to control the trade off between the computational load of the algorithm and performance. Once the tree is deeper than h^s, the algorithm picks the best myopic action as a heuristic to construct the tree's nodes until depth h, since rewards are sufficiently discounted. After candidate

Algorithm 4: Evaluate a Chain of Incidents

```
1 Function ChainEvaluation (Incident Chain c, State s, Depth d, Candidate Actions $\bar{A}$,
    Stochastic Horizon $h^s$, Horizon h)
2     Set scores $u \leftarrow \varnothing$ ;
3     $d \leftarrow d+1$ ;
4     foreach action $a \in \bar{A}$ do
5         Next state $s' \leftarrow UpdateState(s,a,c)$ ;
6         Utility util $= s'$.responseTime ;
7         Root ← new Node(State = $s$‘, util = util) ;
8         Update $u[a] \leftarrow CreateStateTree(Root,c,d,h^s,h)$ ;
9     Return $u$
```

actions are found for the sampled incidents of the chain, Algorithm 4 is used to evaluate possible decision-making courses - each available action is tried and the MDP is simulated to generate future decision-making states, from which the entire process is repeated. This gradually builds a tree, where each edge is an action and each node is a decision-making state. We explain this procedure in Algorithm 5.

The key steps of the procedure are as follows. First, costs are tracked for every branch as the tree is built (refer to steps 10 and 14 in Algorithm 5), which is based on the response time in seconds for the assigned responder to the current incident. A lower cost is better, as it corresponds to lower response times. For any given node that was generated by action a from parent node p, the cost is

$$cost = u_p + (\gamma^t)((t - u_p)/(d+1)) \tag{3.4}$$

where u_p is the parent node‘s cost, γ is the discount factor for future decisions, and t is the time elapsed between taking action a at the parent node and the occurrence of the current node. This is essentially an updated weighted average of the response times to incidents given the dispatch actions.

Once the tree is completed, the cost for each candidate action for the dispatch incident is determined by the cost of the best leaf node in it's sub-tree, as this represents the result

Algorithm 5: Generate State Tree

```
1  Function CreateStateTree (Parent Node n, Incident Chain c, Depth d, Stochastic Horizon
      h^s, Horizon h)
2      if d ¿ horizon h then
3          Return n.util ;
4      else
5          $A = SelectCandidateActions(n.state, d, h^s)$ ;
6          $d \leftarrow d + 1$ ;
7          Let ChildUtils $\leftarrow \varnothing$ ;
8          foreach candidate action $a_i \in A$ do
9              Next state $s' \leftarrow UpdateState(n.state, a, c)$ ;
10             Let cost$_i \leftarrow UtilityUpdate(s', n.cost, d)$ ;
11             Let $x \leftarrow$ Node($s'$, cost$_i$, $d$, $t$) ;
12             ChildUtils $\leftarrow$ ChildUtils $\cup$ $CreateStateTree(x, c, d, h^s, h)$
13         Return min (ChildUtils)
14 Function UtilityUpdate (State s, Parent Utility $u_p$, Depth d, time t)
15     Return $u_p + (\gamma^t)(t - u_p)/(d + 1))$ ;
```

of the best sequence of future actions that could be taken given the dispatch action. Finally, the algorithm averages the costs for each dispatch action across the *b* generated incident chains, and selects the candidate action with the minimum overall cost as the best action in the current state (refer to step 10).

If all the responders are busy when an incident occurs, the incident is placed in a waiting queue. As soon as a responder becomes available, it is assigned to the incident at the front of the queue. This continues until the queue is emptied, after which the algorithm returns to using the heuristic policy above.

3.4.3 Predicting Environmental Factors

We now look at the final component of the proposed pipeline - in order to capture the effect of environment, we must learn how the environment evolves. We specifically focus our attention to traffic conditions, that directly affect the movement of responders. While information about current traffic conditions can be collected while making decisions, it does not

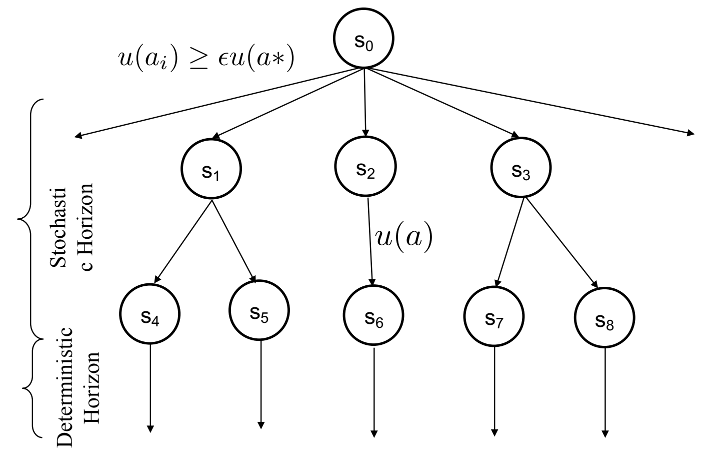

Figure 3.2: State-Action Tree

suffice for long-term planning. As the dispatch algorithm builds the state-action tree into the future, estimates of environmental variables are needed ahead of time, thereby making it imperative to learn predictive models for such variables. We therefore, design an algorithmic approach to predict future traffic conditions, and highlight how it can be used with an appropriate route-finding algorithm to predict travel times for emergency responders.

Traffic Prediction Model: We model the urban area as a set of road segments. For each segment, we assume that the dataset contains an associated set of features, which include data about the number of lanes, length of a segment, vehicular speed at different times and so on. Using this data set and features we learn a function over vehicular speed on a segment, conditional on the set of features using a Long Short-Term Memory Neural Network (LSTM) [59] model. The primary capability of such a framework is to model long-term dependencies and determine the optimal time lag for time series problems, which is especially desirable for traffic prediction in the transportation domain.

Route Finding Algorithm: Armed with a model that can predict vehicular speed on road segments, we now look for an approach to find the optimal route between two given points in the urban area. Specifically, given a source, destination and departure time, we seek to find the route with minimum *expected* travel time. To this end, we design a router

Table 3.2: Final Hyper-Parameter Choices

Number of Stations (Fraction of Nashville Count)	26 (full)	13 (1/2)	6 (1/4)	3 (1/8)
Simulation Budget b	10	10	10	10
Candidate Action Factor ε	1.5	1.5	2.5	1.5
Stochastic Horizon h^s	1	1	2	1
Discount Factor γ	0.9	0.9	0.99999	0.99999

based on A* search with landmarks (*ALT*) [60]. *ALT* improves upon euclidean-based A* search [61] by introducing landmarks to compute feasible potential functions using the triangle inequality, thereby improving the computational cost involved with such a procedure.

3.5 Performance

3.5.1 Data and Methodology

Our evaluation uses traffic accident data obtained from the fire and police departments of Nashville, TN, which has a population of approximately 700,000. We trained the generative survival model on 9345 incidents occurring between 1-1-2016 and 2-1-2017, and evaluated the algorithm on 1386 incidents occurring between 2-1-2017 and 4-1-2017. We gathered information about road segments and their geographical locations using real-time traffic data collected from HERE Traffic API [62] for Nashville area. The granularity of this dataset lets us access real-time vehicular speed for all segments in Nashville, which is sampled every minute throughout the day.

Caching the Router Results: While we recommend using a *router* in real-time using the exact locations of responders and incidents to make decisions, it is not feasible for our experiments. Our preliminary analysis showed that each router request takes approximately 0.2 seconds on average. In order to reduce the query time needed to find vehicular speed between arbitrary locations, we cached travel times between locations for different times of the day, with time discretized every 30 minutes. Our experiments showed that travel times in Nashville do not change significantly at this interval (ranging from 2-7 mph).In order to actualize caching, we used the same grid system described in section 3.3 for locations, with any location in the city discretized to the centroid of it's grid.

3.5.2 Experimental Setup

We begin by evaluating the streaming survival model separately by comparing it to a batch-learning approach [53]. Then, we evaluate the performance of the optimizers that are used for the router, and finally evaluate the dispatch algorithm. There are two considerations that need to be made during the evaluation -

1. **Decreased Responder Availability**: It is reasonable to assume that the base policy of dispatching the closest responder is correct most of the time and it is only rarely that non-greedy actions are needed. We hypothesize that such situations occur more frequently in practice as the strain on the system is greater: i.e. the incident to responder ratio is higher. This happens since responders attend not only to traffic accidents, but to a variety of other incidents (crimes for example). To take this into account, we ran several experiments with different number of responders: The full Nashville responder count of 26, and then cutting it by a factor of half three times to simulate test-beds with 13, 6, and 3 responders. The locations of the stations in each of these test-beds compared to Nashville's incident density heatmap are shown in figures 3.3, 3.4, 3.5, and 3.6.
2. **Hyper-Parameters**: We performed a hyper-parameter search (refer to the appendix for a concise summary of the hyper-parameters) for each of the test-beds based on the number of stations. The parameters that gave the best response time savings were chosen for each set, shown in table 3.2. We note that each hyper-parameter is important and strongly prescribe that each should be tested and tuned carefully for a new environment and hardware the system is deployed on, as these values may not be optimal for more constrained hardware, different responder distributions, or different cities with other incident arrival models.

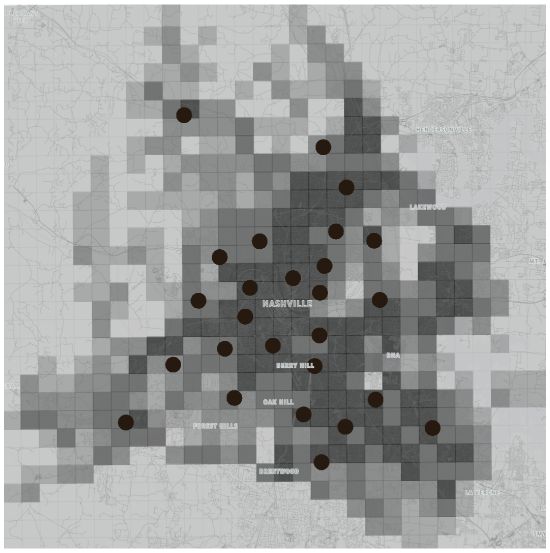

Figure 3.3: Distribution of the actual stations overlaid on an incident occurrence heatmap of Nashville, TN.

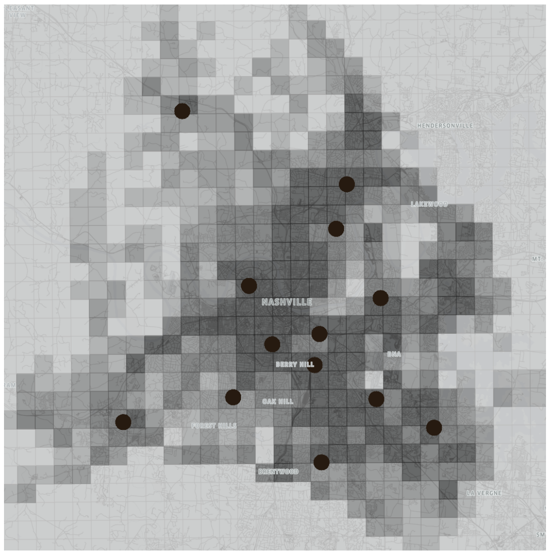

Figure 3.4: Distribution of the stations in the 13 station experiment overlaid on an incident occurrence heatmap of Nashville, TN.

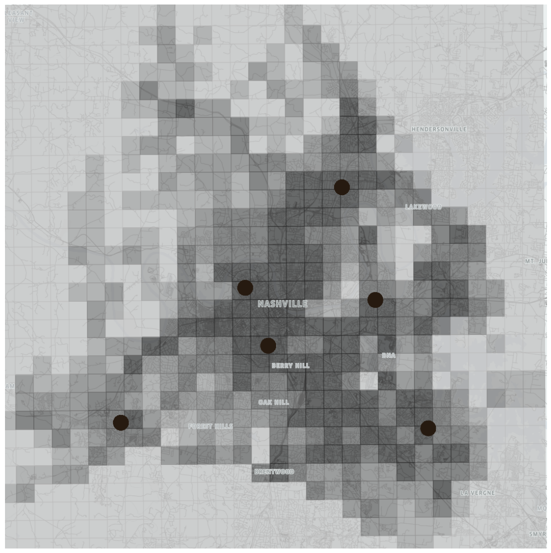

Figure 3.5: Distribution of the stations in the 6 station experiment overlaid on an incident occurrence heatmap of Nashville, TN.

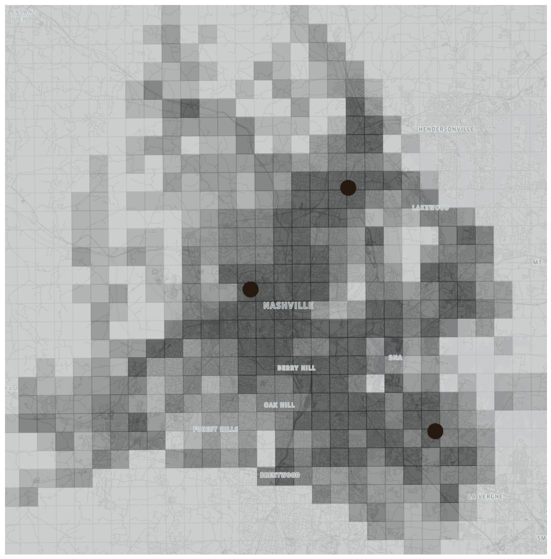

Figure 3.6: Distribution of the stations in the 3 station experiment overlaid on an incident occurrence heatmap of Nashville, TN.

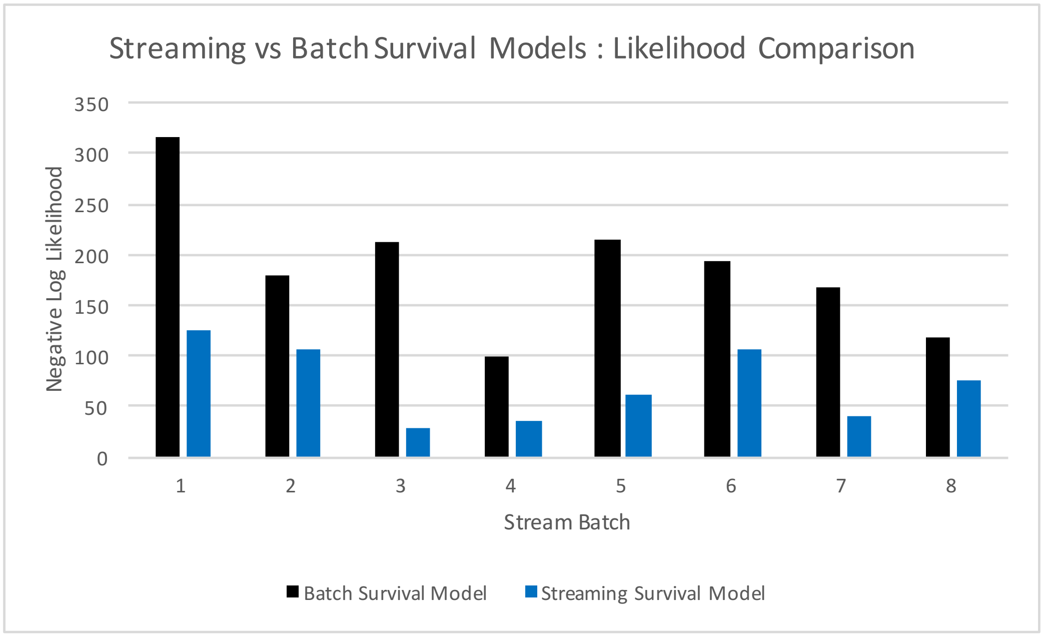

Figure 3.7: Negative Log Likelihood comparison (lower is better) between Batch and Streaming Survival Models. The Streaming model outperforms the batch model by a significant margin

3.5.3 Results and Discussion

3.5.3.1 Streaming Survival Analysis

We learned the batch model using the entire training data set and then, in the evaluation set, we considered each week as a *stream*, and further split 80% of the stream into a training set and 20% as the test set. We evaluated the batch model as well as the streaming model on the test set of each of the streams. Note that the batch model has access to all the data in the streams in the form of features; the stream model, on the other hand, gets updated after each data stream is received according to Algorithm 1. We use the *de-facto* standard of comparing likelihoods for evaluation, and present the results in in Figure 3.7. The streaming model results in a significant increase in likelihood (we plot the negative log likelihoods, hence lower is better) and convincingly outperforms the batch model. We point out a minor caveat - the updates can be used in practice only if the time taken to update

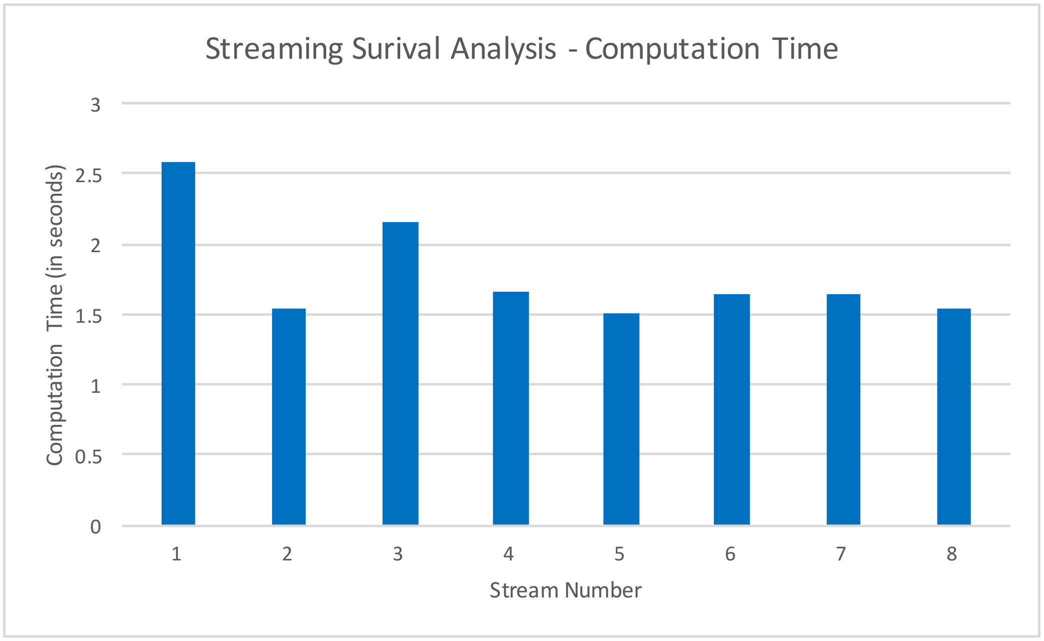

Figure 3.8: Computational Time for Streaming Survival Analysis.

the model is small as compared to the latency that emergency responders can afford. To illustrate this, we present the computational run-times of the stream model (for each stream) in Figure 3.8, and observe that it usually takes less than 2 seconds for an update to run, which justifies the usage of such models in practice.

In order to visually illustrate the benefit of a streaming model, we look at a fabricated example, where we feed the incident prediction models with data that is deviant from standard accident patterns. We show these results as heatmaps in Figure 3.9. Note that a brighter color corresponds to higher density of incidents. We see that the batch model *weakly* learns the pattern since it has access to the updated dataset only in the form of features; the streaming model on the other hand, identifies the current trend and predicts higher density of incidents in the concerned region, thereby highlighting the importance of such models in dynamic environments.

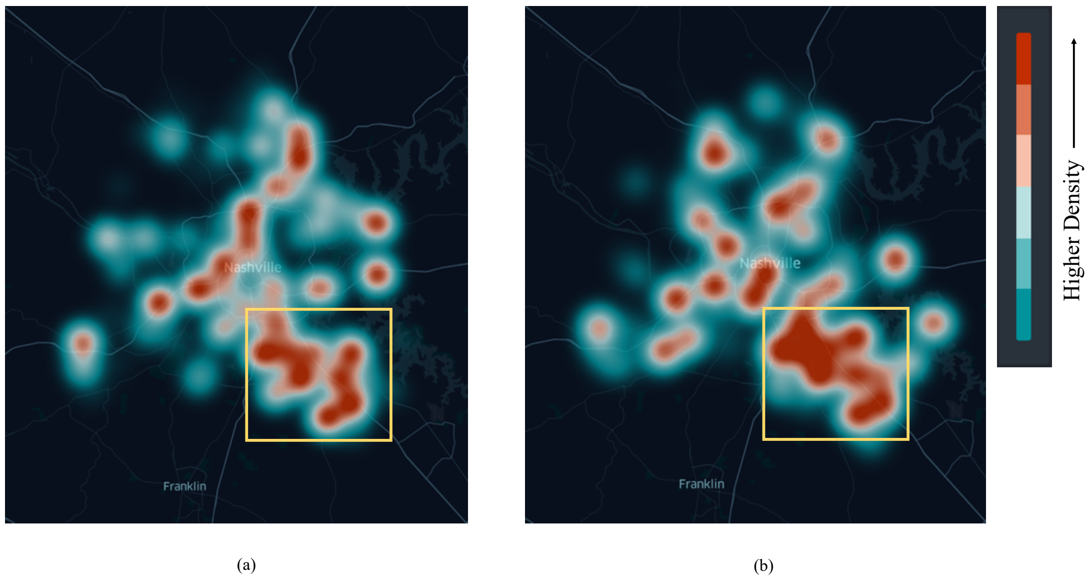

Figure 3.9: Batch Model (a) vs Streaming Model (b): These predicted heatmaps demonstrate that the streaming model adjusts more quickly to new incident distributions. Starting with a survival model learned from the training set, we fed the models synthetic incident data with incidents only occurring in the yellow boxed area, which means that the model should learn that there is now a higher incident likelihood in this area. The batch model picks up the new pattern weakly, whereas the streaming model shows higher likelihood in marked box.

3.5.3.2 Predicting Travel Times

We briefly show results of our traffic router before moving to the dispatch algorithm. We compared the LSTM using three different optimizers (Adam [63], SGD [64], and Adagrad [65]), and model performance was evaluated using five-fold shuffled cross validation. Adam, SGD and Adagrad showed Mean Absolute Errors of 5.47, 4.27 and 6.16 miles/hours respectively. Therefore, we chose SGD for our router described in Section 3.4.3. While evaluating the router on unseen data, the model with SGD optimizer showed MAE of only **6.419 miles/hour**.

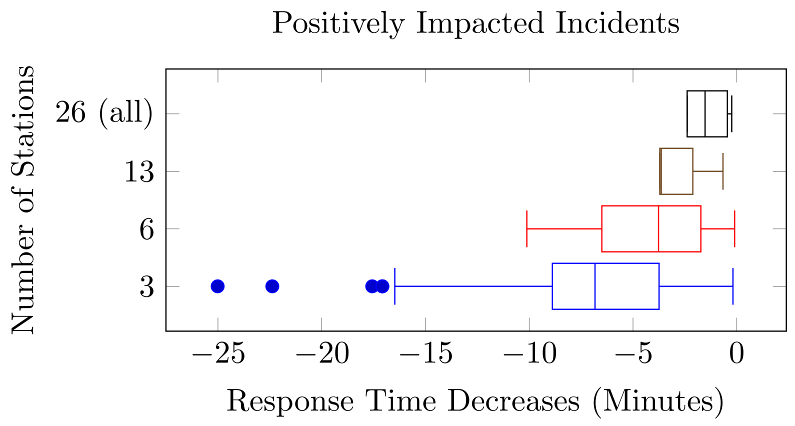

Figure 3.10: Incident response time difference between the base policy and our solution. This chart shows the distribution of response time decreases for positively impacted incidents. The distribution of time difference in minutes (x axis) is compared across each experiment involving the various station counts (y axis).

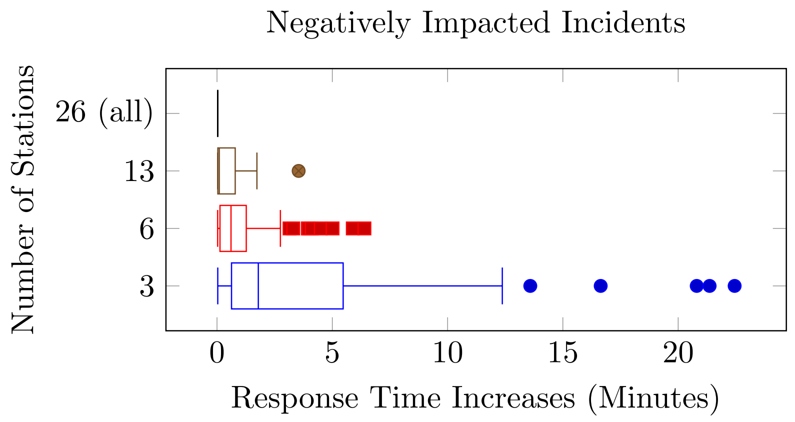

Figure 3.11: Incident response time difference between the base policy and our solution. This chart shows the distribution of response time increases for negatively impacted incidents. The distribution of time difference in minutes (x axis) is compared across each experiment involving the various station counts (y axis).

3.5.3.3 Responder Dispatch

In table 3.3 we present the results of comparing the tuned algorithms for each stations configuration. We compare our solution against the base policy (sending the nearest responder) using the average response time reduction for incidents impacted by the algorithm (i.e. incidents with different response times than the base policy), the number of incidents impacted, and the average computation time. The first observation is that the computation times are all well within acceptable limits, as they are near the human decision maker's visual reaction times [66]. This demonstrates that the system overcomes the technical challenge of running and updating in real-time, and can integrate into emergency response systems described in section 3.2.

We observe that when there is high responder coverage, demonstrated by the experiment with 26 stations, the baseline policy is nearly always used, with only 5 of the 1386 incidents serviced being impacted by the policy. But as the number of responders decreases, the baseline policy is sub-optimal for an increasing number of incidents, capping at 150 with 3 stations. This shows that the system can respond to changing responder availability, and that it is most useful when the system is strained by high incident demand.

Last, the average time saved for impacted incidents is significant, particularly for the experiments with 26 and 3 stations, as 30 seconds can be the difference between mortality and survival in response situations [67]. However, these represent average savings, and to dissect the performance of our approach, we plot the distributions of the response time savings for incidents that benefited from our solution, and response time increases for negatively impacted incidents in figures 3.10 and 3.11.

Comparing the box plots, the negatively impacted response times are more dense near zero compared to the savings. This shows that in general, the algorithm is not making large sacrifices for individual incidents in comparison to the savings generated, which is reinforced by the overall distribution of response times shown in figures 3.12 and 3.13. The response times for the positively impacted incidents are generally much improved; the

Table 3.3: Performance of System Compared to Base Policy

Number of Stations (Fraction of Nashville Count)	26 (full)	13 (1/2)	6 (1/4)	3 (1/8)
Average Response Time Savings for Incidents Impacted by Policy (seconds)	38.705	2.231	15.917	34.871
Number of Incidents Impacted by Policy	5	14	99	150
Average Computation Time per Incident (seconds)	0.384	0.198	0.350	0.0343

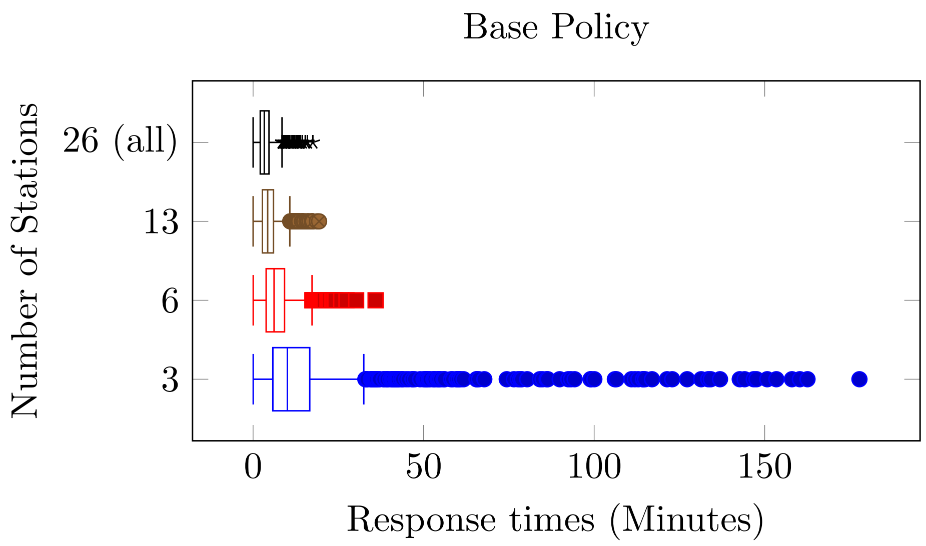

Figure 3.12: Response time distribution of the base policy. It is very similar to our solution on average since most incidents have the same responder, demonstrating that our solution is not worse than the base policy. The benefits of our solution are clear when looking at the response times for incidents with different dispatching decisions, as shown in figures 3.10 and 3.11

median improvement is over 200 seconds for the experiment with 13 stations, for example. Unfortunately, however, there are some outliers with unacceptably large sacrifices. For example, there is an incident in the experiment with 13 stations that took over 200 additional seconds to respond to compared to the base policy, which significantly increases the potential mortality of that incident if it is severe. This raises the question of integrating severity of incidents into the SMDP model, and we plan to consider the integration of prioritization of incidents in future work.

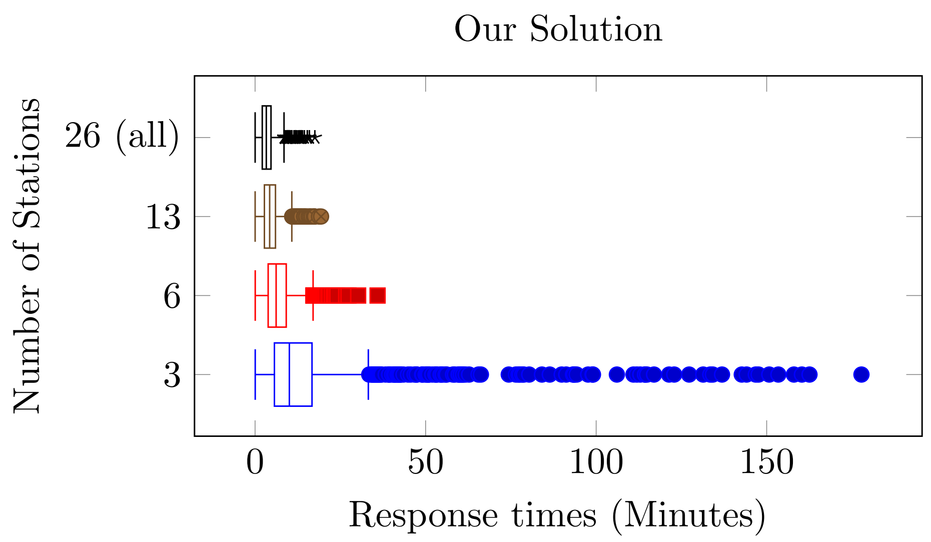

Figure 3.13: Response time distribution of our solution. It is very similar to the baseline policy on average since most incidents have the same responder, demonstrating that our solution is not worse than the base policy. The benefits of our solution are clear when looking at the response times for incidents with different dispatching decisions, as shown in figures 3.10 and 3.11

3.6 Related Work

A Traffic Incident Management Decision Support System is an information system that supports the process of preparing for, responding to, and managing the effects of traffic incidents. It must support functions such as stationing emergency response resources, dispatching to incidents in real time, and routing resources [68]. Most of these sub-problems have been studied in an orthogonal manner. We look at prior work for each of the sub-problems. First, we look at the problem of dispatching responders given a model of incident arrival. Traditionally, problem has been looked at as a part of the responder allocation problem [69, 50], in which an allocation of responders to depots naturally creates an algorithm for dispatch. The problem has also been studied as part of a joint optimization problem that balances distribution of resources and response times [70]. Finally, principled decision-theoretic models have also been used to study the problem [53, 57], that look at learning a policy of actions for all states the urban area can be in.

The second sub-problem is that of predicting incidents like traffic accidents, crimes, fires and others, that need emergency response. The availability of such a mechanism is crucial to the first sub-problem as decision-theoretic approaches can be aided by mechanisms that can simulate the world in which such responders operate. This specific problem has been widely studied in the past. One of the most widely studied types of incidents is crime, and a variety of approaches [71, 72] have been taken to tackle this problem. The problem of predicting traffic accidents has also received significant attention [73, 23]. Recently, freeway accidents have been predicted using panel data analysis approach that predicts incidents based on both time-varying and site-specific factors [19]. A survey of the literature on crash prediction models is presented in [74], which highlights the prevalence of Poisson distribution based models, and multiple linear regression approaches. There are also approaches that use clustering techniques to differentiate between incident types [75]. Finally, there are generic approaches that can work with multiple incident types [50, 49].

3.7 Conclusion

We designed a complete pipeline for the responder dispatch problem. We created an online incident prediction model that can consume streaming data and efficiently update existing incident arrival models. Then, we designed a framework for finding near-optimal decisions of an SMDP by using Monte-Carlo Tree Search, that bridges an important gap in literature by making such models computationally tractable. To aid the decision-making algorithm, we designed a Recurrent Neural Network architecture to learn and predict traffic conditions in urban areas. Our experiments showed significant improvements over prior work and existing strategies in both incident prediction and responder dispatch. We would like to highlight that while we treated incidents with equal severity, an interesting direction of future work involves designing the SMDP reward structure based on priorities, and directing responders based on incident prediction models that take severity into effect.

Chapter 4

Hierarchical SCPS Planning

4.1 Overview

Chapter 3 presents an online, planning-based decision support framework for SCPS. This enables a decision-maker to adapt to an environment as it evolves, since planning occurs at decision time using generative models that are continually updated using observations from the environment. It used a centralized planning approach (MCTS) – i.e., there is a monolithic state-action space that fully encapsulates the system and all its agents. The advantage of this approach is that it fully captures all possible agent interactions, allowing the decision-maker to converge to the optimal action trajectory in the limit.

However, such an approach has limited scalability, since all possible permutations of the state-action space must be considered at each decision epoch. Consider the problem of ERM resource allocation. In a problem with 20 resources and 30 possible waiting locations, there are $Permutations(30,20) = \frac{30!}{(30-20)!} = \frac{30!}{10!} = 7.31 \times 10^{25}$ possible assignments at each decision epoch that a centralized approach must consider. Instead, consider if the problem could be broken into smaller sub-problems. If the above example is split into five evenly sized sub-problems, each sub-problem will have 4 resources to assign to 6 depots. This gives a total of $P(6,4) = 360$ possible allocations in each sub-problem, which comes to a total of $360 \times 5 = 1800$ possible actions across all sub-problems. This example illustrates the potential of a "divide and conquor" approach by reducing the problem's complexity by approximately 22 orders of magnitude.

This chapter presents a hierarchical planning approach that leverages the spatial structure of SCPS environments to create subsets of the problem's state-action space. It incorporates a principled algorithmic approach that partitions the spatial area under consideration into smaller areas, and then treats each sub-area as individual planning problems which are smaller than the original problem by construction, ensuring the scalabilty of the approach. It then focuses on learning local policies, known as *macros*, over these sub-problems with a low-level planner using MCTS. To handle any situations that require coordination across regions, a top-level planner detects states where this “interaction” is necessary and finds actions such that the system’s overall utility can be maximized. The approach is applied to ERM responder allocation, and is demonstrated to scale significantly better than centralized planning approaches while being adaptive to non-stationary environments.

The work comprising this chapter has been published in Transactions on Cyber-Physical Systems (TCPS) [28] and the Proceedings of the 12th IEEE International Conference on Cyber-Physical Systems (ICCPS) [16].

- G. Pettet, A. Mukhopadhyay, M. Kochenderfer, and A. Dubey (2021). “Hierarchical Planning for Dynamic Resource Allocation in Smart and Connected Communities,” in *ACM Trans. Cyber-Phys. Syst.*

- G. Pettet, A. Mukhopadhyay, M. Kochenderfer, and A. Dubey (2021). “Hierarchical Planning for Resource Allocation in Emergency Response Systems,” in *Proceedings of the 12th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2021, Nashville, TN, USA.*

4.2 Introduction

Dynamic resource allocation (DRA) in anticipation of uncertain demand is a common problem in city-scale cyber-physical systems (CPS) [46]. In such a scenario, the decision-maker optimizes the spatial location of resources (typically called *agents*) to maximize utility over time while satisfying constraints specific to the problem domain. This optimization requires the design and development of procedures that can estimate how the system will evolve and evaluate the long-term value of actions. DRA manifests in many problems at the intersection of urban management, CPS, and multi-agent systems. These include emergency response management (ERM) for ambulances and fire trucks [76], allocating helper trucks [32], designing on-demand transit [77], and positioning electric scooters [78]. All such use cases typically deal with particular incidents of interest which correspond to specific calls for service. For example, road accidents require calls for emergency responders. In anticipation of such calls, planners proactively optimize the allocation of resources. The allocation problem can be modeled as a sequential decision-making problem under uncertainty, which can be solved to maximize a domain-specific utility function. For example, maximizing the total demand that can be serviced or minimizing the expected response time to incidents are commonly used objectives [76].

While we present a general approach for dynamic resource allocation in urban areas, we focus mainly on the problem of emergency response as a case study since it is a critical problem faced by communities across the globe. With road accidents accounting for 1.25 million deaths globally and 240 million emergency medical services (EMS) calls made in the USA each year, there is a critical need for a proactive and effective response to these emergencies [76, 79]. In addition to responding to frequent incidents each day, emergency response management (ERM) addresses large-scale disasters due to natural hazards (climate driven disasters caused more than $90 billion of losses in the US in 2018 [80]) and man made attacks. The lack of active and timely response to emergencies constitutes a threat to human lives and has resulted in mounting costs. Minimizing the time to respond to emergency incidents is critical in emergency response. Therefore, governments and private agencies often try to proactively optimize ambulance locations while considering constraints on the number of available ambulances and locations where they can be stationed. In addition, they address the problem of resource dispatch, in which agents[1] must be selected and asked to spatially move from their location to the location of the demand to address the task at hand. For example, ambulances must move to the scene of

[1]The actors of decision making are referred to as agents (e.g., ambulances)

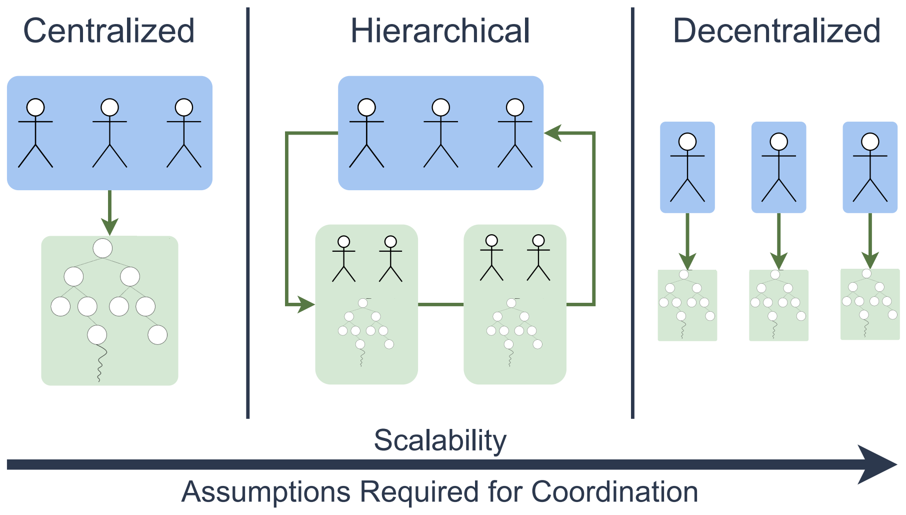

Figure 4.1: A spectrum of approaches for solving a dynamic resource allocation problem under uncertainty. A completely centralized approach uses a monolithic state representation. In a completely decentralized approach, each agent simulates what other agents do and performs its own action estimation. Our hierarchical approach segments the planning problem into sub-problems to improve scalability without agents estimating other agents' actions.

the incidents to provide assistance to patients. Effectively, ERM pipelines are an important example of human-in-the-loop CPS (H-CPS), which introduces problem specific structure and constraints.

State of the art: DRA and dispatch problems are typically modeled as Markov decision processes (MDP) [81, 53, 76]. The decision maker's goal is to find an optimal *policy*, which is a mapping between system states and actions to be taken. There is a broad spectrum of approaches available to address resource allocation under uncertainty as shown in figure 4.1. In our previous work, we applied the extremes of this continuum to ERM, with each approach having strengths and weaknesses [53, 46, 82].

The most direct approach is to model the state of the MDP as a monolithic entity that captures the entire system in consideration, as shown on the left in figure 4.1. When real world problems are modeled as MDPs, state transitions are typically unknown. The standard method to address this issue is to use a simulator (a model of the world) to estimate an empirical distribution over the state transitions [53], and use the estimates to learn a policy using dynamic programming [81]. Unfortunately, this offline approach does not scale to realistic problem settings [46]. Another method is to use an online algorithm like Monte Carlo tree search (MCTS) [46]. While adaptable to dynamic environments, this approach still suffers from poor scalability, taking too long to converge for realistic problem scenarios.

On the other side of the spectrum is an entirely decentralized methodology, as shown in the extreme right in figure 4.1. In such an approach, each agent determines its course of action. As the agents cooperate to achieve a single goal, they must estimate what other agents will do in the future as they optimize their actions. For example, Claes et al. [83] show how each agent can explore the efficacy of its actions locally by using MCTS. While such approaches are significantly more scalable than their centralized counterparts, they are sub-optimal as agents' estimates of other agents' actions can be inaccurate. Note that high fidelity models for estimating agents' actions limit scalability, and therefore decentralized approaches rely on computationally inexpensive heuristics. While decentralized approaches can be helpful in disaster scenarios where communication networks can break down, agents in urban areas (ambulances) typically have access to reliable networks, and communication is not a constraint. Therefore, approaches that ensure scalability but do not fully use available information during planning are not suitable for emergency response in urban areas, especially when fast and effective response is critical.

We explore hierarchical planning [17], which focuses on learning local policies, known as *macros*, over subsets of the state space by leveraging the spatial structure in the problem. Our idea is motivated by the concept of *jurisdictions* or *action-areas* used in public policy, which create different zones to partition and better manage infrastructure. In particular, we design a principled algorithmic approach that partitions the spatial area under consideration into smaller areas. We then treat resource allocation in each resulting sub-area (called regions) as individual planning problems which are smaller than the original problem by construction. While this ensures scalability, it hurts performance because agents constrained in one region might be needed in the other region. We show how hierarchical planning can be used to facilitate the transfer of agents across regions. A top-level planner, called the inter-region planner, identifies and detects states where "interaction" between regions is necessary and finds actions such that the system's overall utility can be maximized. A low-level planner, called the intra-region planner, addresses allocation and dispatch within a region. A challenge with hierarchical planning is that the high-level planner must estimate rewards further along the planning pipeline to make decisions. However, such rewards can only be computed *after* the low-level planner optimizes its objective function; this constraint defeats the purpose of hierarchical planning because it does not segregate the overall planning problem between two different levels. We leverage the structure of DRA and dispatch problems to address this challenge.

Contributions: the specific contributions of this article include:

1. We propose a hierarchical planning approach for resource allocation under uncertainty for smart and connected communities that scales significantly better than prior approaches.

2. We show how exogenous constraints in real world resource allocation problems can be used to naturally partition the overall decision problem into sub-problems. We create a low-level planner that focuses on finding optimal policies for the sub-problems.

3. We show how a high-level planner can facilitate the exchange of resources between the sub-problems (spatial areas in our case).

4. We propose two models, a data-driven surrogate model and a queue based approximation, that can estimate rewards to aid the high-level planner. In practice, responders can remain busy even after they leave the scene of the incident (e.g., ambulances transport victims of accidents to hospitals). Unlike prior work [46, 84], the data-driven approach we propose can accommodate such drop-off times.

5. We use real world emergency response data from Nashville, Tennessee, a major metropolitan area in the United States, to evaluate our approach and show that it performs better than state-of-the-art approaches both in terms of efficiency, scalability, and robustness to agent failures.

The paper is organized as follows. We present a general decision-theoretic formulation for spatial-temporal resource allocation under uncertainty in section 4.3. Section 4.4 describes the overall approach, the high-level, and the low-level planner. Section 4.5 details the implementation of our decision support system for ERM responder allocation. We present our experimental design in section 4.6 and the results in section 4.7. We present related work in section 4.8 and summarize the paper in section 4.9. Table 4.1 summarizes the notation used throughout the paper.

4.3 Problem Formulation

Resource allocation in anticipation of spatial-temporal requests is a common problem in urban areas. For example, ambulances respond to road accidents and emergency calls, helper trucks respond to vehicle failures, taxis provide transit service to customers, and fire trucks respond to urban fires. We refer to responders as *agents* to be consistent with the terminology used in multi-agent systems [85]. Once an incident is reported, agents are dispatched to the scene of the incident. The decision to select an agent to be dispatched can either be performed by a human expert (e.g., dispatching towing trucks), human-algorithm collaboration (e.g., dispatching ambulances), or completely by an algorithmic approach (e.g., dial-a-ride services). If no agent is available, the incident typically enters a waiting queue and is responded to when an agent becomes free. Each agent is typically housed at specific locations distributed throughout the spatial area under consideration (these could be fire stations or rented parking spots, for example). The number of such locations can vary by the type of incident and agent in consideration. For example, while taxis can wait at arbitrary locations or move around areas with high historical demand, ambulances are typically housed at rented parking lots or designated stations. We refer to such locations as *depots*. Once an agent finishes servicing an incident (which might involve transporting the victim of an accident to a nearby hospital), it becomes available for service again. If there are no pending incidents, it is directed back to a depot. Therefore, there are two broad actions that the decision-maker can optimize: (1) which agent to dispatch once an incident occurs (dispatching action) and (2) which depots to send the agents to in anticipation of future demand (allocation action). Next, we introduce our assumptions and the notation we use for problem formulation. While we present a formulation that is broadly applicable to resource allocation problems under uncertainty in urban areas, we use emergency response as a case study to provide concrete examples and use-cases.

We begin with several assumptions on the problem structure. First, we assume that we are given a spatial map broken up into a finite collection of equally sized cells G, a set of agents Λ that need to be allocated across these cells and dispatched to demand points, and a travel model that describes how the agents move throughout G. We also assume that we have access to a spatial-temporal model of demand over G, which is homogenous within each spatial cell. Our third assumption is that agent allocation is restricted to *depots* D, which are located in a fixed subset of cells. Each depot $d \in D$ has a fixed capacity $\mathscr{C}(d)$, which is the number of agents it can accommodate. While the state space in this resource allocation problem evolves in continuous-time, it is convenient to view the dynamics as a set of finite decision-making states that evolve in discrete time. For example, an agent moving through an area continuously changes the state of the *world* but presents no scope for decision-making unless an event occurs that needs a response or the planner redistributes the agents. As a result, the decision-maker only needs to find optimal actions for a subset of the state space that provides the opportunity to take actions.

A key component of response problems is that agents physically move to the site of the request, which makes temporal transitions between states non-memoryless. This component causes the underlying stochastic process governing the system's evolution to be semi-Markovian. The dynamics of a set of agents working to achieve a common goal can be modeled as a Multi-Agent Semi-Markov Decision Process (MSMDP) [86], which can be represented as the tuple $(S, \Lambda, \mathscr{A}, P, T, \rho(s,a), \alpha, \mathscr{T})$, where S is a finite state space, $\rho(s,a)$ represents the instantaneous reward for taking action a in state s, P is a state transition function, T is the temporal distribution over transitions between states, α is a discount factor, and Λ is a finite collection of agents where $\lambda_j \in \Lambda$ denotes the jth agent. The action space of the jth agent is represented by A_j, and $\mathscr{A} = \prod_{i=1}^{m} A_j$ represents the joint action space of all agents. We assume that the agents are cooperative and work to maximize the system's overall utility. $\mathscr{T}$ represents a termination scheme. Since agents take different actions with different times to completion, they may not all terminate at the same time [86]. We focus on asynchronous termination, where actions for a particular agent are chosen as and when the agent completes its last assigned action.[2]

States: A state at time t is represented by s^t and consists of a tuple $(I^t, \mathscr{Q}(s^t))$, where I^t is a collection of cell indices that are waiting to be serviced, ordered according to the relative times of incident occurrence. $\mathscr{Q}(s^t)$ corresponds to information about the set of agents at time t with $|\mathscr{Q}(s^t)| = |\Lambda_r|$. Each entry $q_j^t \in \mathscr{Q}(s^t)$ is a set $\{p_j^t, g_j^t, u_j^t, r_j^t, d_j^t\}$, where p_j^t is the position of agent λ_j, g_j^t is the destination cell that it is traveling to (which can be its current position), u_j^t is used to encode its current status (busy or available), r_j^t is the agent's assigned region, and d_j^t is its assigned depot, all observed at time t. A diagram of the state is shown in figure 4.4 and discussed in detail in section 4.5. We assume that no two events occur simultaneously in our system model. In such a case, since the system model evolves in continuous time, we can add an arbitrarily small time interval to create two separate states. We overload the notation for states for convenience; an arbitrary state is denoted by s when the time at which the state occurs is not required for discussion.

Actions: Actions correspond to directing agents to valid cells to respond to incidents (demand) or wait at a depot. For a specific agent $\lambda_i \in \Lambda_r$, valid actions for a specific

[2]Different termination schemes are discussed by Rohanimanesh and Mahadevan [86].

state s are denoted by $A^i(s)$ (some actions are naturally invalid, for example, if an agent is at cell k in the current state, any action not originating from cell k is unavailable to the agent). Actions can be broadly divided into two categories: *dispatching* actions which direct agents to service an active demand point, and *allocation* actions which assign agents to wait in particular depots in anticipation of future demand.

The manner in which agents are dispatched to the scene of the incidents varies with the type of incident. For example, an essential aspect of emergency response is that if any free agents are available when an incident is reported, then the nearest one must be greedily dispatched to attend to the incident. This constraint is a direct consequence of the bounds within which emergency responders operate and the critical nature of such incidents [76, 87, 82]. On the other hand, taxis can optimize dispatch based on long-term rewards. We focus on emergency response in our experiments and validate the approach using data collected from ambulances. As a result, the problem we consider focuses on proactively redistributing agents across a spatial area under future demand uncertainty. Nonetheless, dispatch actions are still necessary to model since they are the foundation of our reward function.

Transitions: The resource allocation system model evolves through several stochastic processes. First, incidents occur at different points in time and space governed by some arrival distribution. We assume that the number of incidents in a cell $r_j \in R$ per unit time can be approximated by a Poisson distribution with mean rate γ_j (per unit time), a commonly used model for spatial-temporal incident occurrence [76]. Second, agents travel from their locations to the scene of incidents governed by a model of travel times. We assume that agents then take time to service the incident at some exogenously specified service rate. Finally, the system itself takes time to plan and implement the allocation of agents. We refrain from discussing the mathematical model and expressions for the temporal transitions and the state transition probabilities since our algorithmic framework only needs a generative model of the world (in the form of a black-box simulator) and not explicit estimates of transitions themselves.

Rewards: Rewards in an SMDP can have two components: a lump sum immediate reward for taking actions and/or a continuous-time reward as the process evolves. The specific reward structure is highly domain-dependent. For example, taxi services might prioritize maximizing revenue, while paratransit services might focus on maximizing the total number of ride requests that can be served. The metric we are concerned with for ERM is *incident response time* t_r, which is the time between the system becoming aware of an incident and when the first agent arrives on the scene. This is modeled using only an immediate reward, which we denote by $\rho(s,a)$, for taking action a in state s:

$$\rho(s,a) = \begin{cases} \alpha^{t_h}(t_r(s,a)) & \text{if dispatching} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

where α is the discount factor for future rewards, t_h the time since the beginning of the planning horizon t_0, and $t_r(s,a)$ is the response time to the incident due to a dispatch action. The benefits of allocation actions are inferred from improved future dispatching.

Problem Definition: Given a state s and a set of agents Λ, our goal is to find an action recommendation set $\sigma = \{a_1, ..., a_m\}$ with $a_i \in A^i(s)$ that maximizes the expected reward. The ith entry in σ contains a *valid* action for the ith agent. In our ERM case study, this corresponds to finding an allocation of agents to depots that minimizes the expected response times to incidents.

4.4 Approach

Our approach to spatiotemporal resource allocation divides planning tasks into a two-stage hierarchy: "high-level" and "low-level" planning. First, high-level planning divides the overall decision-theoretic problem into smaller sub-problems. Our high-level planner accomplishes this by creating meaningful spatial clusters (which we call regions) and optimizing the distribution of agents among these regions. In order to do so, the high-level

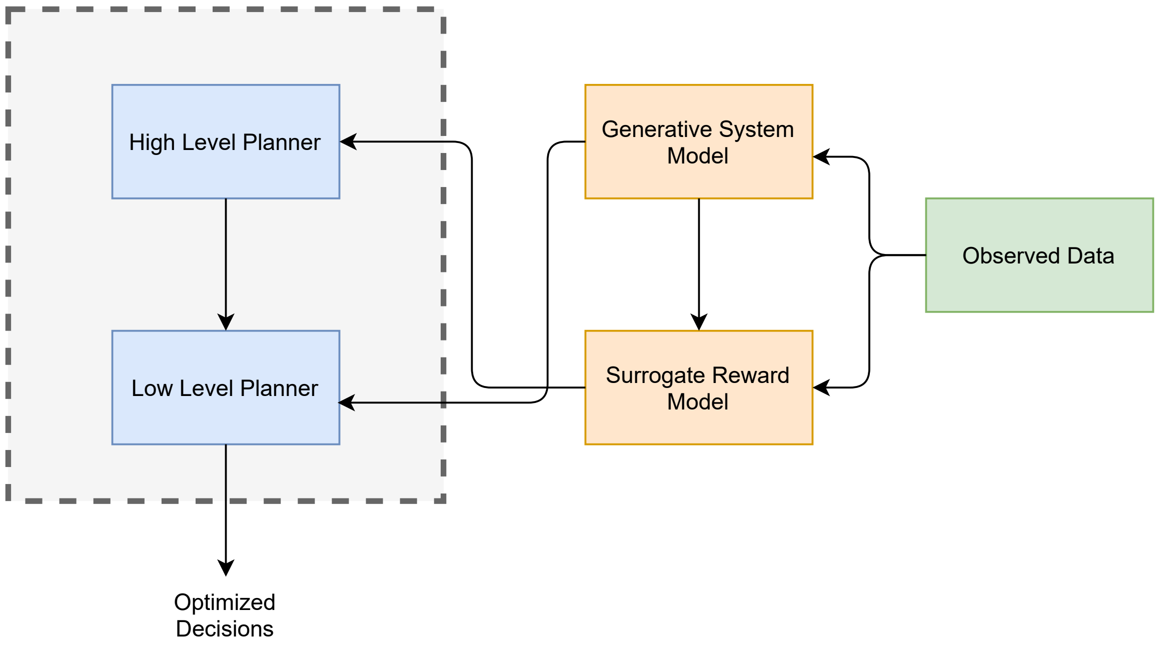

Figure 4.2: An overview of the proposed planning approach. We use the observed data to learn a generative model over when and where incidents occur. The generative model lets us simulate agent behavior, which aids the creation of a surrogate model for the high-level planner to segment the overall problem into a set of smaller problems. The low-level planner tackles each sub-problem independently. As we show, the surrogate reward model can also be estimated based on closed-form expressions of waiting times based on a queuing model.

planner must assess the quality of a specific distribution of agents, which requires the low-level planner itself. Indeed, the actual reward generated from an allocation depends on how the low-level planner optimizes the distribution of responders *within each region*. Clearly, this dependency is detrimental to creating an approach that seeks to divide the overall planning problem into two stages to achieve scalability. In order to tackle this challenge, we learn a surrogate model over waiting times that the high-level planner can use to estimate rewards. The high-level planner can perform inference using the model to estimate rewards as it optimizes the distribution of agents among the regions. Then, an instance of the low-level planner is instantiated for each of the regions. The low-level planner for a specific region optimizes the spatial locations of agents within that region. We assume the availability of an integrated simulation framework that models the dynamics of agents as they travel throughout the environment and a probabilistic generative model of incident occurrence learned from historical incident data. Our simulation framework and the incident model are described in section 4.5.

Segmenting allocation into smaller sub-problems significantly reduces complexity compared to a centralized problem structure. Consider a city with $|\Lambda| = 20$ agents and $|D| = 30$ depots, each of which can hold one agent. With a centralized approach, any agent can go to any depot, so there are $Permutations(|D|, |\Lambda|) = \frac{|D|!}{(|D|-|\Lambda|)!} = \frac{30!}{10!} = 7.31 \times 10^{25}$ possible assignments at each decision epoch. Now consider a hierarchical structure where the problem is split into 5 evenly sized sub-problems, $|\Lambda_h| = 4$ and $|D_h| = 6$ for each region. There are now $P(|D_h|, |\Lambda_h|) = 360$ possible allocations in each region, so there are $360 \times 5 = 1800$ possible actions across all regions. This reduction in complexity is about 22 orders of magnitude compared to the centralized problem, at the cost of abstracting the interactions between agents in different regions through the high-level planner. Alternatively, a decentralized approach in which each responder plans only its actions reduces the complexity further to only $|D| = 30$ possible allocations for each agent [82]. However, this reduction in complexity comes at a cost as each agent must make assumptions regarding other agent behavior, leading to sub-optimal planning. Hierarchical planning offers a balance between decentralized and centralized planning.

An important consideration when designing approaches for resource allocation under uncertainty in city-scale CPS problems is adapting to the dynamic environments in which such systems evolve. In our decision support system, a decision coordinator (an automated module) invokes the high-level planner at all states that allow the scope for making decisions. For example, consider that an agent is unavailable due to maintenance. The coordinator triggers the high-level planner and notifies it of the change. The high-level planner then checks if the spatial distribution of the agents can be optimized to best adapt to the situation at hand. We describe the exact optimization functions, metrics, and approaches that we use to design the planners below.

4.4.1 High-Level Planner

We seek to decompose the overall MSMDP into a set of tractable sub-problems through the high-level planner. A natural decomposition for spatiotemporal resource allocation is to divide the overall problem's spatial area into discrete regions and allocate separately for each region. This decomposition allows the low-level allocation planner to evaluate potential interactions between nearby agents and, therefore, likely to interact. The first goal of the high-level planner is to define these spatial regions. Consider that the high-level planner seeks to divide the overall problem in to m regions, denoted by the set $R = \{r_1, r_2, \ldots, r_m\}$, where $r_j \in R$ denotes the jth region. To achieve this goal, we use the locations of historical incident data. Intuitively, we want to create regions based on *hotspots* of incident occurrence [88]. Recall that our goal is to identify spatial areas where planning (the allocation and distribution of agents) can be performed independently. By identifying spatial incident clusters, we achieve the following: 1) areas close to each other with similar patterns of incident occurrence are grouped together, and 2) areas of high incident occurrence that are further apart get segregated from each other. While any standard spatial clustering approach can be used to achieve this goal, we use the well-known k-means algorithm in our analysis [89]. The k-means algorithm partitions a given set of points in $\mathbb{R}^d$ into k clusters such that each point belongs to its closest cluster (defined by distance to the center of the cluster). While the problem is known to be NP-hard even when $d = 2$ (our case) [90], heuristic-based iterative approaches can be used to achieve good solutions. Typically, the solution process repeatedly performs two steps after initializing an initial set of clusters. In the first step, each incident from the training set is assigned to the cluster whose center is the closest to the incident's location by Euclidean distance. Then, given an assignment, the centers of the clusters are computed again. The process is repeated until convergence. Clusters are then mapped to the cells G. Each cell in G is assigned to the cluster containing the most incidents that occurred within the cell. We use these clusters of cells in G as separate sub-problems for low-level planning.

The high-level planner's second task is to determine the distribution of agents across the decomposed spatial regions. If the regions are homogeneous, agents could be split evenly across them. In practice, however, regions will differ with respect to properties such as size, incident rate, and depot distributions, all of which impact the number of responders needed to cover each region. For example, the number of agents assigned to cover a dense downtown area should likely be different from sparsely populated suburbs. Recall that the overall goal of the parent MSMDP is to reduce the expected incident response times. Response times to emergency incidents consist of two parts: a) the time taken by an agent to travel to the scene of the incident, and b) the time taken to service the incident. Suppose we assume that incidents are homogeneous, meaning that the time taken by agents to service incidents follows the same distribution; in that case, the sole criterion that a planner needs to optimize is the travel time of the agents to incidents, which we refer to as *waiting times* (achieving zero waiting times is clearly infeasible in practice, so we seek to minimize waiting times). Therefore, the high-level planner seeks to distribute agents to different regions to minimize the expected incident waiting time. It is important to note that the real world is more complex; features such as incident severity can impact service times as well as the priority of responding to each incident. Incorprating such features and modeling service time variations in the high-level planner is a topic for future work.

We denote the expected waiting time for incidents in region $r_j \in R$ by $w_j(p_j, \gamma_j)$, where p_j is the number of responders assigned to the region r_j and γ_j is the total incident rate across r_j. Since the arrival process is assumed to be Poisson distributed, γ_j can be calculated as $\sum_{g_i \in G} \mathbb{1}(g_i \in r_j)\gamma_i$, where $\mathbb{1}(g_i \in r_j)$ denotes an indicator function that checks if cell g_i belongs to region $r_j \in R$. We consider two approaches to estimate $w_j(p_j, \gamma_j)$: a *queuing model* that approximates the system using an *m/m/c* queuing formulation, and a *surrogate model* that uses machine learning. We detail these models in sections 4.4.1.1 and 4.4.1.2, and then describe our high-level agent distribution algorithm, which uses an iterative greedy approach to assign agents across regions, in section 4.4.1.3.

4.4.1.1 Queuing Model

A multi-server queue model is one approach to model waiting times in a region. Recall that incident arrivals are distributed according to a Poisson distribution, thereby making inter-incident times exponentially distributed. We make the standard assumption that service times are exponentially distributed as well [50]. One potential issue with using well-known queuing models to estimate waiting times in emergency response is that travel times are not memoryless. In this approach, we use an approximation from prior work to tackle this problem [50]. Specifically, travel times to emergency incidents are typically much shorter than service times. Thus, the sum of travel times and service times can be considered to be approximated by a memoryless distribution (provided that the service time itself can be modeled by a memoryless distribution). The average waiting time for a region $r_j \in R$ can then be estimated by considering a $m/m/c$ queuing model (using Kendall's notation [91]), where $c = p_j$.

Let the average service time be T_s and let $\mu = 1/T_s$ denote the mean service rate. Then, the mean waiting time $w_j(p_j, \gamma_j)$ is [92]:

$$w_j(p_j, \gamma_j) = \frac{P_0(\frac{\gamma_j}{\mu})^{p_j}\gamma_j}{c!(1-\rho)^2 c}$$

where P_0 denotes the probability that there are 0 incidents waiting for service and can be represented as

$$P_0 = 1/\Big[\sum_{m=0}^{p_j-1} \frac{(p_j\rho)^m}{m!} + \frac{(c\rho)^c}{c!(1-\rho)}\Big]$$

While this approach is straightforward, computationally efficient, and works out of the box with any city of interest, it abstracts away many environmental dynamics that can affect response times. For example, the agents' allocation within a region, travel times due to traffic, and behaviors such as dropping off patients at the hospital can all affect the response times observed in the real world. To capture such factors, we explore an alternative method using machine learning to create a surrogate model over expected waiting times.

4.4.1.2 Surrogate Model

Another approach for estimating waiting times in a region is to use simulated data to learn a model of waiting times conditioned on a set of relevant covariates. For example, we simulate emergency response in a region under various conditions (namely, the number of agents assigned to the region and the incident rate in the region) and record the simulated incident response times. After generating many such samples, we use a supervised learning approach to fit a parameterized function to maximize the likelihood of the simulated data. The trained model can then be used to estimate waiting times for a region, given the number of responders. An advantage of this approach is that it captures subtle environmental dynamics that are ignored by the queue model, such as the travel times of agents as they respond to incidents and their location within the region due to factors such as depot locations and dropping off patients at hospitals. It is also adaptable to changes in the system model (e.g. extending the model to consider severity or heterogeneous service times), as these can be incorporated into the underlying simulation used to train the surrogate model.

The first step in creating the surrogate model is generating response time samples by simulating emergency response in each region with different incident rates γ_j and available agents. We sample incidents using the model described in section 4.5, which uses historical rates of incidents to create a sampling distribution. We refer to sampled incidents as *chains*. For each such chain, region r_j, and potential number of agents $p_j \in \{1, \dots, |D_j|\}$ (where D_j is the set of depots in region r_j), we simulate response to the incidents occurring within r_j using the simulation framework described in section 4.5. The simulation provides us with labeled training data where each observation captures the response time (output) given the number of responders and incident rate (inputs).

A crucial factor that affects simulated response times is the initial location of the agents within each region. Recall that in our proposed framework, the location of the agents is optimized by the low-level planner, which is naturally infeasible to use for every step during sampling. As a result, we solve the standard p-median problem [93], which is often applied to ambulance allocation, to determine the initial conditions of the simulation. The objective of the p-median formulation is to locate p_j agents (in the region r_j) such that the average demand-weighted distance between demand points and their nearest agent is minimized. Formally, we solve the following optimization problem to allocate the agents to depots:

$$\min \sum_{i=1}^{|G_j|} \sum_{k=1}^{|D_j|} a_i d_{ik} Y_{ik} \tag{4.2a}$$

$$\text{s.t.} \sum_{k=1}^{|D_j|} Y_{ik} = 1, \quad \forall i \in \{1, \dots, |G_j|\} \tag{4.2b}$$

$$\sum_{k=1}^{|D_j|} X_k = p \tag{4.2c}$$

$$Y_{ik} \leq X_k, \quad \forall i \in \{1, \dots, |G_j|\}, \forall k \in \{1, \dots, |D_j|\} \tag{4.2d}$$

$$X_k, Y_{ik} \in \{0, 1\}, \quad \forall i \in \{1, \dots, |G_j|\}, \forall j \in \{1, \dots, |D_j|\} \tag{4.2e}$$

where G_j is the set of cells in region r_j, D_j is the set of depots in r_j, p_j is the number of agents to be located in r_j, a_i is the likelihood of accident occurrence at cell $g_i \in G_j$, and d_{ik} is the distance between cell $g_i \in G_j$ and location $d_k \in D_j$. Y_{ik} and X_k are two sets of decision variables; $X_k = 1$ if an agent is located at $d_k \in D_j$ and 0 otherwise, and $Y_{ik} = 1$ if cell $g_i \in G_j$ is covered by an agent located at $d_k \in D_j$ (i.e. the agent at d_k is the nearest placed agent to g_i) and 0 otherwise.

The p-median problem is NP-hard [94]; therefore, heuristic methods are employed to find approximate solutions in practice. We use the Greedy-Add algorithm [95] to optimize the locations of agents. We show the algorithm in Algorithm 6. First, we initialize the iteration counter z and the set of allocated agent locations X_z to the empty set (step 1). Then, as long as there are responders awaiting allocation, we iterate through the following

Algorithm 6: Greedy-Add Algorithm

```
Input: Region cells $G_j$, region depots $D_j$, cell incident likelihoods
       $a_i \ \forall i \in \{1,\dots,|G_j|\}$, cell to depot distances $d(i,k) \ \forall i \in \{1,\dots,|G_j|\}$ and
       $k \in \{1,\dots,|D_j|\}$, number of agents $p_j$
Output: Agent depot assignments $X$
1  Initialize $z := 0$, $X_z := \emptyset$ ;
2  while $z < p_j$ do
3      $z := z+1$;
4      for location $d_{k'} \in D_j$,  where $k' \notin X_{z-1}$ do
5          $X'_z := X_{z-1} \cup d_{k'}$;
6          Find nearest facilities $y_i \ \forall i \in \{1,\dots,|G_j|\}$, where $y_i \in X'_z$;
7          Compute $U^z_{k'} := \sum_{g_i \in G_j} a_i d(g_i, y_i)$;
8      end
9      Best location $d^*_k := \text{argmin}_k \ U^z_k$;
10     $X_z := X_{z-1} \cup k^*$;
11     Return $X_z$
12 end
```

loop: (1) update counter z to the current iteration (step 3), (2) for each potential location not already in the allocation, compute the p-median score (equation 4.2a) of the allocation which includes the potential location (steps 5 - 7), and (3) find the location that minimizes the p-median score (step 9) and add it to the set of allocated agent locations (step 10). After locating the agents to depots within the region, we simulate emergency response using greedy dispatch and record the average response times $w(r_j, p_j, \lambda_j)$. The complexity of this algorithm is $O(p_j|D_j||G_j|)$ because assigning each agent requires evaluating each potential depot in D_j, which requires summing over the weighted distances between each cell in G_j and its nearest populated depot.

Given a set of samples of waiting times (w), agent allocations (p), and incident rates (γ), the second step in creating the surrogate model is to learn an estimator over w given p and γ. We learn a different model for each region to capture any latent features that can effect response times such as the region's depot distribution and roadway network. Specifically, we use random forest regression [96], which is an ensemble learning method based on constructing several decision trees at training time. During inference, the average

```
Algorithm 7: High-Level Planner
Input: Sorted regions $R_s$, arrival rates $\{\gamma_1, \gamma_2, \dots, \gamma_m\}$, service rate $\eta$
Output: Responder allocation $P = \{p_1, p_2, \dots, p_m\}$
1  assigned $:= 0, i := 0, J := \emptyset$;
2  while assigned $\leq |\Lambda|$ and $i \leq m$ do
3      $p_i := p_i + 1$;
4      assigned := assigned + 1;
5      if $\eta \times (p_i) \geq \sum_{g_i \in G} \not\Vdash (g_i \in r) \gamma_i$ then
6          $i := i + 1$;
7      end
8  end
9  while assigned $\leq |\Lambda|$ do
10     for $i \in \{1, \dots, m\}$ do
11         $J[i] := w_i(p_i, \gamma_i) - w_i(p_i + 1, \gamma_i)$;
12     end
13     $r^* := \arg\max_{i \in \{1, \dots, m\}} J[i]$;
14     $p_{r^*} := p_{r^*} + 1$;
15     assigned := assigned + 1;
16 end
```

prediction of the trained trees is used as the output (for regression problems).

4.4.1.3 Optimization

Given estimated waiting times for incidents in each region, the high-level planner seeks to minimize the cumulative response times across all regions. The optimization problem can be represented as:

$$\min_{p} \sum_{j=1}^{m} w_j(p_j) \tag{4.3a}$$

$$\text{s.t.} \sum_{i=1}^{m} p_i = |\Lambda| \tag{4.3b}$$

$$p_i \in \mathbb{Z}^{0+} \; \forall i \in \{1, \dots, m\} \tag{4.3c}$$

The objective function in mathematical program 4.3 is non-linear and non-convex. We use an iterative greedy approach shown in algorithm 7. We begin by sorting regions according to total arrival rates. Let this sorted list be R_s. Then, we assign agents iteratively to regions in order of decreasing arrival rates (step 3). After assigning each agent to a region $r_j \in R$, we compare the overall service rate (p_j times the mean service rate by one agent) and the incident arrival rate for the region (step 5). Essentially, we try to ensure that given a pre-specified service rate, the expected length of the queue is not arbitrarily large. Once a region is assigned enough responders to sustain the arrival of incidents, we move on to the next region in the sorted list R_s (step 6). Once all regions are assigned agents in this manner, we check if there are surplus agents (step 9). The surplus agents are assigned iteratively according to the incremental benefit of each assignment. Specifically, for each region, we calculate the marginal benefit J of adding one agent to the existing allocation (step 13). Then, we assign an agent to the region that gains the most (in terms of reducing waiting times) by the assignment.

The complexity of the algorithm is $O(|\Lambda| m\xi)$, where ξ is the complexity of the wait time estimation method, as the algorithm computes the potential wait times $w_j(r_j, p_j, \lambda_j)$ from adding an agent to each region when assigning said agent. The overall complexity is $O(|\Lambda|^2 m)$ using the queuing model, since equation P_0 sums over all assigned agents to estimate w_j. When using the random forest surrogate model, the overall complexity is $O(|\Lambda| m\varepsilon\beta)$, where ε is the number of trees in the forest and β is the maximum tree depth. An important note is that in our problem instances, we assume that there are enough agents to process the incident demand generated in G. The proposed greedy approach may leave some regions with no responders in environments where this assumption does not hold. Future work can examine if the proposed approach or an alternative (such as proportional assignment with respect to incident rate) performs better in such situations.

4.4.2 Low-Level Planner

The fine-grained allocation of agents to depots within each region is managed by the low-level planner, which induces a decision process for each region that is smaller than the original MSMDP problem described in section 4.3 by design. The MSMDP induced by each region $r_j \in R$ contains only state information and actions that are relevant to r_j, i.e., the depots within r_j, the agent's assigned to r_j by the inter-region planner, and incident demand generated within r_j. Decomposing the overall problem makes each region's MSMDP tractable using many approaches, such as dynamic programming, reinforcement learning (RL), and Monte Carlo Tree Search (MCTS). Each approach has advantages and trade-offs that must be examined to determine which is best suited for the specific problem domain under consideration.

Spatial-temporal resource allocation has a key property that informs the choice of the solution method—a highly dynamic environment that is difficult to model in closed-form. To illustrate, consider a travel model for the agents. While there are long-term trends for travel times, precise predictions are difficult due to the complex interactions between features such as traffic, weather, and events occurring in the city. Furthermore, a city's traffic distribution changes with changes in the road network and population shifts, so it needs to be updated periodically with new data. This dynamism is true for many pieces of the domain's environment, including the demand distribution of incidents. Importantly, it is also true of the system itself: agents can enter and leave the system due to mechanical issues or purchasing decisions, and depots can be closed or opened. Whenever underlying environmental models change, the solution approach must consider the updates to make correct recommendations. Approaches that require long training periods, such as reinforcement learning and value iteration, are challenging to apply to such scenarios since they must be re-trained each time the environment changes. This challenge motivates using Monte Carlo Tree Search (MCTS), a probabilistic search algorithm, as our solution approach. Being an anytime algorithm, MCTS can immediately incorporate any changes in the underlying generative environmental models when making decisions.

MCTS represents the planning problem as a "game tree", where nodes in the tree represent states. The decision-maker is given a state of the world and is tasked with finding a promising action. The current state is treated as the root node, and actions that are taken from one state to another are represented as edges between corresponding nodes. The core idea behind MCTS is that the tree can be explored asymmetrically, with the search being biased toward actions that appear promising. To estimate the value of an action at a state node, MCTS simulates a “playout” from that node to the end of the planning horizon using a computationally inexpensive *default policy* (our simulated system model is shown in figure 4.4 and described in detail in section 4.5). This policy is not required to be very accurate (indeed, a standard method is random action selection), but as the tree is explored and nodes are revisited, the estimates are re-evaluated and converge toward the actual value of the node. This asymmetric tree exploration allows MCTS to search large action spaces quickly. When implementing MCTS, there are a few domain specific decisions to make—the *tree policy* used to navigate the search tree and find promising nodes to expand, and the *default policy* used to quickly simulate playouts and estimate the value of a node.

Tree Policy: When navigating the search tree to determine which nodes to expand, we use the standard Upper Confidence bound for Trees (UCT) algorithm [97], which defines the score of a node n as

$$\mathrm{UCB}(n) = \overline{u}(n) + c\sqrt{\frac{\log(\mathrm{visits}(n))}{\mathrm{visits}(n')}} \tag{4.4}$$

where $\overline{u}(n)$ is the estimated utility of state at node n, visits(n) is the number of times n has been visited, and n' is node n's parent node. When deciding which node to explore in the tree, the child node with the maximum UCB score is chosen. The term $\overline{u}(n)$ corresponds to the exploitation objective that favors nodes that have produced higher rewards in the past. The second term on the right-hand side of the equation corresponds to the exploration objective, encouraging the exploration of nodes with low visit counts. The constant c controls the tradeoff between these two opposing objectives and is domain-dependent.

Default Policy: When working outside the MCTS tree to estimate the value of an action, i.e., rolling out a state, a fast heuristic *default policy* is used to estimate the score

Algorithm 8: Low-Level Planner

```
Input: Regions $R$, state $s$, generative demand model $E$, number of samples $n$
Output: Recommended allocation actions $\sigma_r \; \forall r \in R$
1  for region $r_j \in R$ do
2      Decompose $s$ into region specific state $s_j$;
3      Action score map $\widetilde{\mathscr{A}} := \emptyset$;
4      eventChains $:= E$.sample$(s_j, n)$;
5      Action scores $A :=$ MCTS$(s_j,$ eventChains$)$;
6      for action $a \in A$ do
7          $\widetilde{\mathscr{A}}[a]$.append(score$(a)$);
8      end
9      $\overline{\mathscr{A}} := \emptyset$;
10     for potential action $a \in A$ do
11         $\overline{\mathscr{A}}[a] = \text{mean}(\widetilde{\mathscr{A}}[a])$;
12     end
13     Recommended action $\sigma_r := \text{argmax}_a \; \overline{\mathscr{A}}[a]$
14 end
```

of a given action. Rather than using a random action selection policy, we exploit our prior knowledge that agents generally stay at their current depot unless significant shifts in incident distributions occur. Therefore, we use greedy dispatch without redistribution of responders as our heuristic default policy.

It is important to note that performing MCTS on one sampled chain of events is insufficient in practice, as traffic incidents are inherently sparse. Any particular sample is too noisy to determine the value of an action accurately. To handle this uncertainty, we use *root parallelization*. We sample many incident chains from the prediction model and instantiate separate MCTS trees to process each. We then average the scores across trees for each potential allocation action to determine the optimal one. Our low-level planning approach is shown in algorithm 8. The inputs for low-level planning are the regions R, the current overall system state s (which includes each agent's region assignment), a generative demand model E, and the number of chains to sample and average over for each region n. For each region $r_j \in R$, we first extract the state s_j in the region's MSMDP from the current overall system state s (step 2). Then we perform root parallelization by sampling n incident chains from the demand model E and performing MCTS on each to score each

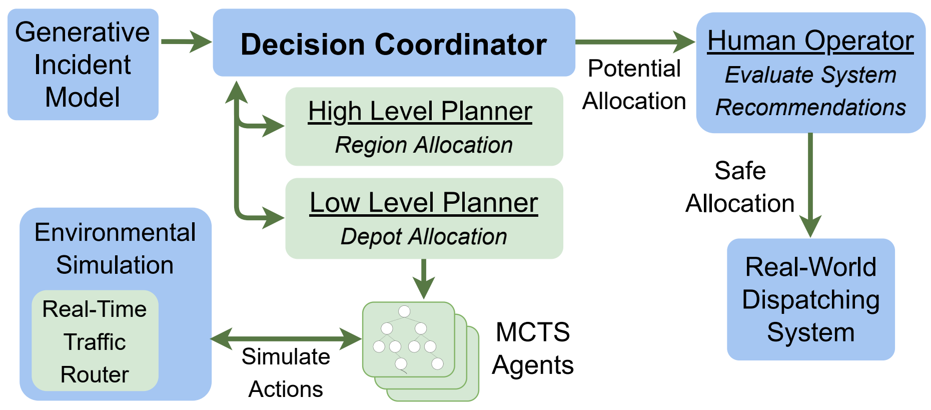

Figure 4.3: Emergency response decision support framework.

potential allocation action (step 4). It is important to note that the sampled incident chains are specific to the region under investigation, and demand is only generated from the cells in that region. We then average the scores across samples for each action and choose the allocation action with the maximum average score (step 13).

4.5 Integration Framework

Figure 4.3 shows a schematic representation of our decision support system. Realizing a such as system for online emergency responder allocation requires a framework of interconnected processes, including:

- A traffic routing model to support routing requests (section 4.5.1).
- A probabilistic generative model of incident occurrence (section 4.5.2).
- A model of the ERM system and its environment, including the dynamics of responders, depot locations, and hospital locations (section 4.5.3).
- A simulation of the ERM system built on the above components (section 4.5.4).
- A hierarchical decision process that makes allocation and dispatching recommendations based on the current state of the environment, responder locations, and projected incident distributions (section 4.4).

- A human operator to access the planning mechanism and act as an interfaced with a real-world computer-aided dispatch system.

4.5.1 Travel Model

We consider two travel time models in our implementation. The first model is based on the Euclidean distance between two points of interest (the centers of the cells in the grid). We also develop a more principled travel model that uses contraction hierarchies [98] and an open-source routing machine engine [99] to look up travel times at different times of the day from the center of each cell in the spatial grid to other cells. We collect such travel times across a week. The final travel model considers the median of the accumulated travel times and develops a lookup table which the overall planning process can use to query the travel times. This approach is better than a Euclidean distance-based travel time as it considers the average road congestion and the maximum travel speed across the road. The pipeline we propose and our framework are flexible to accommodate other travel time models; for example, a modular component that estimates travel times based on advanced graphical neural networks can be used with a richer set of covariates to provide sensitive estimates of time and weather.

4.5.2 Incident Prediction

Recall that we need samples of incidents for the low-level planner and learning the surrogate model for the high-level planner. We assume that incidents are generated by a Poisson distribution. A Poisson model has been widely used to model the occurrence of accidents [76]. The Poisson distribution is a discrete probability distribution over the number of events in a fixed time interval. We learn a separate Poisson model for each cell $g_i \in G$. The rate parameter of each model can be learned by maximizing the likelihood of historical data in the cell. The learned Poisson model can then be used to sample incidents in a given period of time.

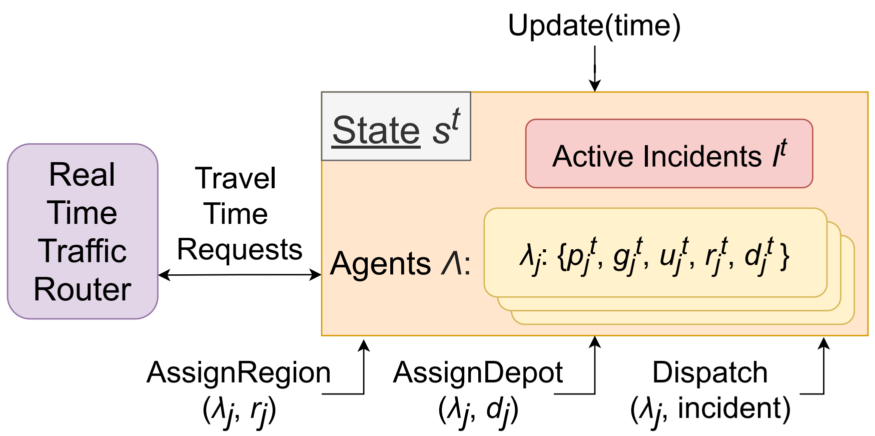
Update(time)
State s^t
Active Incidents I^t
Real Time Traffic Router
Travel Time Requests
Agents Λ:
λ_j: $\{p_j^t, g_j^t, u_j^t, r_j^t, d_j^t\}$
AssignRegion (λ_j, r_j)
AssignDepot (λ_j, d_j)
Dispatch (λ_j, incident)

Figure 4.4: System state and actions.

4.5.3 ERM System Model

As shown in figure 4.4, our system state at time t is captured by a queue of active incidents I^t and agent states Λ. I^t is the queue of incidents that have been reported but not yet serviced. The state of each agent $\lambda_j \in \Lambda$ consists of the agent's current location p^t_j, status u^t_j, destination g^t_j, assigned region r^t_j, and assigned depot d^t_j. Each agent can be in several different internal states (represented by u^t_j), including *waiting* (waiting at a depot), *in_transit* (moving to a new depot and not in emergency response mode), *responding* (the agent has been dispatched to an incident and is moving to its location), and *servicing* (the agent is currently servicing an incident). These states dictate how the agent is updated when moving the simulator forward in time.

4.5.4 Simulation Framework

Our simulator is designed as a discrete event simulator, meaning that the state is only updated at discrete time steps when certain events occur. These events include incident occurrence, re-allocation planning steps, and responders becoming available for dispatch. Between these events, the system evolves based on prescribed rules. Using a discrete event simulator saves valuable computation time compared to a continuous-time simulator. At each time step when the simulator is invoked, the system's state is updated to the current time. First, if the current event of interest is an incident occurrence, it is added to the active incidents queue I^t. Then, each agent's state and locations are updated accordingly. For example, agents that are in the *waiting* state stay at the same position, while agents that are *responding* or *in_transit* are updated according to the travel time model. If they reach their intended destinations, their states are updated to *servicing* or *waiting* at their respective locations. If such responders have not reached their destination at the time under consideration, their locations are updated using the travel model. If an agent is in the *servicing* state and finishes servicing an incident, its state is updated to the *in_transit* state, and its destination g_j^t is set to the assigned depot.

After the state is updated, a planner has several actuations available to control the system. The *Dispatch*$(\lambda_j, incident)$ function will dispatch the agent λ_j to the given incident which is in I^t. Assuming the responder is available, the system sets agent λ_j's destination g_j^t to the incident's location, and its status u_j^t is set to *responding*. The incident is also removed from queue I^t since it is being serviced, and the response time is returned to the planner for evaluation. The planner can also change the allocation of the agents. *AssignRegion*(λ_j, r_j) assigns agent λ_j to region r_j by updating λ_j's r_j^t. *AssignDepot*(λ_j, d_j) similarly assigns agent λ_j to depot d_j by updating λ_j's d_j^t and setting its destination g_j^t to the depots location. These functions allow a planner to try different allocations and simulate various dispatching decisions.

4.6 Experiments

We evaluate the proposed hierarchical framework's effectiveness on emergency response data obtained from Nashville, Tennessee, a major metropolitan area in the United States, with a population of approximately 700,000. We use historical incident data, depot locations, and operational data provided by the Nashville Fire Department [42]. We construct a grid representation of the city using 1×1 mile square cells; this choice resulted from the fact that local authorities follow a similar granularity of discretization. These

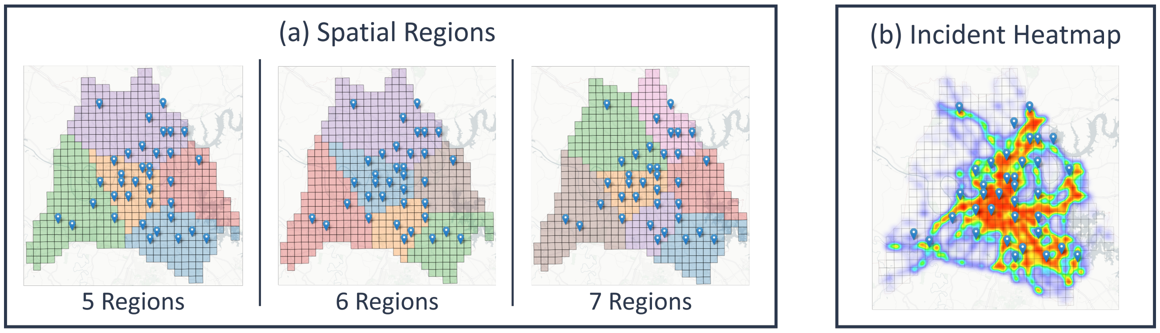

Figure 4.5: Subfigure (a) – The various spatial regions under consideration. Pins on the map represent depot locations, and different colors represent different spatial regions. Subfigure (b) – Nashville's historic incident density from January 2018 to May 2019 overlaid on the spatial grid environment.

cells, as well as the city's 35 depot locations, can be seen in figure 4.5.

Configuration and hyper-parameters: We make a few important assumptions when configuring our experiments. First, we limit the capacity of each depot $C(d)$ to 1. This constraint encourages responders to be geographically spread out to respond quickly to incidents occurring in any region of the city, and it models the usage of ad-hoc stations by responders, which are often temporary parking spots.[3] We assume there are 26 available responders to allocate, which is the actual number of responders in the urban area under consideration [42]. Third, we assume that the mean rate to service an incident is 20 minutes based on actual service times in practice in Nashville (we hold this constant in our experiments to compare the planning approaches directly). Fourth, as mentioned in section 4.4, we assume that incidents are homogeneous.

The number of MCTS iterations performed when evaluating potential actions on a sampled incident chain is set to 1000, and the number of samples from the incident model that are averaged together using root parallelization during each decision step is set to 50. We run the hierarchical planner after each test incident to re-allocate responders. Further, if the planner is not called after a pre-configured time interval, we call it to ensure that the allocations are not stale. In our experiments, this maximum time between allocations is set to 60 minutes. We ran experiments on an Intel i9-9980XE, with 38 logical processors

[3]In theory, we could always add dummy depots at the same location to extend our approach to a situation where more than one responder per depot is needed.

running at a base clock of 3.00 GHz and 64 GB RAM. Our experimental hyper-parameter choices are shown in table 4.2. In our experiments, we vary the number of spatial regions to examine how their size and distribution effects performance of the hierarchical planner; the resulting region configurations can be seen in figure 4.5.

Incident Model: Our incident model is learned from historical incident data. We learn a Poisson distribution over incident arrival for each cell based on historical data. The maximum likelihood estimate (MLE) of the rate of the Poisson distribution is simply the empirical mean of the number of incidents in each unit of time. To simulate our system model, we access the Poisson distribution of each cell and sample incidents from it. In reality, emergency incidents might not be independently and identically distributed; however, the incident arrival model (and the black box simulator of the system in general) is entirely exogenous to our model and does not affect the functioning of our approach. To validate the robustness of our approach, we create three separate testbeds based on domain knowledge and preliminary data analysis of historical incident data.

Region Segmentation: We use the k-means algorithm [100] implemented in scikit-learn [101] on historical incident data provided by the Tennessee Department of Transportation, which consists of 47862 incidents that occurred from January 2018 to May 2019 in Nashville. We vary the parameter k to divide the total area in consideration into 5, 6, and 7 regions. The cluster centers are initialized uniformly at random from the observed data, and we use the classical expectation-minimization-based iterative approach to compute the final clusters [101].

Surrogate Model: To learn the surrogate model over waiting times conditional on the number of responders in a region and mean incident arrival rate, we use the random forest regression model [96]. We use the mean squared error (MSE) to measure the quality of a node split, use 150 estimators, and consider $\sqrt{|n|}$ random features for each split, where n is the number of features. The following hyper-parameters were tuned using a grid search: the maximum depth of each tree, the minimum number of observations in a node required to split it, and the minimum number of samples required to be at a leaf node to split its parent.

Table 4.2: Experimental hyper-parameter choices.

Parameter	**Value(s)**
Number of Regions	{5, 6, 7}
Maximum Time Between Re-Allocations	60 Minutes
Incident Service Time	20 Minutes
Responder Speed	30 Mph
MCTS Iteration Limit	1000
Discount Factor	0.99995
UCT Tradeoff Parameter c	1.44
Number of Generated Incident Samples	50

Stationary incident rates: We start with a scenario where our forecasting model samples incidents from a Poisson distribution that is stationary (for each cell), meaning that the rate of incident occurrence for each cell is the empirical mean of historical incident occurrence per unit time in the cell. This means that the only utility of the high-level planner in such a case is to divide the overall spatial area into regions and optimize the initial distribution of responders among them. Since the rates are stationary, the initial allocation is maintained throughout the test period under consideration. This scenario lets us test the proposed low-level planning approach in isolation. The experiments were performed on five chains of incidents sampled from the stationary distributions, which have incident counts of $\{939, 937, 974, 1003, 955\}$ respectively (for a total of 4808 incidents), and are combined to reduce noise.

Non-stationary incident rates: We test how our model reacts to changes in incident rates. We identify different types of scenarios that cause the dynamics of spatial temporal incident occurrence and traffic to change in specific areas of Nashville. We look at rushhour traffic on weekdays (which affects the center of the county), football game days (which affects the area around the football stadium, typically on Saturdays), and Friday evenings (which affects the downtown area). Then, we synthetically simulate spikes in incident rates in the specific areas at times when the areas are expected to see spikes. To further test whether our approach can deal with sudden spikes, we randomly sample the spikes from a Poisson distribution with a rate that varies between two to five times the historical rates of the regions. We create five different trajectories of incidents with varying incident rates, which have incident counts of $\{873, 932, 865, 862, 883\}$ respectively (for a total of 4415 incidents). In these experiments we compare using the low-level planner with fixed responder distributions across regions to a full deployment that incorporates the high-level planner to dynamically balance responders across regions.

Responder failures: An important consideration in emergency response is to quickly account for situations where some responders might be unavailable due to maintenance and breakdowns. We randomly simulate failures of responders lasting 8 hours to understand how our approach deals with such scenarios.

Dispatch Policy: As discussed in section 4.3, the critical nature of incidents necessitates the use of a *greedy dispatch policy*—if there are any free agents available when an incident is reported, the nearest agent (regardless of region assignment) is dispatched to the incident. If no agents are available, then incidents enter a waiting queue. Once an agent becomes available, if there are any incidents waiting in the queue, it is dispatched to the incident at the front of the queue and that incident is removed from the queue.

Baseline Policy: We compare our approach with a baseline policy that has no responder re-allocation. This baseline emulates current policies in use by cities in which responders are statically assigned to depots and rarely move. The initial responder placement is determined using our proposed high-level policy to ensure all the policies begin with similar responder distributions. The baseline uses the same greedy dispatch policy as our approach.

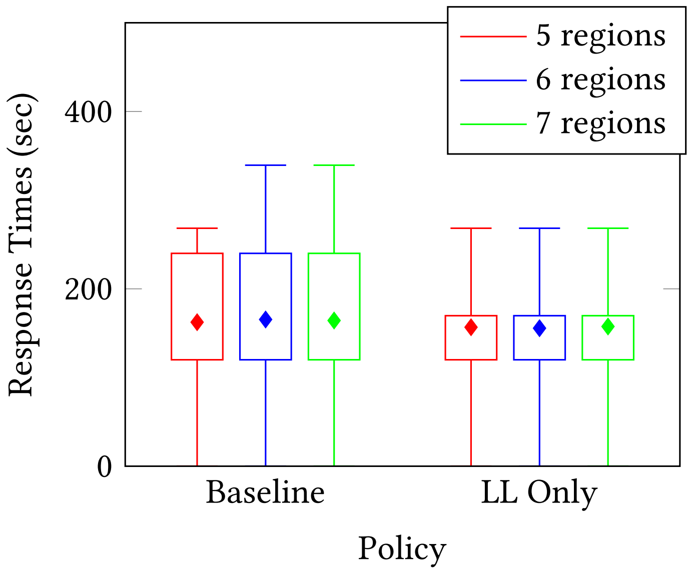

Figure 4.6: Results when applying the baseline and low-level planners to incidents sampled from a stationary rate distribution. This figure presents the full response time distributions; the boxplot represents the data's Inter-Quartile Range (IQR $= Q_3 - Q_1$), and the whiskers extend to the 9th and 91st percentiles.

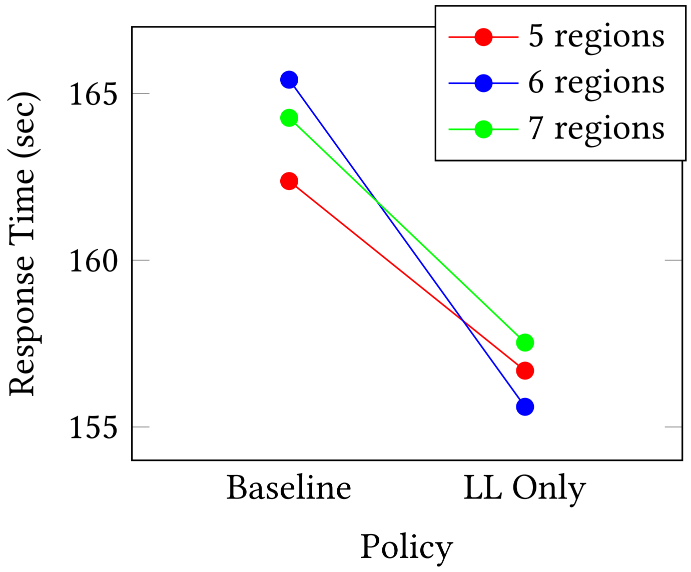

Figure 4.7: Results when applying the baseline and low-level planners to incidents sampled from a stationary rate distribution. This figure presents a zoomed in view of the average response times.

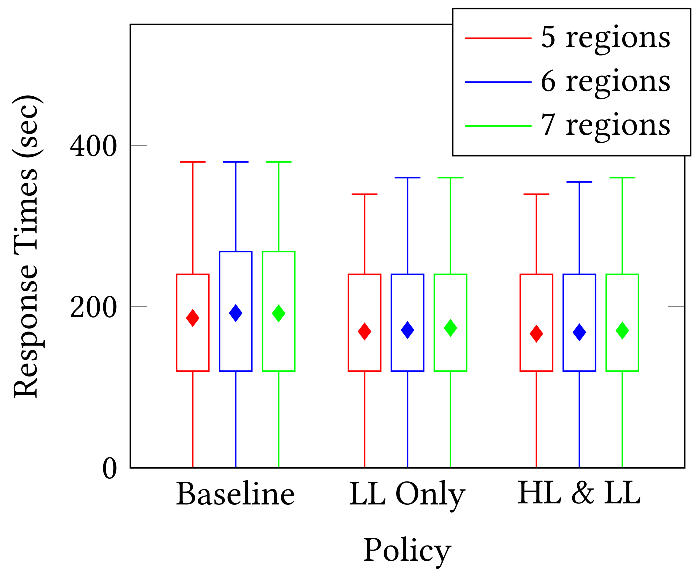

Figure 4.8: Results when applying the baseline, low-level planner (LL Only), and complete hierarchical planner (HL & LL) when applied to incidents sampled from a non-stationary rate distribution. This figure presents the full response time distributions; the boxplot represents the data's Inter-Quartile Range ($\text{IQR} = Q_3 - Q_1$), and the whiskers extend to the 9th and 91st percentiles.

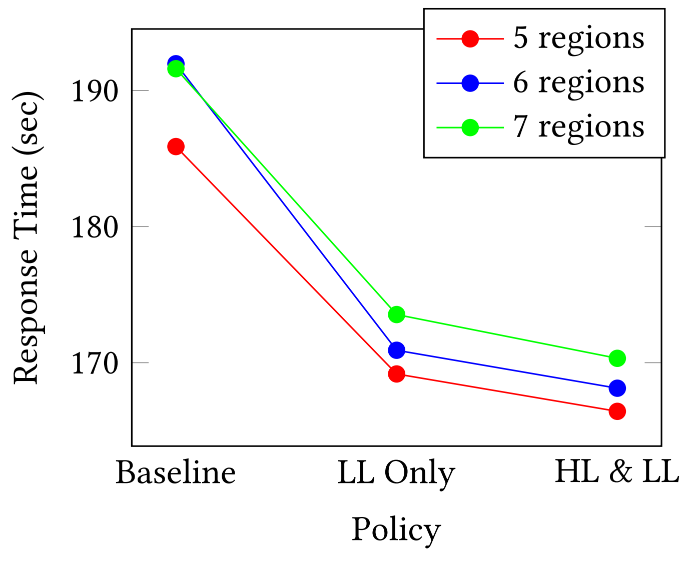

Figure 4.9: Results when applying the baseline, low-level planner (LL Only), and complete hierarchical planner (HL & LL) when applied to incidents sampled from a non-stationary rate distribution. This figure presents a zoomed in view of the average response times.

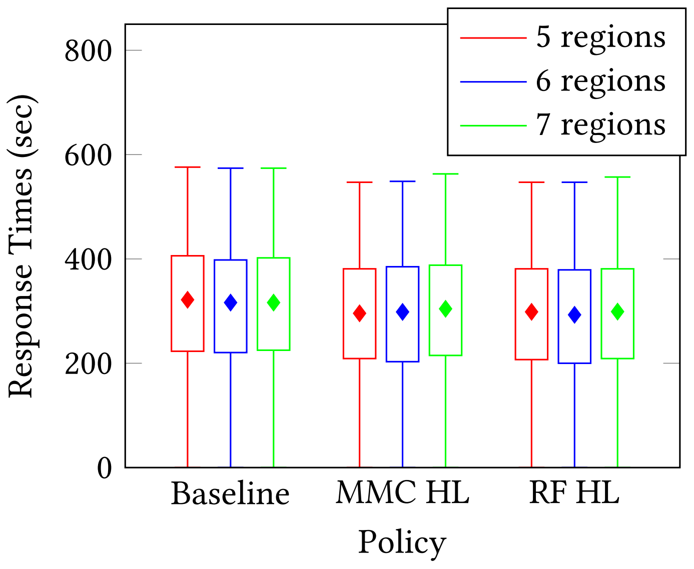

Figure 4.10: Results when applying the baseline, complete hierarchical planner using the MMC queuing high level planner (MMC HL), and complete planner using the surrogate model high level planner (RF HL) when applied to incidents sampled from a non-stationary rate distribution and using a data-driven travel time router. This figure presents the full response time distributions; the boxplot represents the data's Inter-Quartile Range (IQR $= Q_3 - Q_1$), and the whiskers extend to the 9th and 91st percentiles.

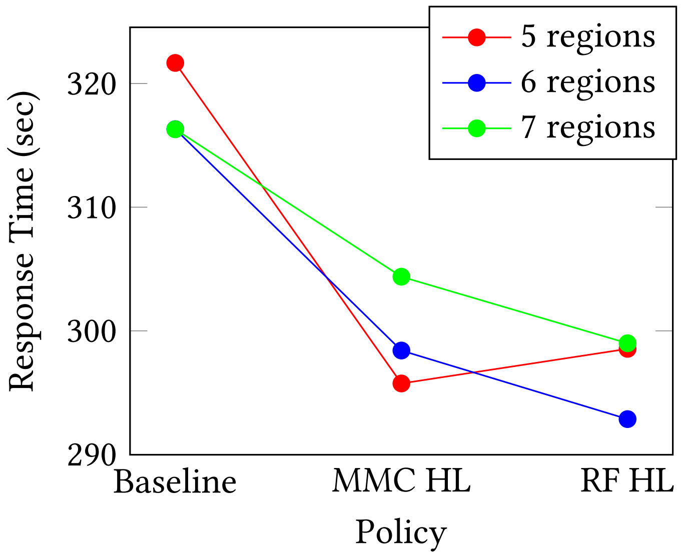

Figure 4.11: Results when applying the baseline, complete hierarchical planner using the MMC queuing high level planner (MMC HL), and complete planner using the surrogate model high level planner (RF HL) when applied to incidents sampled from a non-stationary rate distribution and using a data-driven travel time router. This figure presents a zoomed in view of the average response times.

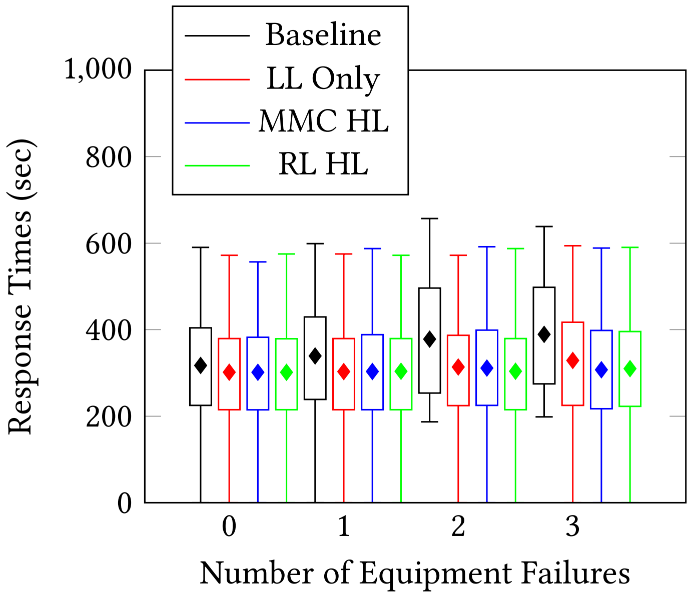

Figure 4.12: Results when applying the low-level planner only (LL Only), complete hierarchical planner using the MMC queuing high level planner (MMC HL), and complete planner using the surrogate model high level planner (RF HL) when subjected to increasing numbers of simultaneous equipment failures. This figure presents the full response time distributions; the boxplot represents the data's Inter-Quartile Range (IQR $= Q_3 - Q_1$), and the whiskers extend to the 9th and 91st percentiles.

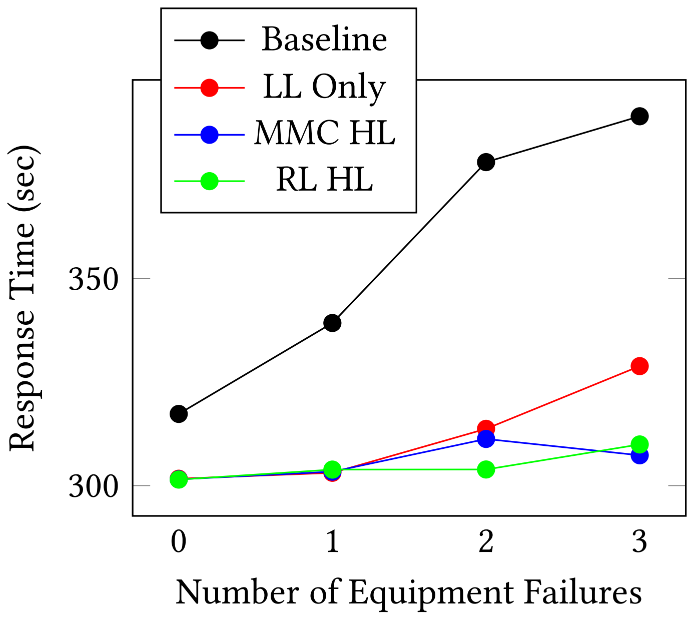

Figure 4.13: Results when applying the low-level planner only (LL Only), complete hierarchical planner using the MMC queuing high level planner (MMC HL), and complete planner using the surrogate model high level planner (RF HL) when subjected to increasing numbers of simultaneous equipment failures. This figure presents a zoomed in view of the average response times.

Stationary Incident Rates: We begin by comparing the baseline policy with the proposed low-level planner on incidents sampled from stationary incident rates. Instead of using the data-driven surrogate and travel-time models, we test the low-level planner in isolation, i.e., we use the simpler queue-based model for initial allocations and a travel-time model based on Euclidean distance (we present results with the data-driven models later). The results are shown in figures 4.6 and 4.7. Our first observation is that using the low-level planner reduces response times for all region configurations, improving upon the baseline by **7.5** seconds on average. This reduction is a significant improvement in the context of emergency response since it is well-known that paramedic response time affects short-term patient survival [102]. We also observe a significant shift in the distribution of response times, with the upper quartile of the low-level results being reduced by approximately **71 seconds** for each region configuration. This reduction in variance indicates that the proposed approach is more consistent than the baseline. As a result, a lesser number of incidents experience large response times.

Non-Stationary Incident Rates: We now examine the results of experiments using incidents generated from non-stationary incident distributions, which are shown in figures 4.8 and 4.9. Again, we begin by using the simple queue-based allocation. Our first observation is that response times generally increase relative to the stationary experiments for both the baseline and the proposed approach. This result is expected since the response to incidents sampled from a non-stationary distribution is more challenging to plan for. However, we also observe that our approach can better adapt to the varying rates. The low-level planner in isolation improves upon the baseline's response times by **18.6 seconds** on average. Introducing the complete hierarchical planner (i.e., both the high-level and low-level planners) improves the result further, reducing response times by **3 seconds** compared to using only the low-level planner, and **21.6 seconds** compared to the baseline. We again observe that the region configuration has a small effect on the efficiency of the proposed approach. This result shows that our approach reduces lower response times irrespective of how the original problem is divided into regions. Finally, we also observe that the variance of the response time distributions achieved by the proposed method is not as low as compared to the stationary experiments, which is likely due to the high strain placed on the system from the non-stationary incident rates.

We now evaluate the surrogate model and its effect on the planner. First, we show how the model performs while forecasting waiting times in unseen test data with respect to the queuing model. We show the results in figure 4.15. We observe that the surrogate model based on random forest regression significantly outperforms the queuing-based model; this improvement is expected as the regression model considers travel times and the time taken to drop victims to hospitals through the simulated data. Even though the queuing-based model has large prediction errors, we show that using it as a heuristic to guide the high-level planner outperforms the baseline approach, most likely because such an estimator learns the proportion of waiting times among the regions fairly well.

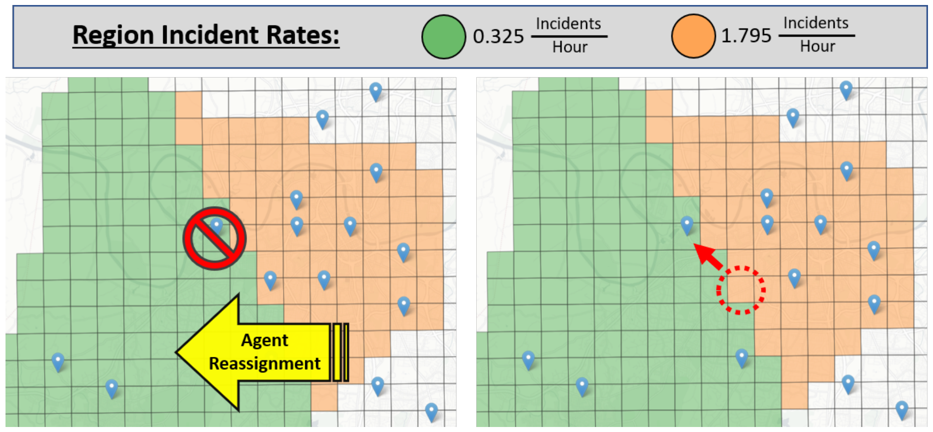

Figure 4.14: Example of the high-level planner resolving an equipment failure. In subfigure (left), the agent positioned at the depot marked by the red circle in the green region fails, and the high-level planner determines there is an imbalance across regions. In subfigure (right), we see the planner move an agent from the depot marked by the red dotted circle to the green region to ensure that the upper left of the region can be serviced.

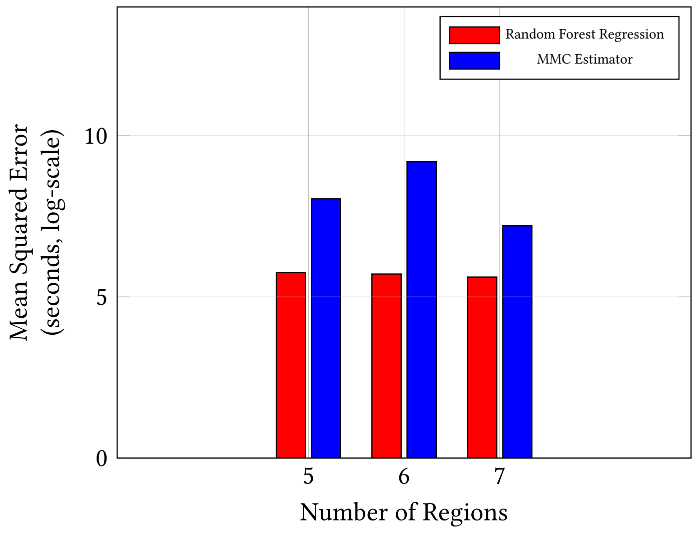

Figure 4.15: Mean Squared Error in logarithmic scale for the proposed estimators. The random forest regression model performs significantly better in comparison to the queuing based estimator. However, as the queuing based estimator learns the proportion of wait times among the regions fairly well, it serves as a meaningful heuristic to guide the high-level planner.

Finally, we test the entire hierarchical planning pipeline (with both the surrogate model and the queuing-based model for initial allocation) and compare it with the baseline approach. We also use the data-driven travel time router to replicate realistic travel times. We present the results in figures 4.10 and 4.11. We see that the hierarchical planner with the data-driven surrogate model usually outperforms the other approaches. On average, it improves response times by about **23 seconds** with respect to the baseline model and by about **6 seconds** with respect to a hierarchical planner that uses a queuing model for the high-level planner when using each model's best region segmentation. We note that the high-level planner using the queuing model outperforms the surrogate model in one case (5 regions). A potential cause for this might be shifts in the underlying environmental distributions, which have been shown to cause learning-based approaches to perform poorly in other domains [103, 104, 105]. Future analyses are needed to examine the impact of such shifts on the surrogate model and identify other potential causes for the queuing model to perform better in some situations.

Responder Failures: Results on the non-stationary incident distribution demonstrate the effectiveness of the hierarchical planner when there are shifts in the spatial distribution of incidents. We now examine its response to equipment failures within the ERM system. Figure 4.14 illustrates an example (from our experiments) of how the planner can adapt to equipment failures. When a responder in the green region fails, the high-level planner determines that imbalance in the spatial distribution of the responders. Intuitively, due to the failure incidents occurring in the upper left cells of the green region could face long response times. Therefore, the planner reallocates a responder from the orange region to the green region. To examine how equipment failures impact the proposed approach, we simulated several responder failures and compared system performance using the baseline policy, the low-level planner in isolation, the full hierarchical approach using the MMC queue high-level planner, and the full approach using the surrogate model. We show the results in figures 4.12 and 4.13. Naturally, as the number of failures increases, response times increase with fewer responders. However, we observe that the proposed hierarchical approach intelligently allocates the remaining responders to outperform the baseline and low-level planner in isolation. Indeed, when there are three simultaneous failures, using the MMC queuing-based hierarchical planner improves response times by about **82 seconds**

compared to the baseline policy and about **22 seconds** compared to using only the low-level planner.

Allocation Computation Times: Decisions using the proposed approach take **180.29** seconds on average. Note that this is the time that our system takes to optimize the allocation of responders. Dispatch decisions are greedy and occur instantaneously. Hence, our system can easily be used by first responders on the field without hampering existing operational speed.

4.8 Related work

Markov decision processes can be directly solved using dynamic programming when the transition dynamics of the system are known [81]. The transition dynamics are typically unknown for resource allocation problems in complex environments like urban areas, [76]. The Simulate-and-Transform (*SimTrans*) algorithm [53] can be used to address this issue; it performs canonical policy iteration with an added computation. In order to estimate values (utilities) of states, the algorithm simulates the entire system of incident occurrence and responder dispatch and keeps track of all states, transitions, and actions, and gradually builds statistically confident estimates of the transition probabilities.

While *SimTrans* finds a close approximation of the optimal policy (assuming that the estimates of the transition probabilities are close to the true probabilities), the process is extremely slow and unsuited to dynamic environments. As an example, even if a single agent (ambulance in this case) breaks down, the entire process of estimating transition probabilities and learning a policy must be repeated. To better react to dynamic environmental conditions, decentralized and online approaches have been explored [83, 46]. For example, Claes et al. [83] entrust each agent to build its own decision tree and show how computationally cheap models can be used by agents to estimate the actions of other agents as the trees are built.

An orthogonal approach to solve large-scale MDPs is using hierarchical planning [17].

Such an approach focuses on learning local policies, known as *macros*, over subsets of the state space. The concept of macro-actions was introduced separately from hierarchical planning as means to reuse a learned mapping from states to actions to solve multiple MDPs when objectives change [106, 107]. Later, the macro-policies were used in hierarchical models to address the issue of large state and action spaces [108, 17].

We also describe how allocation and dispatch are handled in emergency response. First, note that the distinction between allocation and response problems can be hazy since any solution to the allocation problem implicitly creates a policy for response (greedy response based on the allocation) [76]. We use a similar approach in this paper because greedy response satisfies the constraints under which first responders operate. A common metric for optimizing resource allocation is coverage [109, 40, 110]. Waiting time constraints are often used as constraints in approaches that maximize coverage [111, 50]. Decision-theoretic models have also been widely used to design ERM systems. For example, Keneally et al. [57] model the resource allocation and dispatch problem in ERM as a continuous-time MDP, while we have previously used a semi-Markovian process [53]. Allocation in ERM can also be addressed by optimizing the distance between facilities and demand locations [93, 46], and explicitly optimizing for patient survival [7, 112].

4.9 Conclusion

We present a hierarchical planning approach for dynamic resource allocation in city scale cyber-physical system (CPS). We formulate a general decision-theoretic problem for that can be used in a variety of resource allocation settings. We model the overall problem as a Multi-Agent Semi-Markov Decision Process (MSMDP), and show how to leverage the problem's spatial structure to decompose the MSMDP into smaller and tractable sub-problems. We then detail how a hierarchical planner can employ a low-level planner to solve these sub-problems, while a high-level planner identifies situations in which resources must be moved across region lines. We use emergency response as a case-study and validate the proposed approach with data from a major metropolitan area in the USA. Our experiments show that our proposed hierarchical approach offers significant improvements when compared to the state-of-the-art in emergency response planning, as it maintains system fairness while significantly decreasing average incident response times. We also find that it is robust to equipment failure and is computationally efficient enough to be deployed in the field without hampering existing operational speed. While this work demonstrates the potential of hierarchical decision making, several non-trivial technical challenges remain, including how to optimally divide a spatial area such that the solutions of the sub-problems maximize the overall utility of the original problem. We will explore these challenges in future work.

Table 4.1: Notation.

Symbol	Definition
Λ	Set of agents
D	Set of depots
$\mathscr{C}(d)$	Capacity of depot d
G	Set of cells
R	Set of regions
S	State space
A	Action space
P	State transition function
T	Temporal transition distribution
α	Discount factor
$\rho(s,a)$	Reward function given action a taken in state s
$\mathscr{A}$	Joint agent action space
$\mathscr{T}$	Termination scheme
s^t	Particular state at time t
I^t	Set of cell indices waiting to be serviced
$\mathscr{Q}(\Lambda)$	Set of agent state information
p_j^t	Position of agent j
g_j^t	Destination of agent j
u_j^t	Current status of agent j
s_i, s_j	Individual states
σ	Action recommendation set
η	Service rate
γ_g	Incident rate at cell g
t_h	Time since beginning of planning horizon
$t_r(s,a)$	Response time to an incident given action a in state s
p_j	Number of agents assigned to region r_j
γ_j	Total incident rate in region r_j
$w_j(p_j,\gamma_j)$	Expected waiting time for incidents in region r_j
D_j	Set of depots in region r_j
G_j	Set of cells in region r_j

Chapter 5

Decentralized SCPS Planning

5.1 Overview

Chapter 4 presents a hierarchical decision support framework that breaks up the SCPS into smaller, more manageable sub-problems that are each optimized using standard Monte-Carlo tree search. A high-level planner is used to create the sub-problems and coordinate between them as the environment evolves. This approach is scalable to complex multi-agent SCPS problems such as spatio-temporal resource allocation. However, a limitation of this approach is that it requires both complete communication and coordination between agents within a subproblem, and a framework for communication between the sub-problems for the high-level planner. Many SCPS, such as emergency response services, are safety critical systems that must be robust to failures in an area's communication infrastructure. To illustrate, consider the bombing that occurred in Nashville on December 25th, 2020 [113]. AT&T's central office in the city's downtown area was targeted, and the attack disrupted 911 call centers for several days. Communication infrastructure could also be disrupted by large scale events such as natural disasters; emergency services and other critical SCPS must be robust to communication failure in such situations.

This chapter presents a decentralized decision-making framework for multi-agent SCPS that enables each agent to independently determine its own course of action. The technique can be thought of as an extreme version of hierarchical planning, where each agent is considered its own sub-problem. This allows the system to function with access to limited inter-agent communication while being extremely scalable in terms of the number of agents. However, due to the limited communication, the hierarchical framework's high-level planner that coordinates between the sub-problems is not feasible in this domain.

Instead, each agent must estimate the behavior of the other agents using a computationally cheap model. This chapter describes techniques to model this behavior, ensure system constraints are satisfied, and construct a decentralized MCTS framework.

The work comprising this chapter has been published in the Proceedings of the 19th Conference on Autonomous Agents and MultiAgent Systems (AAMAS) [29].

- G. Pettet, A. Mukhopadhyay, M. Kochenderfer, Y. Vorobeychik, and A. Dubey (2020). "On Algorithmic Decision Procedures in Emergency Response Systems in Smart and Connected Communities," in *Proceedings of the 19th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2020, Auckland, New Zealand*.

5.2 Introduction

Emergency response management (ERM) is a critical problem faced by communities across the globe. First responders must attend to many incidents dispersed across space and time using limited resources. ERM can be decomposed into the following sub-problems — forecasting, planning, and dispatching. Although these have been examined independently, planning and dispatch decisions are dependent on accurate incident forecasting. Therefore, it is imperative that principled approaches are designed to tackle all three sub-problems. However, it is fairly common for ERM systems to follow myopic and straight-forward decision policies. For decades, the most common dispatching approach was to send the closest available responder to the incident (in time or space), after which the responder would return to its base or be reassigned. Such methods do not necessarily minimize expected response times [50]. As cities grow, population density, traffic dynamics and the sheer frequency of incidents make such methods stale and inaccurate. We systematically investigate the nuances of algorithmic approaches to ERM and describe how principled decision-making can aid emergency response.

Naturally, algorithmic approaches to emergency response typically combine a data-driven forecasting model to predict incidents with a decision-making process that provides dispatch recommendations. Canonical approaches towards modeling the decision process involve using a Continuous-Time Markov Decision Process (CT-MDP)[57] or a Semi-Markovian Process (SMDP)[53], which are solved through dynamic programming. While the SMDP model provides a more accurate representation of ERM dynamics, it does not scale well for dynamic urban environments[114]. The trade-off between optimality and computational time has also been investigated by the use of Monte-Carlo based methods[114].

Despite such algorithmic progress and attention in recent years from the AI community [115, 53, 75, 114, 116, 117], there are still issues that impede the adoption of principled algorithmic approaches. We argue that a major problem lies in the very focus of most algorithmic approaches. Most ERM systems seek to perform decision-making *after* incidents occur. While such approaches guarantee optimality in the long run (with respect to response times), they de-prioritize response to some incidents. Our conversations with first-responders[42] revealed two crucial insights about this problem: 1) it is almost impossible to gauge the severity of an incident from a call for assistance and de-prioritize immediate response in anticipation of higher future rewards, and 2) Computer-Aided Dispatch systems (CAD)[118] typically enable a human agent to dispatch a responder in the span of 5-10 seconds. These insights explain why the closest responder is usually dispatched to an incident; it is too risky to de-prioritize incidents of unknown severity.

We raise an important conceptual question about algorithmic approaches to emergency response - is it feasible to optimize over dispatch decisions once an incident has happened? In this paper we argue that the crucial, practical period of principled decision-making is *between* incidents. This avoids the potential consequences of explicitly choosing to de-prioritize response to an incident to achieve future gain, but accommodates the scope of principled decision-making. Most ERM systems do not exploit the scope of dynamically *rebalancing* the spatial distribution of responders according to the need of the hour. This problem is challenging since optimizing responder distribution and response as a multi-objective optimization problem is usually computationally infeasible. Indeed, even Monte-Carlo based methods have previously been used with a restricted action space (only responding to incidents) to achieve acceptable computational latency[114]. We address this challenge by proposing two efficient algorithmic approaches to optimize over the spatial distribution of responders dynamically.

The second set of problems that impedes the adoption of algorithmic decision-making in ERM is related to resilience and efficiency. Data processing and decision-making for algorithmic dispatching usually occur in a centralized manner (typically at a central data processing center), which is then communicated to responders. ERM, however, clearly evolves in a multi-agent setting, in which the agents have the capacity to perform independent computation (most modern ambulances are equipped with laptops). In an extremely time-critical setting, especially during communication breakdowns often caused by disasters, it is crucial that such computing abilities are used, and distributed and parallelized algorithmic frameworks are designed. Also, centralized decision-making systems treat all agents as part of a monolithic object or state. This is redundant, as agents often operate independently (for example, an ambulance in one part of the city is usually not affected by an incident in a completely different or distant part). In this paper, we argue that decentralized planning could identify and utilize structure in the problem and save vital computational time.

Contributions: We focus on two problems in this paper 1. designing an approach that can accommodate rebalancing of resources to ensure efficient response, and 2. designing the ability for an emergency response system to be equipped to deal with scenarios that require decentralized planning with very limited communication. To this end, we start by modeling the problem of optimal response as a Multi-Agent Semi-Markov Decision Process (M-SMDP)[119, 120]. Then, we describe a novel algorithmic approach based on Multi-Agent Monte-Carlo Tree Search (M-MCTS)[83] that facilitates parallelized planning to dynamically rebalance the spatial distribution of responders. Our approach utilizes the computation capacity of each individual agent to create a *partially* decentralized approach to planning. Finally, we evaluate our framework using real-world data from Nashville, TN. We find that these approaches maintain system fairness while decreasing the average and variance of incident response times when compared to the standard procedure.

Outline: Through the rest of the paper, we describe the overall problem of emergency response and explain the algorithmic framework. We begin by providing a brief background regarding how ERM pipeline can be modeled technically, and how theoretical approaches to solution work in such situations. Then, we describe our algorithmic framework in detail, and finally, evaluate our framework using incident and response data from Nashville, TN. Table 5.1 can be used as a reference for the symbols we use.

5.3 System Model

Our goal is to develop an approach for emergency responder placement and incident response in a dynamic, continuous-time and stochastic environment. We begin with several assumptions on the problem structure and information provided *a-priori*. First, we assume that we are given a spatial map broken up into a finite collection of equally-sized cells G, and that we are given an exogenous spatial-temporal model of incident arrival in continuous time over this collection of cells (we describe one such model later). Second, we assume that for each spatial cell, the temporal distribution of incidents is homogeneous. Our third assumption is that emergency responders are allowed to be housed in a set of fixed and exogenously specified collection of depots D. Depots are essentially a subset of cells that responders can wait in, and are analogous to fire-stations in the real-world. Each depot $d \in D$ has a fixed capacity $\mathscr{C}(d)$ of responders it can accommodate at a time. We assume that when an incident happens, a free responder (if available) is dispatched to the site of the incident. Once dispatched, the time to service consists of two parts: 1) time taken to travel to the scene of the incident, and 2) time taken to attend to the incident. If no free responders are available, then the incident enters a waiting queue.

5.3.1 Incident Arrival

An important component of a decision-theoretic framework to aid emergency response is the understanding of *when* and *where* incidents occur. While our algorithmic framework can work with any forecasting model, we briefly describe the one that we choose to use: a continuous-time forecasting model based on survival analysis. It has recently shown state-of-the-art performance in prediction performance for a variety of spatial-temporal incidents (crimes, traffic accidents etc.)[50, 49, 75]. Formally, the model represents a probability distribution over inter-arrival times between incidents, conditional on a set of features, and can be represented as

$$f_t(T = t|\gamma(w))$$

where f_t is a probability distribution for a continuous random variable T representing the inter-arrival time, which typically depends on covariates w via the function γ. The model parameters can be estimated by the principled procedure of Maximum Likelihood Estimation (MLE) [121].

5.3.2 Decision-Making Process

The evolution of incident arrival and emergency response occur in continuous-time, and can be cohesively represented as a Semi-Markov Decision Process (SMDP) [53]. An SMDP system can be described by the tuple $(S, A, P, T, \rho(i,a), \alpha)$ where S is a finite state space, A is the set of actions, P is the state transition function with $p_{ij}(a)$ being the probability with which the process transitions from state i to state j when action a is taken, T denotes the temporal transition with $t(i, j, a)$ representing a distribution over the time spent during the transition from state i to state j under action a, ρ represents the reward function, and α is the discount factor.

To adapt this formulation to a multiagent setting, we model the evolution of incidents and responders together in a Multi-Agent SMDP (MSMDP)[86], which can be represented as the tuple $(\Lambda, S, \mathscr{A}, P, T, \rho(i,a), \alpha, \mathscr{T})$, where Λ is a finite collection of agents and $\lambda_j \in \Lambda$ denotes the j^{th} agent. The action space of the j^{th} agent is represented by A_j, and $\mathscr{A} = \prod_{i=1}^{m} A_j$ represents the joint action space. We assume that the agents are cooperative and work to maximize the overall utility of the system. The components S, ρ and P are defined as in a standard SMDP. $\mathscr{T}$ represents a termination scheme; note that since agents each take different actions that could take different times to complete, they may not all terminate at the same time. An overview of such schemes can be found in prior literature [86]. We focus on asynchronous termination, where actions for a particular agent are chosen as and when the agent completes it's last assigned action. Next, we define the important components of the decision process in detail.

States: A state at time t is represented by s^t which consists of a tuple (I^t, R^t), where I^t is a collection of cell indices that are waiting to be serviced, ordered according to the relative times of incident occurrence. R^t corresponds to information about the set of agents at time t with $|R^t| = |\Lambda|$. Each entry $r_j^t \in R^t$ is a set $\{p_j^t, g_j^t, u_j^t\}$, where p_j^t is the position of responder λ_j, g_j^t is the destination cell that it is traveling to (which can be its current position), and u_j^t is used to encode its current status (busy or available), all observed at the state of our world at time t. For the sake of convenience, we abuse notation slightly and refer to an arbitrary state simply by s and use the notation s_i and s_j to refer to multiple states. We point out that our model revolves around states with specific events that provide the scope of decision-making. Specifically, decisions need to be taken when incidents occur, when responders finish servicing and while rebalancing the distribution of responders. We also make the assumption that no two events can occur simultaneously in our world. In case such a scenario arises, since the world evolves in continuous time, we can add an arbitrarily small time interval to segregate the two events and create two separate states.

Actions: Actions in our world correspond to directing the responders to a valid cell to either respond to an incident or wait. Valid locations include cells with pending incidents or any depot that has capacity to accommodate additional responders. For a specific agent λ_i, valid actions for a specific state s_i are denoted by $A^i(s_i)$ (some actions are naturally invalid, for example, if an agent is at cell k in the current state, any action not originating from cell k is unavailable to the agent). Actions can be broadly divided into two categories - *responding* and *rebalancing*. Responding actions refer to an agent actually going to the scene of an incident to service it. But agents could also be directed to wait at certain depots based on the likelihood of future incidents in the proximity of the said depot. We refer to such actions as rebalancing. Finally, we reiterate that the joint valid action space of all the agents and a particular instantiation of it are defined by $\mathscr{A}$ and a respectively, and that of a specific agent λ_j by A_j and a_j.

Transitions: Having described the evolution of our world, we now look at both the transition time between states, as well as the probability of observing a state, given the last state and action taken. We define the former first, denoting the time between two states s_i and s_j by the random variable t_{ij}. There are four random variables of interest in this context. We denote the time between incidents by the random variable t_a, the time to service an incident by t_s, the time taken for a balancing step as t_b and the time taken for a responder to reach the scene of an incident by t_r. We overload these notations for convenience later. Specifically, we model t_a using a survival model described in section 5.3.1. We model the service times (t_s) by learning an exponential distribution from service times using historical emergency response data, and we model rebalancing time (t_b) simply by the time taken by an agent to move to the directed cell.

We refrain from focusing on the transition function P, as our algorithmic framework only needs a generative model of the world and not explicit estimates of state transition probabilities.

Rewards: Rewards in SMDP usually have two components: a lump sum instantaneous reward for taking actions, and a continuous time reward as the process evolves. Our system only involves the former, which we denote by $\rho(s,a)$, for taking action a in state s. We define the exact reward function in section 5.4.3.

5.3.3 Problem Definition

Given state s and an agent set Λ, the problem is to determine an action recommendation set $\sigma = \{a_1, ..., a_m\}$, $s.t.\ a_i \in A^i(s)$, that maximizes the expected reward. The ith entry in σ contains a *valid* action for the ith agent.

Sovling this problem directly is hard due to its intractable state space. Further, the state transition functions are unknown and difficult to model in closed form, which is typical of urban scenarios where incidents and responders are modeled cohesively [53]. Finally, we have to consider the following practical constraints and limitations.

- Temporal constraints — emergency response systems can afford minimum latency (5-10 seconds in practice).
- Capacity constraints — each depot has a fixed agent capacity.
- Uniform severity constraint — all incidents must be responded to 'promptly', without making a judgement about its severity based on a report or a call.
- Wear and Tear — The overall distance agents travel should be controlled to limit vehicle wear and tear.
- Limited Communication - ERM systems must be equipped to deal with disaster situations, where communication is limited.

The temporal and uniform severity constraints make it difficult to justify implementing dispatch policies other than greedy; in order to improve upon greedy dispatch, some 'good' myopic rewards must be sacrificed for an increase in expected future rewards. Since it is very hard to predict the severity of an incident pre-dispatch, the decision process cannot determine if this sacrifice is acceptable. Therefore, in this work we focus on *inter-incident* planning while maintaining greedy dispatch decisions when an incident is reported. This approach gives the decision-maker more flexibility, as it can proactively position resources rather than reacting to incidents. Our problem then becomes how to distribute responders between incidents such that the greedy dispatching rewards are maximized.

5.4 Rebalancing Approach to ERM

5.4.1 Problem Complexity

Dynamic rebalancing's flexibility comes with an increase in complexity. Consider an example city with $|\Lambda|$ responders (i.e. agents) and $|D|$ locations where responders could be stationed (called *depots*) that each can hold one responder. When making a dispatch decision at the time of an incident, a decision maker has at most $|\Lambda|$ possible choices: which responder to dispatch. If instead it is re-assigning responders across depots, there are significantly more choices. For example, with $|\Lambda| = 20$ and $|D| = 30$, there are 20 dispatching choices per incident, but $P(|D|, |\Lambda|) = \frac{|D|!}{(|D|-|\Lambda|)!} = \frac{30!}{10!} = 7.31 \times 10^{25}$ possible assignments. This will only increase if depots have higher responder capacities.

Approaching the problem from this perspective requires solutions that can cope with this large complexity. One possible approach is to directly solve the SMDP model. Although the state transition probabilities are unknown, one can estimate the transition function by embedding learning into policy iteration[53]. This approach is unsuitable for rebalancing, as it is too slow even for the dispatch problem. A centralized MCTS approach suffers from the same shortcoming, barely satisfies the computational latency constraints in case of the dispatch problem[24]. Instead, we seek to exploit meaningful heuristics to propose computationally feasible rebalancing strategies. We begin by presenting our first approach, which focuses on using historical frequencies of incident occurrence across cells to assign responders.

5.4.2 Multi-Server Queue Based Rebalancing

One way to address the complexity of rebalancing is by considering an informed heuristic. A natural heuristic for ERM rebalancing is *incident rate* — each depot can be assigned responders based on the total rate of incidents it serves. Ultimately, our goal is to find a rebalancing strategy that minimizes expected response times. As a result, we first estimate the response time given a specific assignment of responders. Such a scenario can be modeled as a multi-server M/M/c queue [122]. For a given cell and depot, the response time for an M/M/c queue can be represented as

$$\begin{aligned} &\text{responseTime}(c_d, \upsilon, \mu) = \frac{\omega(c_d, \upsilon/\mu)}{c_d\mu - \upsilon} + \frac{1}{\mu} \\ &\text{where } \omega(c_d, \upsilon/\mu) = \frac{1}{1 + (1 - \frac{\upsilon}{c_d\mu})(\frac{c_d!}{(c_d q)^c})\sum_{k=0}^{c_d-1}\frac{(c_d q)^k}{k!}} \end{aligned} \tag{5.1}$$

where $\mu = \mathbb{E}(t_s)$ is the mean service time of responders, c_d is the number of responders stationed at the depot, υ denotes the rate of incident occurrence at the concerned cell, and $q = \frac{\upsilon}{c\mu}$ is server utilization. The standard M/M/c model above needs slight adjustment to account for the fact that incidents at a cell g can potentially be serviced by any depot, which are located at different distances from g. Therefore, we consider a multi-class queue formulation in which a cell's incident rate is split among each depot. Since depots closer to a cell g are more likely to service its incidents, we split g's incident arrival rate such that the fraction of rate incurred by a depot is inversely proportional to the distance to g.

The following system of linear equations can be used to split the arrival rate of a cell g among depots D.

$$\sum_{d \in D} \upsilon_g^d = \upsilon_g \tag{5.2a}$$

$$\text{dist}(\widetilde{d}, g)\upsilon_g^{\widetilde{d}} = \text{dist}(d_i, g)\upsilon_g^{d_i} \;\; \forall d_i \in D \backslash \widetilde{d} \tag{5.2b}$$

where the variable υ_g^d is the fraction of arrival rate of cell g that is shared by depot d, $\text{dist}(d,g)$ denotes the distance between depot d and cell g, and $\widetilde{d}$ is the depot closest to g. Equation 5.2a ensures that the split rates for each cell $g \in G$ sum to its actual arrival rate υ_g, and equation 5.2b ensures that the weighted υ's are inversely proportional to the relative distances between the depots and the cell. For convenience, we refer to the entire set of split rates by Υ.

The split rates Υ provide a foundation for a responder rebalancing approach, given a few considerations. First, we might not have enough responders to meet the total demand based on Υ. Secondly, the problem of evaluating response times in the context of emergency response is different than the standard M/M/c queue formulation, since travel times are not memoryless, and must be modeled explicitly. To address these issues, we design a scoring mechanism for evaluating a specific allocation of responders to depots for a given Υ. We denote this score by π_Υ. Using Υ, a responder allocation can be scored by summing each depot d's expected response time based on the queuing model (calculated using equation 5.1) and the overall time taken by responders to complete the rebalancing:

$$\pi_\Upsilon = \sum_{d \in D} \sum_{g \in G} \nVdash(d,\Lambda)\{responseTime(c_d, \upsilon_g^d, \mu) + travelTime(d,g)\} \tag{5.3}$$

where $\nVdash(d,\Lambda)$ is an indicator function which set to 1 only if depot d has at least one responder, and the functions $responseTime$ and $travelTime$ are used to denote the expected response time of a depot and travel times needed by agents to respond to incidents. The goal of an assignment method is then to find a responder allocation that minimizes this heuristic score. To minimize the total score we employ an iterative greedy approach, shown in algorithm 9. Once the best depots are found, responders are assigned to them based on their current distance from the depots.

The approach dramatically decreases the computational complexity of rebalancing compared to a brute force search. The complexity for solving the system of linear equations $\{$5.2a, 5.2b$\}$ is $\mathscr{O}(|\Lambda|^3)$, as there are at most $|\Lambda|$ depots that could have a resource allo-

Algorithm 9: Iterative Greedy Action Selection

```
INPUT: number of agents $|\Lambda|$, depots $D$, depot capacities $C$, grid rates $\upsilon_g \forall g \in G$;
final_depot_occupancy := Hash $\{d : 0\}\ \forall d \in D$ ;
do
    candidate_depots := Set $\emptyset$;
    candidate_scores := Hash $\emptyset$;
    for $d \in D$ do
        if final_depot_occupancy[d] $< C(d)$ then
            temp_occ := final_depot_occupancy;
            temp_occ[$d$] $+= 1$;
            find $\Upsilon_d$ by solving system of linear equations {5.2a, 5.2b} given temp_occ;
            $\pi_{\Upsilon_d} := \sum_{d \in D} \sum_{g \in G} \mathbb{1}(d,\Lambda)\{responseTime(\text{temp\_occ}[d], \upsilon_g^d, \mu) + travelTime(d,g)\}$;
            candidate_depots := candidate_depots $\cup d$;
            candidate_scores := candidate_scores $\cup \{d : \pi_{\Upsilon_d}\}$
    best_depot := argmin $\pi_{\Upsilon_d}\ \forall d \in$ candidate_depots;
    final_depot_occupancy[best_depot] $+= 1$;
while sum(final_depot_occupancy) $< |\Lambda|$;
return chosenDepots;
```

cated. The rates are split for each cell $g \in G$ and new depot under consideration d during each iteration of the greedy search in algorithm 9, which is repeated $|\Lambda|$ times to place each responder. This gives the overall algorithm a complexity of $\mathcal{O}(|G||D||\Lambda|^5)$. Taking the same example given above with $|\Lambda| = 20$ and $|D| = 30$ and assuming $|G| = 900$ (based on our geographic area of interest and patrol areas chosen by local emergency responders), the complexity is 1×10^{15} times less than a brute force search.

While this approach is not inherently decentralized, each agent can perform these computations and take actions themselves, requiring minimal coordination. While straightforward and tractable, there are a few potential downsides to this approach. First, this policy does not take into account the internal state of the system. For example, a responder might be on its way to respond to an incident, thereby rendering it unavailable for rebalancing. Secondly, it assumes that historical rates of incident arrival can be used to optimize responder placement for the future, thereby not considering how future states of the system affect a particular rebalancing configuration. To address these issues, we propose a decentralized Monte-Carlo Tree Search algorithm.

5.4.3 Decentralized MCTS Approach

Monte-Carlo Tree Search (MCTS) is a simulation-based search algorithm that has been widely used in game playing scenarios.

MCTS based algorithms evaluate actions by sampling from a large number of possible scenarios. The evaluations are stored in a search tree, which is used to explore promising actions. Typically, exploration policy is dictated by a principled approach like UCT[97]. A standard MCTS-based approach is not suitable for our problem due to the sheer size of the state-space in consideration coupled with the low latency that ERM systems can afford. Instead, we focus on a decentralized multi-agent MCTS (MMCTS) approach explored by Claes et. al [83] for multi-robot task allocation during warehouse commissioning. In MMCTS individual agents build separate trees focused on their own actions, rather than having one monolithic, centralized tree. This dramatically reduces the search space: in our case, at each evaluation step of a Monte-Carlo based approach, using a decentralized multi-agent search reduces the total number of choices from the number of permutations $P(|D|,|\Lambda|) = \frac{|D|!}{(|D|-|\Lambda|)!}$ to only the number of depots $|D|$.

To realize MMCTS for an ERM domain, some extensions need to be made to standard UCT [123]. While an agent is building its own tree, it must model other agents' behavior. Since this estimation is required at every step of every simulation by each agent, finding a model that strikes a balance between computation time and accuracy of predicted actions is vital.

There are also global constraints on the system which mandate agents maintain a minimal degree of coordination. For example, the number of resources assigned to a depot cannot be higher than its capacity. We take this into account by adding a filtering step to the decision process. Similar to Map-Reduce [124], each agent sends their evaluated actions to a central planner which makes the final decisions while satisfying global system constraints.

Next, we describe the architecture of our decentralized MMCTS based algorithm.

- **Reward Structure:** At the core of an MCTS approach is an evaluation function that measures the reward of taking an action in a given state. For a state s in the tree of agent λ_j, we design the reward ρ of taking an action a in s as

$$\rho(s,a) = \begin{cases} \rho_{s-1} - \alpha^{t_h}(t_r(s,a)), & \text{if responding to an incident} \\ \rho_{s-1} - \alpha^{t_h}\psi\frac{\sum_{\lambda_k\in\Lambda}(\phi_k(s,a))}{|\Lambda|}, & \text{if balancing at } s \end{cases} \tag{5.4a}$$

  where ρ_{s-1} refers to the total accumulated reward at the parent of state s in the tree, α is the discount factor for future rewards, and t_h the time since the beginning of the planning horizon t_0. The evaluation function is split into cases reflecting the separate *incident dispatch* and *balancing* steps in our solution approach. In a dispatch step, the reward is updated with the discounted response time to the incident $t_r(s,a)$. In a balancing step, we update the reward by the average distance traveled by the agents (we denote the distance traveled by agent λ_k while balancing due to action a in s by $\phi_k(s,a)$). ψ is an exogenous parameter that balances the trade-off between response time and distance traveled for balancing, and is set by the user depending on their priorities. Distance is not included during dispatch actions, as we always send the closest agent.
- **Evaluating other agents' actions:** Agents must have an accurate yet computationally cheap model of other agents' behavior; we explore two such possible policies — (1) a naive policy that other agents will not rebalance, remaining at their current depot (referred to as *Static Agent Policy*), and (2) an informed policy, which is in the form of the *Queue Rebalancing Policy* described in the section 5.4.2. These are used to select actions for the other agents $\Lambda\backslash\{\lambda_i\}$ when building agent λ_i's search tree, and are represented by the ActionSelection(available agents, state) function in line 5 of algorithm 12.
- **Rollout:** When working outside the MMCTS tree, i.e. rolling out a state, a fast heuristic is used to estimate the score of a given action. We use greedy dispatch without balancing as our heuristic.

Algorithm 10: Decision Process

```
1 INPUT: state $s$, time limit $t_{lim}$;
2 $I$ := Sample Incidents(s)
3 $E$ := $I$ + rebalancing events
4 ranked action set $\widetilde{\mathscr{A}} := \emptyset$;
5 for Agent $\lambda_j \in \Lambda$ do
6     $\widetilde{\mathscr{A}}[\lambda_j]$ := MMCTS(s, $\lambda_j$, $E$, $t_{lim}$);
7 recommended actions $\sigma$ := CentralizedActionFilter($s$, $\widetilde{\mathscr{A}}$);
8 apply $\sigma$ to $s$;
9 Return s;
```

Algorithm 11: MMCTS

```
1 INPUT: state s, agent $\lambda_j$, sampled events $E$, time limit $t_{lim}$;
2 create root of search tree at s;
3 do
4     select most promising node $n$ from tree using UCB1;
5     childNode := Expand($n$, $\lambda_j$, next event $e \in E$ after state($n$));
6     $r_c$ := Rollout(childNode);
7     back-propagate(child, $r_c$)
8 while within time limit $t_{lim}$;
9 return actions $\lambda_j$ could take ranked by average reward
```

- **Action Filtering:** The dispatching domain has several global constraints to adhere to, including ensuring that an incident is serviced if agents are available and that depots are not filled over capacity. To meet these constraints, we propose a filtering step be added to the MMCTS workflow, similar to Map-Reduce. Once each individual agent has scored and ranked each possible action, these are sent to a centralized filter that chooses the final actions for each agent to maximize utility without breaking any constraints.

Another way global constraints affect the workflow is that the set of valid actions for an agent when they build their search tree may not be the same as the valid actions when it comes time for them to make a decision. For example, consider two agents λ_1 and λ_2; if agent λ_1 moves to a station and fills it to capacity, then agent λ_2 cannot move to that station. To address this, we have agents evaluate every action they could possibly take when expanding nodes in the tree, even if those actions would cause an invalid state. As the filter assigns actions to other agents, some of these actions can become valid.

Algorithm 12: Expand

```
1  INPUT: Search Tree Node $n$, agent $\lambda_j$, next important event $e$;
2  if $e$ is balancing step then
3      select un-explored action $a \in A_j$ ;
4      $\lambda_j$ takes action $a$;
5      actions available to other agents are updated ActionSelection($\Lambda \backslash \{$unavailable agents$\}$,
         state($n$));
6  else if $e$ is an incident then
7      dispatch nearest agent to incident
8  create new child node $n_c$ from selected actions;
9  update the child's reward based on the response times (if any) and agent balancing movement
10 update $n_c$ to the time of the next event $e$, fast forwarding the state;
11 return $n_c$;
```

Algorithm 13: Centralized Action Filter

```
1  INPUT: state $s$, ranked actions $\widetilde{\mathscr{A}}$;
2  $\Lambda_{avail}$ := agents($s$) do
3      candidate_actions := $\emptyset$;
4      for Agent $\lambda_j \in \Lambda_{avail}$ do
5          candidate_actions[$\lambda_j$] := valid action $a_j \in \widetilde{\mathscr{A}}[\lambda_j]$ with highest reward $\rho(s,a)$;
6      find agent $\lambda_j$ with highest scored action $a_j \in$ candidate_actions;
7      $\lambda_j$ takes action $a_j$;
8      update actions available to other agents accordingly;
9      remove $\lambda_j$ from $\Lambda_{avail}$;
10 while there are unassigned agents;
```

5.5 Integration Framework

To realize an online ERM decision support system requires a framework of interconnected processes. Our integration framework is built on our prior modular ERM pipeline work[114]. It includes the following components:

- A traffic routing model to support routing requests.
- A model of the environment and how it changes over time, which is used by the incident prediction model.
- A model of the spatio-temporal distribution of incidents.
- A decision process that makes dispatching recommendations based on the current state of the environment, responder locations, and future incident distributions.

This framework is a natural choice as it decouples the decision process (our focus in this work) from other components. As it was designed for the centralized, post-incident dis-

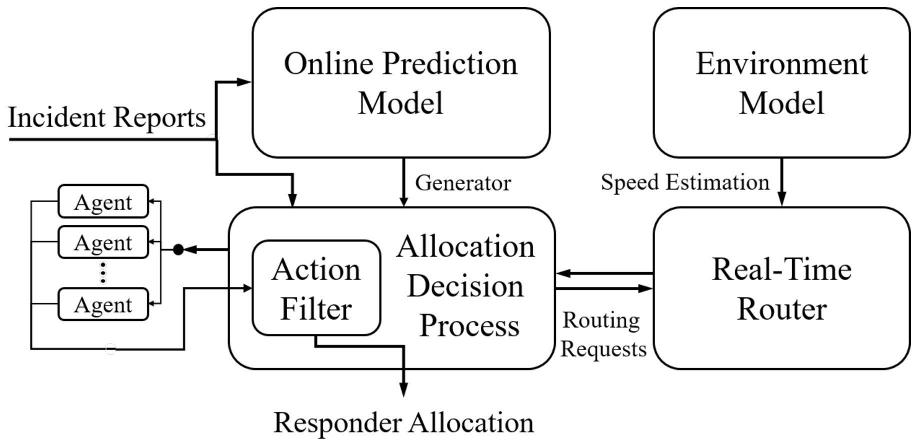

Figure 5.1: Extended Decentralized ERM Framework Overview

patching approach, we make necessary changes to adapt it to our needs. The underlying discrete event simulation was generalized to accept events other than incident occurrence, such as periodic balancing events. The decision process was also extended to handle distributed, multi-agent approaches. An overview of the extended framework can be seen in figure 5.1.

In our experiments we use a Euclidean distance based router, and the incident prediction model outlined in section 5.3. Due to the framework's modularity, these components can be replaced without affecting the decision process.

Incident Prediction Model: While the broader approach of rebalancing the spatial distribution of responders is flexible enough to work with any modular incident forecasting model, we provide a brief evaluation of forecasting using survival analysis. To this end, we generate forecasts 4 hours into the future at intervals of every half an hour for the entire test set, and then repeat the procedure 5 times to reduce variance and increase our confidence in the forecasts. Finally, we create a heatmap (average of all forecasted rates in the test set) to visualize the performance of the model in comparison to actual incidents (see figure 5.2). The forecasting models captures the high and low density areas fairly accurately, as well as the spatial spread of the incidents.

5.5.1 Experimental Design

We perform our evaluation on data from Nashville, TN, a major metropolitan area of USA, with a population of approximately 700,000. The depot locations are based on actual ambulance stations obtained from the city. Traffic accident data was obtained from the Tennessee Department of Transportation, and includes the location and time of each incident. The incident prediction model was trained on 35858 incidents occurring between 1-1-2018 and 1-1-2019, and we evaluated the decision processes on 2728 incidents occurring in the month of January, 2019.

Experimental Configuration and Assumptions: We limit the capacity of each depot to 1 in our experiments. This is motivated by two factors — first, it encourages responders to be geographically spread out to respond quickly to incidents occurring in any region of the city, and it models the usage of ad-hoc stations by responders, which are often temporary parking spots. While the responder service times to incidents are assumed to be exponential in the real world, we set them to a constant for these experiments. This ensures that the experiments across different methods and parameters are directly comparable. If deployed, however, proper service time distributions should be learned and sampled from for each ERM system. We set the total number of responders to 26, which is the actual number of responders in Nashville. We split the geographic area into 900, 1x1 mile square cells. This choice was a consequence of the fact that a similar granularity of discretization is followed by local authorities. To smooth out model noise, each agent evaluates 5 sampled incident chains from the generative model and averages the scores for each action across the playouts. The standard UCB1 [125] algorithm is used to select the most promising node during MCTS iterations. Finally, we augment the queue based rebalancing policy by adding a *radius of influence (RoI)* for each cell. Only depots within a cell's RoI are considered when splitting its rate to encourage even agent distribution and reduce computation time.

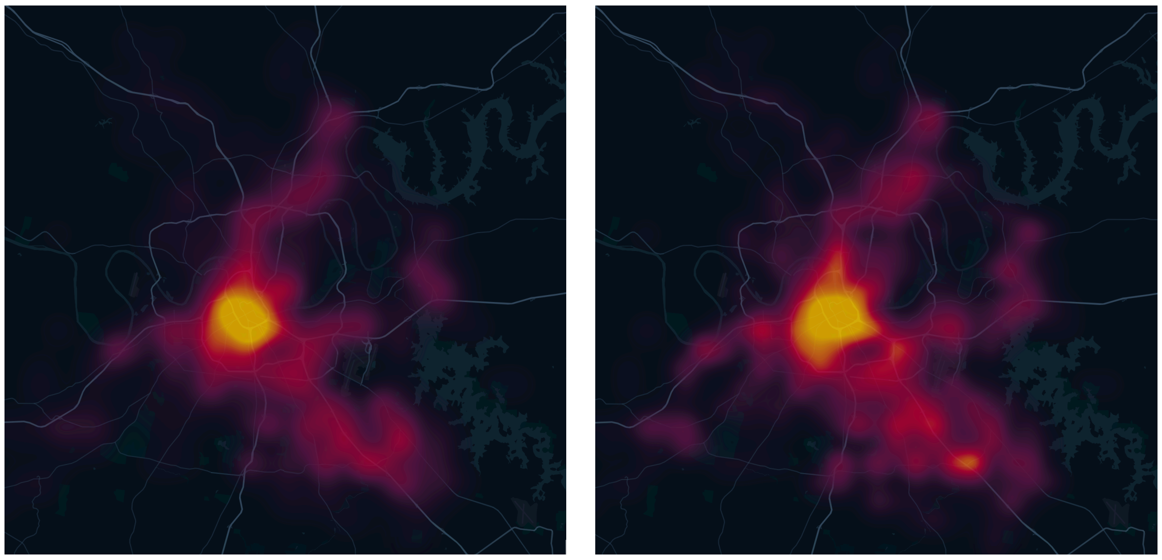
Figure 5.2: Heatmaps comparing average incident rates for the forecasting model (left) with actual incidents in Nashville, TN (right)

.

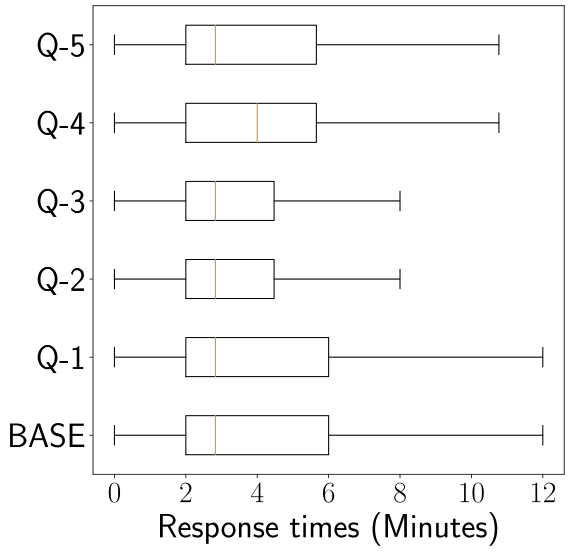

Figure 5.3: The response time distributions for each queue rebalancing policy experiment.

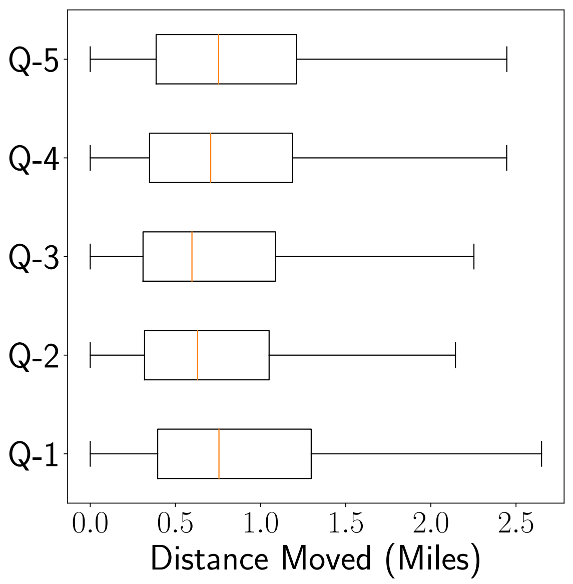

Figure 5.4: Distribution of average miles traveled by each responder at each balancing step in the queue rebalancing policy experiments. The baseline approach has no rebalancing, so it is excluded.

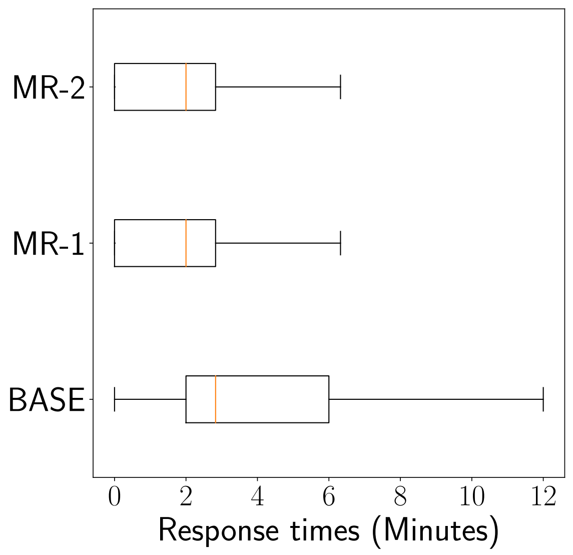

Figure 5.5: The response time distributions for each MMCTS experiment using an oracle.

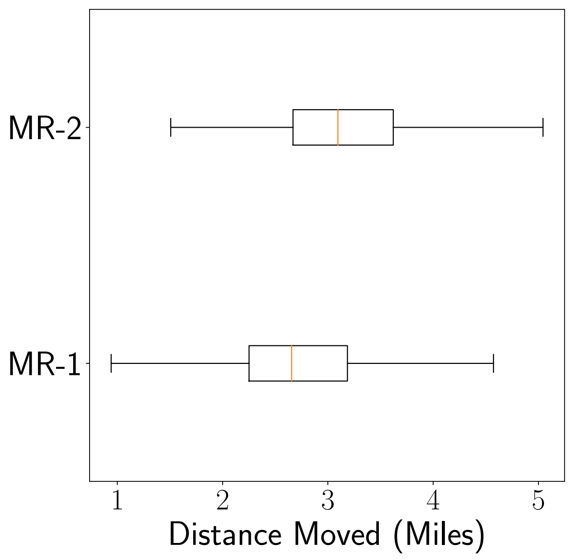

Figure 5.6: Distributions of average miles traveled by each responder at each balancing step of the MMCTS experiments using an oracle.

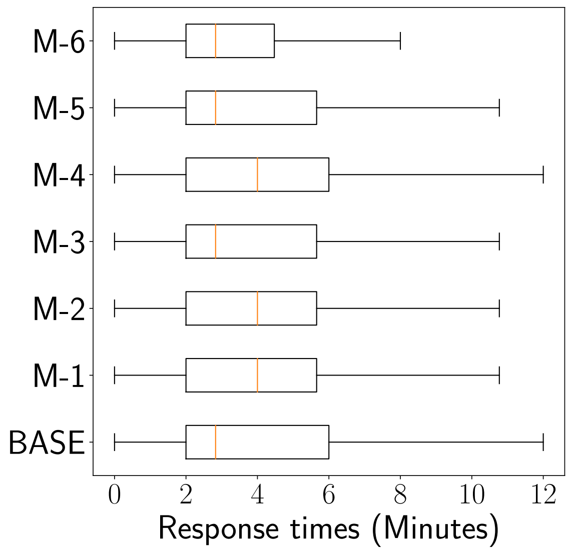

Figure 5.7: The response time distributions for each MMCTS parameter search experiment.

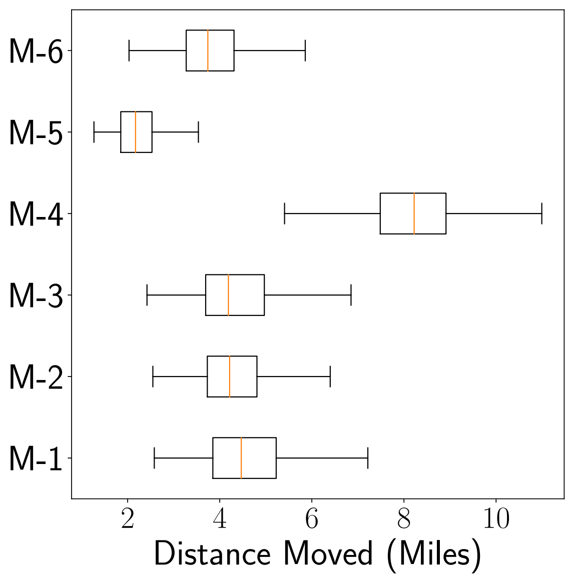

Figure 5.8: Distributions of average miles traveled by each responder at each balancing step of the MMCTS parameter search experiment.

5.6 Results and Discussion

We now discuss the results of the experiments for the two policies.

5.6.1 Queue Based Rebalancing Policy

We first compare the queue based rebalancing policy described in section 5.4.2 to the baseline policy of no rebalancing. In these experiments rebalancing occurred every half hour, and the incident rates υ were average historical rates from the training data. We tested several values (in miles) for the depots' RoI, and compared the distributions of response times (figure 5.3) and the rebalancing distance traveled by each responder (figure 5.4).

Our first observation is that increasing the RoI does not necessarily increase performance; there is an optimal zone around RoI=3, implying that encouraging responders to spread out is beneficial. We also see that while Q-3's median and 1st quartile response times remained fairly consistent with the baseline, the upper quartiles are reduced. This decreases the response time's mean and variance, making the system more fair to all incidents. We also observe that Q-2 and Q-3's responders traveled less than 1 mile on average each balancing step.

5.6.2 MMCTS Rebalancing

To determine the potential of the MMCTS rebalancing approach, we first compare the two agent action models described in section 5.4.3 (*Static Agent Policy* and *Queue Rebalancing Policy*) using **an oracle**, which has complete information regarding future incidents (this assumption takes the errors of the prediction model out of comparison and enables us to observe the best results that we can obtain) . We present the results for the response time distributions in figure 5.5 and the average responder distance traveled per rebalancing step in figure 5.6.

Our first observation is that the MMCTS approach has high potential. Using an oracle, it is able to significantly decrease the response time distribution compared to the queue based policy above. This is not surprising given that a standard MCTS algorithm given perfect information should perform well given adequate time, but it demonstrates that the MMCTS extensions of independent action evaluation for each agent and action filtering are valid. Secondly, we see that MR-1 (using a static agent policy) outperforms MR-2 (using the queue rebalancing policy). Last, we observe that responders traveled between 2 and 4 miles on average each during balancing step in these experiments, which is significantly higher than the queuing approach.

Next, we examine a more the practical approach using the **incident prediction model** based on survival analysis. Since the static agent policy performed better in the oracle experiments, we use it for these experiments. There are several hyper-parameters that can affect the performance of the algorithm, including 1. MCTS Iteration Limit 2. Rebalancing Period - the amount of time between rebalancing steps 3. Distance Weight in Reward Function ψ - this represents the importance of distance traveled for rewards 4. Look-ahead Horizon for MCTS.

We vary these parameters to see their effect on the system (see table 5.2). We present the response time distributions of MMCTS using the incident model in figure 5.7, and the average responder distance traveled per rebalancing step in figure 5.8. We observe that different parameter choices lead to different performance characteristics. For example, we see that changing the distance weight has a large impact on the distance responders travel; users with tight budgets for responder movement and maintenance will want to pay close attention to this parameter. Comparing the queue based policy with MMCTS, we see that both improve the response time distributions compared to the baseline. MMCTS is more configurable, but is also more sensitive to poor hyper-parameter choices. With proper hyper-parameter choices, both fulfil the constraints discussed in section 5.3.3 by having quick dispatching decisions, allowing for limited communication, and allowing users to control for distance traveled (i.e. wear and tear).

5.7 Conclusion

Principled approaches to Emergency Response Management (ERM) decision making have been explored, but have failed to be implemented into real systems. We have identified that a key issue with these approaches is that they focus on post-incident decision making. We argue that due to fairness constraints, planning should occur between incidents. We define a decision theoretic model for such planning, and implement both a heuristic search using queuing theory and a Multi Agent Monte Carlo Tree Search planner. We find that these approaches maintain system fairness while decreasing the average response time to incidents.

While the focus of this work is in the ERM domain, there are important takeaways for general agent-based systems: (1) Planning performance is dependent on the quality of the underlying event prediction models. (2) It is imperative to understand the needs and constraints for a target domain when designing a planning approach for it to be accepted in practice. (3) The computational capacity of “agents” has evolved in recent decades, and should be used to create decentralized planning approaches. Given these takeaways, we will explore the applicability of this framework to other domains where planning occurs over a spatial-temporally evolving process.

Table 5.1: Notation lookup table

Symbol	Definition
Λ	Set of agents
D	Set of depots
$\mathscr{C}(d)$	Capacity of depot d
G	Set of cells
S	State space
A	Action space
P	State transition function
T	Temporal transition distribution
α	Discount factor
$\rho(s,a)$	Reward function given action a taken in state s
$\mathscr{A}$	Joint agent action space
$\mathscr{T}$	Termination scheme
s^t	Particular state at time t
I^t	Set of cell indices waiting to be serviced
R^t	Set of agent states at time t
p_j^t	Position of agent j
g_j^t	Destination of agent j
u_j^t	Current status of agent j
s_i, s_j	Individual states
t_{ij}	Transition time between states s_i, s_j
t_a	Time between incidents
t_s	Time to service an incident
t_b	Time to a balance step
r	Reward
t_r	Incident response time (the time between incident awareness and first agent's arrival on scene)
σ	Action recommendation set
μ	Mean agent service time
c_d	Number of agents at depot d
υ_g	Incident rate at cell g
υ_g^d	The fraction of cell g's incident rate shared by depot d
Υ	Set of occupied depots and their split incident rates
π_Υ	Utility of Υ
t_h	Time since beginning of planning horizon
$t_r(s,a)$	Response time to an incident given action a in state s
$\phi_k(s,a)$	Distance traveled by agent k while balancing
ψ	Exogenous parameter balancing response time and distance traveled
RoI	Radius of Influence of a cell (used in queue based rebalancing policy).

Table 5.2: Outline of the experimental runs performed and their corresponding hyper-parameter choices. (*When not indicated, parameters are set to values of M-1, the MMCTS Baseline in the table.)

Identifier	**Description**	**Hyper-Parameter Choices**
BASE	Greedy Baseline Without Rebalancing	N/A
Q-1	Queue Based Rebalancing Policy with RoI of 1	RoI = 1
Q-2	Queue Based Rebalancing Policy with RoI of 2	RoI = 2
Q-3	Queue Based Rebalancing Policy with RoI of 3	RoI = 3
Q-4	Queue Based Rebalancing Policy with RoI of 4	RoI = 4
Q-5	Queue Based Rebalancing Policy with RoI of 5	RoI = 5
MR-1	MMCTS - using an oracle for future incidents and a Static Agent Policy	*Same as MMCTS Baseline M-1*
MR-2	MMCTS - using an oracle for future incidents and a Queue Rebalancing Policy	*Same as MMCTS Baseline M-1*
M-1	MMCTS - Baseline The foundation for the parameter search. Each parameter varies independently while other parameters retain these values. (All M-* experiments use generated incident chains and a Static Agent Policy)	MCTS Iteration Limit = 250 Lookahead Horizon = 120 min Reward Distance Weight ψ = 10 Reward Discount Factor = 0.99995 Rebalance Period = 60 min
M-2	MMCTS - Iteration Limit of 100	MCTS Iteration Limit = 100*
M-3	MMCTS - Iteration Limit of 500	MCTS Iteration Limit = 500*
M-4	MMCTS - Reward Distance Weight ψ of 0	Reward Distance Weight ψ = 0*
M-5	MMCTS - Reward Distance Weight ψ of 100	Reward Distance Weight ψ = 100*
M-6	MMCTS - Rebalance Period of 30 minutes; Lookahead Horizon of 30 minutes	Lookahead Horizon = 30 min Rebalance Period = 30min*

Chapter 6

Combining Learning and Planning for Adaptive Decision Making

6.1 Overview

Chapters 3, 4, and 5 present a spectrum of online planning approaches for multi-agent SCPS, each with advantages and limitations: Chapter 3's centralized approach completely captures the set of possible interactions between agents, but is not scalable to complex environments. Chapter 4's hierarchical approach can scale to much more complex environments by splitting the problem into tractable sub-problems, but requires domain specific segmentation approaches and communication between agents within a sub-problem. Finally, Chapter 5's decentralized approach is extremely scalable and resilient to communication disruptions, but requires a computationally cheap model of agent behavior to replace coordination that can lead to sub-optimal decisions.

A limitation that is shared by all of these methods is that, being pure planning approaches, they must perform all of their computation at decision time. Even decentralized algorithms can take too long to converge for time-sensitive domains such as emergency response management in large cities.

As discussed in Chapter 1, learning-based decision-making algorithms such as reinforcement learning (RL) are an alternative to online planning. These algorithms interact with the environment offline to learn a general mapping from states of the environment to actions that should be taken (called a *policy*). The advantage of these methods is that once a policy is learned, it can be invoked nearly instantaneously at decision time. Unfortunately, non-stationary environments can cause a policy to become stale and result in sub-optimal decisions.

This chapter presents a hybrid decision-making approach called *Policy Augmented*

Monte Carlo tree search (PA-MCTS) which combines learning-based RL with online planning. The intuition behind the approach is that if the environment has not changed too much between when an optimal policy was learned and when a decision needs to be made, the policy can still provide useful information for decision-making. Combining an *old* policy (i.e., a policy learned on an environment that has changed since) with a *current* online search (i.e., search using the current environmental parameters) results in a two-fold advantage. First, given a specific computational budget, the framework converges to significantly better decisions than standard MCTS. Second, we show how the online search makes the approach significantly more robust to environmental changes than standard state-of-the-art RL approaches. Several properties of the approach are proven, including when PA-MCTS will return the optimal action, when it will choose better actions than either pure MCTS or greedy Q action selection, and the bound on the total deviation in cumulative rewards from an optimal policy when used for sequential decision-making.

The work comprising this chapter has been submitted to the Thirty-sixth Conference on Neural Information Processing Systems (2022), and is currently pending review. It builds upon work published at the 5th Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM) [30].

- G. Pettet, A. Mukhopadhyay, K. Wray, A. Dubey (2022). "Decision Making in Non-Stationary Environments with Policy-Augmented Monte Carlo Tree Search," in *Proceedings of the 36th Conference on Neural Information Processing Systems, NeurIPS 2022, New Orleans, LA, USA* ***(Submitted, pending review)***.

- G. Pettet, A. Mukhopadhyay, A. Dubey (2022). "Decision Making in Non-Stationary Environments with Policy-Augmented Monte Carlo Tree Search," in *Proceedings of the 5th Multidisciplinary Conference on Reinforcement Learning and Decision Making, RLDM 2022, Providence, RI, USA*.

6.2 Introduction

Sequential decision-making is present in many important problem domains, such as autonomous driving, emergency response, and medical diagnosis [126]. An open challenge in such settings is non-stationary environments, where the dynamics of the environment can change over time. A decision agent must adapt to these changes or take sub-optimal actions. Two well known approaches for sequential decision-making are reinforcement learning (RL) and online planning [126]. In RL approaches, an agent learns a policy π, i.e., a mapping from states to actions, through interacting with the environment. The learning can also take place offline using environmental models. Once a policy is learned, it can be invoked nearly instantaneously at decision time. Deep RL methods, which use a neural network as a function approximator for the policy, have achieved state-of-the-art performance in many applications [13, 126]. However, when faced with non-stationary environments, a policy can become stale and result in sub-optimal decisions. Moreover, retraining the policy on the new environment takes time and considerable computational effort, particularly in problems with complex state-action spaces. While RL algorithms have been designed to operate in non-stationary environments [127, 128], there is a delay between when a change is detected and when the learning framework converges to the updated policy. Depending on the problem setting, such delays might be very expensive.

An alternative approach is to perform online planning using algorithms such as Monte Carlo tree search (MCTS). Given the current environment, these approaches perform their computation at decision time using high-fidelity models to determine promising action trajectories. These models can be updated as soon as environmental changes are detected, and such changes can be immediately incorporated in decision-making (assuming that the generative model used to build the search tree can be updated quickly). MCTS has been proven to converge to optimal actions given enough computation time [97], but convergence can be slow for domains with large state-action spaces. The slow convergence is a particular issue for problem settings with tight constraints on the time allowed for decision-making;

e.g., in emergency response, when an incident occurs, any time used for decision-making increases the time until a responder is dispatched [84]. While several approaches have been designed to leverage problem structure to scale MCTS [129, 84], such approaches make assumptions about the environment to prune the search space, which can lead to sub-optimal decisions.

As we point out, both RL and MCTS have weaknesses when applied to complex decision-making problems in non-stationary environments. In this paper, we present a novel hybrid decision-making approach that combines the strengths of RL and online planning while mitigating some of these weaknesses. The intuition behind our approach is that if the environment has not changed too much between when an optimal policy was learned and when a decision needs to be made, the policy can still provide useful information for decision-making. Our approach, called *Policy Augmented Monte Carlo tree search* (PA-MCTS), is extremely simple—it combines a policy's action-value estimates and the returns generated from MCTS. We show that combining an *old* policy (i.e., a policy learned on an environment that has changed since) with a *current* online search (i.e., search using the current environmental parameters) results in a two-fold advantage. First, given a specific computational budget, our framework converges to significantly better decisions than standard MCTS. Second, we show how the online search makes our approach significantly more robust to environmental changes than standard state-of-the-art RL approaches.

In this paper, we first introduce the notion of *Q-bounded non-stationary Markov decision processes* (Q-NSMDP), which is a general framework for quantifying how MDPs evolve in non-stationary settings (Section 6.4). We describe prior work in this domain in Section 6.3. Then, in Section 6.5, we explain how our approach, PA-MCTS, can combine old action-value estimates with online search and present several theoretical results. Finally, we present experimental results in Section 6.6. We show that under non-stationary settings, PA-MCTS outperforms baselines in terms of utility and robustness, and converges faster than standard MCTS.

6.3 Related Work

Sequential decision-making in non-stationary and dynamic environments has been studied from several perspectives. Satia and Lave [130] and White and Eldeib [131] consider transition matrices constrained within a pre-specified polytope. However, as pointed out by Iyengar [132], they do not discuss how such polytopes can be constructed. Iyengar introduced the idea of robust MDPs by using the concept of uncertain priors [132], where the transition function can change within a set of functions due to uncertainty [133]. Choi et al. [134] introduced hidden-mode MDPs, that considered a formal model for representing changes in the environment, confined with a set of modes. A broader notion of non-stationarity was introduced by Lecarpentier and Rachelson [133], who considered that both the reward function and the transition function can change over time and that the rate of change is bounded through Lipschitz continuity.

Our formulation is inspired by that of Lecarpentier and Rachelson [133]; sequential decision-making in real-world settings is a complex task and we also consider that both the reward and the transition functions can change over time. The key difference in our formulation is the representation of that change. Explicitly specifying the changes in the transition function makes it imperative that we know the transition function explicitly in the first place; as pointed out by Satia and Lave [130] and Mukhopadhyay et al. [114], this is often not the case in practice. Therefore, we use the Q-function to specify how much the decision-process has changed due to environmental changes. We also assume that while the transition function might not be available explicitly, the environmental change can be detected and that a black-box simulator of the system under consideration is available, as in prior work [135]. We also point out that decision-making by an agent which is trained on one task (or environmental conditions) and subsequently provided with another task has also been explored in the domains of transfer learning [136] and lifelong reinforcement learning [137]. However, we specifically look at settings where "learning" a new policy is not feasible, or even when it is feasible, decisions must be taken while the updated policy is being learned. Finally, we recognize that approaches combining model-based online search with learning methods have been explored – for example, AlphaZero integrates MCTS with a policy iteration framework [18], while the Search with Amortized Value Estimates (SAVE) algorithm combines model-free Q-learning with MCTS [138]. To the best of our knowledge, our approach is the first that applies such an approach to non-stationary environments.

6.4 Markov Decision Processes in Non-Stationary Settings

Markov decision processes (MDP) provide a general framework for sequential decision-making under uncertainty. An MDP can be defined by the tuple $(\mathscr{S}, \mathscr{T}, \mathscr{A}, P(s,a), R(s,a))$, where $\mathscr{S}$ is a finite state space, $\mathscr{A}$ is a discrete action-space, $P(s' \mid s,a)$ is the probability of reaching state s' when taking action a in state s, and $R(s,a)$ is the scalar reward when action a is taken in state s. The goal of an agent is to learn a policy π that maps states to actions (or, more generally, states to a distribution over actions) that can maximize a specified utility function. Typically, the utility function is simply the expected reward, i.e., $\mathbb{E}_P[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s))]$, where γ denotes the discount factor that weighs immediate rewards more than future rewards. We are interested in settings where the environment with which the agent interacts changes over time. Specifically, we consider an agent that is *trained* on a specific task given a particular setting, e.g., consider the problem of proactively allocating ambulances in a city in anticipation of accidents modeled as an MDP [139]. Now, consider that the city experiences unexpected congestion, changing the transition and reward functions of the underlying MDP. We seek to analyze and understand how an agent trained to operate in an environment before the change can adapt to the new environment.

We point out that our interest lies in problem settings where *learning* a new policy immediately is infeasible in practice. In complex real-world tasks, learning a new policy takes time, and decisions must be made as the policy is being updated based on the new environmental conditions. We are interested in understanding how agents can optimize decision-making during such delays, i.e., between when a change is detected in the environment and when the agent learns a new (near-optimal) policy. One way to model such a problem is a non-stationary Markov decision process (NSMDP) [133]. An NSMDP, as defined by Lecarpentier and Rachelson [133], can be viewed as a stationary MDP where the state space is enhanced with time. Depending on whether the task is episodic and the agent is allowed to explore along the temporal axis, this enhancement can be trivial or difficult [133]. The rate of change in the transition and reward functions is typically bounded in an NSMDP by assuming Lipschitz continuity [133]. However, changes to the transition and reward functions are difficult to observe and analyze in practice, especially in real-world applications where an agent is deployed to solve complex tasks [84].

We propose a new approach to bound the non-stationarity of an MDP by directly considering the action-value function Q. Let $s_0 \in \mathscr{S}$ and $a_0 \in \mathscr{A}$ denote the initial state and action (respectively) at time step 0. The Q function, represented as $Q^{\pi}(s,a) = \mathbb{E}_P\{R(s_0,a_0) + \sum_{t=1}^{\infty} \gamma^t R(s_t, a_t = \pi(s_t))\}$, measures the value of a state s and an action a under policy π. Our key hypothesis is that with small changes in the environment, the Q function under an optimal policy does not change much, i.e., "good" actions remain valuable, and "bad" actions do not suddenly become promising. It is trivial to see that there are exceptions to our hypothesis and its rather simplified explanation; if the Q function stayed the same as before, there would not be any need for the agent to adapt to changes in the environment. In this paper, we study how agents can adapt to such changes, given that the change in the optimal Q function is bounded.

In order to differentiate our definition from the one used by Lecarpentier and Rachelson [133], we refer to our setting as the *Q-bounded non-stationary Markov decision process* (Q-NSMDP). We first introduce some notation before formally defining our problem setting. Consider without loss of generality that the agent learns the optimal policy π_0^* for the given environment at time step 0. We denote the Q function under this optimal policy by $Q_0^{\pi_0^*}$, where the subscript denotes the time step in consideration. Now, consider that the environment undergoes some change between time 0 and t. We assume that:

$$|Q_0^{\pi_0^*}(s,a) - Q_t^{\pi_t^*}(s,a)|_\infty \leq \varepsilon \quad \forall \ t \in \mathscr{T} \tag{6.1}$$

where $Q_0^{\pi_0^*}$ is the Q-function under the optimal policy π_0^* at time step 0 (i.e., the last time step or decision epoch where the optimal Q-function is known by the decision agent), $Q_t^{\pi_t^*}$ is the Q-function under the optimal policy π_t^* at time step $t \in \mathscr{T}$, and $\varepsilon \in \mathbb{R}^+$ is a scalar bound. In other words, the expected return after taking some action a in state s at time step t (under the updated environmental conditions) will be off by no more than ε from the expected return of taking action a in state s at time step 0 (under the old environmental conditions). We also point out that while our problem definition is agnostic to whether the change is continuous or discrete (we could define an analogous setting for continuous-time MDPs), our algorithm only tackles discrete changes for now.

6.5 Policy Augmented Monte Carlo Tree Search

Our approach is based on Monte Carlo Tree Search (MCTS), an anytime search algorithm that builds a search tree in an incremental and asymmetric manner [140, 141]. The fundamental idea of MCTS is that the tree can be explored asymmetrically, with the search being biased toward actions that appear promising. To estimate the value of an action at a node in the tree (nodes denote states), MCTS uses a model of the environment at the current decision epoch ($\mathscr{M}_t$) to simulate a "playout" to the end of the planning horizon [141]. Using these estimates, a *tree policy*, such as the standard Upper Confidence bound for Trees (UCT) algorithm [97], is used to determine which action trajectories to explore. As the tree is explored and nodes are revisited, the estimates are re-evaluated and they converge toward the actual *value* of the node, i.e., for a node that represents state s, the value converges to $\arg\max_a Q_t^{\pi_t^*}(s,a)$. Once a computational budget is reached, the tree is used to select which action to take in the current state based on a selection criteria. For example, a common selection method is to choose the action with the highest expected return, i.e., $a^* = \arg\max_{a \in \mathscr{A}_s} \overline{G}_t(s,a)$, where s is the current state at decision epoch t, $\mathscr{A}_s$ is the set of valid actions that can be taken at s, and $\overline{G}_t(s,a)$ is MCTS's estimate of the expected return for taking each action $a \in \mathscr{A}_s$ at state s.

MCTS is proven to converge to the optimal action given infinite time [97]. However, the number of iterations required can become impractical as the state-action space of the environment grows. The convergence also holds true for an environment that has changed—this consideration is actually meaningless for a purely online approach such as MCTS, assuming that the generative model that is used to build the tree has been updated based on environmental changes.[1] On the other hand, a trained policy has exactly the opposite advantage (and disadvantage): while it can be invoked in constant time during decision-making, it cannot accommodate environmental changes without re-training, which is computationally expensive. We raise the following question: *can the advantages of online search procedures (e.g., MCTS) be combined with those of offline policy learning (e.g., reinforcement learning) to tackle discrete changes in the environment?*

Our approach presents a natural solution to the question we raised. *Policy-Augmented Monte Carlo Tree Search* (PA-MCTS) addresses this challenge by integrating Q-values learned on the environment at an earlier decision epoch with an online search, even if the environment has changed. Rather than selecting an action based on the highest expected return estimated by the online search, PA-MCTS instead chooses the action that maximizes a convex combination of the previously learned Q-values and the MCTS estimates $\overline{G}$:

$$\underset{a \in \mathscr{A}_s}{\arg\max} \quad \alpha Q_0^{\pi_0^*}(s,a) + (1-\alpha)\overline{G}_t(s,a) \tag{6.2}$$

where $Q_0^{\pi_0^*}(s,a)$ is the previously learned optimal[2] Q-function learned by the decision

[1]Environmental models can be updated in a purely online manner, as shown by Mukhopadhyay et al. [114]

[2]In principle, we do not require the optimal Q-function. As we show in the experiments, an approximation also works well in practice.

Algorithm 14: Policy-Augmented Monte Carlo Tree Search

```
Input  : Current state $s$, initial $Q$-values $Q_0^{\pi_0^*}$, weight hyperparameter $\alpha$,
         environment model $\mathscr{M}_t$ at current decision epoch $t \in \mathscr{T}$
Output: Action to take at state $s$
1 $\mathscr{A}_s :=$ Actions that can be taken in state $s$;
2 Run MCTS using $\mathscr{M}_t$ to obtain the average returns $\overline{G}_t(s,a) \quad \forall\, a \in \mathscr{A}_s$;
3 Return $\arg\max_{a \in \mathscr{A}_s} \alpha Q_0^{\pi_0^*}(s,a) + (1-\alpha)\overline{G}_t(s,a)$;
```

agent. The hyper-parameter α, chosen such that $0 \leq \alpha \leq 1$, controls the tradeoff between the learned Q-values and the returns generated through MCTS estimates: if $\alpha = 1$, PA-MCTS reduces to greedy action selection using $Q_0^{\pi_0^*}$; whereas if $\alpha = 0$, it reduces to standard MCTS. We essentially seek to balance the dichotomy between using accurate estimates generated using an older environment (through $Q_0^{\pi_0^*}$) and inaccurate estimates generated using the current environment (through $\overline{G}_t$). If $\alpha \in (0,1)$, then both estimates are considered (we refer to the action-value from the Q-function as "estimates" due to the change in the environment). We hypothesize that when the error in Q-values is bounded by ε (as described in Eq. (6.1)), $Q_0^{\pi_0^*}$ likely embeds useful information about the updated environment. By combining $Q_0^{\pi_0^*}$ with the estimates from MCTS (which are sampled using an updated model of the environment), PA-MCTS can utilize the knowledge from these outdated Q-values in addition to updated information through the online search.

We present the PA-MCTS algorithm in Algorithm 14. The algorithm requires $Q_0^{\pi_0^*}$ and an updated model of the environment $\mathscr{M}_t$ at the current decision epoch t as inputs. Given the inputs, MCTS is run using $\mathscr{M}_t$ to estimate the expected returns $\overline{G}_t(s,a) \ \forall\, a \in \mathscr{A}_s$ where $\mathscr{A}_s$ is the set of actions that can be taken at state s (line 2). Then, these estimates are combined with the initial Q-values for each action to determine each action's score. This is a convex combination, weighted by hyperparamter α, which controls the balance between the initial Q-values and the updated return estimates from MCTS (line 3). The algorithm returns the action that maximizes the combined score.

6.5.1 Theoretical Analysis

We prove three properties of PA-MCTS: (1) the conditions under which PA-MCTS will return the optimal one-step action, (2) the conditions under which PA-MCTS will choose an action with a higher estimated return than either MCTS or selection using Q-values $Q_0^{\pi_0^*}$, and (3) a bound on the total deviation of the expected return from an optimal (updated) policy when following PA-MCTS. We present brief sketches for the proofs in the main body of the paper. Detailed descriptions for each proof can be found in the appendix. We begin by defining some additional notation.

Definition 6.5.1. *Let $a_t^* := \arg\max_{a \in \mathscr{A}_s} Q^{\pi_t^*}(s,a)$ be the optimal action at an arbitrary time step t.*

Definition 6.5.2. *Let $a_t' := \arg\max_{a \in \mathscr{A}_s} Q_t^{\pi_t^*}(s,a)$, $a_t' \neq a_t^*$, denote the second best action[3]at an arbitrary time step t.*

Definition 6.5.3. *Let $\psi_t := Q_t^{\pi_t^*}(s,a_t^*) - Q_t^{\pi_t^*}(s,a_t')$; for a state s, ψ_t denotes the difference in Q values when taking actions a_t^* and a_t' at time t and following the optimal policy π_t^* thereafter.[4]*

Finally, while MCTS is guaranteed to converge to the optimal expected returns for a given state and action given infinite time, actions must be taken after limited time in practice. Below, we define how "far" the estimates of MCTS are from the optimal values:

Definition 6.5.4. *Let δ denote the bound on the error of the values estimated by MCTS when it is stopped, i.e., $|Q_t^{\pi_t^*}(s,a) - \overline{G}_t(s,a)|_\infty \leq \delta \quad \forall\ t \in \mathscr{T}$.*

We first present Theorem 1, which describes the conditions with respect to ε, δ, and ψ_t under which PA-MCTS returns the optimal one-step action.

[3]We assume that there are no ties in Q-values for any actions at a given state.

[4]Recall that the optimal policy is indexed with the time step as the environment might change in our setting, i.e., π_t^* denotes the optimal policy given the environmental conditions at time step t.

Theorem 1. *If* $\alpha\varepsilon + (1-\alpha)\delta \leq \frac{\psi_t}{2}$*, PA-MCTS is guaranteed to select the optimal action at time step* t*.*

Proof. We know that PA-MCTS is guaranteed to select the optimal action when the following inequality holds:

$$\alpha Q_0^{\pi_0^*}(s,a_t') + (1-\alpha)\overline{G}_t(s,a_t') \leq \alpha Q_0^{\pi_0^*}(s,a_t^*) + (1-\alpha)\overline{G}_t(s,a_t^*) \tag{6.3}$$

Given Eq. (6.1) and Definition 6.5.4, $Q_t^{\pi_t^*}(s,a)$ can be bounded with respect to the estimates $Q_0^{\pi_0^*}(s,a)$ and $\overline{G}_t(s,a)$:

$$\begin{aligned} Q_0^{\pi_0^*}(s,a) - \varepsilon \leq Q_t^{\pi_t^*}(s,a) \leq Q_0^{\pi_0^*}(s,a) + \varepsilon \\ \overline{G}_t(s,a) - \delta \leq Q_t^{\pi_t^*}(s,a) \leq \overline{G}_t(s,a) + \delta \end{aligned} \tag{6.4}$$

By substituting $Q_t^{\pi_t^*}(s,a)$ for $Q_0^{\pi_0^*}$ and $\overline{G}_t$ in Eq. (6.3) by using inequalities in Eq. (6.4), rearranging, and using Definition 6.5.3, we find that PA-MCTS chooses the optimal action when

$$\alpha\varepsilon + (1-\alpha)\delta \leq \frac{\psi_t}{2}$$

□

Theorem 1 essentially shows that PA-MCTS is guaranteed to choose the optimal action if the sum of α-weighted errors is low enough that the decision agent can differentiate between the best and next best action. While ψ_t is not observable at decision time, we can use Eq. (6.4) to find the relationship between ψ_t and ψ_0:

Corollary 1.1. $\psi_t \leq \psi_0 + 2\varepsilon$

Using Corollary 1.1, we substitute ψ_t in terms of ψ_0 (which can be computed at decision time using known $Q_0^{\pi_0^*}$ values) in Theorem 1. Then, we can determine when PA-MCTS will select the optimal action using information that can be computed at decision time:

Corollary 1.2. *If $\alpha\varepsilon + (1-\alpha)\delta \leq \frac{\psi_0}{2} + \varepsilon$, PA-MCTS is guaranteed to select the optimal action at decision epoch t.*

Using $0 \leq \alpha \leq 1$, Corollary 1.2 can be rearranged to solve for α:

Corollary 1.3. *PA-MCTS will choose the optimal one-step action if*

$$\begin{cases} \frac{-\delta}{\varepsilon-\delta} \leq \alpha \leq \frac{\psi_0}{2(\varepsilon-\delta)} + 1 & \text{if } \varepsilon > \delta \\ \frac{-\delta}{\varepsilon-\delta} \geq \alpha \geq \frac{\psi_0}{2(\varepsilon-\delta)} + 1 & \text{if } \varepsilon < \delta. \end{cases}$$

Therefore, given ε and δ, a decision agent can determine what values of the hyperparameter α (if any) would guarantee that PA-MCTS chooses the optimal action at decision time.

Theorem 1 guarantees the conditions under which PA-MCTS will choose the optimal action. We now seek to answer a more practical question: when does PA-MCTS choose a one-step action with a higher $Q_t^{\pi_t^*}$ value than MCTS or the learned policy in isolation? First, we list conditions under which PA-MCTS chooses a better action than pure MCTS:

Proposition 1. *If PA-MCTS and MCTS choose different actions, PA-MCTS's chosen action will have a higher $Q_t^{\pi_t^*}$ value than MCTS if $2\varepsilon \leq \zeta_t^m$, where*[5] *$\zeta_t^m := (Q_0^{\pi_0^*}(s,\tilde{a}) + \overline{G}_t(s,\tilde{a})) - (Q_0^{\pi_0^*}(s,a_m) + \overline{G}_t(s,a_m))$, $\tilde{a} := \arg\max_{a\in\mathscr{A}_s} \alpha Q_0^{\pi_0^*}(s,a) + (1-\alpha)\overline{G}_t(s,a)$, and $a_m := \arg\max_{a\in\mathscr{A}_s} \overline{G}_t(s,a)$.*

Proof. PA-MCTS's chosen action $\tilde{a}$ has a higher $Q_t^{\pi_t^*}$ value than MCTS's chosen action a_m when

$$0 \leq Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_m) \tag{6.5}$$

If PA-MCTS and MCTS choose different actions, then it must be that $\overline{G}_t(s,a_m) \geq \overline{G}_t(s,\tilde{a})$ and $\alpha Q_0^{\pi_0^*}(s,a_m) + (1-\alpha)\overline{G}_t(s,a_m) \leq \alpha Q_0^{\pi_0^*}(s,\tilde{a}) + (1-\alpha)\overline{G}_t(s,\tilde{a})$. Then, by using Eq. (6.4),

[5]the superscript m in ζ_t^m represents the bounds with respect to MCTS and is not an index.

we find that

$$\zeta_t^m - 2\varepsilon \leq Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_m)$$

Therefore, it follows from Eq. (6.5) that PA-MCTS chooses an action with higher $Q_t^{\pi_t^*}$ value when $0 < \zeta_t^m - 2\varepsilon$, i.e., $2\varepsilon \leq \zeta_t^m$. □

Since ζ_t^m is constructed from terms that can be computed at decision time, Proposition 1 can be used by the decision agent to determine if it should use PA-MCTS or MCTS when they choose different actions at decision time. A similar statement can be made regarding action selection using $Q_0^{\pi_0^*}$ values.

Proposition 2. *If the action chosen by PA-MCTS is different than* $\arg\max_a Q_0^{\pi_0^*}(s,a)$ *at state s, then PA-MCTS's chosen action will have a higher* $Q_t^{\pi_t^*}$ *value than selection through* $Q_0^{\pi_0^*}$*-values if* $2\delta \leq \zeta_t^q$ *where*[6] $\zeta_t^q := (Q_0^{\pi_0^*}(s,\tilde{a}) + \overline{G}_t(s,\tilde{a})) - (Q_0^{\pi_0^*}(s,a_g) + \overline{G}_t(s,a_g))$*,* $\tilde{a} := \arg\max_{a\in\mathscr{A}_s} \alpha Q_0^{\pi_0^*}(s,a) + (1-\alpha)\overline{G}_t(s,a)$*, and* $a_g := \arg\max_{a\in\mathscr{A}_s} Q_0^{\pi_0^*}(s,a)$*.*

We present the proof in the appendix. Using Propositions 1 and 2, we can determine the conditions under which PA-MCTS chooses an action that is better than either of its constituent policies.

The above propositions consider only one decision epoch. However, typically, training a new policy takes time, which might consist of several decision epochs. Therefore, we compute the total error in the expected return when following PA-MCTS compared to an optimal (updated) policy.

Theorem 2. *When PA-MCTS is used for sequential decision making, the maximum difference between the return from an optimal policy and the return from following PA-MCTS is at most* $\frac{2(\alpha\varepsilon - \alpha\delta + \delta)}{1-\gamma}$.

Proof. Let $\mathscr{A}_s'$ denote the set of actions that can be taken by PA-MCTS in state s at any decision epoch. Given ε, δ, and $\mathscr{A}_s'$, it must be that $\alpha Q_0^{\pi_0^*}(s,a) + (1-\alpha)\overline{G}_t(s,a) \geq$

[6] Again, the superscript q only denotes bounds with respect to the Q-values and is not an index.

$\alpha Q_0^{\pi_0^*}(s, a_t^*) + (1-\alpha)\overline{G}_t(s, a_t^*)$. Substituting $Q_t^{\pi_t^*}$, we get $2(\alpha\varepsilon + (1-\alpha)\delta) \geq V_t^{\pi_t^*}(s) - Q_t^{\pi_t^*}(s,a)$, where $V_t^{\pi_t^*}(s) = \arg\max_a Q_t^{\pi_t^*}(s,a)$ is the optimal value function. Therefore, $\mathscr{A}_s' = \{a \in \mathscr{A}_s \,|\, V_t^{\pi_t^*}(s) - Q_t^{\pi_t^*}(s,a) \leq 2(\alpha\varepsilon + (1-\alpha)\delta)\}$. Then, through proof by induction by using the Bellman operator [142], we find that

$$V_t^{\pi^*}(s) - V_t^{\pi_{\text{PA-MCTS}}}(s) \leq 2(\alpha\varepsilon - \alpha\delta + \delta)/(1-\gamma) \tag{6.6}$$

where $V^{\pi_{\text{PA-MCTS}}}(s)$ denotes the expected rewards by following PA-MCTS as an action selection policy from state s. The full proof is presented in the appendix. □

6.6 Experiments

To evaluate the efficacy of PA-MCTS, we first learn a policy π for the open-source OpenAI Gym CartPole-v1 environment [143], a version of the inverted pendulum control problem, using a double deep Q algorithm [144]. We vary the maximum length of each episode from 500 to 2500 time steps to evaluate all approaches. The environment parameters were set to default values for the stationary experiments, i.e., gravity $g = 9.8 m/s^2$, the cart's mass $m = 1.0 kg$, and a reward function of $R(s,a) = 1.0$ for each time step before reaching a terminal state. The double Q-learning agent's learning rate was set to 0.001, and a Boltzmann control policy was used. The agent learned for 300,000 steps. The details for the neural network and its implementation are presented in the appendix. We implemented PA-MCTS using UCT [97] as the tree policy. We use the following PA-MCTS hyperparameters in all experiments: the exploration-exploitation tradeoff parameter $c = 50$, the planning horizon is 500 time steps, and the decay rate $k = 0.0$. We perform a hyperparameter search on the discount factor γ for both RL and MCTS; as a result, we present undiscounted cumulative returns in the results. We implemented MCTS in the Python programming language [145]. Our complete implementation is available online[7]. We ran

[7]See https://anonymous.4open.science/r/NeurIPS_PA_MCTS-5595/

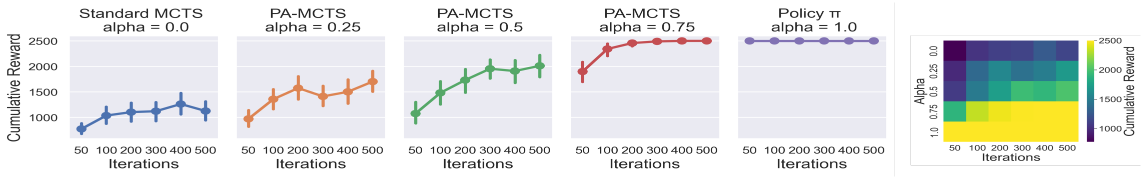

Figure 6.1: PA-MCTS results when applied to the default CartPole environment. Columns denote values for α. The individual plots' horizontal axis is the number of MCTS iterations per decision epoch. The vertical axis is the cumulative reward. We show the mean cumulative reward over 50 samples with 95% confidence intervals. The heatmap compactly presents the cumulative reward w.r.t. various combinations of α and the number of iterations.

experiments using the Chameleon testbed [146] on 4 Linux systems with 32–96 logical processors. A full description of the hardware used is presented in the appendix.

Stationary Environment: We begin by comparing PA-MCTS with a policy π learned through RL ($\alpha = 1.0$) and MCTS ($\alpha = 0.0$). We consider $\alpha \in \{0.25, 0.5, 0.75\}$ and vary the MCTS iteration budget ({50, 100, 200, 300, 400, 500}) to evaluate the convergence of each approach. The results are shown in Fig. 6.1. Our first observation is that the policy π ($\alpha = 1.0$) achieves the best possible return of 2500. This is expected, given a fixed environmental condition, the RL policy outperforms both MCTS and PC-MCTS. We also observe that PA-MCTS, by utilizing the learned policy, converges in far fewer iterations than standard MCTS (i.e. with $\alpha = 0.0$) for $\alpha \in \{0.25, 0.5, 0.75\}$, with $\alpha = 0.75$ converging to the optimal return at 300 iterations.

Non-Stationary Environments: We then introduce non-stationarity to the environment by modifying three environmental parameters: we change the gravitational constant g from the default value of 9.8 m/s^2 to the values {20.0, 30.0, 50.0, 500.0} m/s^2, the cart's mass m from the default of 1.0 kg to the values {2.0, 5.0, 10.0, 25.0} kg, and shrink the cart's track X_t from the default length of 2.4m to {2.0, 1.5, 1.0}m. We present results for the first two changes in Figs. 6.2 and 6.3 and for the last one in the appendix. Our first observation is that the learned policy's performance in isolation degrades significantly as hypothesized: with $g = 30$ m/s^2 and $\alpha = 1.0$, the mean return does not reach the optimal value, while for $g \in \{50.0, 500.0\}$ m/s^2, $m \in \{5.0, 10.0, 25.0 \text{ kg}\}$ kg, and $X_t \in \{1.5, 1.0\}$

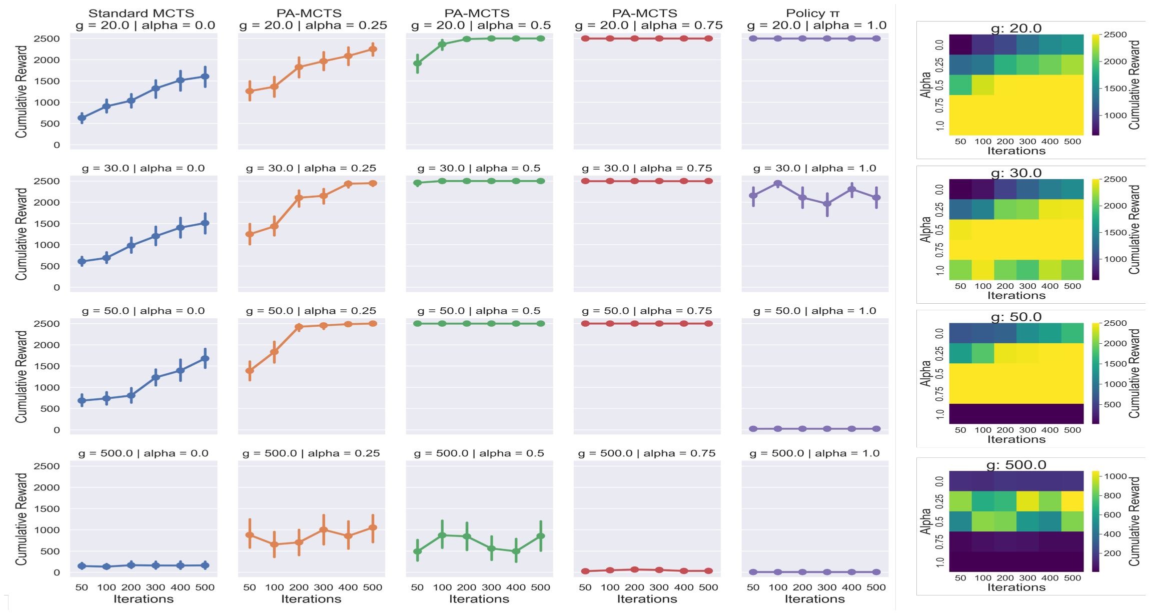
Figure 6.2: PA-MCTS results with gravity modified from its default value of 9.8 m/s^2. Rows represent different gravity values. Columns represent values for α. The individual point plots' horizontal axis is the number of MCTS iterations per decision epoch. The vertical axis is the cumulative reward. We show the mean cumulative reward over 50 samples with 95% confidence intervals. The heatmap compactly presents the cumulative reward w.r.t. various combinations of α and the number of iterations.

the policy obtains returns close to 0. We also observe that PA-MCTS again converges in significantly fewer iterations than standard MCTS in most cases, notably achieving the optimal return within 50 iterations for $g \in \{30.0 \text{ m/s}^2, 50.0 \text{ m/s}^2\}$ and $\alpha = 0.75$ in Fig. 6.2. There are a few exceptions, such as $m = 25.0$ kg in Fig. 6.3, where PA-MCTS does not significantly improve upon standard MCTS with any tested α. There are several possible explanations for this: we may not have chosen the optimal value for α, the environment may have shifted too much for the policy to be useful, or it may be that balancing the pole with an extremely heavy cart is too challenging given the default action space.

Modified Reward Function: Finally, we evaluate PA-MCTS with a modified reward function. In the standard cartpole environment, the agent receives a reward of 1 for each time-step before the episode is terminated. We modify the reward function to return $1 - (|x_p|/X_t)$, where x_p is the cart's x coordinate and X_t is the track's boundary. In the cartpole environment, the center of the track is at $x = 0$ and the boundaries are at $x = -X_t$ and

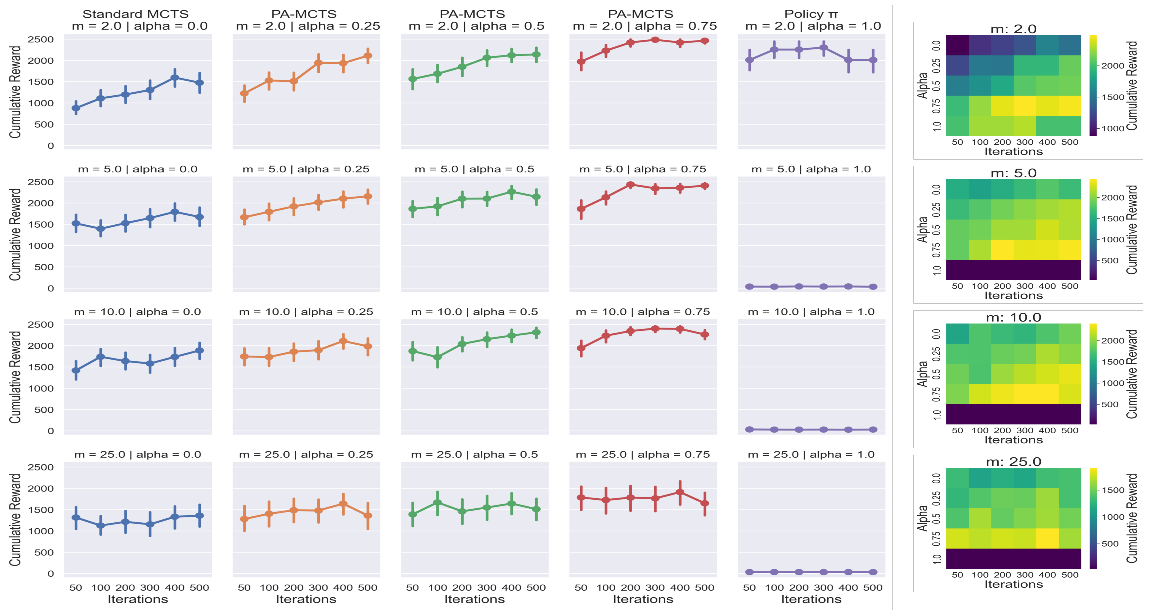

Figure 6.3: PA-MCTS results with the cart's mass modified from its default value of 1.0 kg. Rows represent different cart masses. Columns represent values for α. The individual point plots' horizontal axis is the number of MCTS iterations per decision epoch. The vertical axis is the cumulative reward. We show the mean cumulative reward over 50 samples with 95% confidence intervals. The heatmap compactly presents the cumulative reward w.r.t. various combinations of α and the number of iterations.

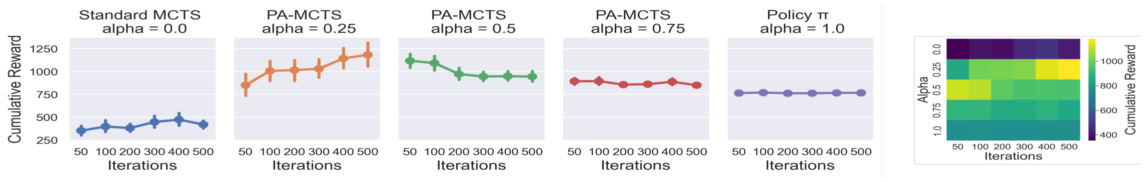

Figure 6.4: PA-MCTS results with a modified reward function which incentivizes staying near the center of the track. The setting is the same as Fig. 6.2 and Fig. 6.3.

$x = X_t$. Therefore, this new reward function provides higher rewards the closer the cart is to the center of the track. The results with this reward function are shown in Fig. 6.4. We again observe that PA-MCTS obtains higher returns than the policy while converging in fewer iterations than standard MCTS.

Limitations: We point out that our approach focuses on settings where the environmental changes *can be* detected. While this is the case in practice for many problem settings [84], there might be problems where this is not the case. Also, we need to design practical "detectors" to gauge the values of ε and δ at decision time. We plan to explore these problems in future work.

6.7 Conclusion

We explore sequential decision-making in non-stationary environments, where the decision-maker faces the dilemma of choosing between accurate but obsolete state-action values that are available from learning approaches and inaccurate state-action values (given limited computation time) from an online search algorithm. We present a novel approach, *Policy Augmented MCTS* (PA-MCTS), that combines the strengths of reinforcement learning and online planning in non-stationary environments. We present theoretical results characterizing the hyper-parameter values that are better than the baselines (i.e., pure learning or MCTS), demonstrating that there is a range of values that will work well in practice. We also present bounds on the error accrued by following PA-MCTS as a policy for sequential decision-making. Through extensive experiments using the classical cart-pole environment, we show that PA-MCTS results in higher cumulative rewards than an RL agent in isolation under environmental shifts while converging to better solutions in significantly fewer iterations than pure MCTS. Our complete implementation is available online.

Chapter 7

Scalable Heterogeneous Demand Prediction

7.1 Overview

The previous chapters present scalable decision-making approaches that use online planning algorithms. The foundation of these planning algorithms are the models they use to forecast how the environment will evolve and estimate the long-term value of actions. Modeling SCPS environments is difficult due to the challenges of heterogeneity and non-stationarity. To be useful for proactive decision-making, these models must also have high spatio-temporal resolution, so that a decision algorithm can have a precise understanding of the locations and times where events are likely to occur. Unfortunately, increasing the resolution of models introduces *sparsity*, meaning that there are significant class imbalances between positive and negative samples in the data, since each spatiotemporal window is less likely to see any significant samples.

This chapter presents a generative modeling approach that enables event forecasting over large, heterogeneous geographic areas using mixed-typed features (i.e., categorical and numeric features). It uses the Similarity Based Agglomerative Clustering (SBAC) algorithm to find groups of similar events spread across the spatial area. These groups of similar events also tend to have similar arrival distributions, which makes forecasting for each group more accurate. These groups are then mapped to spatial locations to achieve models with high spatial-temporal resolution.

The work comprising this chapter has been published in the Proceedings of the 3rd IEEE International Conference on Smart City Innovations [31].

- G. Pettet, S. Nannapaneni, B. Stadnick, A. Dubey, and G. Biswas (2017). "Incident analysis and prediction using clustering and Bayesian network," in *Proceedings of* *the 3rd IEEE International Conference on Smart City Innovations, SCI 2017, San Francisco, CA, USA,* pp. 1-8.

7.2 Introduction

Emerging Trends: The advancement in sensors and information transfer technologies provide opportunities to collect large amounts of data on complex operations and processes that govern various aspects of city life. In such situations, where it is an almost intractable task to build complex models, data driven approaches can produce more informed solutions to problems than using heuristics and ad-hoc approaches. Such data driven approaches have been successfully applied in a number of areas, ranging from monitoring industrial processes [147] to identifying students at risk of emotional disorders [148]. Today, advances in data collection (such as wireless sensor networks [149]) and storage infrastructure (such as distributed hash rings [150]) have allowed information to be collected and analyzed for applications not possible before, such as urban analytics [151] and emergency response services.

In this paper, we describe a toolchain that will enable fire departments to analyze multiple, distributed incident occurrences that they must respond to. Our goal is to analyze historical incident data and develop predictive models that can help the department efficiently allocate and route emergency vehicles to incidents as they occur over a large, distributed area. Minimizing response times increases victim survival rates [67] and frees vehicles to respond to other incidents more quickly. Any such optimal dispatch algorithm is typically based on a sequential optimization that requires prediction of the likelihood of future incidents occurring in a given area, so it can plan ahead.

Incident prediction using the negative binomial distribution [20], artificial neural networks [21], and hierarchical analysis [22] have been used to great effect when attempting to predict incident frequency for specific areas, and have helped determine features of roadways that affect incident occurrence. Prediction methods such as the negative binomial regression [23] and random effect probit models [19] have also been used to analyze feature effects on accident frequency, and generating predictive models for specific areas. Unfortunately, these studies generally make assumptions about the locations that they are analyzing. For example, they study a specific length of freeway, or look at only intersections and their features in a specific city. To create a predictive model for an entire metro area however, location agnostic features such as weather and time must be considered for the incidents.

Contributions: This paper describes a toolchain to forecast the likelihood of incidents, specially motor vehicle incidents, occurring in a large, geographical area. This paper describes major components of our prediction and analysis toolchain:

1. An unsupervised clustering approach for grouping incidents with similar characteristics. We hypothesize that incidents within each group will have similar arrival times, making forecasting for each group more accurate. This is validated by our results described in Table 7.4: the average log-likelihood of cluster survival model accuracy is significantly higher (-16,100.8) than models built from the entire dataset (-180,243.8).
2. Predictive models for each cluster using survival analysis.
3. A mapping of these cluster predictive models to spacial locations using a Bayesian network.
4. Compose all of the data preprocessing, analysis, and prediction routines in the form of a toolchain to facilitate analysis of data received from the Nashville fire department from February 2014 to February 2016, and then validate our toolchain on data from February 2016 to December 2016. This toolchain will facilitate future emergency vehicle dispatch optimization analysis.

Paper Outline Section 7.3 presents prior work on incident prediction. Section 7.4.1 describes the data used in our case study for Nashville Motor Vehicle Accident (MVA) response dispatching. In section 7.4.2 we formally describe the problem specification and then detail each step in the prediction toolchain, and simultaneously present the results of the case study. Section 7.5 presents a discussion, and Section 7.6 presents the conclusions of the paper.

7.3 Related Research

Vehicle accident analysis has been an important area of research due to their large safety and monetary costs and their impact on human life. Miaou and Lum found that due to over dispersion often present in accident data, the more general Negative Binomial distribution is often superior to the Poisson regression in this area [20]. Abdel-Aty and Radwan applied the negative binomial technique to model accident frequency for a principal arterial in Central Florida [152]. Using data from 1606 accidents over three years, they found eight features to be significant in determining the frequency of accidents, including segment length, shoulder width, and annual average daily traffic (AADT). Ackaah and Salifu used negative binomial regression to model 76 rural highways in Ghana [23]. They found that the negative binomial distribution fit their data reasonably well, and that five features (including traffic flow and road segment length) were significant in determining accident frequency.

Chin and Quddus built on the research by suggesting that the random effect negative binomial (RENB) model is superior to the negative binomial model [153]. They reasoned that the negative binomial's assumption that accident data is uncorrelated in time is inappropriate for accidents, due to serial correlation in the accident data. The RENB model accounts for temporal effects by treating the data in a time-series cross-section panel. Their model, based on signalized intersections in Singapore, found that eleven features had a significant impact on accident frequency, and improved on the model found with the negative binomial model.

Chang explored artificial neural networks (ANN) as another alternative to the negative binomial regression model [21]. The negative binomial model assumes a predefined underlying relationship between the dependent and independent variables. If this assumption is violated, it will lead to erroneous accident estimations. ANN avoids this by making no assumptions regarding the variable's relationship. Raut and Karmore have combined an Artificial Neural Network to analysis a large amount of input data with a fuzzy logic system that uses this data to predict accident severity [154]. Unfortunately this technique requires large amounts of external traffic information that may not be available to dispatch services.

Researchers have also explored applying various classification methods to traffic problems. For example, Moreira-Matias [155] uses boosted decision trees to classify traffic jam events. Unfortunately these methods do not apply to our problem, since we are looking for the probability that an accident will occur in each location, not to classify accidents to a location.

Survival analysis has also been widely used and is the one of the core components of our toolchain. This analysis has been applied by researchers to predict accident duration. Chung applied the log-logistic accelerated failure time model to 2 years of Korean freeway accident duration data [156]. Using eight features to characterize accidents, they found that the model provided reasonable duration prediction based on the mean absolute percentage error scale. Kang and Fang (2011) similarly applied the Weibull prediction model to 3 years of Jiaxing city's freeway incident data to find predictive factors affecting incident duration, and found the model to be reasonable.

The research presented thus far makes strong assumptions about the location and/or type of accident during analysis. For example, they might only look at accidents occurring at intersections or along a single stretch of highway. While these assumptions are helpful for small scale or specific analysis, they make generalizing the results difficult. In the past, our group has studied the accident prediction problem, focusing on spatial grids [157]. However, in this paper we explore a generalized, yet unsupervised approach to categorizing incidents. Therefore, the method presented in this paper clusters on *individual incidents* rather than *grid sections*. This leads to tighter, more meaningful clusters for dispatch allocation, which is shown by our improved likelihood results shown in the Table 7.4. This is helpful since an important aspect of emergency response is ensuring that appropriate equipment is dispatched based on the incident type.

7.4 Our Approach

7.4.1 Data Specification

The majority of the data used in this study was provided by the Metro Nashville Fire Department in the form of a relational database scrubbed to eliminate personally identifiable information. The database contained approximately two years of incidents occurring from February 2014 to February 2016. In total, there were 477,837 unique incidents recorded in the database, the majority of which occurred in Davidson County, Nashville, Tennessee.

Motor vehicle accidents were extracted from the database according to the following criteria: the location of the incident was fully specified by GPS coordinates, the incident occurrence time was known, the first unit arrival time that occurred after the incident occurrence time was also known, and the incident was classified by emergency medical dispatch card numbers starting with '29' to ensure it was a motor vehicle accident. Using these criteria 19,910 motor vehicle accidents were extracted for the clustering algorithm.

In addition to the information obtained from the database, weather condition information and information regarding the type of road on which the accident occurred was obtained from DarkSky and OpenStreetMaps, respectively. Using these three sources of information, the features described in the clustering analysis subsection 7.4.3 were extracted for clustering.

7.4.2 Overview of the Approach

The toolchain described in this paper consists of 4 major components as shown in figure 7.1: data preprocessing, clustering incidents (modeled in figure 7.2), learning survival models for each incident cluster, and mapping these predictive models to locations (Both

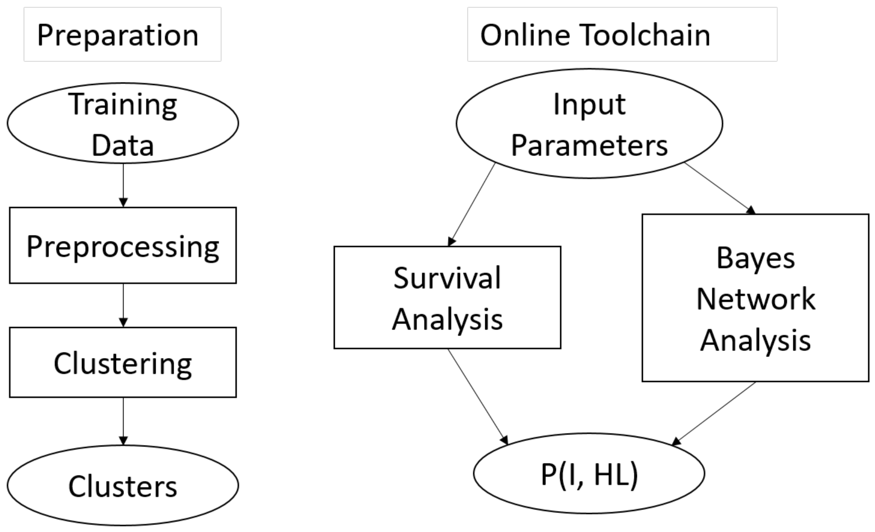

Figure 7.1: Toolchain Block Diagram. P(I, HL) refers to the joint probability of an incident occurring in a particular hex cell

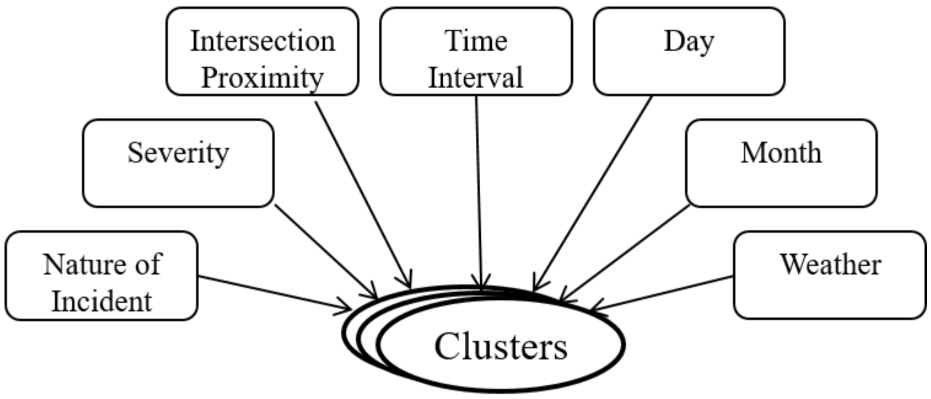

Figure 7.2: Cluster Generation Model

shown in figure 7.3). Each step is explained in detail in the following sections.

For prediction, we assume that the city is divided into a grid of regularly sized half mile hexagonal sections, which are referred to as **hex cells** in the remainder of the paper. For each hex cell, we have data regarding incidents that took place in that grid in the given period. We also assume that the incidents in each cell are independent of incidents in other cells. This information enables us to train the Bayesian network that we describe later in the paper.

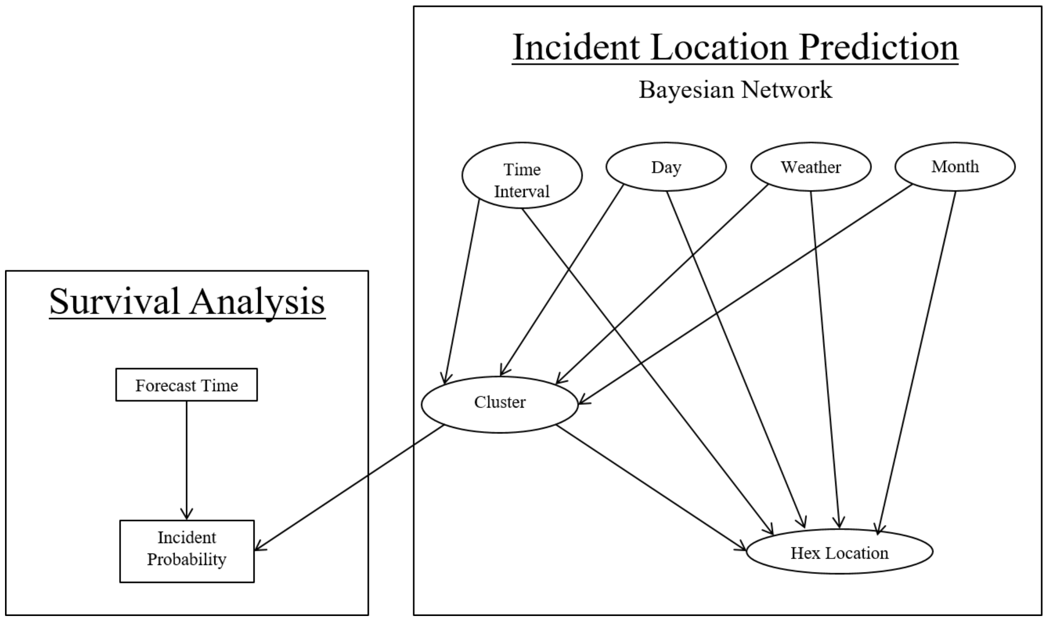

Figure 7.3: Prediction Toolchain Model

7.4.3 Clustering Analysis

The first step of the toolchain is clustering the incidents into similar groups. There are three primary steps to this process: (1) choosing incident features to cluster on, (2) calculating the similarity values between each pair of incidents and (3) running a hierarchical clustering algorithm, SBAC [158] to generate a dendrogram, and then establishing a minimum distance criterion as separation among clustering, and therefore, establishing the number of clusters that make up the dataset. forming a dendrogram from these values, and cutting that dendrogram to form the optimal number of clusters.

Incident Feature Selection: Table 7.1 describes the features we chose to categorize the incidents. We discretized the continuous values of time and the distance to the nearest intersection to reflect trends in the data.

Similarity Calculation: Once features of interest are chosen for the incidents, they are used to calculate a similarity measure between incident pairs. Traditional clustering methodologies generally focus on either numeric valued data [159] (k-means clustering [160], for example) or nominal valued data (known as conceptual clustering [161]), but are not designed for mixed numeric and nominal data. While the features in our case study are

Table 7.1: Incident Features Considered

Feature	Description	Source
Road type	Type of road incident took place on, such as freeway or primary	Obtained from OpenStreetMaps based on the GPS coordinates and street address
Weather	Weather conditions at the time of the incident	Obtained from DarkSky based off GPS coordinates and time of incident
Severity	Severity measure based off Fire Department codes, with 'A' as least severe to 'D' being most severe	Obtained from the incident's associated emergency medical dispatch card number
Nature of Accident	Description of incident	Obtained from emergency medical dispatch card number
Time of Day	time in which incident occurred: early morning, late morning, afternoon, or night	Obtained from Nashville fire department data
Day	The day of the week the incident occurred on	Obtained from Nashville fire department data
Month	The month in which the incident occurred	Obtained from Nashville fire department data
Intersection Proximity	How close the incident occurred to an intersection: On, near, or far from the intersection	Obtained from OpenStreetMaps based on the GPS coordinates of the incident

all nominal, other applications of this toolchain may require some numeric features to be considered. For this reason, we use a similarity measure that works well with mixed typed data.

Specifically, we use a similarity measure created by Li and Biswas that has been shown to work well for mixed data types [158]. Their method, which is a generalized version of a measure proposed by Goodall for biological taxonomy [162], uses unusual characteristics shared by data objects to determine their similarity. Specifically, "a pair of objects *(i,j)* is considered more similar than a second pair of objects *(l,m)* if *i* and *j* exhibit a greater match in feature values that are less common in the population. In other words, similarity among objects is decided by the uncommonality of their feature value matches" [158]. This helps create tight clusters likely to share unique feature values.

To demonstrate how to calculate the similarity of a pair of objects using this method, it is helpful to use a toy example. Consider the objects described in Table 7.2: there are 6 ball objects, each of which have a color (a nominal feature) and a weight (a numerical feature).

To calculate the similarity of a pair of objects, the first step is to determine the individual feature similarities - the color and weight, in this case. The technique used to calculate this

Table 7.2: Toy Example for Similarity Calculation

Ball ID	**Color (Nominal)**	**Weight (Numeric)**
Ball 1	Red	15.0 kg
Ball 2	Red	10.0 kg
Ball 3	Red	10.0 kg
Ball 4	Yellow	10.0 kg
Ball 5	Yellow	7.5 kg
Ball 6	Blue	5.0 kg

depends on if the feature is nominal or numeric. These individual feature similarities are then combined into a total similarity for the pair of objects. The techniques for each of these steps are described below.

Nominal Feature Similarity: Let us first consider nominal feature similarity. There are two cases for nominal features: the two objects have either the same feature value, or different feature values. If the two objects' feature values are not equivalent their similarity score is 0 (i.e. they are not similar at all), but if the values are the same then the similarity score is somewhere between 0 and 1. Applied to our toy example, Balls 1 and 4 have 0 similarity for the Color feature since red and yellow are different colors, and there is no way to relate the two. Balls 1 and 2, on the other hand, have some non-zero similarity for Color since they are both red.

The exact feature similarity score when the two feature values are equivalent is a function of that value's uncommonality in the population: the more common a feature value, the less similar any pairs with that value are considered to be. For example, yellow balls are considered more similar than red balls in our toy dataset, since there are more red balls.

This information is used to create the object pair's *More Similar Feature Value Set (MSFVS(*$(v_i)_k$*)*, which is the set of all values for nominal feature k that are equally or more similar than the object pair's feature value. For example, the *MSFVS* for balls 1 and 2 would be {Red, Yellow} since yellow is rarer than red (blue is omitted, as there are no pairs with that color). The *MSFVS* for balls 4 and 5, however, would only be {yellow}, since yellow is the rarest color represented by a pair of ball objects.

Equation 7.1 shows how the *MSFVS* is used to calculate the similarity score between the two objects:

$$(S_{ii})_k = 1 - (D_{ii})_k = 1 - \sum_{l \in MSFVS((V_i)_k)} (p_l)_k^2 \tag{7.1}$$

where $(p_l)_k^2)$ is the probability of picking a pair $((V_i)_k, (V_i)_k) \in MSFVS((V_i)_k)$ for feature k at random, and $(D_{ii})_k$ is the dissimilarity of the objects.

Let's apply this to balls 1 and 2 in our example. As discussed earlier, the balls both have the Color red, and their MSFVS is $\{$Red, Yellow$\}$. Given this set, the similarity for the Color value of red is $S_{(ball1,ball2)})_{Red} = 1 - (D_{(ball1,ball2)})_{Red} = 1 - (p_{red}^2 + p_{yellow}^2) = 0.733$. This means that the color red contributes 0.733 to the similarity score between balls 1 and 2.

Numeric Feature Similarity: The method for calculating the similarity for numeric features is slightly different from nominal features. When feature values are not equivalent, their similarity is determined in the traditional manner: one pair of objects is more similar than another if their feature values are closer together. The weights of balls 5 and 6 are considered more similar than 4 and 6, for example, since the difference in their weights is smaller. It is only when two pairs of values have equivalent differences that the uniqueness of the values is considered.

This information is used to create a pair of value's *More Similar Feature Segment Set (MSFSS$((V_i)_k, (V_j)_k)$)*, which includes all pairs of values that are more similar due to the criteria described above.

Similar to nominal features, this is used to calculate the feature value similarity as follows: the probability of picking two objects from the population having values $(V_l)_k$ and $(V_m)_k$ for feature k where $((V_l)_k, (V_m)_k) \in MSFSS((V_i)_k, (V_j)_k)$ is

$$\alpha_{lm} = \begin{cases} 2(p_l)_k(p_m)_k = \frac{2(f_l)_k(f_m)_k}{n(n-1)}, (p_l)_k \neq (p_m)_k \\ (p_l)_k(p_m)_k = \frac{(f_l)_k((f_l)_k-1)}{n(n-1)}, (p_l)_k = (p_m)_k \end{cases} \tag{7.2}$$

where f_l and f_m are the frequency of values $(V_l)_k$ and $(V_m)_k$ and n is the total number of objects in the population. The similarity is then computed as in equation 7.3:

$$(S_{ij})_k = 1 - (D_{ij})_k = 1 - \sum_{l,m \in MSFSS((V_i)_k,(V_j)_k)} \sigma_{lm} \tag{7.3}$$

where σ_{lm} is the appropriate probability distribution from equation 7.2 and $(D_{ij})_k$ is the dissimilarities of the two objects. For example, take balls 1 and 2 again. The MSFSS for their value segment (10.0, 15.0) is {(5.0, 7.5), (7.5, 10.0), (10.0, 10.0), (10.0, 15.0)}. Notice that even though balls 6 and 4's values (5.0, 10.0) are as close together as (10.0, 15.0), they are not included in the set. This is because there are more balls with weights in the range [5.0, 10.0] then [10.0, 15.0], making the weights of balls 5 and 6 more unique (and therefore more similar).

This makes the ball's similarity
$(S_{(b1,b2)})_w = 1 - (D_{(b1,b2)})_w = 1 - (2p_{5.0}p_{7.5} + 2p_{7.5}p_{10.0} + p_{10.0}p_{10.0} + 2p_{10.0}p_{15.0}) = 0.333$.
This means that the weight feature contributes 0.333 to balls 1 and 2's similarity.

Total Object Similarity Aggregation: To aggregate these individual feature similarities for a pair of objects into the total similarities, we apply Fisher's χ^2 transformation [163] to numeric features, assuming that individual results are expressed as the square of a standard normal deviate. Continuing our simple example with balls 1 and 2, their aggregate numeric similarity would be $(\chi_c)^2_{(Ball1,Ball2)} = -2(ln(0.667) = 0.8109$.

For nominal features, Lancaster's mean value χ^2 transformation [164] is applied, as the traditional Fisher's transformation has been shown to cause deviations in the mean and standard deviation when applied to features with a small number of possible observations [164]. The nominal aggregate for balls 1 and 2 is therefore $(\chi_d)^2_{(Ball1,Ball2)} = 2(1 - \frac{(0.267*ln(0.267))-0}{0.267-0}) = 4.641$

These two χ^2 distributions can be added to determine the aggregate χ^2_{agg} distribution. The significance value of this distribution gives the aggregate dissimilarity of the two objects, which can be looked up in standard tables. This makes the final dissimilarity for our example balls 1 and 2 approximately 0.1, giving them a similarity of 0.9 in this population.

The result of performing this analysis on each pair of objects is a *Dissimilarity Matrix* defining the similarity relation between each pair. We then use these dissimilarities to perform *agglomerative hierarchical clustering*. Hierarchical clustering algorithms construct a *dendrogram*, which is a hierarchy of possible clusters. In agglomerative clustering, these groups are built from the bottom up: each data point starts in its own leaf group. During each iteration of the algorithm, the most similar pair of groups are merged into one group at the next level. This constructs a tree from the bottom up, and continues until there is only one root group containing all incidents [165]. We now determine the optimal level to cut this dendrogram.

Establishing the number of Clusters: To determine the optimal level to cut the dendrogram we score each possible clustering with a weighted silhouette value. Silhouette analysis compares each incident's similarity with its assigned cluster to its similarity to the next most similar cluster [166]. Formally, for each object *i*, let *a(i)* be the average dissimilarity of *i* with all data within the same cluster, and *b(i)* the lowest average dissimilarity of *i* to any cluster of which *i* isn't a member. The silhouette value of *i* is:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{7.4}$$

which produces silhouette values in the range $-1 \leq s(i) \leq 1$, where a high value indicates that the incident is well matched with its assigned cluster, while a low value indicates it is more similar to objects in its neighboring cluster. Finding the average silhouette score across all objects shows how well the objects are clustered in general:

$$\frac{1}{n} \sum_{i \in dataObjects} s(i), \tag{7.5}$$

where n is the total number of objects being clustered. To lower the complexity of our

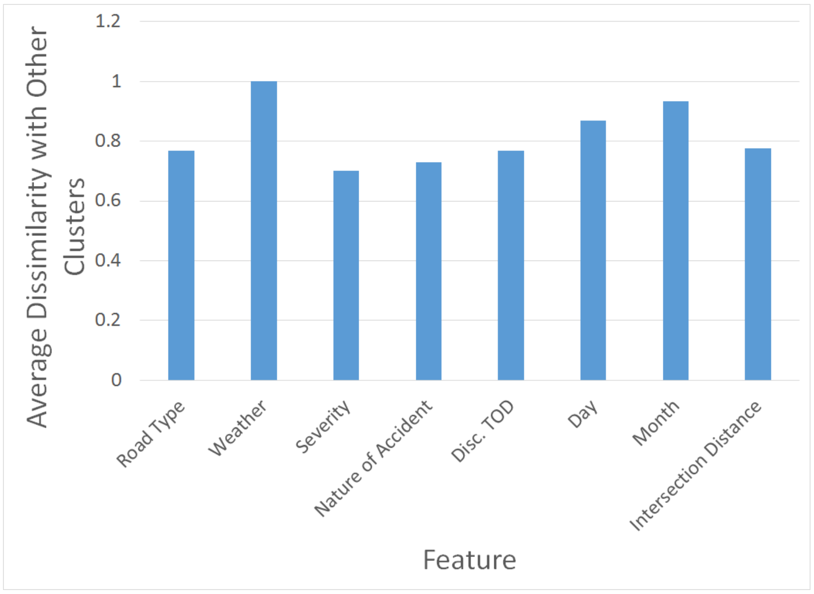

Figure 7.4: Cluster 7 - Average Feature Dissimilarity

groupings, we augment traditional silhouette analysis with a complexity weight, favoring cuts with fewer clusters:

$$w \cdot m + (\frac{1}{n} \sum_{i \in dataObjects} s(i)), \tag{7.6}$$

where the weight w is multiplied by the number of clusters in the current cut m.

Applying this complexity-weighted silhouette scoring to groupings produced by the dendrogram helps us find the optimal number of clusters. *By applying this to our data, we found the optimal number of clusters to be 13*.

Individual Cluster Analysis: Each of the clusters can be analyzed to determine the characteristics found to be unique to that cluster by the Similarity Based Agglomerative Clustering (SBAC) algorithm described earlier. For some clusters, this characterization is simple to visualize using feature dissimilarity - the more dissimilar a feature, the more unique it is.

Take our 7th cluster, for example. When examining the average feature dissimilarities for the cluster in figure 7.4, we see that the weather values are most unique. The weather represented in figure 7.5 shows this cluster is dominated by incidents that occurred in snowy

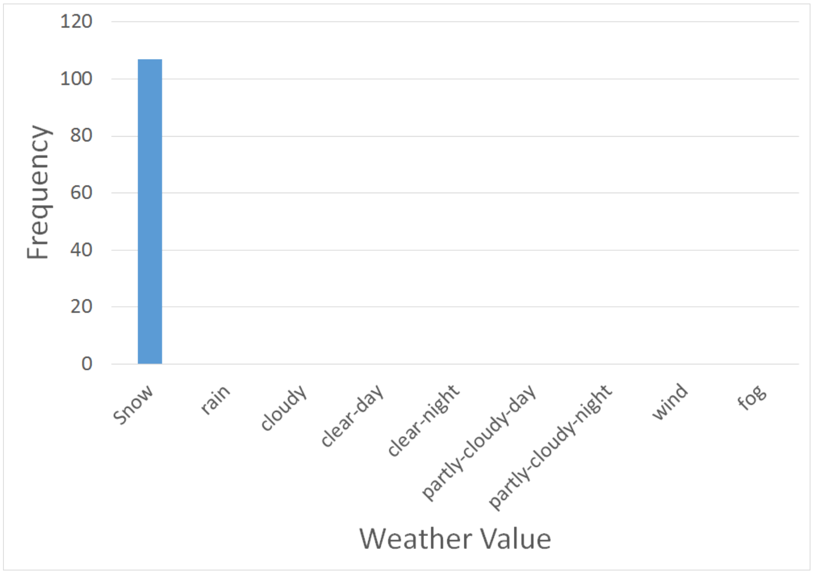

Figure 7.5: Cluster 7 - Weather Feature Values

conditions. This coincides with our goal, as it makes sense that incidents occur at a different rate in snowy weather.

7.4.4 Survival Analysis per Cluster

The next step of the toolchain is to create predictive models for each group. A group of statistical methods that are particularly well suited to this is survival analysis, which analyze if and when an event of interest is likely to take place [167]. More precisely, "the analysis of data that correspond to the time from a well-defined *time origin* until the occurrence of some particular event or *end-point*" [168]. In this case, the *time origin* is the current time, and the event whose occurrence we are interested in is an incident. In particular, we use an accelerated failure time model, which regresses the logarithm of the survival time over the covariates [169].

Formally, the survival function is defined as $S(t) = 1 - F_t(t)$ where $F_t(t)$ is the cumulative distribution function of the arrival time variable T. To model our survival function, we use an exponential distribution for our regression due to its memoryless property: the predicted time to the next event does not depend on the elapsed time since the last event [170]. We hypothesize that an incident's arrival time does not generally depend on any past

Table 7.3: Comparison of Cluster vs. Non-clustered Prediction

Cluster	**Log-Likelihood**
1	-89,780.2
2	-52,030.0
3	-5,041.5
4	-15,694.4
5	-11,912.6
6	-22,706.9
7	-1,788.1
8	-719.7
9	-957.5
10	-763.5
11	-164.4
12	-316.9
13	-7,434.8

incidents, making this property desirable and used in other motor incident studies [171].

We applied this regression analysis to each cluster's incident data to learn their predictive models. For comparison, we also applied predictive models to the entire dataset: the same survival analysis as well as the popular negative binomial analysis discussed earlier in the paper. We compare the model's accuracy using the log-likelihood scale: the likelihood of a model is the probability of some observed values (the past data, in this case) occurring given said model and its parameters. The natural logarithm of this likelihood is known as log likelihood, and easier to handle mathematically. By comparing two model's log likelihoods, we can determine which model better fits the historical data (as it will have a high log-likelihood value).

The Log likelihoods of each cluster's survival models are shown in table 7.3, while the comparisons are shown in table 7.4. The average log likelihood for the 13 clusters was **-16,100.8**, compared to the values of **-180,243.8** and **-178,488.9** of the survival model and negative binomial model applied to the entire dataset, respectively. As we are attempting to maximize log likelihood, our clusters' models showed to be an order of magnitude more accurate than models applied to the entire dataset. This reinforces our hypothesis that similar incidents have similar arrival rates.

7.4.5 Bayesian Network Analysis for Associating Clusters with Hex Cells

The last step to the prediction toolchain is using the newly created survival models for each cluster to determine the likelihood of an incident occurring in a particular hex cell. To be able to predict the most likely hex cell, we first learn the distribution of incidents pertaining to each cluster across the cells conditioned on the features shown in Fig. 7.3 (Time interval, Day, Weather and Month). As these features are categorical in nature, we learn the conditional probability distribution of hex cells, represented as $P(HL|T,D,W,M,C)$ for every combination of the incident features. Here, HL,T,D,W,M,C refer to Hex Location, Time Interval, Day, Weather, Month and Cluster respectively. In the Nashville Fire Incident dataset, the number of levels (distinct possible values) for these incident features are 4, 7, 10 and 12 respectively, totaling to 3360 distinct combinations. After learning the conditional relationships in the Bayesian network (Fig. 7.3), we use it to find the probabilities of incidents in each location. The systematic procedure for incident procedure is given below.

1. Cluster Identification: The probability of choosing a particular cluster $P(C|T,D,W,M)$ is based on current time and environment parameters: for example, cluster 7 described in section 7.4.3 would become much more likely in snowy weather than clear conditions.
2. Survival probability: We then determine the probability of an incident occurring of this type via the cluster's survival model $P(I|C,t)$. If the probability is below a threshold, we ignore the cluster.
3. Region Identification: We then calculate the likelihood that the incident occurs in a hex cell $P(I,HC|T,D,W,M,C,t)$ (Eq. 7.7) using the learned conditional distributions of cells and the cluster-specific survival models.

$$\begin{aligned} P(I,HL|T,D,W,M,C,t) = {} & P(HL|T,D,W,M,C,t,I) \\ & \times P(I|T,D,W,M,C,t) \end{aligned} \tag{7.7}$$

Eq. 7.7 can be further simplified to Eq. 7.8 since we know which incident features

Table 7.4: Comparison of Cluster vs. Non-clustered Prediction

Method	**Log-Likelihood**
Average of Cluster Survival Models	-16,100.8
Entire Dataset - Survival Model	-180,243.8
Entire Dataset - Neg. Binomial	-178,488.9

affect the hex cells and incident probability.

$$\begin{aligned} P(I,HL|T,D,W,M,C,t) = P(HL|T,D,W,M,C) \\ \times P(I|C,t) \end{aligned} \tag{7.8}$$

Since we are predicting incidents at a future time, the weather at a future time may not be known precisely. In such cases, we predict the incident probabilities by summing over all possible weather conditions as given in Eq. 7.9.

$$\begin{aligned} P(I,HL|T,D,M,C,t) = \sum_{W} P(I,HL|T,D,W,M,C,t) \\ \times P(W|T,D,M,C,t) \end{aligned} \tag{7.9}$$

In Eq. 7.9, $P(W|T,D,M,C,t)$ represents the prior probability of the weather conditioned on Time, Day, Month, Cluster and prediction time. Since weather is independent of cluster type and prediction time, $P(W|T,D,M,C,t)$ can further be simplified to $P(W|T,D,M)$. The prior probability of weather can be obtained from historical weather data sets or any available weather prediction models. We used the same DarkSky database that provided incident weather data.

7.5 Discussion

The final prediction toolchain consists of two major components: the survival models for each cluster and the Bayesian network mapping cluster probabilities to the hex cells. We can validate each of these separately to show the correctness of the toolchain. We have already demonstrated the accuracy of the survival models: the likelihood analysis displayed in tables 7.3 and 7.4 shows that the survival models for each cluster match the incident data better than the popular negative binomial method [152]. This establishes that the survival models accurately represent the arrival times of accidents for each cluster.

To determine the accuracy of the Bayesian network analysis at predicting the distribution of incidents we compared its results to a validation set. This validation set consists of approximately 10 months of recent Nashville incident data, ranging from February 6th to December 23rd 2016. We ran the Bayesian analysis over this range of dates, and then compared its predicted accident distribution across hex locations against the actual distribution of the validation data. The results are presented in the figure 7.6. The bars represent the difference between the predicted probability and the actual probability for incidents in each hex cell. With a few exceptions, most cell's predicted incident probability is within 2% of the actual probability. There are a few cells that have slightly worse prediction performance, with the difference generally being less than 10%. These inaccuracies could be due to properties changing in the hex cells (increasing population density, for example), or may be due to having only two years of training data: future data should increase this accuracy. The normalized root mean squared error of the predicted distribution was **1.656425**, which shows that overall the predicted results match the validation data well.

7.5.1 Using the Toolchain

Now that we have demonstrated the accuracy of the toolchain components, we will discuss the toolchain's use. Since the toolchain is split into two major components, its use is also split into these two functions.

The first step for a user is inputting their current environmental factors (the current weather and time) and how much time they want to look into the future. The survival models are then consulted according to the analysis time. This determines the likelihood that an incident of each cluster will occur *at any location* within the given time. Then the Bayesian Network analysis is run using the input parameters to determine where an incident is likely to happen. By using these two components together, we can predict

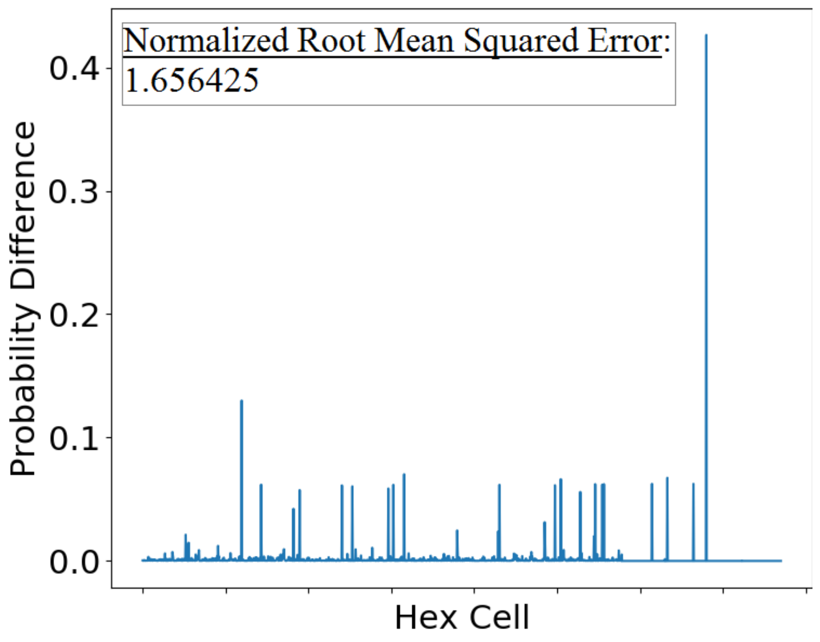

Figure 7.6: Error in Predicted Hex Incident Probabilities vs. Validation Data

the likelihood of a next incident at a given time and location. Note that both of these components are necessary to use the toolchain. The survival likelihood for each incident of each cluster describes how likely each type of incident is, while the Bayesian analysis shows the probable distribution of any incident that might occur across the hex cells.

Example analysis from the trained toolchain: In table 7.5 we show the predicted accident distribution starting at 15:00 on March 23rd, on a Thursday, in clear weather, with an analysis time of 2 hours. The survival models indicate that there are a few clusters that have over a 50% likelihood of having an accident during this time segment. The Bayesian analysis then shows that cells 3523 and 4140 are tied for the most likely cells for an incident to take place in given the parameters, with cells 5491, 4703, and 4699 following close behind.

For comparison we provide the same analysis but during rainy weather in table 7.6. Because the analysis time is the same, the survival models for each cluster give the same accident probabilities. When determining which clusters are more likely, however, the Bayesian analysis considers the rainy weather. This changes the most likely cells to 3513, 4332, 5290, 4803, and 3587. These cells have a higher probability than in the clear weather

Table 7.5: Prediction Ran with following Properties: Weather='Clear-Day', Day='Thursday', Month='March', Date='23rd', StartTime='15:00', AnalysisTime='2 hours'

Survival Models

Rank	**Cluster**	**Incident Likelihood**
1	1	0.6554
2	13	0.6294
3	7	0.49461
4	2	0.4448
5	6	0.2114

Hex Mapping

Rank	**HexCell**	**Hex Probability**
1	3523	0.05884
2	4140	0.05884
3	5491	0.04682
4	4703	0.04682
5	4699	0.04682

case due to the increased likelihood of incidents during rain.

Table 7.6: Prediction Ran with following Properties: Weather='Rain', Day='Thursday', Month='March', Date='23rd', StartTime='15:00', AnalysisTime='2 hours'

Survival Models

Rank	**Cluster**	**Incident Likelihood**
1	1	0.6554
2	13	0.6294
3	7	0.49461
4	2	0.4448
5	6	0.2114

Hex Mapping

Rank	**HexCell**	**Incident Probability**
1	3513	0.13108
2	4332	0.13108
3	5290	0.13108
4	4803	0.13108
5	3587	0.13108

Table 7.7: Prediction Ran with following Properties: Weather='Snow', Day='Thursday', Month='January', Date='10th', StartTime='17:00', AnalysisTime='6 hours'

Survival Models

Rank	**Cluster**	**Incident Likelihood**
1	1	0.9591
2	13	0.9491
3	7	0.8710
4	2	0.8288
5	6	0.5100

Hex Mapping

Rank	**HexCell**	**Incident Probability**
1	4334	0.201918
2	3862	0.174190
3	4350	0.036615
4	3943	0.036615
5	3792	0.036615

One last example demonstrated in table 7.7 shows snowy weather in mid-January, given 6 hours of analysis time. Notice that due to the increased analysis time the survival models predict that incident's belonging to several clusters are very likely to happen. Looking at the cell distribution shows that there are two likely cells, but other cells are less likely than rainy or clear conditions. This might be caused by people using their cars less in snowy conditions (except in a few areas), although this is just speculation.

7.6 Conclusion

We have demonstrated that by combining clustering, survival analysis, and Bayesian network inference techniques a toolchain can be created that accurately forecasts incidents in both space and time. Unlike many popular techniques that focus on particular situations, the toolchain is shown to well over the spatially diverse Nashville metropolitan area. By leveraging this predictive model, in the future we will create more accurate dispatching algorithms to respond appropriately to motor vehicle accidents as they occur.

Chapter 8

Generalizability: Applying the Decision Framework to Other Applications

8.1 Overview

A goal of this dissertation is to present decision-making approaches and frameworks that are applicable to a diverse set of multi-agent SCPS. However, up to this point we have primarily used the problems of emergency resource allocation and dispatch, which are instances of the Spatio-Temporal Resource Allocation (STRM) class of problems, to illustrate the challenges of SCPS decision-making and the proposed solution approaches.

This chapter details the other STRM problems that we have applied our decision-making approaches to, which include electric fleet charge scheduling and dynamic vehicle routing for paratransit vehicles. Each of these domains pose unique challenges both when modeling the environment and during decision-making, which we detail below. However, despite these unique challenges, we demonstrate that the structure of our decision support framework and the spectrum of planning approaches can be generalized to these domains, illustrating the broad applicability of our methods to SCPS problems.

The work comprising the description of the electric fleet charge scheduling problem and our solution framework (Section 8.2) has been published in the 2020 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference [25].

- G. Pettet, M. Ghosal, S. Mahserejian, S. Davis, S. Sridhar, A. Dubey, and M. Meyer (2021). "A Decision Support Framework for Grid-Aware Electric Bus Charge Scheduling," in *2020 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference, ISGT 2020*.

The work comprising the description of the dynamic vehicle routing problem for paratransit services and our solution framework (Section 8.3) has been published in the 13th

ACM/IEEE International Conference on Cyber-Physical Systems [27].

- M. Wilbur, S. Kadir, Y. Kim, G. Pettet, A. Mukhopadhyay, P. Pugliese, S. Samaranayake, A. Laszka, and A. Dubey (2022). "An Online Approach to Solve the Dynamic Vehicle Routing Problem with Stochastic Trip Requests for Paratransit Services," in *ACM/IEEE 13th International Conference on Cyber-Physical Systems, ICCPS 2022*.

8.2 Electric Fleet Charging

8.2.1 Introduction

Many municipalities have begun exploring the challenges of converting public transit fleets to electric buses (EVs). EVs increase fleet management complexity since the system now interacts with the power grid. Operators must determine where to build charging stations and when to charge the EVs.

Several works have previously examined EV charge scheduling from the perspective of energy cost optimization. Techniques that have been applied to the domain include genetic programming [172], greedy algorithms [173], linear optimization [174], and solving a Markov Decision Process using policy iteration [175].

While minimizing energy cost is important, it is equally important to consider the strain charging decisions place on the power grid. Bus fast charging may have significant impact on the grid and potentially cause thermal overloading, phase imbalances, and voltage violations [176, 177, 178]. This can lead to insulation breakdown in transformers and result in blackouts in the transformer's service area [179].

To accurately account for the energy needs of buses throughout the day with respect to the grid's condition, an integrated traffic and grid model is needed [180], [181], but is understudied in the context of charge scheduling – a recent paper examines charge scheduling with respect to grid constraints [182], but does not include the grid's health in the cost function or consider the impact of traffic on bus movement, for example. In this paper, we present a principled fleet management policy that incorporates such models. A traffic model provides precise travel time and power use information given different traffic conditions, informing the policy when buses will need to be recharged. A power grid model then quantifies how charging decisions impact the grid at various times and locations, and whether the charging behavior causes equipment thermal overloading or unacceptable system conditions [178, 177].

We present an anytime algorithm which leverages these traffic and grid models to estimate both the long term operational cost and power grid strain of charging decisions. Unlike methods such as reinforcement learning that require extensive offline training, our approach requires no training and does not assume a stationary environment. This is crucial in dynamic city environments where traffic or grid loads can change due to unexpected demand or emergencies.

Contributions: To realize such a charge scheduling policy we (1) construct a traffic simulation of the fleet's operating area to extract expected travel times and energy use for each bus route segment, (2) create a power grid model that captures the effect of a charging policy's demand on the grid, and (3) define a control process that utilizes the above simulations to find an optimal charging policy. We implement this framework on the transit system of Richland, WA as a case study and compare it to a greedy charging approach.

8.2.2 Decision Support Framework

Grid aware charge scheduling requires many inter-connected components (Figure 8.4). A traffic model provides bus State of Charge (SOC) and travel time estimates while a power grid model quantifies the grid impact of charging actions and determines if there are infrastructure constraint violations. These models are incorporated in a simulation of the transit system which is used by a decision theoretic planner to look ahead and determine

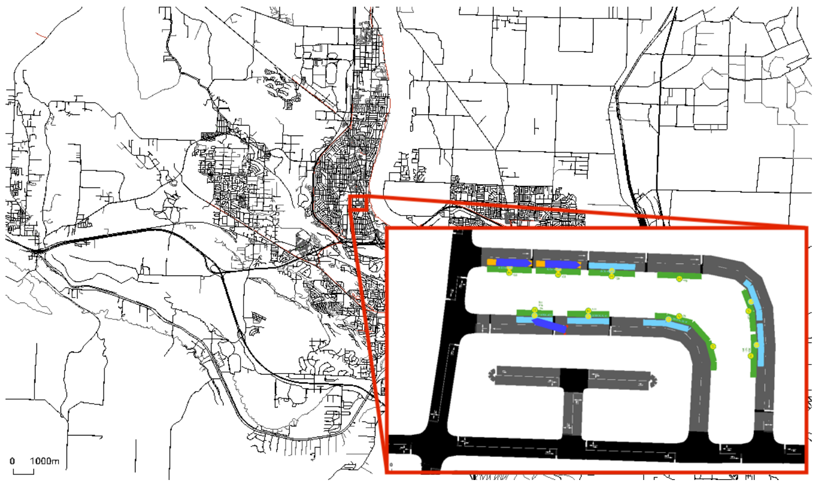

Figure 8.1: Street map of Tri-Cities region; the inset screenshot shows buses arriving and parked at the Knight Street Transit Center.

an optimal charging policy. We discuss each of these components in detail below.

8.2.2.1 Traffic Model

The traffic model simulates local transit busses and their SOC while following a daily schedule in realistic traffic conditions. The simulation domain for our case study is the Ben Franklin Transit (BFT) service area, which includes Richland, WA, and the encompassing cities of West Richland, Kennewick, and Pasco [183].

The street map data for the considered Tri-Cities region was obtained from © OpenStreetMap (OSM) [184], and included metadata for traffic light programs, public transit routes, and bus stop locations. The street map and zoomed in region of the Knight Street Station transit hub is illustrated in Figure 8.1.

The OSM map data was loaded into SUMO, an open source, agent-based, microscopic and continuous simulation package that can handle large networks [185]. The Traffic Control Interface (TraCI) allowed for controlling the SUMO simulation to extract detailed information at every time step in Python. Vehicles in SUMO follow common driving rules, including interactions with other vehicles such as changing lanes and maintaining a minimum space between vehicles [186]. The built in "ElectricBus" vehicle type was used to model the buses and their electricity use, and allows for custom bus attributes such as the

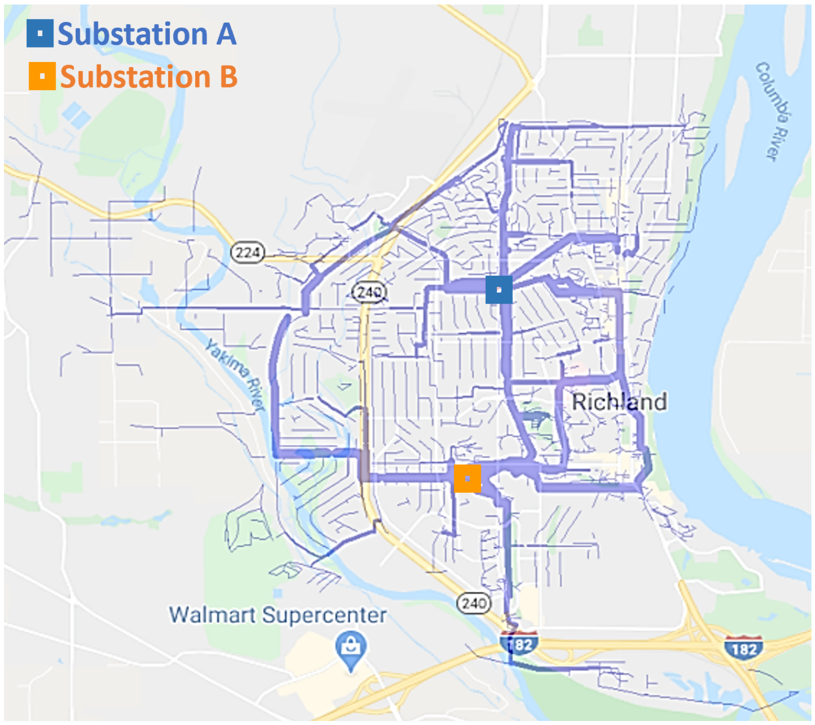

Figure 8.2: GIS overlay of the feeder model.

attributes listed in [186].

The electric bus flows were generated by repeating routes. Simulation output was collected for the electric buses including travel time between stops and and SOC at each stop.

8.2.2.2 Power Grid Model

The power grid representing the 12.47kV distribution feeder network in Richland, WA is modeled using GridLAB-D [187]. The substation and feeder layouts, equipment ratings, and historical hourly load for every customer was attained from the local utility. In this study scenario, two different distribution substations, each with 12 radial feeders, supply power to the majority of loads within the study footprint. Fig. 8.2 illustrates the location of these two substations and their feeder network stretching over the city of Richland.

As charging stations are connected to the distribution feeder system, various power system conditions will be monitored throughout the day. Each charging station is mapped to the nearest power grid node, and their impact to power grid depends on their location and power consumption relative to the time of day. Figure 8.6 illustrates the one-line diagram

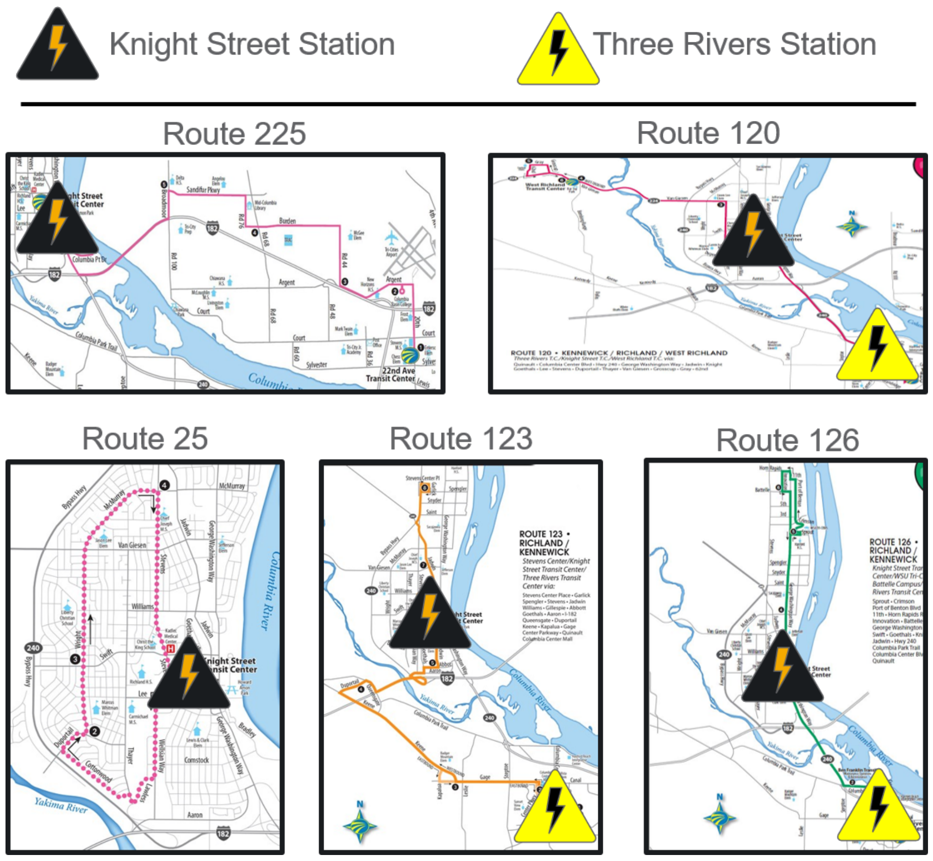

Figure 8.3: Simulated electric bus routes. Triangles represent the charger locations on each route – black for the Knight Street station, yellow for the Three Rivers station.

of the two distribution substations and an example placement of a new charging station placed mid-way down the feeder.

To be consistent with industry practice of building the system to ensure reliability under worst case scenarios, a peak day's hourly load profile is used to represent the power grid load without bus charging. The charging load is then added on top of the peak day load, and load flow analysis is performed to monitor nodal voltage deviations, phase imbalances, line losses, and the apparent power drawn from the feeder head to analyze equipment thermal loading. These measurements are:

- Nodal voltage deviation of the phases $\phi \in \{a,b,c\}$,

$$\Delta v_{i,\phi} = \frac{v_{i,\phi} - v_{base}}{v_{base}} \tag{8.1}$$

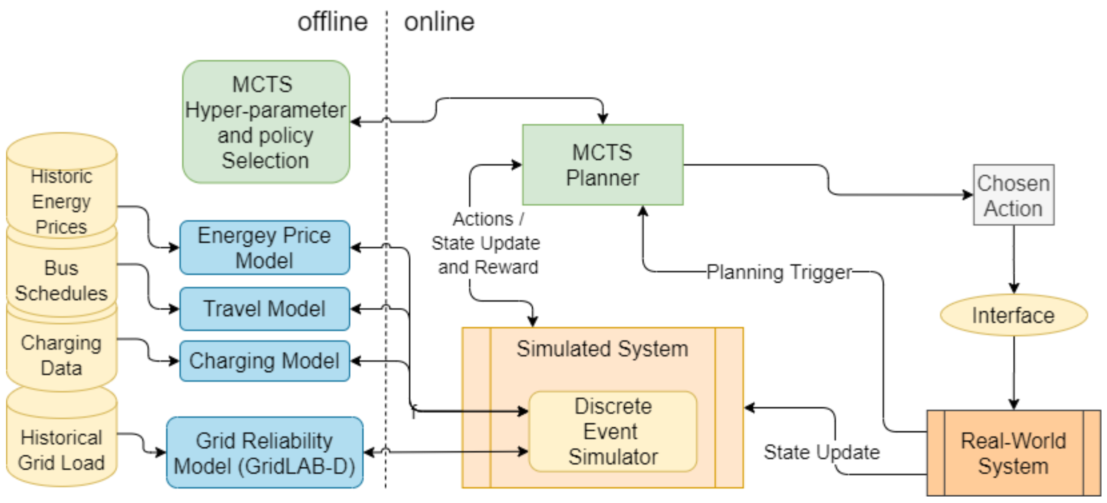

Figure 8.4: Grid-aware Decision Support Framework for public transit EV charge scheduling.

- Imbalance factor [176] of the circuit after charging at node i at time t approximated by

$$\mathscr{I}_i = \frac{v_2}{v_1} \approx \sqrt{\frac{1-\sqrt{3-6\alpha}}{1-\sqrt{3+6\alpha}}}, \tag{8.2}$$

where,

$$\alpha = \frac{v_{ab}^4 + v_{bc}^4 + v_{ca}^4}{(v_{ab}^2 + v_{bc}^2 + v_{ca}^2)^2}, \tag{8.3}$$

and v_1 and v_2 are the positive and negative sequence voltage,

- Total line losses L in underground cables (L^{ug}) and overhead lines (L^{oh}) after charging at node i,

$$L_i = L_i^{ug} + L_i^{oh} \tag{8.4}$$

- Apparent power drawn from the feeder f head corresponding to the node i where the charger is placed at time t,

$$\mathbb{S}_f = \sum_{\phi} V_{f,\phi} I_{f,\phi}, \;\; \forall \phi \tag{8.5}$$

where, the complex voltage and current at the feeder f are denoted by V_f and I_f.

The constraints which should not be violated are:

$$\left.\begin{array}{c} |\Delta v_{i,\phi}| \leq \Delta v_{max}, \; \mathscr{I}_i \leq \mathscr{I}_{max}, \\ L_i \leq L_{max}, \; \mathbb{S}_f \leq \mathbb{S}_{max}. \end{array}\right\} \tag{8.6}$$

where, the suffix *max* denotes the limit of the measurements. Based on the above measurands in (8.2)-(8.5), a novel grid score metric of charging at node i at time t is defined using the following terms,

The grid score g_i is given by,

$$g_i = \begin{cases} 0, \text{ if any of the inequalities in (8.6) is violated,} \\ 1 - \dfrac{1}{\sum_n w_n} \left[w_1 \displaystyle\sum_{j \in \phi} \frac{1}{3} \frac{|\Delta v_{i,j}|}{\Delta v_{max}} + w_2 \frac{\mathscr{I}_i}{\mathscr{I}_{max}} + w_3 \frac{L_i}{L_{max}} \right. \\ \left. + w_4 \dfrac{\mathbb{S}_f}{\mathbb{S}_{max}} \right], \text{ otherwise.} \end{cases} \tag{8.7}$$

where, w_n, $\forall n \in \{1, 2, 3, 4\}$'s are the various weights associated with the four additive terms in (8.7) (normalized voltage violation, imbalance factor, loss and apparent power drawn at feeder head). These weights are introduced so that the planner can choose to prioritize each contributing factor in the metric differently. Uniform contribution from the contributing factors would require each weight to be equal to 1. The joint grid score of multiple chargers' charging impact at a particular time can be derived using an extension of the expression in (8.7). The time variable t is dropped for simplicity in the above derivation. An example of grid related inputs used are shown in Fig. 8.5. The individual charging impact is given in Fig 8.5a, whereas Fig. 8.5b shows the plot of the assumed time of use (TOU) price of the

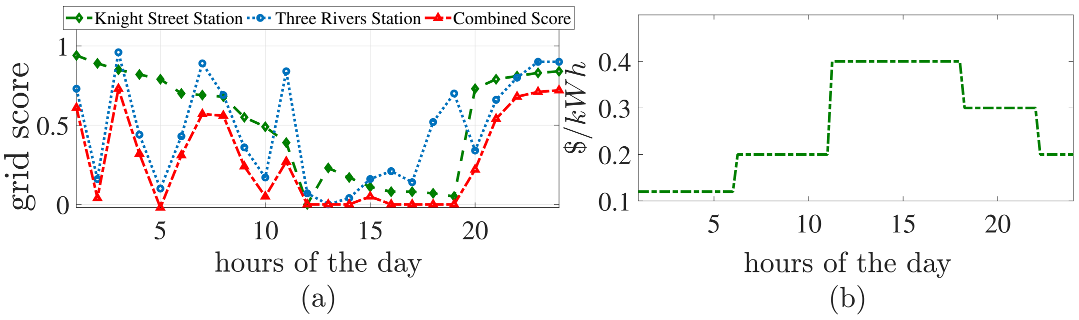

Figure 8.5: Grid related input. (a) Individual and combined grid score of the chargers in different hours of the day. (b) The Time of Use price of Electricity.

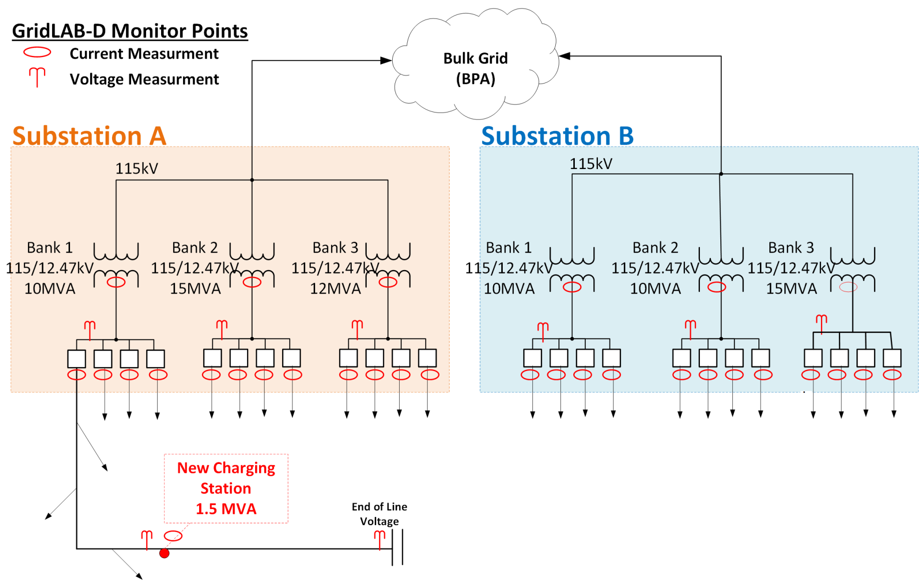

Figure 8.6: Richland distribution substations diagram

electricity throughout the day.

8.2.2.3 Decision Theoretic Planner

The overall system goal is to find an electric bus charging policy that minimizes both the impact on the power grid and operating costs. We begin with several assumptions. First, we assume that bus routes are set in advance, that each bus is assigned to a particular route, and that there is a travel model which describes each bus's travel time and battery discharge throughout their routes (Section 8.2.2.1). Next, we assume there is a set of pre-defined chargers C placed on the routes. We assume that we are given a model that captures how different charging actions effect the health of the power grid (Section 8.2.2.2). Last, we assume we have access to a time of use energy price model.

8.2.2.4 Markov Decision Process

We model the bus charging problem as a Markov Decision Process (MDP), which is a model commonly used to describe state and control dynamics for systems with intrinsic uncertainty[188, 189]. MDPs are described by the tuple $(S, A, P(s,a), \rho(s,a))$ where S is a finite state space, A is a set of actions, $P(s,a)$ is the state transition function for taking action a in state s, and $\rho(s,a)$ is the reward function for taking a at s.

States: A state captures environmental information which is needed for decision making, including each bus's SOC and position as well as energy pricing. Our model is limited to states which are relevant to decision making – when buses arrive at and leave chargers. Formally, a state at time t is represented by s^t and consists of a tuple (B^t, ε), where ε^t is the current time of use energy pricing and B^t is SOC and position information about the set of buses B at time t.

Actions: Actions in our model correspond to assigning a charger $c \in C$ to charge an available bus. A bus $b_i \in B$ is available to charge at c at time t if it is in the set of buses located at c at time t, $\gamma(c,t)$. A valid action at a time t is represented as $a^t = \{c-> b_i | c \in C\}$ where $b_i \in \gamma(c,t)$. $c-> b_i$ represents assigning bus b_i to charge at charger c.

Transitions: State transitions depend on the travel model and charging actions. A bus b_i's location will update based on its position along its assigned route $POS(b_i)$ and the current traffic. We assume that buses do not deviate from their schedule to charge, and spend the same amount of time at the charger weather they charge or not. b_i's SOC change depends on both the travel model as well as charging actions.

Rewards: Our reward function captures an action's impact on both the power grid and operational (i.e. energy) costs:

$$\rho(s,a) = -\bar{\varepsilon}_a + \beta \bar{g}(s,a) + \psi n_f(s) \tag{8.8}$$

where $\bar{\varepsilon}_a$ is the total energy cost for taking action a, $\bar{g}(s,a)$ is the total impact to the power grid of taking a at state s (which is mapped to the joint grid score $g(i,h)$ in Section 8.2.2.2), and β is a hyper-parameter that determines the tradeoff between the two. The last term is a penalty given anytime a bus runs out of charge – ψ is a hyper-parameter, and $n_f(s)$ is the number of 'empty' buses in state s.

8.2.2.5 Solution Approach

When choosing an approach to solve the above MDP, there are two required properties. First, the approach needs to be *adaptive* to unexpected changes in the environment such as equipment failure. Second, it must be capable of handling uncertainty in the environment, including uncertainty in travel times or power grid demand. While the current model does not include such uncertainty, incorporating it is a future goal.

With these requirements in mind, we solve the MDP using Monte Carlo Tree Search (MCTS), a simulation based search algorithm that evaluates actions by sampling from a large number of possible scenarios. The evaluations are stored in a search tree, which is used to explore promising actions. Unlike approaches like reinforcement learning that require offline training, MCTS performs its computation online by sampling from underlying simulations, making it flexible to changes in the environment. There is also substantial research on handling uncertainty with MCTS using techniques such as sparse sampling [190] and information set theory [191]. These properties make MCTS a good choice to solve our MDP.

When implementing MCTS, there are a few domain specific considerations: the Tree Policy and the Default Policy. The Tree Policy governs how the algorithm explores the search tree. We use the standard Upper Confidence bounds applied to Trees (UCT) algorithm [97], which is a principled approach that balances exploiting the most promising actions with exploring other actions. The Default Policy estimates the value of a new node by quickly simulating to a terminal node. The simplest default policy is uniform random action selection, but domain specific information can be incorporated to make these estimates more accurate. Buses with lower SOC's are more likely to be charged at any given moment, therefore our default policy chooses to charge each bus with probability inversely proportional to their SOC.

8.2.3 Performance

8.2.3.1 Experimental Design

To evaluate the framework, we examine the Tri-Cities area in Washington, USA. This mid-sized metropolitan area's transit system services the cities of Richland, Pasco, and Kennewick WA. Of the transit system's 18 bus routes, we selected 5 to simulate as EVs, which combined have 14 buses assigned to them on a typical day. We simulated charging stations at two transit hubs – the Knight Street station, which all 5 bus routes pass through, and the Three Rivers station, which 3 routes pass through. This setup simulates one main charging hub, with one secondary hub for a subset of routes, and is shown in Fig. 8.3. Our experimental runs are for one day of operation lasting from 6am to 10pm, and assume that each bus starts with batteries at half maximum capacity from overnight charging. To reduce noise, we run 10 experiments for each hyper parameter combination and average their scores.

For this case study, initial hyper-parameter values and environmental constants were selected from experience and are shown in Table 8.1. We focus on the effect of one key parameter: the reward tradeoff β. It controls the balance between minimizing the system's energy cost with minimizing the system's impact on the power grid, as explained in Section 8.2.2.3. The other hyper-parameters are kept constant.

Table 8.1: Experimental Parameters

Hyper-Parameter	**Value(s)**
MCTS Iteration Limit	3000
Look ahead time horizon	3.5 hours
UCT exploit / explore tradeoff	3.5
Bus failure reward penalty ψ	-500
Reward tradeoff parameter β	$\{1,2,3,4,5\}$
Battery capacity	150 KWh
Charging rate	300 KW

To understand the efficacy of our framework, we compare it to a greedy bus charging policy which charges any bus when it stops at a charger if its SOC is under a set threshold. If there are multiple buses that could be charged, the bus with the lowest SOC is chosen. The threshold ensures that buses are only charged when needed. For our experiments we chose a threshold of 41kWh, as this was the lowest threshold that did not lead to bus failure.

8.2.3.2 Results and Discussion

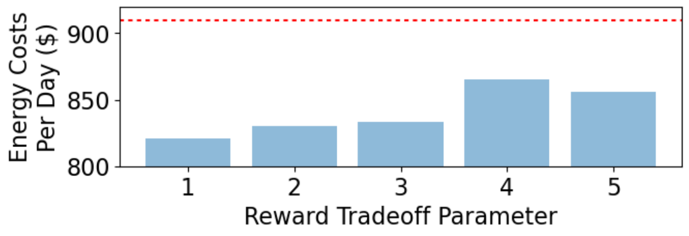

Figure 8.7: Reward tradeoff parameter β's effect on the energy cost (a lower score is better) to run the transit system per day. Red dashed line represents the greedy approach for comparison.

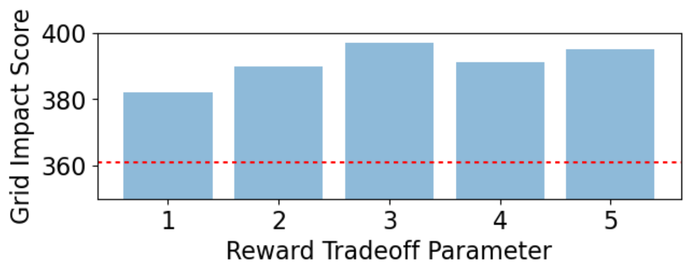

Figure 8.8: Reward tradeoff parameter β's effect on the cumulative grid impact (a higher score is better) to run the transit system per day. Red dashed line represents the greedy approach for comparison.

Results are shown in Figs. 8.7 and 8.8. Fig. 8.7 plots the cumulative energy cost to charge the buses for the 1 day scenario with different values for β, and compares them to the greedy baseline policy. Figure 8.8 does the same but for the cumulative power grid impact metric. Our first observation is that in all cases our framework outperforms the baseline greedy policy. For energy costs, lower values are better and indicate that less money is needed to charge the buses. The maximum cost using our framework of \$860 is \$50 lower than the greedy policy's \$910. For the power grid metric, a higher value is better and indicates that the charging decisions had a more favorable impact on the grid. Here our lowest power grid score of 376 was better than the greedy policy's score of 362. These values represent a significant savings when scaled to an entire transit system. For example, scaling our results to the full 75 buses in Richland's transit system could save over \$100k per year without considering the avoided cost for grid upgrades or peak demand charges.

Our second observation is the effect of the reward tradeoff parameter β. Generally, the higher the value of β the more emphasis is given to the power grid impact metric as opposed to the energy cost. This is reflected in our results, as $\beta = 1$ has the lowest grid impact score. As β is increased, the framework increasingly sacrifices energy costs to achieve better grid impacts. The takeaway is that our framework is flexible to the needs of different operators – if a city has very low tolerances for impacts to their power grid but can spare extra operating costs, they can use higher values for β. On the other hand cities with tighter budgets can decrease β to save operating costs at the cost of increasing the stress on their power grid.

8.2.4 Conclusion

When managing an electric bus transit fleet, it is crucial that charging policies take the power grid into consideration. We argue that to realize such a policy requires a decision support framework which incorporates both traffic and power grid models. We discuss how these models are used by a decision theoretic planner that evaluates possible charging schedules with regard to their operational costs and impact on the grid. We implement said framework on mid-sized transit network and found that our approach improves both costs and grid impact compared to a greedy scheduling policy. The positive outcome of this case study motivates future extensions in this domain. For example, our system assumes a conservative worst case scenario of peak historic demand in the grid model – this can be extended to a probabilistic demand model that allows more flexibility in decision making.

8.3 Dynamic Vehicle Routing Problem

The vehicle routing problem (VRP) is a well-known combinatorial optimization problem that assigns a fleet of vehicles to serve a set of user trip requests. The dynamic version of this problem (DVRP) introduces a subset of requests that are not known at the time of planning. The decision agent's goal can be to maximize the number of requests served, minimize the passenger's travel time, minimize the total distance traveled while serving all requests, or some combination of these objectives. In this work we focused on the socially beneficial real-world DRVP of *paratransit services* in Chattanooga, Tennessee. Paratransit services are a curb-to-curb transportation service provided by public transit agencies for passengers who are unable to use fixed-route transit (e.g., passengers with disabilities).

A fundamental challenge in solving a DVRP with planning approaches is computational tractability. For example, for the paratransit service we studied in Chattanooga with five vehicles, each with a capacity of eight passengers, the action space is of the order of 10^{22}.

While requests are relatively sparse in this problem setting (with one request occurring every few minutes on average), each decision must be made fairly quickly to keep customers from waiting for too long. One potential planning approach that could scale to a problem of this complexity the hierarchical framework from Chapter 4. However, trip requests often take users across large portions of the environment, making it difficult to spatially segment the problem into well-isolated sub-problems for such a hierarchical approach.

Instead, we leverage the structure of the problem to find promising actions. We define a *budget heuristic* that scores an action based on the time each vehicle has no passengers on board – a higher score means that vehicles spend more time with no riders. This is based on the idea that it is easier for empty vehicles to serve unexpected, dynamic requests, therefore maximizing the slack in the system. Then, given a set of routes and a new requests, we create a weighted graph based on the budget heuristic whose edges reqpresent: (a) which vehicles can serve the new requests, and (b) which vehicles can swap unpicked requests from their routes to maximize utility. From this graph we can sample feasible actions that are weighted by their heuristic score, allowing our decision maker to focus on the best promising actions during its search.

We applied our online planning framework using this heuristic sampling method to manage a simulation of Chattanooga's paratransit service, and compared its performance to the actual decisions that were made by the service. We examined both the total number of vehicle runs that were needed to service the requests, as well as the distance each vehicle traveled. These metrics for one week of data are shown in figures 8.9 and 8.10. Overall, we found that our decision support framework could save **\$145k** per year in operation costs and reduce CO_2 emissions by **576 metric tons** per year by using the vehicles more efficiently than the methods used in practice today. This was achieved with a computational time of less than **60 seconds** on average for each request. This successful implementation of our framework demonstrates its adaptability to complex SCPS problems.

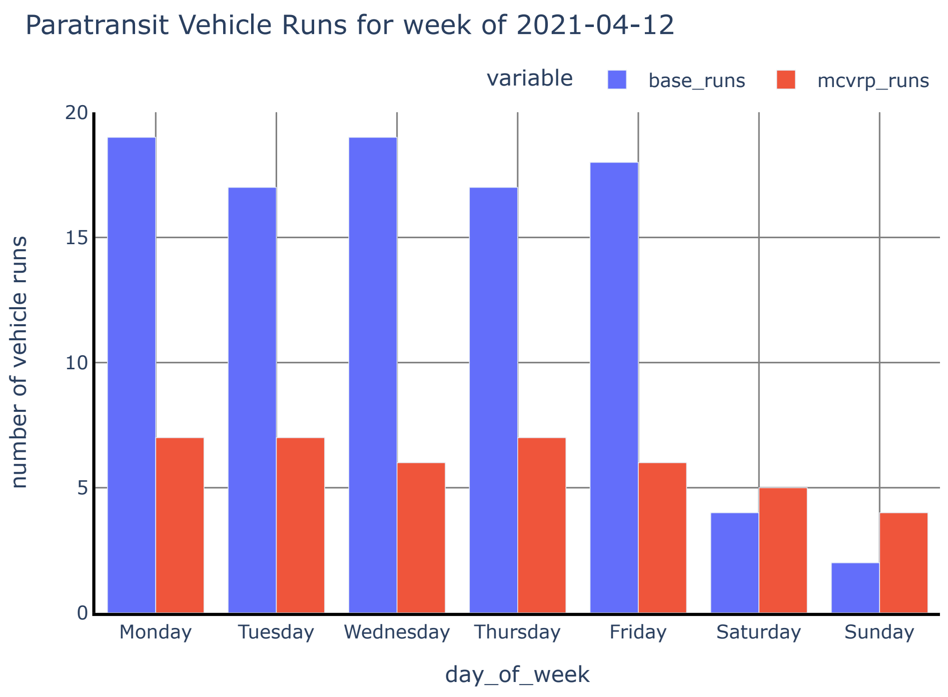

Figure 8.9: The number of vehicle runs that were performed

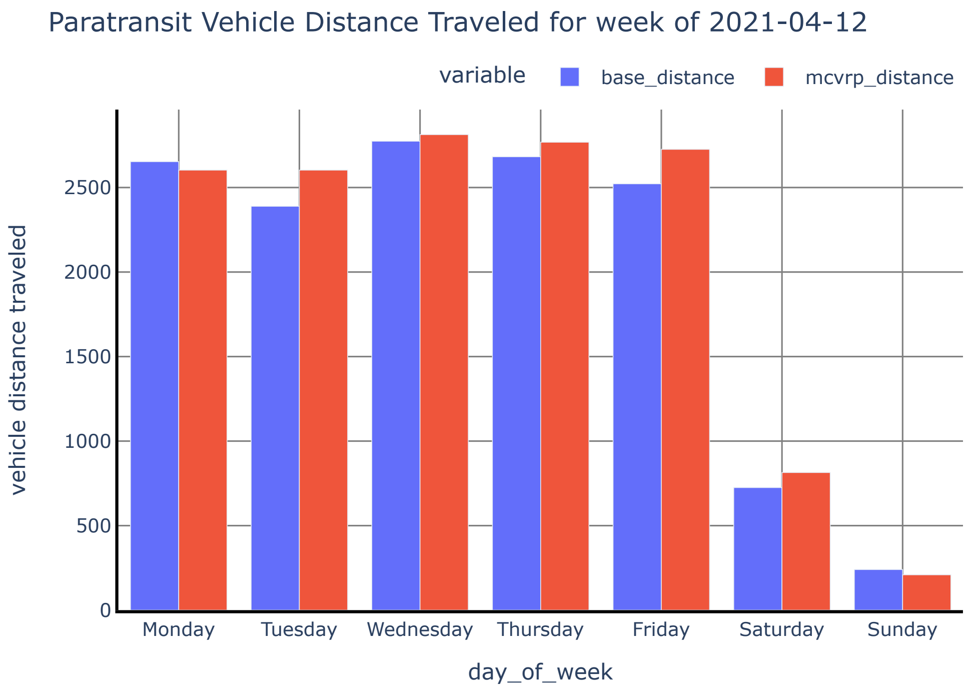

Figure 8.10: distance

Chapter 9

Conclusion

This dissertation makes several important contributions toward creating scalable and adaptive decision-making frameworks for Societal-scale Cyber-Physical Systems (SCPS).

First, this dissertation addresses the problem of creating an integrated decision support framework that combines generative models of the environment and procedures to update these models as the environment evolves with online planners. Chapter 3 presents a modular framework that that integrates these components, applies the framework to the emergency response responder dispatch problem, and shows that it outperforms classical optimization approaches. The standard centralized decision-making approach presented in Chapter 3 is applicable only to multi-agent SCPS with relatively small state-action spaces, but for these problems it completely captures the set of possible interactions between agents. This framework forms the foundation for each scalable planning approach to follow.

Second, this dissertation addresses the challenge of creating online planning algorithms that can scale to the complex environments present in many multi-agent SCPS. Chapter 4 presents a hierarchical planning approach that splits a large SCPS into smaller sub-problems that can be solved tractably using standard online planners. It utilizes two planning levels: a high-level planner leverages the spatial structure of the environment to spit the problem into sub-problems and then coordinates between these sub-problems as the environment evolves. A low-level planner then uses Monte-Carlo tree search to plan within each sub-problem. This approach is applied to the emergency response resource allocation problem, and is demonstrated to scale significantly better than centralized planning approaches while being adaptive to non-stationary environments.

Third, this dissertation addresses the challenge of designing decision-making approaches that are robust to communication failures. Chapter 5 presents a decentralized decision-making framework for multi-agent SCPS that enables each agent to independently determine its own course of action. It allows the system to function with access to limited inter-agent communication while being extremely scalable in terms of the number of agents. It examines techniques that cheaply model the behavior of other agents while each agent is locally planning to compensate for loosing inter-agent coordination, and designs a centralized filter that requires minimal communication with each agent while ensuring that system-wide constraints are satisfied.

Fourth, this dissertation addresses the challenge of utilizing the knowledge encoded in learning-based decision-making policies even when the environment has changed. Chapter 6 presents a hybrid decision-making approach called *Policy Augmented Monte Carlo tree search* (PA-MCTS) which combines learning-based RL with online planning. Given a specific computational budget, this hybrid framework converges to significantly better decisions than standard online planning approaches, making it ideal for situations with tight time constraints on decision-making. The online search also makes the approach significantly more robust to environmental changes than standard state-of-the-art RL approaches. Several properties of the approach are proven, including when PA-MCTS will return the optimal action, when it will choose better actions than either pure MCTS or greedy Q action selection, and the bound on the total deviation in cumulative rewards from an optimal policy when used for sequential decision-making.

Finally, this dissertation addresses the challenge of creating generative models with high-spatiotemporal resolution for SCPS environments. Chapter 7 presents a generative modeling approach that enables event forecasting over large, heterogeneous geographic areas using mixed-typed features (i.e., categorical and numeric features). It uses the Similarity Based Agglomerative Clustering (SBAC) algorithm to find groups of similar events spread across the spatial area. These groups of similar events also tend to have similar arrival distributions, which makes forecasting for each group more accurate. These groups are then mapped to spatial locations to achieve models with high spatial-temporal resolution.

This dissertation has shown that the above components, when integrated into a cohesive decision support system, can significantly improve the performance of several SCPS: (1) when the hierarchical framework was applied to emergency responder allocation in a simulation of Nashville, TN (see Chapter 4), our experiments show that it improves response times by **21.6 seconds** on average when compared to the policies in use today (Fig. 9.1), and improves response times by about **82 seconds** when there are three simultaneous vehicle failures (Fig. 9.2) – demonstrating the framework's adaptability to non-stationary conditions. (2) Applying the proposed framework to schedule when electric buses should charge throughout the day in a simulation of Richland, WA (see Section 8.2) resulted in an estimated savings of over **$100k** in operating costs compared to greedy methods. (3) Applying the framework to dynamically match and route paratransit vehicles to users in a simulation of Chattanooga, TN (see Section 8.3) resulted in a savings of **$145k** in operating costs and a reduction of **576 metric tons** of CO_2 emissions per year. The framework's success on this broad set of problems demonstrates it's generalizability across SCPS domains.

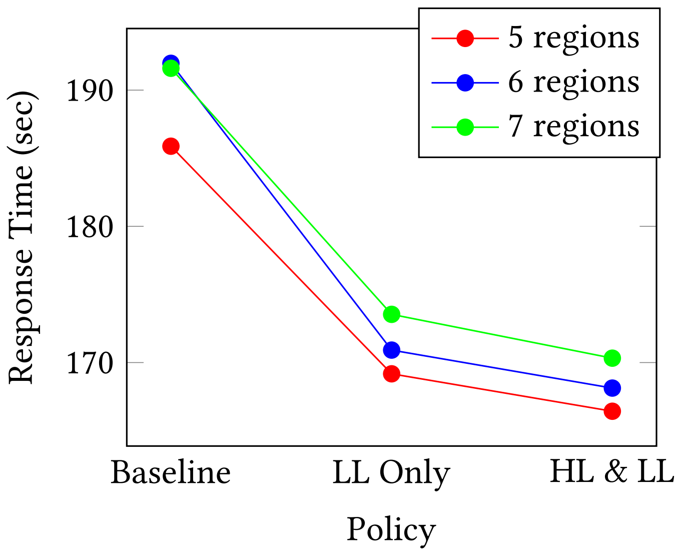

Figure 9.1: Results when the decision support framework is applied to emergency responder allocation in Nashville, TN. We compare the policy used by responders today (baseline), the hierarchical framework without the high-level inter-region coordination (LL Only), and the complete complete hierarchical planning framework described in Chapter 4 (HL & LL) when applied to incidents sampled from a non-stationary rate distribution. This figure presents a zoomed in view of the average response times.

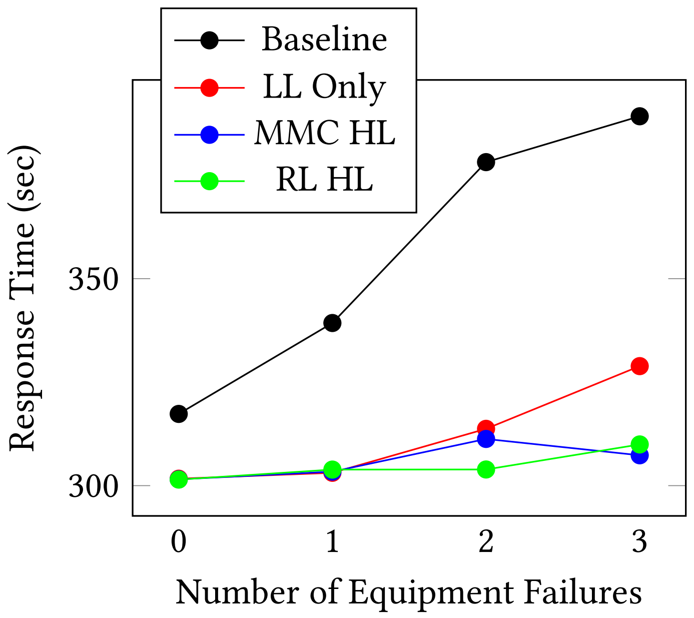

Figure 9.2: Results when the decision support framework is applied to emergency responder allocation with vehicle failures in Nashville, TN. We compare the policy used by responders today (baseline), the hierarchical framework without the high-level inter-region coordination (LL Only), and the complete complete hierarchical planning framework described in Chapter 4 using both an MMC queuing high level planner (MMC HL) and a surrogate model high level planner (RF HL) when subjected to increasing numbers of simultaneous equipment failures. This figure presents a zoomed in view of the average response times.

9.1 Future Work

As data collection and processing techniques improve, cloud and fog compute infrastructure is extended, and the internet of things becomes reality, Societal-scale Cyber-Physical Systems are becoming a pervasive aspect of our communities. Many such systems have an immense influence on our quality of life: for example, transit services can enable environmentally friendly and equitable movement throughout a community, while emergency response services are critical for the health and safety of community members. Optimizing these systems makes our communities more efficient and pleasant. However, SCPS present several challenges to overcome. They often cover very large, heterogeneous areas that make learning forecasting models difficult. Since they closely interact with the real world, the environments are constantly changing, which requires models that are constantly updating as well as adaptive decision making approaches. Finally, they can involve large numbers of agents that interact, leading to complex state-action spaces.

This dissertation has addressed several of these challenges, but there are still significant work yet to be accomplished. One key area that requires further exploration is the hybrid decision-making approach presented in Chapter 6. This approach has shown much promise for dealing with non-stationary environments with tight time constraints for decision-making. However, so far the approach has only been applied to the simple inverted pendulum control problem. The extensions necessary to apply this approach to complex SCPS should be investigated. So far the approach has only used centralized planning – would a decentralized or hierarchical framework further improve its scalability?

Another important gap is the explainability of these planning frameworks. These frameworks will be approved and used by people who are not experts in decision-making theory and data science. This hurts the adoption of the frameworks, particularly in safety critical applications such as emergency response. While there has been efforts to increase the explainability of machine learning and reinforcement learning approaches, there is little research on the same for planning approaches.

The overarching goal of this line of research is to create generalizable, scalable decision support frameworks for SCPS. This dissertation has addressed many challenges with creating such a framework, but there is still work to be done before such methods can be practically applied to many problems and accepted by stakeholders in these domains.

Bibliography

[1] Usdoe - the smart grid. https://www.smartgrid.gov/the_smart_grid/smart_grid.html, 2021. Accessed: 2021-04-21.

[2] Fangzhou Sun, Yao Pan, Jules White, and Abhishek Dubey. Real-Time and Predictive Analytics for Smart Public Transportation Decision Support System. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–8, May 2016.

[3] Trista Lin, Hervé Rivano, and Frédéric Le Mouël. A Survey of Smart Parking Solutions. *IEEE Transactions on Intelligent Transportation Systems*, 18(12):3229–3253, December 2017. ISSN 1558-0016.

[4] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, January 2017.

[5] Sean K. Keneally, Matthew J. Robbins, and Brian J. Lunday. A markov decision process model for the optimal dispatch of military medical evacuation assets. *Health Care Management Science*, 19(2):111–129, June 2016.

[6] M. S. Maxwell, S. G. Henderson, and H. Topaloglu. Ambulance redeployment: An approximate dynamic programming approach. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 1850–1860, December 2009.

[7] Erhan Erkut, Armann Ingolfsson, and Güneş Erdoğan. Ambulance location for maximum survival. *Naval Research Logistics*, 55(1):42–58, 2008.

[8] Amir Ali Nasrollahzadeh, Amin Khademi, and Maria E. Mayorga. Real-Time Am-

bulance Dispatching and Relocation. *Manufacturing & Service Operations Management*, 20(3):467–480, April 2018.

[9] X. Yu and S. Shen. An Integrated Decomposition and Approximate Dynamic Programming Approach for On-Demand Ride Pooling. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3811–3820, September 2020.

[10] Sanket Shah, Meghna Lowalekar, and Pradeep Varakantham. Neural Approximate Dynamic Programming for On-Demand Ride-Pooling. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):507–515, April 2020.

[11] H. Shuai and H. He. Online Scheduling of a Residential Microgrid via Monte-Carlo Tree Search and a Learned Model. *IEEE Transactions on Smart Grid*, 12(2):1073–1087, March 2021.

[12] S. Yoshida, M. Ishihara, T. Miyazaki, Y. Nakagawa, T. Harada, and R. Thawonmas. Application of Monte-Carlo tree search in a fighting game AI. In *2016 IEEE 5th Global Conference on Consumer Electronics*, pages 1–2, October 2016.

[13] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[14] D. Lenz, T. Kessler, and A. Knoll. Tactical cooperative planning for autonomous highway driving using Monte-Carlo Tree Search. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 447–453, June 2016.

[15] C. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer. Combining Planning and Deep Reinforcement Learning in Tactical Decision Making for Autonomous Driving. *IEEE Transactions on Intelligent Vehicles*, 5(2):294–305, June 2020.

[16] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel J. Kochenderfer, and Abhishek Dubey. Hierarchical planning for resource allocation in emergency response systems. In *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, pages 155–166, Nashville Tennessee, May 2021. ACM.

[17] Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas L Dean, and Craig Boutilier. Hierarchical solution of Markov decision processes using macro-actions. *arXiv preprint arXiv:1301.7381*, 2013.

[18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362 (6419):1140–1144, 2018.

[19] Yi Qi, Brian L Smith, and Jianhua Guo. Freeway accident likelihood prediction using a panel data analysis approach. *Journal of transportation engineering*, 133(3): 149–156, 2007.

[20] Shaw-Pin Miaou and Harry Lum. Modeling vehicle accidents and highway geometric design relationships. *Accident Analysis & Prevention*, 25(6):689–709, 1993.

[21] Li-Yen Chang. Analysis of freeway accident frequencies: negative binomial regression versus artificial neural network. *Safety science*, 43(8):541–557, 2005.

[22] Do-Gyeong Kim, Yuhwa Lee, Simon Washington, and Keechoo Choi. Modeling crash outcome probabilities at rural intersections: Application of hierarchical binomial logistic models. *Accident Analysis & Prevention*, 39(1):125–134, 2007.

[23] Williams Ackaah and Mohammed Salifu. Crash prediction model for two-lane rural highways in the ashanti region of ghana. *IATSS research*, 35(1):34–40, 2011.

[24] Ayan* Mukhopadhyay, Geoffrey* Pettet, Chinmaya Samal, Abhishek Dubey, and Yevgeniy Vorobeychik. An online decision-theoretic pipeline for responder dispatch. In *ACM/IEEE International Conference on Cyber-Physical Systems*, pages 12–pages. ACM, 2019.

[25] Geoffrey Pettet, Malini Ghosal, Shant Mahserejian, Sarah Davis, Siddharth Sridhar, Abhishek Dubey, and Michael Kintner-Meyer. A decision support framework for grid-aware electric bus charge scheduling. In *2021 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, pages 1–5, 2021.

[26] Geoffrey Pettet, Hunter Baxter, Sayyed Mohsen Vazirizade, Hemant Purohit, Meiyi Ma, Ayan Mukhopadhyay, and Abhishek Dubey. Designing decision support systems for emergency response: Challenges and opportunities. *Workshop on Cyber Physical Systems for Emergency Response in conjunction with CPS-IOT Week 2022*, 2022.

[27] Michael Wilbur, Salah Kadir, Youngseo Kim, Geoffrey Pettet, Ayan Mukhopadhyay, Philip Pugliese, Samitha Samaranayake, Aron Laszka, and Abhishek Dubey. An online approach to solve the dynamic vehicle routing problem with stochastic trip requests for paratransit services. In *ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2022.

[28] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel J. Kochenderfer, and Abhishek Dubey. Hierarchical planning for dynamic resource allocation in smart and connected communities. *ACM Trans. Cyber-Phys. Syst.*, nov 2021. ISSN 2378-962X.

[29] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel Kochenderfer, Yevgeniy Vorobeychik, and Abhishek Dubey. On algorithmic decision procedures in emergency response systems in smart and connected communities. In *Proceedings of the 19th Inter-*

national Conference on Autonomous Agents and MultiAgent Systems, pages 1046–1054, 2020.

[30] Geoffrey Pettet, Ayan Mukhopadhyay, and Abhishek Dubey. Decision making in non-stationary environments with policy-augmented monte carlo tree search. In *The 5th Multi-disciplinary Conference on Reinforcement Learning and Decision Making*, pages 490–494, 2022.

[31] Geoffrey Pettet, Saideep Nannapaneni, Benjamin Stadnick, Abhishek Dubey, and Gautam Biswas. Incident analysis and prediction using clustering and bayesian network. In *2017 IEEE International Conference on Smart City Innovations*, pages 1–8, 2017.

[32] Sayyed Mohsen Vazirizade, Ayan Mukhopadhyay, Geoffrey Pettet, Said El Said, Hiba Baroud, and Abhishek Dubey. Learning incident prediction models over large geographical areas for emergency response systems. *IEEE Conference on Smart Computing*, 2021.

[33] Henrik Jaldell. How important is the time factor? saving lives using fire and rescue services. *Fire Technology*, 53(2):695–708, 2017.

[34] Henrik Jaldell, Prachaksvich Lebnak, and Anurak Amornpetchsathaporn. 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.

[35] Center of Disease Control and Prevention. Road traffic injuries and deaths — a global problem. https://www.cdc.gov/injury/features/global-road-safety/index.html, 2019.

[36] Association for Safe International Road Travel. Road Safety Facts. https://www.asirt.org/safe-travel/road-safety-facts/, 2019.

[37] John A Deacon, Charles V Zegeer, and Robert C Deen. Identification of hazardous rural highway locations. *Transportation Research Record*, 543, 1974.

[38] Dominique Lord, Simon P Washington, and John N Ivan. Poisson, poisson-gamma and zero-inflated regression models of motor vehicle crashes: Balancing statistical fit and theory. *Accident Analysis & Prevention*, 37(1):35–46, 2005.

[39] Jie Bao, Pan Liu, and Satish V. Ukkusuri. A spatiotemporal deep learning approach for citywide short-term crash risk prediction with multi-source data. *Accident Analysis & Prevention*, 122:239–254, 2019. ISSN 0001-4575. doi: 10.1016/J.AAP.2018.10.015.

[40] Richard Church and Charles ReVelle. The maximal covering location problem. In *Papers of the Regional Science Association*, volume 32, pages 101–118, 1974.

[41] Matthew S. Maxwell, Mateo Restrepo, Shane G. Henderson, and Huseyin Topaloglu. Approximate dynamic programming for ambulance redeployment. *INFORMS Journal on Computing*, 22(2):266–281, 2010.

[42] Nashville Fire Department. Private Communication, 2018.

[43] Yi Qi, Brian L. Smith, and Jianhua Guo. Freeway accident likelihood prediction using a panel data analysis approach. *Journal of Transportation Engineering*, 133 (3):149–156, 2007. ISSN 0733-947X. doi: 10.1061/(ASCE)0733-947X(2007)133:3(149).

[44] Qi Shi and Mohamed Abdel-Aty. Big data applications in real-time traffic operation and safety monitoring and improvement on urban expressways. *Transportation Research Part C: Emerging Technologies*, 58:380–394, 2015. ISSN 0968-090X. doi: 10.1016/J.TRC.2015.02.022.

[45] Yuanchang Xie, Dominique Lord, and Yunlong Zhang. Predicting motor vehicle collisions using Bayesian neural network models: An empirical analysis. *Accident Analysis & Prevention*, 39(5):922–933, 2007. ISSN 0001-4575. doi: 10.1016/J.AAP.2006.12.014.

[46] Ayan Mukhopadhyay, Geoffrey Pettet, Chinmaya Samal, Abhishek Dubey, and Yevgeniy Vorobeychik. An online decision-theoretic pipeline for responder dispatch. In *International Conference on Cyber-Physical Systems*, pages 185–196, 2019.

[47] Christos G Cassandras. Smart cities as cyber-physical social systems. *Engineering*, 2(2):156–158, 2016.

[48] Megan K. Sutherland and Meghan E. Cook. Data-driven smart cities: A closer look at organizational, technical and data complexities. In *Proceedings of the 18th Annual International Conference on Digital Government Research*, dg.o '17, pages 471–476, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5317-5. doi: 10.1145/3085228.3085239. URL http://doi.acm.org/10.1145/3085228.3085239.

[49] Ayan Mukhopadhyay, Chao Zhang, Yevgeniy Vorobeychik, Milind Tambe, Kenneth Pence, and Paul Speer. Optimal allocation of police patrol resources using a continuous-time crime model. In *International Conference on Decision and Game Theory for Security*, pages 139–158, 2016.

[50] Ayan Mukhopadhyay, Yevgeniy Vorobeychik, Abhishek Dubey, and Gautam Biswas. Prioritized allocation of emergency responders based on a continuous-time incident prediction model. In *Conference on Autonomous Agents and Multiagent Systems*, pages 168–177, 2017.

[51] Mohammad Abu-Matar and John Davies. Data driven reference architecture for smart city ecosystems. In *2017 IEEE SmartWorld*, pages 1–7, San Francisco, CA, USA, 2017. IEEE, IEEE.

[52] Chao Zhang, Victor Bucarey, Ayan Mukhopadhyay, Arunesh Sinha, Yundi Qian, Yevgeniy Vorobeychik, and Milind Tambe. Using abstractions to solve opportunistic crime security games at scale. In *2016 AAMAS Proceedings*, pages 196–204, Bologna, Italy, 2016. International Foundation for Autonomous Agents and Multiagent Systems, IFAAMAS.

[53] Ayan Mukhopadhyay, Zilin Wang, and Yevgeniy Vorobeychik. A decision theoretic framework for emergency responder dispatch. In *Conference on Autonomous Agents and Multiagent Systems*, pages 588–596, 2018.

[54] Qiying Hu and Wuyi Yue. *Markov decision processes with their applications*, volume 14. Springer Science & Business Media, New York, NY, USA, 2007.

[55] Ayan Mukhopadhyay, Chao Zhang, Yevgeniy Vorobeychik, Milind Tambe, Kenneth Pence, and Paul Speer. Optimal allocation of police patrol resources using a continuous-time crime model. In *International Conference on Decision and Game Theory for Security*, pages 139–158, New York, NY, USA, 2016. Springer, Springer.

[56] Ciro Caliendo, Maurizio Guida, and Alessandra Parisi. A crash-prediction model for multilane roads. *Accident Analysis & Prevention*, 39(4):657 – 670, 2007. ISSN 0001-4575. doi: https://doi.org/10.1016/j.aap.2006.10.012. URL http://www.sciencedirect.com/science/article/pii/S0001457506001965.

[57] Sean K Keneally, Matthew J Robbins, and Brian J Lunday. A markov decision process model for the optimal dispatch of military medical evacuation assets. *Health Care Management Science*, 19(2):111–129, 2016.

[58] David Roxbee Cox and David Oakes. *Analysis of survival data*, volume 21. CRC Press, New York, NY, USA, 1984.

[59] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[60] Andrew V Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *16th ACM-SIAM symposium on Discrete algorithms*, pages 156–165, Vancouver, BC, Canada, 2005. SIAM, ACM.

[61] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[62] Here api. https://developer.here.com/, 2018. Accessed: 2018-11-17.

[63] Diederik P Kingma and Jimmy Lei Ba. Adam: Amethod for stochastic optimization. In *Proc. 3rd Int. Conf. Learn. Representations*, 2014.

[64] Herbert Robbins and Sutton Monro. A stochastic approximation method. In *Herbert Robbins Selected Papers*, pages 102–109. Springer, 1985.

[65] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[66] George T. Taoka. Brake reaction times of unalerted drivers. *ITE Journal*, 59(3): 19–21, 1989.

[67] Thomas H Blackwell and Jay S Kaufman. Response time effectiveness: comparison of response time and survival in an urban emergency medical services system. *Academic Emergency Medicine*, 9(4):288–295, 2002.

[68] Konstantinos G Zografos, Konstantinos N Androutsopoulos, and George M Vasilakis. A real-time decision support system for roadway network incident response logistics. *Transportation Research Part C: Emerging Technologies*, 10(1):1–18, 2002.

[69] Xueping Li, Zhaoxia Zhao, Xiaoyan Zhu, and Tami Wyatt. Covering models and

optimization techniques for emergency response facility location and planning: a review. *Mathematical Methods of Operations Research*, 74(3):281–310, 2011.

[70] Hector Toro-DíAz, Maria E Mayorga, Sunarin Chanta, and Laura A Mclay. Joint location and dispatching decisions for emergency medical services. *Computers & Industrial Engineering*, 64(4):917–928, 2013.

[71] Leslie W Kennedy, Joel M Caplan, and Eric Piza. Risk clusters, hotspots, and spatial intelligence: risk terrain modeling as an algorithm for police resource allocation strategies. *Journal of Quantitative Criminology*, 27(3):339–362, 2011.

[72] Martin B Short, Maria R D'orsogna, Virginia B Pasour, George E Tita, Paul J Brantingham, Andrea L Bertozzi, and Lincoln B Chayes. A statistical model of criminal behavior. *Mathematical Models and Methods in Applied Sciences*, 18(supp01): 1249–1267, 2008.

[73] Praprut Songchitruksa and Kevin Balke. Assessing weather, environment, and loop data for real-time freeway incident prediction. *Transportation Research Record: Journal of the Transportation Research Board*, 1(1959):105–113, 2006.

[74] Vasin Kiattikomol. *Freeway crash prediction models for long-range urban transportation planning*. PhD thesis, The University of Tennessee, Knoxville, 2005.

[75] Geoffrey Pettet, Saideep Nannapaneni, Benjamin Stadnick, Abhishek Dubey, and Gautam Biswas. Incident analysis and prediction using clustering and bayesian network. In *2017 IEEE SmartWorld*, pages 1–8, San Francisco, CA, USA, 2017. IEEE, IEEE.

[76] Ayan Mukhopadhyay, Geoffrey Pettet, Sayyed Mohsen Vazirizade, Di Lu, Alejandro Jaimes, Said El Said, Hiba Baroud, Yevgeniy Vorobeychik, Mykel Kochenderfer, and Abhishek Dubey. A review of incident prediction, resource allocation, and

dispatch models for emergency management. *Accident Analysis & Prevention*, 165: 106501, 2022.

[77] Olfa Chebbi and Jouhaina Chaouachi. Modeling on-demand transit transportation system using an agent-based approach. In *IFIP International Conference on Computer Information Systems and Industrial Management*, pages 316–326. Springer, 2015.

[78] Stefan Gössling. Integrating e-scooters in urban transportation: Problems, policies, and the prospect of system change. *Transportation Research Part D: Transport and Environment*, 79:102230, 2020.

[79] National Emergency Number Association. 911 Statistics. www.nena.org/page/911Statistics, 2021.

[80] AON Impact Forecasting. Weather, Climate and Catastrophe Insight. Technical report, AON Impact Forecasting, 01 2018.

[81] Mykel J Kochenderfer. *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[82] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel Kochenderfer, Yevgeniy Vorobeychik, and Abhishek Dubey. On algorithmic decision procedures in emergency response systems in smart and connected communities. In *Conference on Autonomous Agents and Multiagent Systems*, page 1046–1054, 2020.

[83] Daniel Claes, Frans Oliehoek, Hendrik Baier, and Karl Tuyls. Decentralised online planning for multi-robot warehouse commissioning. In *Conference on Autonomous Agents and Multiagent Systems*, pages 492–500, 2017.

[84] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel J. Kochenderfer, and Abhishek Dubey.

Hierarchical planning for dynamic resource allocation in smart and connected communities. *ACM Transactions on Cyber-Physical Systems*, 2021.

[85] Michael Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2009.

[86] Khashayar Rohanimanesh and Sridhar Mahadevan. Learning to take concurrent actions. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, pages 1651–1658, 2002.

[87] Ayan Mukhopadhyay, Geoffrey Pettet, Mykel Kochenderfer, and Abhishek Dubey. Designing emergency response pipelines: Lessons and challenges. *AI for Social Good Workshop, AAAI Fall Symposium Series*, 2020.

[88] Spencer Chainey, Svein Reid, and Neil Stuart. *When is a hotspot a hotspot? A procedure for creating statistically robust hotspot maps of crime*. Taylor & Francis, London, England, 2002.

[89] Stuart Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[90] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. In *International Workshop on Algorithms and Computation*, pages 274–285. Springer, 2009.

[91] David G Kendall. Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain. *The Annals of Mathematical Statistics*, pages 338–354, 1953.

[92] John F Shortle, James M Thompson, Donald Gross, and Carl M Harris. *Fundamentals of Queueing Theory*. Wiley, 2018.

[93] Michael Dzator and Janet Dzator. An effective heuristic for the P-median problem with application to ambulance location. *OPSEARCH*, 50(1):60–74, 2013.

[94] Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. i: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.

[95] Mark S Daskin. *Network and discrete location: models, algorithms, and applications*. John Wiley & Sons, 1995.

[96] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[97] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, pages 282–293, 2006.

[98] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Workshop on Experimental and Efficient Algorithms*, pages 319–333, 2008.

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

[100] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

[101] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[102] Jonathan D Mayer. Emergency medical service: delays, response time and survival. *Medical Care*, pages 818–827, 1979.

[103] Sangdon Park, Osbert Bastani, James Weimer, and Insup Lee. Calibrated prediction with covariate shift via unsupervised domain adaptation. In *International Conference on Artificial Intelligence and Statistics*, pages 3219–3229, 2020.

[104] Forrest Laine, Chiu-Yuan Chiu, and Claire Tomlin. Eyes-closed safety kernels: Safety for autonomous systems under loss of observability. *arXiv preprint arXiv:2005.07144*, 2020.

[105] Shreyas Ramakrishna, Zahra Rahiminasab, Gabor Karsai, Arvind Easwaran, and Abhishek Dubey. Efficient out-of-distribution detection using latent space of β-vae for cyber-physical systems. *ACM Transactions on Cyber Physical Systems*, 2021.

[106] Richard S Sutton. Td models: Modeling the world at a mixture of time scales. In *International Conference on Machine Learning*, pages 531–539. 1995.

[107] Doina Precup and Richard S Sutton. Multi-time models for temporally abstract planning. In *Neural Information Processing Systems*, pages 1050–1056, 1998.

[108] J-P Forestier and Pravin Varaiya. Multilayer control of large markov chains. *IEEE Transactions on Automatic Control*, 23(2):298–305, 1978.

[109] Constantine Toregas, Ralph Swain, Charles ReVelle, and Lawrence Bergman. The location of emergency service facilities. *Operations Research*, 19:1363–1373, 1971.

[110] Michel Gendreau, Gilbert Laporte, and Frédéric Semet. Solving an ambulance location model by tabu search. *Location Science*, 5(2):75–88, 1997.

[111] Francisco Silva and Daniel Serra. Locating emergency services with different priorities: the priority queuing covering location problem. *Journal of the Operational Research Society*, 59(9):1229–1238, 2008.

[112] V. A. Knight, P. R. Harper, and L. Smith. Ambulance allocation for maximal survival with heterogeneous outcome measures. 40(6):918–926, 2012.

[113] USA Today. Nashville bombing froze wireless communications, exposed 'achilles heel' in regional network. https://www.usatoday.com/story/news/nation/2020/12/29/nashville-bombing-area-communications-network-exposed-achilles-heel/4070797001/, 2020.

[114] Ayan Mukhopadhyay, Geoffrey Pettet, Chinmaya Samal, Abhishek Dubey, and Yevgeniy Vorobeychik. An online decision-theoretic pipeline for responder dispatch. In *ACM/IEEE International Conference on Cyber-Physical Systems*, pages 185–196, 2019.

[115] One Concern. Artificial Intelligence: A GameChanger for Emergency Response. Technical report, One Concern, 2017.

[116] Yisong Yue, Lavanya Marla, and Ramayya Krishnan. An efficient simulation-based approach to ambulance fleet allocation and dynamic redeployment. In *AAAI*, 2012.

[117] H. Purohit, S. Nannapaneni, A. Dubey, P. Karuna, and G. Biswas. Structured summarization of social web for smart emergency services by uncertain concept graph. In *2018 IEEE International Science of Smart City Operations and Platforms Engineering in Partnership with Global City Teams Challenge (SCOPE-GCTC)*, pages 30–35, April 2018. doi: 10.1109/SCOPE-GCTC.2018.00012.

[118] Wikipedia contributors. Computer-aided dispatch — Wikipedia, the free encyclopedia, 2019. URL https://en.wikipedia.org/w/index.php?title=Computer-aided_dispatch&oldid=916096608. [Online; accessed 20-October-2019].

[119] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210. Morgan Kaufmann Publishers Inc., 1996.

[120] Mohammad Ghavamzadeh and Sridhar Mahadevan. Learning to cooperate using hierarchical reinforcement learning. 2006.

[121] Shenyang Guo. *Survival analysis*. Oxford University Press, 2010.

[122] Natarajan Gautam. *Analysis of queues: methods and applications*. CRC Press, 2012.

[123] Johannes Fürnkranz and Tobias Scheffer. *Machine Learning: ECML 2006: 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, volume 4212. Springer Science & Business Media, 2006.

[124] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492. URL http://doi.acm.org/10.1145/1327452.1327492.

[125] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[126] Mykel J Kochenderfer, Tim A Wheeler, and Kyle H Wray. *Algorithms for decision making*. MIT Press, 2022.

[127] Ronald Ortner, Pratik Gajane, and Peter Auer. Variational regret bounds for reinforcement learning. In *35th Uncertainty in Artificial Intelligence Conference*, volume 115, pages 81–90, 2020.

[128] Wang Chi Cheung, David Simchi-Levi, and Ruihao Zhu. Non-stationary reinforcement learning: The blessing of (more) optimism. *Available at SSRN 3397818*, 2019.

[129] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel Kochenderfer, Yevgeniy Vorobeychik, and Abhishek Dubey. On algorithmic decision procedures in emergency response systems in smart and connected communities. In *Conference on Autonomous Agents and Multi-Agent Systems*, pages 1046–1054, 2020.

[130] Jay K Satia and Roy E Lave Jr. Markovian decision processes with uncertain transition probabilities. *Operations Research*, 21(3):728–740, 1973.

[131] Chelsea C White III and Hany K Eldeib. Markov decision processes with imprecise transition probabilities. *Operations Research*, 42(4):739–749, 1994.

[132] Garud N Iyengar. Robust dynamic programming. *Mathematics of Operations Research*, 30(2):257–280, 2005.

[133] Erwan Lecarpentier and Emmanuel Rachelson. Non-stationary Markov decision processes, a worst-case approach using model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 32:7216–7225, 2019.

[134] Samuel PM Choi, Dit-Yan Yeung, and Nevin L Zhang. Hidden-mode Markov decision processes for nonstationary sequential decision making. In *Sequence Learning*, pages 264–287. Springer, 2000.

[135] Michael Kearns, Yishay Mansour, and Andrew Y Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2):193–208, 2002.

[136] Leandro L Minku. Transfer learning in non-stationary environments. In *Learning from Data Streams in Evolving Environments*, pages 13–37. Springer, 2019.

[137] Daniel L Silver, Qiang Yang, and Lianghao Li. Lifelong machine learning systems: Beyond learning algorithms. In *2013 AAAI Spring Symposium Series*, pages 49–55, 2013.

[138] Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Tobias Pfaff, Theophane Weber, Lars Buesing, and Peter W Battaglia. Combining q-learning and search with amortized value estimates. In *International Conference on Learning Representations*, 2019.

[139] Ayan Mukhopadhyay, Zilin Wang, and Yevgeniy Vorobeychik. A decision theoretic framework for emergency responder dispatch. In *Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 588–596, 2018.

[140] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games (CG)*, pages 72–83. Springer, 2006.

[141] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[142] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[143] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[144] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *30th AAAI Conference on Artificial Intelligence (AAAI)*, volume 30, pages 2094–2100, 2016.

[145] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[146] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. Lessons learned from the chameleon testbed. In *USENIX Annual Technical Conference*. USENIX Association, July 2020.

[147] S Joe Qin. Survey on data-driven industrial process monitoring and diagnosis. *Annual Reviews in Control*, 36(2):220–234, 2012.

[148] Kathleen L Lane. Identifying and supporting students at risk for emotional and behavioral disorders within multi-level models: Data driven approaches to conducting secondary interventions with an academic emphasis. *Education and Treatment of Children*, 30(4):135–164, 2007.

[149] Sushant Jain, Rahul C Shah, Waylon Brunette, Gaetano Borriello, and Sumit Roy. Exploiting mobility for energy efficient data collection in wireless sensor networks. *Mobile Networks and Applications*, 11(3):327–339, 2006.

[150] Robert Morris, M Frans Kaashoek, David Karger, Hari Balakrishnan, Ion Stoica, David Liben-Nowell, and Frank Dabek. Chord: A scalable peer-to-peer look-up protocol for internet applications. *IEEE/ACM Transactions On Networking*, 11(1): 17–32, 2003.

[151] D. Dhungana, G. Engelbrecht, J. X. Parreira, A. Schuster, R. Tobler, and D. Valerio. Data-driven ecosystems in smart cities: A living example from seestadt aspern. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 82–87, Dec 2016. doi: 10.1109/WF-IoT.2016.7845434.

[152] Mohamed A Abdel-Aty and A Essam Radwan. Modeling traffic accident occurrence and involvement. *Accident Analysis & Prevention*, 32(5):633–642, 2000.

[153] Hoong Chor Chin and Mohammed Abdul Quddus. Applying the random effect negative binomial model to examine traffic accident occurrence at signalized intersections. *Accident Analysis & Prevention*, 35(2):253–259, 2003.

[154] Saroj Raut and Swapnili Karmore. Review on: Severity estimation unit of automotive accident. In *Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in*, pages 523–526. IEEE, 2015.

[155] Luis Moreira-Matias and Vitor Cerqueira. Cjammer-traffic jam cause prediction using boosted trees. In *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*, pages 743–748. IEEE, 2016.

[156] Younshik Chung. Development of an accident duration prediction model on the korean freeway systems. *Accident Analysis & Prevention*, 42(1):282–289, 2010.

[157] Ayan Mukhopadhyay, Yevgeniy Vorobeychik, Abhishek Dubey, and Gautam Biswas. Prioritized allocation of emergency responders based on a continuous-time incident prediction model. In *Sixteenth International Conference on Antonomous Agents and Multiagent Sytems*, Sao Paulo - Brazil, 05/2017 2017.

[158] Cen Li and Gautam Biswas. Unsupervised learning with mixed numeric and nominal data. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):673–690, 2002.

[159] Richard O Duda and Peter E Hart. *Pattern Elesslfication and Scene Analysis*. Wiley, 1973.

[160] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28 (1):100–108, 1979.

[161] Douglas Fisher and Pat Langley. Methods of conceptual clustering and their relation to numerical taxonomy. Technical report, DTIC Document, 1985.

[162] David W Goodall. A new similarity index based on probability. *Biometrics*, pages 882–907, 1966.

[163] Ronald Aylmer Fisher. *Statistical methods for research workers*. Genesis Publishing Pvt Ltd, 1925.

[164] HO Lancaster. The combination of probabilities arising from data in discrete distributions. *Biometrika*, 36(3/4):370–382, 1949.

[165] Lior Rokach and Oded Maimon. Clustering methods. In *Data mining and knowledge discovery handbook*. Springer, 2005.

[166] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[167] G Shenyang. Survival analysis (pocket guides to social work research methods), 2010.

[168] D Collett. Modelling survival data. In *Modelling Survival Data in Medical Research*, pages 53–106. Springer, 1994.

[169] Lee-Jen Wei. The accelerated failure time model: a useful alternative to the cox regression model in survival analysis. *Statistics in medicine*, 11(14-15):1871–1879, 1992.

[170] Willliam Feller. *An introduction to probability theory and its applications*, volume 2. John Wiley & Sons, 2008.

[171] Yinhai Wang and N Nihan. Quantitative analysis on angle-accident risk at signalized intersections. In *World Transport Research, Selected Proceedings of the 9th World Conference on Transport Research (in print)*, 2001.

[172] Pablo L Durango-Cohen and Elaine C McKenzie. Trading off costs, environmental impact, and levels of service in the optimal design of transit bus fleets. *Transportation research procedia*, 23:1025–1037, 2017.

[173] Topon Paul and Hisashi Yamada. Operation and charging scheduling of electric

buses in a city bus route network. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 2780–2786. IEEE, 2014.

[174] Subramanya P Nageshrao, Jubin Jacob, and Steven Wilkins. Charging cost optimization for ev buses using neural network based energy predictor. *IFAC-PapersOnLine*, 50(1):5947–5952, 2017.

[175] Guang Wang, Xiaoyang Xie, Fan Zhang, Yunhuai Liu, and Desheng Zhang. bCharge: Data-driven real-time charging scheduling for large-scale electric bus fleets. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 45–55. doi: 10.1109/RTSS.2018.00015. ISSN: 2576-3172.

[176] Azhar Ul-Haq, Carlo Cecati, Kai Strunz, and Ehsan Abbasi. Impact of electric vehicle charging on voltage unbalance in an urban distribution network. *Intelligent Industrial Systems*, 1(1):51–60, 2015.

[177] Sanchari Deb, Kari Tammi, Karuna Kalita, and Pinakeshwar Mahanta. Impact of electric vehicle charging station load on distribution network. *Energies*, 11(1):178, 2018.

[178] M Kintner-Meyer, S Davis, S Sridhar, D Bhatnagar, S Mahserejian, and M Ghosal. Electric vehicles at scale–phase I analysis: High EV adoption impacts on the western US power grid. Technical report, 2020.

[179] Ralph Hermans, Mads Almassalkhi, and Ian Hiskens. Incentive-based coordinated charging control of plug-in electric vehicles at the distribution-transformer level. In *2012 American Control Conference (ACC)*, pages 264–269. IEEE, 2012.

[180] Yuping Lin, Kai Zhang, Zuo-Jun Max Shen, Bin Ye, and Lixin Miao. Multistage large-scale charging station planning for electric buses considering transportation network and power grid. *Transportation Research Part C: Emerging Technologies*, 107:423–443, 2019.

[181] Marc Gallet, Tobias Massier, and Daniel Zehe. Developing a large-scale microscopic model of electric public bus operation and charging. In *2019 IEEE Vehicle Power and Propulsion Conference (VPPC)*, pages 1–5. IEEE, 2019.

[182] Avishan Bagherinezhad, Alejandro D Palomino, Bosong Li, and Masood Parvania. Spatio-temporal electric bus charging optimization with transit network constraints. *IEEE Transactions on Industry Applications*, 2020.

[183] BFT Homepage. URL https://www.bft.org/. Ben Franklin Transit Services and Route Schedules.

[184] © OpenStreetMap Contributors. URL https://www.openstreetmap.org. Extracted Tri-Cities street map and metadata.

[185] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie WieBner. Microscopic traffic simulation using SUMO. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582. IEEE, 2018.

[186] SUMO User Documentation. URL https://sumo.dlr.de/docs/index.html. Updated: 2020-07-08.

[187] GridLAB-D Simulation Software. URL https://www.gridlabd.org/.

[188] Ronald A Howard. Dynamic programming and markov processes. 1960.

[189] D. J. White. A survey of applications of markov decision processes. *Journal of the Operational Research Society*, 44(11):1073–1096, 1993.

[190] Ronald Bjarnason, Alan Fern, and Prasad Tadepalli. Lower bounding klondike solitaire with monte-carlo planning. In *Nineteenth International Conference on Automated Planning and Scheduling*. Citeseer, 2009.

[191] Daniel Whitehouse, Edward J Powley, and Peter I Cowling. Determinization and information set monte carlo tree search for the card game dou di zhu. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, pages 87–94. IEEE, 2011.

LIST OF ABBREVIATIONS

$\bar{\varepsilon}_a$	Total energy cost of taking action a
β	Reward tradeoff hyper-parameter
ε^t	Time of use energy pricing at time t
$\gamma(c_i,t)$	Set of buses that can be charged at charger c_i at time t
ψ	Bus failure penalty hyper-parameter
$\rho(s,a)$	Reward function given action a taken in state s
B	Set of buses $\{b_1, b_2, ...\}$
C	Set of chargers $\{c_1, c_2, ...\}$
$g(s,a)$	Power grid impact score of an action a at in state s
$n_f(s)$	The number of failed buses in state s
AADT	annual average daily traffic
ADP	approximate dynamic programming
AFT	Accelerated Failure Model
ALT	A* Search with Landmarks
AMEXCLP	adjusted maximum expected covering location model
CAD	computer aided dispatch
CNN	convolutional neural networks
CPS	cyber physical systems
DSM	double standard model

DTMDP	Discrete Time Markov Decision Process
EE	event extraction
EMS	Emergency Medical Services
ERM	emergency response management
GLM	generalized linear models
GNN	graph neural networks
HCPS	Human-in-the-Loop Cyber-Physical Systems
LSCP	location set covering problem
LSTM	Long Short-Term Memory Neural Network
MALP	maximum availability location problem
MCLP	maximal covering location problem
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MEXCLP	maximum expected covering location model
MLE	Maximum Likelihood Estimation
MVA	motor vehicle accident
NFD	Nashville Fire Department
NLP	natural language processing
OSM	Open Street Maps
POS	part-of-speech

QPLSCP	queuing probabilistic location set covering problem
RL	reinforcement learning
RP	random paramter
SBAC	similarity based agglomerative clustering
SCPS	Societal-scale Cyber-Physical System
SMDP	Semi-Markov Decision Process
STRM	spatiotemporal resource management
SUMO	Simulation of Urban Mobility
TDOT	Tennessee Department of Transportation