Decision Automation for Electric Power Network Recovery

Yugandhar Sarkale, Saeed Nozhati, Edwin K. P. Chong, Fellow, IEEE, and Bruce R. Ellingwood

Abstract—Critical infrastructure systems such as electric power networks, water networks, and transportation systems play a major role in the welfare of any community. In the aftermath of disasters, their recovery is of paramount importance; orderly and efficient recovery involves the assignment of limited resources (a combination of human repair workers and machines) to repair damaged infrastructure components. The decision maker must also deal with uncertainty in the outcome of the resource-allocation actions during recovery. The manual assignment of resources seldom is optimal despite the expertise of the decision maker because of the large number of choices and uncertainties in consequences of sequential decisions. This combinatorial assignment problem under uncertainty is known to be NP-hard. We propose a novel decision technique that addresses the massive number of decision choices for large-scale real-world problems; in addition, our method also features an experiential learning component that adaptively determines the utilization of the computational resources based on the performance of a small number of choices. Our framework is closed-loop, and naturally incorporates all the attractive features of such a decision-making system. In contrast to myopic approaches, which do not account for the future effects of the current choices, our methodology has an anticipatory learning component that effectively incorporates lookahead into the solutions. To this end, we leverage the theory of regression analysis, Markov decision processes (MDPs), multi-armed bandits, and stochastic models of community damage from natural disasters to develop a method for near-optimal recovery of communities. Our method contributes to the general problem of MDPs with massive action spaces with application to recovery of communities affected by hazards.

Note to Practitioners—A significant amount of work has already been done to make communities more resilient against natural or anthropogenic hazards, which can include earthquakes, hurricanes, or a nuclear attack. However, the problem of recovery post-hazard has not been addressed, except in the case of small problems, because of the massive amount of computational resources required to solve the problem. To address this challenging problem, in this work we develop novel decision-making-under-uncertainty algorithms that should be of interest to practitioners in several fields—those that deal with real-world large-scale problem of selecting a single choice given a massive number of alternatives. Even though the applicability of our approach is not limited to the application of community recovery post-hazard, the techniques developed in this work is motivated by, and primarily addresses, post-hazard recovery planning and decision making. We use real-world models of an earthquake to simulate a shock to the community of Gilroy, California. Our automated decision making and optimization framework can be adapted to various types of disasters and different kinds of communities. Our framework accommodates sophisticated stochastic community damage models, handles the stochastic nature of outcomes of human or machine repair actions, takes into account the future impact of current actions, does not suffer from decision fatigue, and incorporates the current policies of the decision makers to automatically plan near-optimal recovery decisions. The current work focuses on optimizing a single recovery policy. In future research, we will extend our framework to simultaneously optimize multiple recovery policies. To further validate the efficacy of our algorithm, we plan to deal with massive stochastic sequential decision-making problems in different domains, future work will include testing the performance of our framework on commercial recommender systems like those used in YouTube and Amazon and large industrial control systems.

Index Terms—Rollout, Monte Carlo, Markov Decision Processes, large MDPs, resilience, sequential decision making, adaptive sampling.

I. INTRODUCTION

AUTOMATIC control systems have had a wide impact in multiple fields, including finance, robotics, manufacturing, and automobiles. Decision automation has gained relatively little attention, especially when compared to decision support systems where the primary aim is to aid humans in the decision-making process. In practice, decision automation systems often do not eliminate human decision makers entirely but rather optimize decision making in specific instances where the automation system can surpass human performance. In fact, human decision makers play a very important role in the selection of models, determining the set of rules, and developing methods that automate the decisions. Nonetheless, decision automation systems remain indispensable in applications where humans are unable to make rational decisions, whether because of the sheer complexity of the system, the enormity of the set of alternatives, or the massive amount of data that must be processed.

Our focus in this paper is to develop a framework that automates decisions for post-disaster recovery of communities. Designing such a framework is ambitious given that it should ideally possess several key properties such as the ability to incorporate sources of uncertainty in the models, information gained at periodic intervals during the recovery...
process, current policies of the decision-maker, and multiple decision objectives under resource constraints [1]. Our framework possesses these desired properties; in addition, our framework uses reasonable computational resources even for massive problems, has the lookahead property, and does not suffer from decision fatigue.

Civil infrastructure systems, including building infrastructure, power, transportation, and water networks, play a major role in the welfare of any community. The interdependence between the recovery of these networks post-hazard and community welfare addressing the issue of food-security, has been studied in [2]–[4]. In this study, we focus on electric power networks (EPNs) because almost all other infrastructure systems rely heavily on the availability of this network. In this study, a stochastic model characterizes the damage to the components of the EPN after an earthquake; similarly, the repair times associated with the repair actions are also given by a stochastic model.

The assignment of limited resources, including repair crews composed of humans and machines, to the damaged components of the EPN after a hazard can be posed as the generalized assignment problem (as defined in [5]), which is known to be NP-hard. Several heuristic methods have been demonstrated in the literature to address this problem [6].

Our Contribution: Instead of these classical methods, we employ Markov decision processes (MDPs) for the representation and solution of our stochastic decision-making problem, which naturally extends its appealing properties to our framework. In our framework, the solution to the assignment problem formulated as a MDP is computed in an online fashion using an approximate dynamic programming method known as rollout [7], [8]. This approach addresses the curse of dimensionality associated with large state spaces [9]. Furthermore, in our framework, the massive action space is handled by using a linear belief model, where a small number of candidate actions are used to estimate the parameters in the model based on a least-squares solution. Our method also employs adaptive sampling inspired by solutions to multi-armed bandit problems to carefully expend the limited simulation budget—a limit on the simulation budget is often a constraint while dealing with large real-world problems. Our approach successfully addresses the goal of developing a technique to deal with problems when the state and actions spaces of the MDP are jointly exceptionally large.

II. THE ASSIGNMENT PROBLEM

A. Problem Setup: The Gilroy Community

The description in this section comes mainly from [10]; we give a complete description here for the sake of being self-contained. We describe the EPN of Gilroy, California, which provides the context for our assignment problem. We also briefly discuss the earthquake model, the EPN restoration model, and the computational challenges associated with the assignment problem.

1) Network Characterization: Gilroy is a moderately-sized growing city located approximately 50 km south of the city of San Jose with a population of 48,821 at the time of the 2010 census [11]. The study area is divided into 36 gridded rectangles to define the community and encompasses 41.9 km² area of Gilroy with a population of 47,905. The average number of people per household in Gilroy in 2010 was 3.4, greater than the state and county averages [12]. A heat map of the population in the grid is shown in Fig. 1. This model has a resolution that is sufficient to study the methodology at the community level under hazard events. The community is susceptible to severe earthquakes on the San Andreas Fault (SAF).

The modeled EPN of Gilroy within the defined boundary is shown in Fig. 2. A 115 kV transmission line supplies the Llagas power substation, which provides electricity to the distribution system. The distribution line components are placed at intervals of 100 m and modeled from the power substation to the centers of the urban grid rectangles. If a component of the EPN is damaged, then along with the damaged EPN component, all the EPN components dependent on the damaged component are rendered nonfunctional or unavailable. If at least one EPN component serving a particular gridded rectangle is unavailable, the entire population of the gridded rectangle does not have electricity.

2) Seismic Hazard Simulation: In this study, we assume that a seismic event of moment magnitude $M_w = 6.9$ occurs at the closest points on the SAF projection to downtown Gilroy with an epicentral distance of approximately 12 km [10]; this event is similar to the devastating Loma Prieta earthquake of 1989 near Gilroy [14]. Ground motion prediction equations (GMPEs) determine the conditional probability of exceeding the ground motion intensity at specific geographic locations within Gilroy given a fault rupture mechanism and epicentral distance for the earthquake [4]. We use the Abrahamson et al. [15] GMPE to estimate the intensity measures (peak ground acceleration) throughout Gilroy.

3) Fragility and Restoration Assessment of EPN: Based on the ground-motion intensities using the above seismic model, we use seismic fragility curves presented in HAZUS-MH [16] to calculate the damages to the components of the EPN. Repair crews, replacement components, and equipment are considered as available units of resources to restore the damaged
components of the EPN following the hazard. One unit of resource (RU) is required to repair each damaged component [17]. To restore the EPN, we use the restoration times based on exponential distributions synthesized from HAZUS-MH, as summarized by expected repair times in Table I.

TABLE I  
EXPECTED REPAIR TIMES (UNIT: DAYS)

| Component                | Undamaged | Minor | Moderate | Extensive | Complete |
|--------------------------|-----------|-------|----------|-----------|----------|
| Electric sub-station      | 0         | 0.5   | 1        | 1         | 30       |
| Transmission line component | 0         | 0.5   | 1        | 1         | 2        |
| Distribution line component | 0         | 0.5   | 1        | 1         | 1        |

4) Challenges: The total number of modeled EPN components is equal to 327, denoted by \( L \). On average, about 60% of these components are damaged after the simulated earthquake event. At each decision epoch \( t = 0, 1, 2, \ldots \), the decision maker has to select the assignment of RUs to the damaged components; each component cannot be assigned more than one RU. Note that the symbol \( t \) is used to denote a discrete-index representing decision-epoch and is not to be confused with the actual time for recovery. Let the total number of damaged components at any \( t \) be represented by \( M_t \), and let the total number of RUs be equal to \( N \), where \( N \ll M_t \) (typically, the number of resource units for repair is significantly less than the damaged components). Then, the total number of possible choices for the assignment at any \( t \) is \( \binom{M_t}{N} \). For 196 damaged components and 29 RUs (15% of the damaged components), the possible choices at the first decision epoch is approximately \( 10^{34} \). In addition, the reassignment of all RUs is done when one component gets repaired so that the total number of choices at the second decision epoch is \( \binom{195}{28} \approx 10^{34} \).

Note that the repair time associated with a damaged component will depend on the level of damage, as determined from the fragility analysis described in Section II-A.3. This repair time is random and is exponentially distributed with expected repair times shown in Table I. Therefore, the outcomes of the repair actions are also random. It is difficult for a human decision maker to anticipate the outcome of repair actions when the outcomes are uncertain; therefore, planning with foresight is difficult. In fact, the problem is difficult to such an extent that assignment of RUs at the first decision epoch itself is challenging. Further, an additional layer of complexity to the problem is manifested owing to the level of damage at each location specified by a probabilistic model [16].

Because of the extraordinarily large number of choices, stochastic initial conditions, and the stochastic behavior of the outcome of the repair actions, our problem has a distinct flavor compared to the generalized assignment problem, and the classical heuristic solutions are not well-suited to this problem. In addition to dealing with these issues, the decision maker has to incorporate the dynamics and the sequential nature of decision making during recovery; thus, our problem represents a stochastic sequential decision-making problem. Last, we would also like our solution to admit most of the desirable properties previously discussed in the Section II. Our framework addresses all these issues.

B. Problem Formulation

In this section, we briefly discuss MDPs and the simulation-based representation pertaining to our problem, previously described in [18], and repeated here for the sake of continuity and completeness. We then specify the components of the MDP for our problem.

1) MDP Framework and Simulation-Based Representation: An MDP is a controlled stochastic dynamical process, widely used to solve disparate decision-making problems. In the simplest form, it can be represented by the 4-tuple \((S,A,T,R)\). Here, \( S \) represents the set of states, and \( A \) represents the set of actions. The state makes a transition to a new state at each decision epoch (represented by discrete-index \( t \)) as a result of taking an action. Let \( s,s' \in S \) and \( a \in A \); then \( T \) is the state transition function, where \( T(s,a,s') = P(s' | s,a) \) is the probability of transitioning to state \( s' \) after taking action \( a \) in state \( s \), and \( R \) is the reward function, where \( R(s,a,s') \) is the reward received after transitioning from \( s \) to \( s' \) as a result of action \( a \). In our problem, \( |S| \) and \( |A| \) are finite; \( R \) is real-valued and a stochastic function of \( s \) and \( a \) (deterministic function of \( s, a, \) and \( s' \)). Implicit in our presentation are also the following assumptions [19]: First-order Markovian dynamics (history independence), stationary dynamics (transition function is not a function of absolute time), and full observability of the state space (outcome of an action in a state might be random, but the state reached is known after the action is completed). The last assumption simplifies our presentation in that we do not need to take actions specifically to reinforce or modify our belief about the underlying state. We assume that recovery actions (decisions) can be taken indefinitely as needed, e.g., until all the damaged components are repaired (infinite-horizon planning). In this setting, we define a stationary policy as a mapping \( \pi : S \rightarrow A \). Our objective is to find an optimal policy \( \pi^\star \). For the infinite-horizon case, \( \pi^\star \) is defined as

\[
\pi^\star = \arg \max_{\pi} V_{\pi}(s_0),
\]
where

$$V^\pi(s_0) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1}) \mid s_0 \right]$$

(2)

is called the value function for a fixed policy $\pi$, and $\gamma \in (0, 1)$ is the discount factor. Note that in (1) we maximize over policies $\pi$, where at each decision epoch $t$ the action taken is $a_t = \pi(s_t)$. Stationary optimal policies are guaranteed to exist for the discounted infinite-horizon optimization criterion [20]. To summarize, our framework is built on discounted infinite-horizon discrete-time MDPs with finite state and action spaces, though the role $\gamma$ is somewhat tangential in our application.

We now briefly explain the simulation-based representation of an MDP [21]. Such a representation serves well for large state, action, and outcome spaces, which is a characteristic feature of many real-world problems; it is infeasible to represent $T$ and $R$ in a simple matrix form for such problems. A simulation-based representation of an MDP is a 4-tuple $(S, A, R, T)$, where $S$ and $A$ are as before. Here, $R$ is a stochastic real-valued function that stochastically returns a reward when input $s$ and $a$ are provided, where $a$ is the action applied in state $s$; $T$ is a simulator, which stochastically returns a state sample $s'$ when state $s$ and action $a$ are provided as inputs. We can think of $R$ and $T$ as callable library functions that can be implemented in any programming language.

2) MDP Specification for EPN Recovery Problem:

**States:** Let $s_t$ denote the state of our MDP at discrete decision epoch $t$: $s_t = (s_t^1, \ldots, s_t^l, a_t^1, \ldots, a_t^l)$, $s_t^l$ is the damage state of the $l$th damaged EPN component (the possible damage states are Undamaged, Minor, Moderate, Extensive, and Complete, as shown in Table I); and $a_t^l$ is the remaining repair time associated with the $l$th damaged component, where $l \in \{1, \ldots, L\}$. The state transition, and consequently the calculation of $r_{t}^{l}$ and $s_{t}^{l}$ at each $t$, is explained in the description of simulator $T$ below.

**Actions:** Let $a_t$ denote the repair action to be carried out at decision epoch $t$: $a_t = (a_t^1, \ldots, a_t^l)$, and $a_t^l \in \{0, 1\} \forall l,t$. When $a_t^l = 0$, no repair work is to be carried out at $l$th component. Conversely, when $a_t^l = 1$, repair work is carried out at the $l$th component. Note that $\sum_l a_t^l = N$, and $a_t^l = 0$ for all $l$ where $s_t^l$ is equal to Undamaged. Let $D_t$ be the set of all damaged components before a repair action $a_t$ is performed. Let $\mathcal{P}(D_t)$ be the powerset of $D_t$. The total number of possible choices at any decision epoch $t$ is given by $|\mathcal{P}(D_t)|$, where

$$|D_t| = M_t, \text{ and } |\mathcal{P}(D_t)| = \left( \frac{M_t}{N} \right).$$

(3)

**Initial State:** The stochastic damage model, previously described in Sections II-A2 and II-A3, is used to calculate the initial damage state $s_{0}$. Once the initial damage states of the EPN components are known, depending on the type of the damaged EPN component, the repair times $\rho_{0}^l$ associated with the damaged components are calculated using the mean restoration times provided in Table I.

**Simulator $T$:** Given $s_t$ and $a_t$, $T$ gives us the new (stochastic) state $s_{t+1}$. We define a repair completion as the instant when at least one of the locations where repair work is carried out is fully repaired. The decision epochs occur at these repair-completion times. A damaged component is fully repaired when the damage state of the component changes from any of the four damage states (except the Undamaged state) in Table I to the Undamaged state. Let us denote the inter-completion time by $r_t$, which is the time duration between decision epochs $t$ and $t + 1$, and let $\Delta_t = \{\rho_t^l : l \in \{1, \ldots, L\}, \rho_t^l > 0\}$. Then, $r_t = \min_{\Delta_t} \text{ and } \rho_{t+1} = \max(\rho_t - r_t, 0)$. Note that it is possible in principle for the repair work at two or more locations to be completed simultaneously, though this virtually never happens in simulation or in practice. When a damaged component is in any of the Minor, Moderate, Extensive, or Complete states, it can only transition directly to the Undamaged state. Instead of modeling the effect of repair via inter-transitions among damage states, the same effect is captured by the remaining repair time $\rho_t$.

Once a damaged component is restored to the Undamaged state, the RU$\text{s}$ previously assigned to it become available for reassignment to other damaged components. Moreover, the RUs at remaining locations, where repair work is unfinished, are also available for reassignment—the repair of a component is preemptive. It is also possible for a RU to remain at its previously assigned unrepaired location if we choose so. Because of this reason, preemption of repair work during reassignment is not a restrictive assumption; on the contrary, it allows greater flexibility to the decision maker for planning. Preemptive assignment is known to be particularly useful when an infrastructure system is managed by a central authority, an example of which is EPN [I].

Even if the same assignment is applied repeatedly to the same system state (let us call this the current system state), the system state at the subsequent decision epoch could be different because different components might be restored in the current system state, because of random repair times; i.e., our simulator $T$ is stochastic. When $M_t$ eventually becomes less than or equal to $N$ because of the sequential application of the repair actions (say at decision epoch $t_0$), the extra RUs are retired so that we have $M_t = N \forall t \geq t_0 + 1$, and the assignment problem is trivial. The evolution of the state of the community as a result of the nontrivial assignments is therefore given by $(s_0, \ldots, s_n)$.

**Rewards:** We define two reward functions corresponding to two different objectives:

In the first objective, the goal is to minimize the days required to restore electricity to a certain fraction ($\zeta$) of the total population ($p$); recall that for our region of study in Gilroy, $p = 47905$. We capture this objective by defining the corresponding reward function as follows:

$$R_1(s_t, a_t, s_{t+1}) = r_t,$$

(4)

where we recall that $r_t$ is the inter-completion time between the decision epochs $t$ and $t + 1$. Let $\hat{r}_t$ denote the decision epoch at which the outcome of repair action $a_{t-1}$ results in the restoration of electricity to $\zeta \cdot p$ number of people. The corresponding state reached resulting from action $a_{t-1}$ is $s_{\hat{r}_t}$, called the goal state for the first objective.

In the second objective, the goal is to maximize the sum (over all the discrete decision epochs $t$) of the product of the
total number of people with electricity \( (n_t) \) after the completion of a repair action \( a_t \) and the per-action time, defined as the time required \( (r_t) \) to complete the repair action \( a_t \), divided by the total number of days \( (t_{\text{tot}}) \) required to restore electricity to \( p \) people. We capture this objective by defining our second reward function as:

\[
R_2(s_t, a_t, s_{t+1}) = \frac{n_t \cdot r_t}{t_{\text{tot}}}. \tag{5}
\]

The terms in (5) have been carefully selected so that the product of the terms \( n_t \) and \( r_t/t_{\text{tot}} \) captures the impact of automating a repair action at each decision epoch \( t \), in the spirit of maximizing electricity benefit in a minimum amount of time. Let \( t_c \) denote the decision epoch at which the outcome of repair action \( a_{t_c} \) results in the restoration of electricity to the entire population. Then the corresponding goal state is \( s_{t_c} \).

Note that both \( t_c \) and \( t_e \) need not belong to the set \( \{0, \ldots, t_{-1}\} \), i.e., both \( s_{t_e} \) and \( s_{t_c} \) need not be reached only with a nontrivial assignment. Also, note that our reward function is stochastic because the outcome of each action is random.

**Discount factor \( \gamma \):** A natural consequence of sequential decision making is the problem of intertemporal choice [22]. The problem consists in balancing the rewards and costs at different decision epochs so that the uncertainty in the future choices can be accounted for. To deal with the problem, the MDP model, specifically for our formulation, accommodates a discounted utility, which has been the preferred method of tackling this topic for over a century. In this study, the discount factor \( \gamma \) is fixed at 0.99. We have selected a value closer to one because of the use of sophisticated stochastic models described in Sections II-A2 and II-A3, the uncertainty in the outcome of the future choices is modeled precisely via these models, and therefore we can evaluate the value of the decisions several decision-epochs in the future accurately to estimate the impact of the current decision. In our framework, it is possible to select a value closer to zero if the decision automation problem demands the use of simpler models. Moreover, the discounting can be done based on \( r_t \)—the real time required for repair in days (the inter-epoch time)—rather than the number of decision epochs, but this distinction is practically inconsequential for our purposes because of our choice of \( \gamma \) being very close to one.

Next we highlight the salient features of our MDP framework; in particular, we discuss the successful mitigation of the challenges previously discussed in Section II-A4.

Recall that we have a probability distribution for the initial damage state of the EPN components for a simulated earthquake. We generate multiple samples from this distribution to initialize \( s_0 \) and optimize the repair actions for each of the initial states separately. The outcomes of the optimized repair action for each initial state constitutes a distinct stochastic unfolding of recovery events (recovery path or recovery trajectory). We average over these recovery paths to evaluate the performance of our methods. In our framework, as long as sufficient samples (with respect to some measure of dispersion) are generated, we can appropriately deal with the probabilistic damage-state model.

Our sequential decision-making formulation also includes modeling the uncertainty in the outcome of repair actions. Thus, our framework can handle both stochastic initial conditions and stochastic repair actions.

We have formulated the impact of the current decisions on the future choices with exponential discounting. In addition, our sequential decision-making framework addresses the issue of making restoration decisions in stages, where feedback (information) gathered at each stage can play an important role in successive decision making. This is essentially a closed-loop design to compute decisions at each decision epoch.

Finally, we have defined the second reward function to account for multiple objectives (benefit of electricity \( (n_t) \) and per-action repair time \( (r_t/t_{\text{tot}}) \) ) without relaxing the constraint on the number of resources.

In the next section, we address the computational difficulties associated with solving the problem, show how to account for the current preferences and policies of the decision maker, and discuss the lookahead property.

### III. PROBLEM SOLUTION

**A. MDP Solution: Exact Methods**

A solution to an MDP is an optimal policy \( \pi^* \). There are several methods to exactly compute \( \pi^* \); here, we discuss the **policy iteration** algorithm because it bears some relationship with the rollout method, which we describe later.

Suppose that we have access to a nonoptimal policy \( \pi \). The value function for this policy \( \pi \) in (2) can be written as

\[
V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s' | s, \pi(s)) \cdot V^\pi(s') \quad \forall s \in S, \tag{6}
\]

where \( V^\pi \) can be calculated iteratively using the **Bellman’s update equation** or by solving a linear program [23]. This calculation of \( V^\pi \) is known as the policy evaluation step of the policy iteration algorithm. The \( Q \) value function of policy \( \pi \) is given by

\[
Q^\pi(s, a) = R(s, a) + \gamma \sum_{s'} P(s' | s, a) \cdot V^\pi(s'), \tag{7}
\]

which is the expected discounted reward in the future after starting in some state \( s \), taking action \( a \), and following policy \( \pi \) thereafter. An improved policy \( \pi' \) can be calculated as

\[
\pi'(s_t) = \arg\max_{a_t} Q^\pi(s_t, a_t). \tag{8}
\]

The calculation of an improved policy in (8) is known as the **policy improvement step** of the policy iteration algorithm. Even if the policy \( \pi' \) defined in (8) is nonoptimal, it is a strict improvement over \( \pi \) [20]. This result is called the **policy improvement theorem**. Note that the improved policy \( \pi' \) is generated by solving, at each state \( s \), an optimization problem with \( Q^\pi(s, \cdot) \) as the objective function. In the policy iteration algorithm, to compute the optimal policy \( \pi^* \), the policy evaluation and improvement steps are repeated iteratively until the policy improvement step does not yield a strict improvement.

Unfortunately, algorithms to compute the exact optimal policy are intractable for even moderate-sized state and actions spaces. Each iteration of the policy evaluation step requires \( O(|S|^3) \) time using a linear program and \( O(|S||A|) \) time using
Bellman’s update for a given $\pi^j$. In the previous example from Section II-A4 where the total number of damaged components after the initial shock is equal to 196, for the five damage states in Table I and two repair actions (repair and no-repair), $|S| = 5^{196}$ and the $|A| = 2^{196}$. Note that our state and action space is jointly massive. In our case, and for other large real-world problems, calculating an exact solution is practically impossible; even enumerating and storing these values in a high-end supercomputer equipped with state-of-the-art hardware is impractical.

B. Rollout: Dealing with Massive S

We now motivate the rollout algorithm in relation to our simulation-based framework and the policy iteration algorithm. When dealing with large $S$ and $A$, approximation techniques have to be employed given the computational intractability of the exact methods. A general framework of using approximation within the policy iteration algorithm is called approximate policy iteration—rollout algorithms are classified under this framework. In rollout algorithms, usually the policy evaluation step is performed approximately using Monte Carlo sampling and the policy improvement step is exact. The policy improvement step is typically exact, because approximating the policy improvement step requires the use of sophisticated techniques tailored to the specific problem being solved by rollout to avoid poor solution quality. A novel feature of our work is that we approximate both the policy improvement and policy evaluation step. The approximation to the policy improvement step is explained in Section III-C.

The policy evaluation step is approximated as follows. An implementable (in a programming sense) stochastic function (simulator) $\text{SimQ}(s_t, a_t, \pi, h)$ is defined in such a way that its expected value is $Q_\pi(s_t, a_t, h)$, where $Q_\pi(s_t, a_t, h)$ denotes a finite-horizon approximation of $Q_\pi(s_t, a_t)$. and $h$ is a finite number representing horizon length. In the rollout algorithm, $Q_\pi(s_t, a_t, h)$ is calculated by simulating action $a_t$ in state $s_t$ and thereafter following $\pi$ for another $h - 1$ decision epochs, which represents the approximate policy evaluation step. This is done for candidate actions $a_t \in A(s_t)$, where $A(s_t)$ is the set of all the possible actions in the state $s_t$. A finite-horizon approximation $Q_\pi(s_t, a_t, h)$ is unavoidable because, in practice, it is of course impossible to simulate the system under policy $\pi$ for an infinite number of epochs. Recall, however, that $V_\pi(s_t)$, and consequently $Q_\pi(s_t, a_t)$, is defined over the infinite horizon. It is easy to show the following result [21]:

$$|Q_\pi(s_t, a_t) - Q_\pi(s_t, a_t, h)| = \frac{\gamma^h R_{\text{max}}}{1 - \gamma},$$

where $R_{\text{max}}$ is the largest value of the reward function (either $R_1$ or $R_2$). The approximation error in (9) reduces exponentially fast as $h$ grows. Therefore, the $h$-horizon calculation approximately approximates the infinite-horizon version, for

we can always choose $h$ sufficiently large such that the error in (9) is arbitrarily small. The algorithm for rollout and the simulator is presented in Algorithms 1 and 2, respectively, where $\alpha = |A(s_t)|$, $a_t,i \in A(s_t)$ (here $i \in \{1, \ldots, \alpha\}$), and $\beta$ is the total number of samples available to estimate $Q_\pi(s_t, a_t, h)$. Algorithm 1 is also called a uniform rollout algorithm because $\beta$ samples are allocated to each action $a_t \in A(s_t)$ uniformly. In essence, rollout uses Monte-Carlo simulations in the policy evaluation step to calculate approximate $Q$ values; the quality of the approximation is often practically good enough even for small $h$.

Algorithm 1 Uniform_Rollout($\pi, h, \beta, s_t, A(s_t)$)

```
for $i = 1$ to $\alpha$
    for $j = 1$ to $\beta$
        $Q^{i,j} \leftarrow \text{SimQ}(s_t, a_t, i, \pi, h)$
    end for
    $Q_i(i) \leftarrow \text{Average}(Q^{i,j})$
end for
$k \leftarrow \text{arg max}_i Q_i$
return $a_t, k$
```

Algorithm 2 Simulator SimQ($s_t, a_t, \pi, h$)

```
t' = 0
$s'_0 \leftarrow s_t$
for $\lambda = 1$ to $h - 1$
    $s'_{t+\lambda} \leftarrow T(s'_t, a_t)$
    $r' \leftarrow R(s'_t, a_t, s'_{t+\lambda})$
end for
return $r'$
```

Rollout fits well in the paradigm of online planning. In online planning, the optimal action is calculated only for the current state $s_t$, reducing the computational effort associated with a large state space. Similarly, in our problem, we need to calculate repair actions for the current state of the EPN without wasting computational resources on computing repair actions for the states that are never encountered during the recovery process. Therefore, the property of online planning associated with Algorithm 1 is important for recovery, and even if the policy is (called the base policy in the context of Algorithm 1) is applied repeatedly (“rolled out”) for $h - 1$ decision epochs, we focus only on the encountered states as opposed to dealing with all the possible states (cf., [6]). In essence, for the recovery problem, rollout can effectively deal with large sizes of the state space because the calculation of the policy is amortized over time.

Consider the following example. In the context of online planning, for the sake of argument suppose that the action space has only a single action. Even for such a superficially trivial example, the outcome space can be massive. However, the representation of the problem in our framework limits the possible outcomes for any $(s, a)$ pair to $N$, bypassing the problem with the massive outcome space.
We can use existing policies of expert human decision makers as the base policy in the rollout algorithm. The ability of rollout to incorporate such policies is reflected by its interpretation as one-step of policy iteration, which itself starts from a nonoptimal policy \( \pi \). In fact, rollout as described here is a “one-step lookahead” approach (here, one-step lookahead means one application of policy improvement) [8]. Despite the stochastic nature of the recovery problem, the uniform rollout algorithm (as defined by Algorithm 1) computes the expected future impact of every action to determine the optimized repair action at each \( t \). Because the policy evaluation step is approximate, rollout cannot guarantee a strict improvement over the base policy; however, the solution obtained using rollout is never worse than that obtained using the base policy [8] because we can always choose the value of \( h \) and \( \beta \) such that the rollout solution is no worse than the base policy solution [25]. In practice, compared to the accelerated policy gradient techniques, rollout requires relatively few simulator calls (Algorithm 2) to compute equally good near-optimal actions [26].

C. Linear Belief Model: Dealing with Massive A

The last remaining major bottleneck with the rollout solution proposed above is that for any state \( s_t \), to calculate the repair action, we must compute the \( \text{argmax} \) of the \( Q \) function at \( s_t \). This involves evaluating the \( Q \) values for candidate actions and searching over the space of feasible actions. Because of online planning, we no longer deal with the entire action space \( A \) but merely \( A(s_t) \). For the example previously discussed in Section II-A4 even though this is a reduction from \( 2^{196} \) to \( (196)^{29} \), the required computation after the reduction remains substantial.

Instead of rolling out all \( a_t \in A(s_t) \) exhaustively, we train a set of parameters of a linear belief model (explained below) based on a small subset of \( A(s_t) \), denoted by \( \tilde{A}(s_t) \). The elements of \( \tilde{A}(s_t) \), denoted by \( \tilde{a}_t \), are chosen randomly and the size of the set \( A(s_t) \), denoted by \( \tilde{\alpha} \), is determined in accordance with the simulation budget available at each decision epoch \( t \). The simulation budget \( B \) at each decision epoch is divided among \( \tilde{\alpha} \) and \( \tilde{\beta} \) such that \( B = \tilde{\alpha} \cdot \tilde{\beta} \). In essence, based on the \( \tilde{a}_t \in \tilde{A}(s_t) \)—which corresponds to the assignment of \( N \) RUs to \( M_t \) damaged components according to (11)—the matrix \( \mathbf{H} \) is constructed. The vector \( \mathbf{y} \) is constructed by computing the \( Q \) values corresponding to \( \tilde{a}_t \) according to Algorithm 2.

We estimate the parameter vector \( \hat{\theta} \) by solving the least squares problem of minimizing \( \| \mathbf{y} - \mathbf{H} \hat{\theta} \|_2 \) with respect to \( \hat{\theta} \). We chose a least squares solution to estimate \( \hat{\theta} \) because least-squares solutions are well-established numerical solution methods, and if the noise is an uncorrelated Gaussian error, then \( \hat{\theta} \) estimated by minimizing \( \| \mathbf{y} - \mathbf{H} \hat{\theta} \|_2 \) is the maximum likelihood estimate. In our framework, the rank of \( \mathbf{H} \) is \((M_t \cdot N) - (N - 1)\). Therefore, the estimated parameter vector \( \hat{\theta} \), which consists of parameters \( \hat{\Theta}_{mn} \) and is calculated using the ordinary least squares solution, is not unique and admits an infinite number of solutions [28]. Even though \( \hat{\theta} \) is not unique, \( \tilde{\mathbf{y}} \) defined by the equation \( \tilde{\mathbf{y}} = \mathbf{H} \cdot \hat{\theta} \) is unique; moreover, the value of \( \| \mathbf{y} - \mathbf{H} \cdot \hat{\theta} \|_2 \) is unique. We can solve our least squares problem uniquely using either the Moore-Penrose pseudo-inverse or singular value decomposition by calculating the minimum-norm solution [29]. In this work, we have used the Moore-Penrose pseudo-inverse. Note that \( \alpha > (M_t \cdot N) - (N - 1) \) (the number of rows of the matrix \( \mathbf{H} \) is much greater than its rank).

Once the parameters \( \hat{\Theta}_{mn} \) are estimated, the optimum assignment of the RUs is calculated successively (one RU at a time) depending on the objective in (4) and (5). In the calculation of the successive optimum assignments of RU in Algorithm 3 let \( \hat{m} \) denote the assigned location at each RU assignment step; then all the estimated parameters corresponding to \( \hat{m} \) (denoted by parameters \( \hat{\Theta}_{m, \text{index}} \), where index \( \in \{1, \ldots, N\} \)) are set to \( \infty \) or \(-\infty \) depending on (4) and (5), respectively. This step ensures that only a single RU is assigned at each location. This computation is summarized in Algorithm 4. Similar to Algorithm 1, the assignment of \( \hat{\beta} \) samples to every action in \( \tilde{A}(s_t) \) is uniform.
In a regression problem, it is also possible to pose the rollout. The belief model is a function approximator for the value function, on the error performance of any algorithm (see [31]). The influence of estimating \( \hat{\beta} \) on \( \hat{\alpha} \) doverages computed using Algorithm 1, nonuniform allocation methods have been used in [35], necessitate sampling over the actions in (4). As the authors in [24] aptly point out, “increasing expressive power can lead to a surprisingly worse performance, which can make feature engineering a counterintuitive and tedious task.”

D. Adaptive Sampling: Utilizing Limited Simulation Budget

Despite implementing best software practices to code fast simulators and deploying the simulators on modern super-computers, the simulation budget \( B \) is a precious resource, especially for massive real-world problems. Algorithm [1] and consequently also Algorithm [3] is not only directly dependent on the speed of Algorithm [2](simulator) but also requires an accurate \( Q \) value function estimate to guarantee performance. Therefore, typically a huge sampling budget in the form of large \( \beta \) is allocated uniformly to every action \( \hat{a}_t \in \hat{A}(s_t) \). This naive approach decreases the value of \( \hat{a}_t \) (which is the size of the set \( \hat{A}(s_t) \)) consequently, the parameter vector \( \theta \) is trained on a smaller number of \( Q \) values. In practice, we would like to get a rough estimate of the \( Q \) value associated with every action in the set \( \hat{A}(s_t) \) and adaptively spend the remaining simulation budget in refining the accuracy of the \( Q \) values corresponding to the best-performing actions; this is the exploration vs. exploitation problem in optimal learning and simulation optimization problems [33]. Spending the simulation budget \( B \) in a nonuniform, adaptive fashion in the estimation of the \( Q \) value function would not only train the parameter vector \( \theta \) on a larger size of the set \( \hat{A}(s_t) \) via the additive model in (10) but also train the parameters \( \Theta_{mn} \) on \( Q \) values corresponding to superior actions (this is because in an adaptive scheme, \( B \) is allocated in refining the accuracy of only those actions that show promising performance), consequently refining the accuracy of the parameters. The nonuniform allocation of simulation budget is the experiential learning component of our method, which further enhances Algorithm [3].

An interesting closed-loop sequential method pertaining to drug discovery that bears some resemblance to the experiential learning component of our method is described in [35], where the alternatives (actions are called alternatives in their work) are selected adaptively using knowledge gradient (KG). Further, in their work, KG is combined with a linear-belief model, and the results are demonstrated on a moderate-sized problem. Unfortunately, the algorithms proposed in [35] are not directly applicable to our problem because the algorithms in [35] necessitate sampling over the actions in \( A(s_t) \), instead of \( \hat{A}(s_t) \).

Instead of uniformly allocating \( \hat{\beta} \) samples to each action in Algorithm [1] nonuniform allocation methods have been explored in the literature to manage the rollout budget [25]. An analysis of performance guarantees for nonuniform allocation

---

Algorithm 3 Uniform_Rollout w/ Linear_Belief

\[(\pi,h,\beta,s_t,\hat{A}(s_t))\]

1. Initialize \( a_t = [0] \)
2. for \( i = 1 \) to \( \hat{\alpha} \) do
   1. for \( j = 1 \) to \( \hat{\beta} \) do
      1. \( Q^{j^*} = \text{Sim}Q(s_t,\hat{a}_t,\pi,h) \) \( \triangleright \) See algorithm 2
   2. end for
3. end for
4. \( y(i) \leftarrow \text{Average}(Q^{j^*}) \) \( \triangleright \) With respect to \( j \)
5. \( \hat{\theta} \leftarrow OLS(y,H) \) \( \triangleright \) Ordinary least squares solution
6. for \( k = 1 \) to \( N \) do
   1. \( (\hat{\theta},\hat{\alpha}) \leftarrow \arg \min_{\alpha,n} \hat{\theta} \) \( \triangleright \) Min for (4) and max for (5)
   2. \( \alpha_i^{\hat{\alpha}} \leftarrow 1 \)
   3. for index = 1 to \( N \) do
      1. \( \bar{\theta}_{\hat{\alpha},\text{index}} \leftarrow \infty \) \( \triangleright \) \(-\infty\) for (5)
   4. end for
7. end for

return \( a_t \)

Our Algorithm [3] has several subtleties, as summarized in the following discussion.

The use of linear approximation for dynamic programming is not novel in its own right (it was first proposed by Bellman et al. [30]). The only similarity between the typical related methods (described in [24]) and our approach is that we are fitting a linear function over the rollout values—the belief model is a function approximator for the \( Q \) value function in Algorithm [1] whereas the primary difference is explained next.

Most of the error and convergence analyses for MDPs use the max-norm \( (\mathcal{L}_\infty \text{ norm}) \) to guarantee performance; in particular, the performance guarantee on the policy improvement step in (8) and the computation of \( a_t \) using rollout in Algorithm [1] are two examples. It is possible to estimate the parameters \( \hat{\theta} \) to optimize the \( \mathcal{L}_\infty \) norm by solving the resultant optimization problem using linear programming (see [31]). The influence of estimating \( \hat{\theta} \) to optimize the \( \mathcal{L}_\infty \) norm, when a linear function approximator is used to approximate the \( Q \) value function, on the error performance of any algorithm that falls in the general framework of approximate policy iteration is analyzed in [32]. Our approach is different from such methods because in our setting, the least squares solution optimizes the \( \mathcal{L}_2 \) norm, which we found to be advantageous.

Indeed, our solution shows promising performance. Three commonly used statistics to validate the use of the linear-belief model and the least squares solution in Algorithm [3] are as follows: residual standard error (RSE), R-squared \( (R^2) \), and F-statistic. The RSE for our model is \( 10^{-5} \), which indicates that the linear model satisfactorily fits the \( Q \) values computed using rollout. The \( R^2 \) value for our model is 0.99, which indicates that the computed features/predictors \( (\hat{\theta}) \) can effectively predict the \( Q \) values. The F-statistic is 4 (away from 1) for a large \( \alpha \) \( (\hat{\alpha} = 10^6) \); whereas, at each \( t \), the rank of \( H \) is never greater than 5850, which indicates that the features/predictors defined

---

1Note that \( B \) is fixed and depends on the simulator runtime and the computational platform on which the algorithm runs. Recall that \( B = \hat{\alpha} \cdot \beta \), and the larger the value of \( \beta \) required to guarantee performance, the smaller the value of \( \hat{\alpha} \).
of the rollout samples remains an active area of research [36]. However, we extend the ideas in [25] and [36], pertaining to nonuniform allocation, to Algorithm [3] based on the theory of multi-armed bandits.

In bandit problems, the agent has to sequentially allocate resources among a set of bandits, each one having an unknown reward function, so that a bandit objective [37] is optimized. There is a direct overlap between managing $B$ and the resource allocation problem in multi-armed bandit theory; the allocation of the simulation budget $B^*$ defined by the equation $B^* = B - \alpha$ sequentially to the state-action pair $(s_t, \tilde{a}_t)$ during rollout is equivalent to a variant of the classic multi-armed bandit problem [25].

In this study, we consider two bandit objectives: probable approximate correctness (PAC) and cumulative regret. In the PAC setting, the goal is to allocate budget $B^*$ sequentially so that we find a near-optimal $(\epsilon, \delta)$ action $\tilde{a}_t$ with high probability $(1 - \delta)$ when the budget $B^*$ is exhausted. Algorithm [1] is PAC optimal when $h$ and $\beta$ are selected in accordance with the fixed algorithm in [36]. For our decision-automation problem, the value of $\beta$ required to guarantee performance is typically large. Nonuniform allocation algorithms like median elimination are PAC optimal [38] (the median elimination algorithm is asymptotically optimal, so no other nonuniform resource-allocation algorithm can outperform the median elimination algorithm in the worst case). However, the choice of $(\epsilon, \delta)$ for the PAC objective is arbitrary; therefore, the PAC objective is not well-suited to our decision automation problem. Further, the parameters of the median elimination algorithm that guarantee performance are directly dependent on the $(\epsilon, \delta)$ pair.

The second common objective function in bandits problems mentioned earlier, cumulative regret is well-suited to our problem. During the optimization of cumulative regret, the budget $B^*$ is allocated sequentially in such a way that when the budget is exhausted, the expected total reward is very close to the best possible reward (called minimizing the cumulative regret). An algorithm in [39] called UCB1 minimizes the cumulative regret; in fact, no other algorithm can achieve a better cumulative expected regret (in the sense of scaling law). Usually, cumulative regret is not an appropriate objective function to be considered in nonuniform rollout allocation [40] because almost all common applications require finding the best (approximately) action $a_t$, whereas in our problem, we would like to allocate the budget nonuniformly so that the parameter vector $\hat{\theta}$ in Algorithm [3] is estimated in the most efficient way. Therefore, it is natural to allocate the computing budget so that the expected cumulative reward over all the $\tilde{a}_t$ ($Q$ values in the vector $\tilde{y}$ in Algorithm [3]) is close to the optimal value.

Based on the simulator runtime, the underlying computational platform, and the actual time provided by the decision maker to our automation system, suppose that we fix $B$ and in turn the size of the set $\tilde{A}(s_t)$. We exhaust a budget of $\tilde{\alpha}$ samples (one per action) from $B$ on getting rough estimates of the $Q$ value function for the entire set $\tilde{A}(s_t)$; the remaining budget $B - \tilde{\alpha}$ (denoted by $B^*$) is allocated adaptively using the UCB1 algorithm. This scheme of adaptively managing $B^*$ in Algorithm [3] is summarized in Algorithm [4].

Algorithm [4] alleviates the shortcomings of Algorithm [3] by embedding the experiential learning component using the UCB1 algorithm. The UCB1 algorithm assumes that the rewards lie in the interval $[0, 1]$. Satisfying this condition is trivial in our case because the rewards are bounded and thus can be always normalized so that they lie in the interval $[0, 1]$. As an important implementation of Algorithm [3] when we use Algorithm [4] in Algorithm [4] not only is $B^* \gg B$, but we can also select $\tilde{\alpha}$ larger than that in Algorithm [3] and train the parameter vector $\theta$ on a larger size of the set $\tilde{A}(s_t)$, which in turn will yield better estimates of $\tilde{\theta}$. Note that Algorithm [4] does not merely manage the budget $B^*$ adaptively (adaptive rollout), but it also handles massive action spaces through the linear belief model described in Section III-C (this is because Algorithm [4] is Algorithm [3] with the UCB1 step appended).

In essence, Algorithm [4] has three important steps: First, $Q$ values corresponding to $\tilde{\alpha}$ actions in the set $\tilde{A}(s_t)$ are computed. Second, the estimates for the $Q$ values corresponding to the most promising actions are refined by nonuniform allocation of the simulation budget using the UCB1 algorithm. Last, based on the ordinary least squares solution to calculate $\hat{\theta}$, the RU$s$ are assigned sequentially just like in Algorithm [3] described in Section III-C.

**Algorithm 4 Adaptive Rollout w/ Linear_Belief $(\pi, h, B^*, s_t, H)$**

1. **Initialize** $a_t = [0]$
2. **for** $i = 1$ to $\tilde{\alpha}$ **do**
   1. $\tilde{y}(i) \leftarrow \text{SimQ}(s_t, \tilde{a}_t, \tilde{a}_t, \pi, h)$  
   **See algorithm 2**
3. **end for**
4. $\text{Count} \leftarrow \tilde{\alpha}$
5. $\text{Count}_i \leftarrow [1]$  
   * Counts the number of samples assigned to the $i^{th}$ action
6. **while** $B^*$ is not zero  
   **UCB1 step**
7. **for** $i = 1$ to $\tilde{\alpha}$ **do**
   1. $d(i) \leftarrow \tilde{y}(i) + \sqrt{\frac{2 \ln(\text{Count}_i)}{\text{Count}_i(i)}}$
   **end for**
8. $\tau \leftarrow \arg\max_i d$
9. $\text{Count}_\tau \leftarrow \text{Count}_\tau + 1$
10. $\tilde{y}(\tau) \leftarrow \frac{\text{SimQ}(s_t, a_t, \pi, h) + \tilde{\alpha} \cdot \text{SimQ}(s_t, \tilde{a}_t, \pi, h)}{\text{Count}_\tau}$
11. $B^* \leftarrow B^* - 1$
12. **end while**
13. $\hat{\theta} = \text{OLS}(\tilde{y}, H)$  
   * Ordinary least squares solution
14. **for** $k = 1$ to $N$ **do**
   1. $(\tilde{m}, \tilde{h}) \leftarrow \arg\max_{m, h} \hat{\theta}$  
      * Min for [4] and max for [4]
   2. $a_t^{\tilde{h}} \leftarrow 1$
   **for index = 1** to $N$ **do**
   3. $\tilde{\Theta}_{\tilde{h}, \text{index}} \leftarrow -\infty$  
      * $\infty$ for [4]
   **end for**
16. **end for**
17. **return** $a_t$
IV. SIMULATION RESULTS: MODELING GILROY RECOVERY

We simulate 25 different damage scenarios (stochastic initial conditions) for each of the figures presented in this section. Calculation of the recovery for a single damage scenario is computationally expensive. Nevertheless, multiple initial conditions are generated to deal with the stochastic earthquake model as discussed in Section I-B2. In case of both Objective 1 and Objective 2, corresponding to \( R_1 \) and \( R_2 \) respectively, there will be a distinct recovery path for each of the initial damage scenarios. To present the results for Objective 1, we do not explicitly show the recovery trajectories. We are only interested in the number of days it takes to provide maximum benefit in the sense of optimizing \( R_1 \). Therefore, the results are presented in terms of a cumulative moving average plot. In Objective 2, for both Algorithm 3 and Algorithm 4, the recovery computed using these algorithms outperform the base policy for every single scenario.

There are several candidates for determining the base policy to be used in the simulation. For a detailed discussion on these candidates in post-hazard recovery planning, see [13]. For the simulations presented in this study, a random base policy is used. The total number of RUs are capped at 15% of the damaged components for each scenario. The maximum number of damaged components in any scenario encountered in this study is 205, i.e., the size of the assignment problem at any \( t \) is less than \( 10^{37} \). The simulators have a runtime of \( 10^{-5} \) s when \( h = 1 \), and this runtime varies with the parameter \( h \). The deeper we rollout the base policy in any variation of the rollout algorithm, the larger the simulation time per-action and the smaller the action space covered to train our parameters.

For Algorithm 3 and the computational platform (AMD EPYC 7451, 2.3 GHz, and 96 cores), the value of \( \beta \) is capped at 100 and the value of \( \tilde{\alpha} \) is capped at \( 10^6 \). Note that it is possible to parallelize Algorithm 3 at two levels. The recovery of each damage scenario can be computed on a separate processor, and then their average can be calculated. Further, Algorithm 5 offers the opportunity to parallelize over \( \hat{A}(s_i) \) because a uniform budget can be allocated to a separate processor to return the average \( Q \) value for each \( a(s_i) \). On the contrary, the allocation of budget \( B^* \) in Algorithm 4 is sequential, and only a single \( Q \) value corresponding to the allocated sample is evaluated (see the UCB1 step in Algorithm 4). Based on the updated \( Q \) value (calculation of \( \hat{y}(\pi) \) in Algorithm 4), further allocation is continued until the budget (\( B^* \)) is exhausted. Therefore, barring the rough estimates at the first iteration, Algorithm 4 cannot be parallelized for allocation. However, just like Algorithm 5, each processor can compute the recovery for a distinct initial condition (\( s_0 \)) separately. Because of reduction in the parallelization in Algorithm 4, the solutions, even though high-quality, are computed at a slower rate. For our simulations, \( B^* \leq 9 \cdot 10^5 \) and \( \tilde{\alpha} \leq 10^5 \) in Algorithm 4.

Fig. 3 compares the performance of Algorithm 3 with the base policy for Objective 1. For the simulations, \( \zeta = 0.8 \); the goal is to calculate recovery actions so that 80% of the population has electricity in minimum time. The figure depicts the cumulative moving average plot of the number of days required to achieve Objective 1. The cumulative moving average plot is computed by averaging the days required to reach the threshold for the total number of scenarios depicted on the X-axis of Fig. 3. The cumulative moving average is used to smooth the data. As the number of scenarios increases in order to represent the stochastic behaviour of the earthquake model accurately, our algorithm saves about half a day over the recovery computed using the base policy. We manage to achieve the performance at scale (without any restriction on the number of workers, whereas all our earlier related work (see [1], [3], [13], and [18]) put a cap on the number of RUs); in addition, this performance is achieved on a local computational machine.

Fig. 4 compares the performance of Algorithm 3 with the base policy for Objective 2. The recovery path (trajectories) for both the base policy and Algorithm 3 are computed by calculating the average of 25 different recoveries over different initial conditions. The recovery path represents the number of people that have electricity after a given amount of time (days) because of recovery actions. Evaluating the performance of our algorithm in meeting Objective 2 (defined in Section I-B2) boils down to calculating the area under the curve of our plots normalized by the total time for the recovery (12 days). The area represents the product of the number of people who have electricity after the completion of each repair action (\( n_t \)) and the time required in days for the completion of that action (the inter-completion time \( r_t \)). A larger value of this area (\( \sum n_t \cdot r_t \)) normalized by total time to recovery \( (t_{tot}) \) represents the situation where a greater number of people were benefitted as a result of the recovery actions. Normalization of the area \( \sum n_t \cdot r_t \) with the time to recovery \( (t_{tot}) \) is important because the amount of time required to finish the recovery \( (t_{tot}) \) using the base policy and rollout with linear belief can be different. It is evident by visual inspection of the figure that recovery with Algorithm 3 results in more benefit than its base counterpart; however, calculating \( \sum n_t \cdot r_t / t_{tot} \) for the plots is necessary when the recovery achieved by the algorithms intersect at several points (see [12]), a behaviour commonly seen with the rollout algorithm because of the lookahead property.

Fig. 5 compares the performance of Algorithm 4 with the
Because the entire framework is closed-loop, Algorithm 4 provides for better recovery actions at each decision epoch. \(B^*\) is particularly appealing in this regard: parallel rollout and policy switching [43]. In parallel rollout, just like in [35], an enhanced recovery. Essentially, the small improvements squeezed at the earlier stages set a better platform for these algorithms to further exploit the anticipatory and experiential learning components at a later point in the recovery.

**V. CONCLUSION**

In this work, we presented a novel, systematic approach to MDPs that have jointly massive finite state and action spaces. When the action space consists of large number of discrete actions, the method of choice has been to embed these actions in continuous action spaces [41], where deep reinforcement learning techniques have shown promising performance on \(|A| \approx 10^6\). In contrast, in this study, we present a unique approach to address the problem, where the size of the discrete action space that we consider is significantly large than that in [41].

We studied an intricate real-world problem, modeled it in our framework, and demonstrated the powerful applicability of our algorithm on this challenging problem. The community recovery problem is a stochastic combinatorial decision-making problem, and the solution to such decision-making problems is critically tied with the welfare of communities in the face of ever-increasing natural and anthropogenic hazards. Our modeling of the problem is general enough to accommodate the uncertainty in the hazard models and the outcome of repair actions. Ultimately, we would like to test the techniques developed in this work on other real-world problems, e.g., large recommender systems (like those in use with the organizations YouTube and Amazon) and large industrial control systems.

**Ongoing Work:** In our work on post-hazard community management (see [1]–[3], [42], [43], and [13]), including this study, we have been focusing on obtaining solutions by the use of a single base policy. Currently, we are developing a framework where we leverage the availability of multiple base polices in the aftermath of hazards. Two algorithms are particularly appealing in this regard: parallel rollout and policy switching [43]. In parallel rollout, just like in [35], the optimization is done over the entire set \(A(s_t)\). In our
ongoing work, we are formulating a non-preemptive stochastic scheduling framework, where the size of set $A(s_t)$ grows linearly with the number of RUs, which circumvents the issue of large action spaces. In addition, we are also exploring heuristic search algorithms to guide the stochastic search, i.e., adaptively select the samples of the parallel rollout algorithm.

There, we consider several infrastructure systems in a community, such as building structures, EPN, WN, and food retailers simultaneously (all these systems are inter-connected), and we compute the recovery of the community post-hazard.

REFERENCES

[1] S. Nozhati, Y. Sarkale, E. K. P. Chong, and B. R. Ellingwood, “Optimal stochastic dynamic scheduling for managing community recovery from natural hazards,” Reliab. Eng. & Syst. Saf., vol. 193, p. 106627, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832019305158

[2] S. Nozhati, Y. Sarkale, B. R. Ellingwood, E. K. P. Chong, and H. Mahmoud, “A modified approximate dynamic programming algorithm for community-level food security following disasters,” in Proc. 9th Int. Congr. Environ. Model. and Softw. (iEMS'18), Ft. Collins, CO, Jun 2018.

[3] “An approximate dynamic programming approach to food security of communities following hazards,” in Proc. 13th Int. Conf. Appl. Statist. and Probab. and Civ. Eng. (ICASP 13), Seoul, S. Korea, May 2019. [Online]. Available: http://Fs-space.snu.ac.kr/handle/10371/153966

[4] S. Nozhati, N. Rosenheim, B. R. Ellingwood, H. Mahmoud, and M. Perez, “Probabilistic framework for evaluating food security of households in an aftermath of a disaster,” Struct. Infrastructure Eng., vol. 15, no. 8, p. 1066, 2019.

[5] M. L. Fisher, R. Jaikumar, and L. N. Van Vassenhove, “A multiplier adjustment method for the generalized assignment problem,” Manag. Sci., vol. 32, no. 9, pp. 1095–1103, 1986. [Online]. Available: https://doi.org/10.1287/mnsc.32.9.1095

[6] D. B. Shmoys and E. Tardos, “An approximation algorithm for the generalized assignment problem,” Math. Program., vol. 62, no. 1, pp. 461–474, Feb 1993. [Online]. Available: https://doi.org/10.1007/BF01585178

[7] D. Silver and J. Veness, “Monte-Carlo planning in large POMDPs,” in Adv. Neural Inf. Process. Syst. 23, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 2164–2172. [Online]. Available: http://papers.nips.cc/paper/4031-monte-carlo-planning-in-large-pomdps.pdf

[8] D. P. Bertsekas and D. A. Castanon, “Rollout algorithms for stochastic dynamic scheduling,” in IEEE Trans. on Aut. Control, vol. 35, no. 1, pp. 89–110, Jan 1990.

[9] W. B. Powell, Approximate Dynamic Programming: Solving the curses of dimensionality. John Wiley & Sons, 2007, vol. 703.

[10] S. Nozhati, B. R. Ellingwood, H. Mahmoud, and J. W. van de Lindt, “Identifying and analyzing interdependent critical infrastructure in post-earthquake urban reconstruction,” in Proc. of the 11th Natl. Conf. in Earthq. Eng. Los Angel., CA: Earthq. Eng. Res. Inst., Jun 2018.

[11] “The Association of Bay Area Governments (City of Gilroy Annex),” in Adopted and Background Report, Dec 2014. [Online]. Available: https://www.hsdl.org/?view&did=11343

[12] M. Ouyang, L. Dueñas-Osorio, and X. Min, “A three-stage resilience analysis framework for urban infrastructure systems,” Struct. Saf., vol. 36, pp. 23–31, 2012.

[13] Y. Sarkale, S. Nozhati, E. K. P. Chong, B. R. Ellingwood, and H. Mahmoud, “Solving Markov decision processes for network-level post-hazard recovery via simulation optimization and rollout,” in 2018 IEEE Int. Conf. Autom. Sci. and Eng. (CASE), Aug 2018, pp. 906–912.

[14] M. L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, 1st ed. N. Y., NY, USA: John Wiley & Sons, Inc., 1994.

[15] R. A. Howard, Dynamic Programming and Markov Processes. Camb., MA: MIT Press, 1960.

[16] Department of Homeland Security, Emergency Preparedness and Response Directorate, FEMA, Mitigation Division, Multi-hazard Loss Estimation Methodology, Earthquake Model: HAZUS-MH MR1, Advanced Engineering Building Module, Wash., DC, Jan 2003. [Online]. Available: https://www.hsdl.org/?view&did=11343

[17] Y. Sarkale, S. Nozhati, E. K. P. Chong, B. R. Ellingwood, and H. Mahmoud, “Solving Markov decision processes for network-level post-hazard recovery via simulation optimization and rollout,” in 2018 IEEE Int. Conf. Autom. Sci. and Eng. (CASE), Aug 2018, pp. 906–912.

[18] Y. Sarkale, S. Nozhati, E. K. P. Chong, B. R. Ellingwood, and H. Mahmoud, “Solving Markov decision processes for network-level post-hazard recovery via simulation optimization and rollout,” in 2018 IEEE Int. Conf. Autom. Sci. and Eng. (CASE), Aug 2018, pp. 906–912.
[39] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” Mach. Learn., vol. 47, no. 2, pp. 235–256, May 2002. [Online]. Available: https://doi.org/10.1023/A:1013689704352

[40] S. Bubeck, R. Munos, and G. Stoltz, “Pure exploration in multi-armed bandits problems,” in Algorithmic Learning Theory, R. Gavaldà, G. Lugosi, T. Zeugmann, and S. Zilles, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 23–37.

[41] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, “Deep reinforcement learning in large discrete action spaces,” arXiv preprint arXiv:1512.07679, 2015.

[42] S. Nozhati, B. Ellingwood, H. Mahmoud, Y. Sarkale, E. K. P. Chong, and N. Rosenheim, “An approximate dynamic programming approach to community recovery management,” in Eng. Mech. Inst. Conf. (EMI 2018), Camb. - Boston, MA, May – Jun 2018.

[43] H. S. Chang, R. Givan, and E. K. P. Chong, “Parallel rollout for online solution of partially observable Markov decision processes,” Discret. Event Dyn. Syst., vol. 14, no. 3, pp. 309–341, Jul 2004. [Online]. Available: https://doi.org/10.1023/B:DISC.0000028109.78776.c4