Mitigating Side Effects in Multi-Agent Systems Using Blame Assignment

Pulkit Rustagi[1] and Sandhya Saisubramanian[1]

***Abstract*— When independently trained or designed robots are deployed in a shared environment, their combined actions can lead to unintended negative side effects (NSEs). To ensure safe and efficient operation, robots must optimize task performance while minimizing the penalties associated with NSEs, balancing individual objectives with collective impact. We model the problem of mitigating NSEs in a cooperative multi-agent system as a bi-objective lexicographic decentralized Markov decision process. We assume independence of transitions and rewards with respect to the robots' tasks, but the joint NSE penalty creates a form of dependence in this setting. To improve scalability, the joint NSE penalty is decomposed into individual penalties for each robot using credit assignment, which facilitates decentralized policy computation. We empirically demonstrate, using mobile robots and in simulation, the effectiveness and scalability of our approach in mitigating NSEs.**

I. INTRODUCTION

In many real-world environments, the actions of individual robots, even when operating independently, can have significant and often unintended consequences on the broader environment and the collective system behavior. Traditional approaches to agent design and training focus on optimizing performance in isolation, prioritizing task completion while overlooking potentially harmful effects their actions can collectively produce, such as *negative side effects* (NSEs). In multi-agent systems, NSEs are the unintended and undesirable outcomes of collective system behavior, caused by the incompleteness of models used for decision making [1], [20], [21]. While the NSEs are difficult to identify before deployment, mitigating them is crucial for overall system efficiency and safety.

This paper focuses on mitigating NSEs in cooperative multi-agent settings where the robots produce no (or negligible) NSEs when executing their policy in isolation, but their joint policy results in NSEs. Consider warehouse robots that optimize moving shelves between two locations. Each robot's model provides the necessary information, including reward and transition dynamics, to complete its task optimally. The models lack information about the effects of robots' joint actions in the environment, such as a narrow corridor being blocked for human access when multiple robots simultaneously move large shelves through it. Thus, even when the robots are adept at their tasks and produce no NSEs when acting in isolation, their joint actions are undesirable.

Mitigating NSEs in multi-agent settings is challenging because NSEs and corresponding penalties are defined over joint actions, creating inter-agent dependencies that did not exist for task completion. Further, the computational complexity of mitigating NSEs, without significantly affecting task completion, increases with the number of agents in the system. Prior research on NSEs primarily target single-agent settings [10], [11], [20], [21], [29] or treat other agents as part of the environment [1]. These techniques do not apply to multi-agent settings as they ignore the agent interactions that produce NSEs. A recent approach uses distributed constraint optimization with Q-learning to mitigate multi-agent NSEs [6]. However, it is not scalable since distributed constraint optimization is intractable for large problems [7].

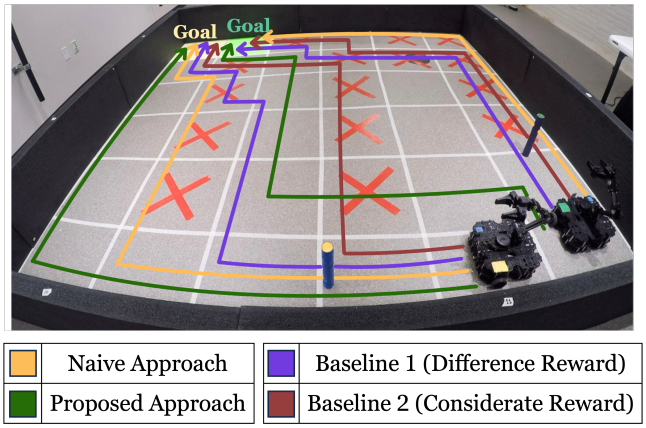


Fig. 1: Comparison of the paths taken by two TurtleBots performing delivery tasks in our indoor setup. The robots, initially unaware about the side effects of their actions, receive a joint penalty when one or both are in NSE states marked by X. The robots must update their behavior to complete tasks while mitigating NSEs.

We formulate the problem of mitigating NSEs in a multi-agent system as a decentralized Markov decision process (DEC-MDP) with two objectives and a *lexicographic ordering* over them. A lexicographic formulation is an intuitive way to model problems with an inherent ordering over objectives [19], [27]. The *primary objective* for each agent is to optimize its assigned task. We assume independence of transition and rewards with respect to the agents' assigned tasks [2]. That is, each agent's reward for task completion is determined by its local state and actions and the overall reward for the system is the sum of individual agent's rewards, and agents' state transitions are affected only by their own actions. The *secondary* objective is to minimize the NSEs. Agents have no prior knowledge of NSEs and only observe the *joint penalty*—a global value based on *all* agents' actions. While they can independently compute

[1]The authors are with the Collaborative Robotics and Intelligent Systems (CoRIS) Institute, Oregon State University, Corvallis OR 97331, USA {rustagip, sandhya.sai}@oregonstate.edu

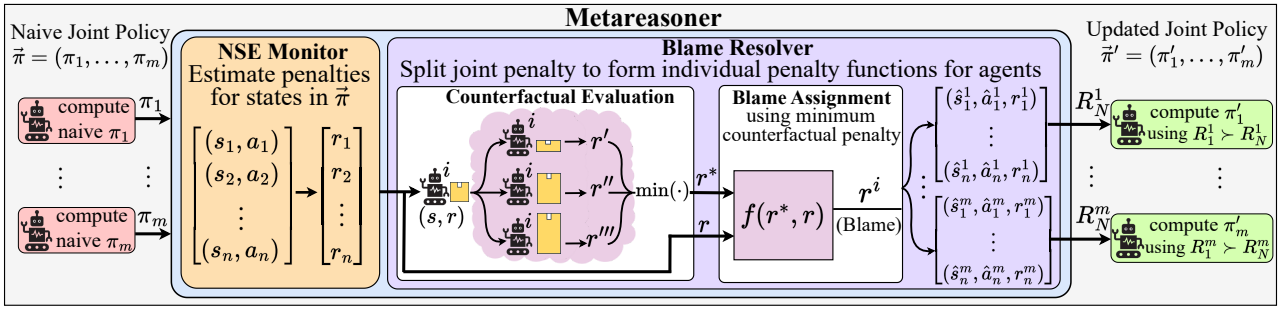


Fig. 2: Overview of our solution approach. Agents independently compute policies to complete tasks described by R_1^i (Naive policy). The NSE Monitor computes the NSE penalty for the joint policy $\vec{\pi}$. The Blame Resolver assigns a blame value for each agent, by evaluating counterfactual scenarios specific to each agent, as illustrated with warehouse robots handling different-sized boxes. Individual penalty functions R_N^i are derived for each agent, based on the estimate blame. Agents then recompute their policies by solving the bi-objective problem with $R_1^i \succ R_N^i$, where $\succ$ denotes preference ordering over the objectives and their associated reward functions.

policies for their tasks, mitigating NSEs requires addressing the dependence induced by the joint penalty (Fig. 1).

We present a *metareasoning* [4], [32] approach to detect and mitigate NSEs. The metareasoner is a centralized entity that monitors the agents' behaviors and has two components: (1) *NSE Monitor* that estimates the NSEs associated with agents' *joint policy*, and outputs the corresponding penalty, and (2) a *Blame Resolver* that decomposes the joint penalty into individual agent penalties, using our algorithm *Reward Estimation using Counterfactual Neighbors (RECON)*. It is assumed that the NSE monitor has access to a model of NSEs and the associated penalty, either provided as part of its design or acquired through human feedback. The blame resolver, using RECON, performs blame (credit) assignment to determine each agent's relative contribution towards the NSE, based on which the joint NSE penalty is decomposed into local penalties for each agent [3], [15], [17]. This decomposition facilitates augmenting the model of each agent with NSE information via a penalty function, thereby enabling decentralized policy computation to mitigate NSEs.

Our solution framework uses a four-step approach to mitigate NSEs (Fig. 2): (1) the agents first calculate optimal policies to complete their assigned tasks (referred to as naive policies); (2) the NSE monitor estimates the NSE penalty associated with the joint policy of the agents; (3) the blame resolver then decomposes the joint penalty into individual penalties for each agent, using blame (credit) assignment; and (4) the agents recompute their policies by solving a decentralized, bi-objective problem, with the prescribed reward for their task and the estimated local penalty for NSE, using lexicographic value iteration [27]. Our experiments using mobile robots and in simulation demonstrate the efficiency and scalability of our approach in mitigating NSEs by updating the policies of a subset of agents in the system.

II. BACKGROUND

Decentralized Markov Decision Processes (Dec-MDPs) are widely used to model decentralized multi-agent decision-making problems [9]. A Dec-MDP is defined by the tuple $\langle \mathcal{A}, S, A, T, R \rangle$ with $\mathcal{A}$ denoting the finite set of k agents in the system; $S = \hat{S}_1 \times \ldots \times \hat{S}_k$ denoting the joint state space, where $\hat{S}_i$ denotes the state space of agent i; $A = \hat{A}_1 \times \ldots \times \hat{A}_k$ denoting the joint action space, where $\hat{A}_i$ denotes agent i's actions; $T : S \times A \times S \rightarrow [0, 1]$ denoting the transition function; and R denoting the reward function. We use s_{-i} to denote joint state excluding agent i. A joint policy $\vec{\pi} = (\pi_1, ..., \pi_k)$ is a set of policies, one for each agent in the system. A Dec-MDP with *transition independence and reward independence* [2] is a class of problems in which agents operate independently but are tied together through a reward structure that depends on all of their execution histories:

$$T(\hat{s}'_i | s, a, s'_{-i}) = T_i(\hat{s}'_i | \hat{s}_i, \hat{a}_i), \forall i \in \mathcal{A},$$

$$R(s, a, s') = \sum_{i \in \mathcal{A}} R_i(\hat{s}_i, \hat{a}_i, \hat{s}'_i).$$

A transition and reward-independent Dec-MDP can be solved as k single agent MDPs [9].

Lexicographic MDP (LMDP) LMDPs are particularly convenient to model problems with potentially competing objectives and an inherent lexicographic ordering over them, such as ours where task completion is prioritized over NSE mitigation [20], [27]. An LMDP is denoted by the tuple $M = \langle S, A, T, \mathbf{R}, o \rangle$ with finite set of states S, finite set of actions A, transition function denoted by $T : S \times A \times S \rightarrow [0, 1]$ and a vector of reward functions $\mathbf{R} = [R_1, ..., R_k]^T$ with $R_i : S \times A \rightarrow \mathbb{R}$, and o denotes the strict preference ordering over the k objectives. The set of value functions is denoted by $\boldsymbol{V} = [V_1, ..., V_k]^T$, with V_i corresponding to o_i,

$$\boldsymbol{V}^\pi(s) = \mathbf{R}(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') \boldsymbol{V}^\pi(s'), \forall s \in S.$$

A slack $\boldsymbol{\Delta} = \langle \delta_1, ..., \delta_k \rangle$ with $\delta_i \geq 0$, denotes the acceptable deviation from the optimal expected reward for objective o_i so as to improve the lower priority objectives. Objectives are processed in the lexicographic order. The set of restricted actions for o_{i+1} is $A_{i+1}(s) = \{a \in A | \max_{a' \in A_i} Q_i(s, a') - Q_i(s, a) \leq \eta_i\}$, where $\eta_i = (1 - \gamma)\delta_i, \gamma \in [0, 1)$. Refer [27] for a detailed background on LMDP.

TABLE I: Overview of different credit assignment techniques for mitigating NSEs. A technique is scalable if it is computationally inexpensive to generate counterfactuals for problems with a few hundreds of agents. "–" indicates that the approach cannot be applied directly but can be modified to meet the requirements of our setting.

Credit Assignment Technique	Scalable	Supports decentralized planning	Compatible with heterogeneous agents	Can generate NSE-specific counterfactuals
Shapley Value	✗	✓	✓	✗
Difference Reward	✓	✓	✓	–
Action Not Taken	✓	✓	✓	✗
D++	✗	✓	✓	✗
Wonderful Life Utility	✗	✓	✗	✗
Value Decomposition	✗	✗	✓	✗
Our Approach	✓	✓	✓	✓

Credit Assignment It is a popular approach to measure the contribution of an agent to team performance so as to convert a joint reward into individual agent rewards [15]. Difference Reward [17] and its variants such as D++ [18], Wonderful Life Utility [15], perform credit assignment by comparing the joint rewards before and after the agent is removed from the system. Another type of credit assignment uses Shapley value to compute a value function for each agent by considering all combinations of possible agent interactions [3], [14], [22], [23]. Table I summarizes the characteristics of different credit assignment techniques. While some of these techniques assess individual contributions, they do not support decentralized planning due to their reliance on other agents' policies. The existing methods calculate counterfactuals by considering all state features, including those that are not associated with NSEs, which leads to incorrect blame assignment for NSEs.

III. PROBLEM FORMULATION

Problem Setting. Consider m agents independently performing their assigned tasks which is their primary objective $o_1 = \{o_1^1, ..., o_1^m\}$. Each agent operates based on an MDP $\hat{M}$ that contains all the information necessary to optimize o_1 but lacks information about the joint effects of agents' actions when it is unrelated to task completion. Due to the limited fidelity of $\hat{M}$, *negative side effects* (NSEs) occur when agents execute their policies simultaneously. A meta-level process, *metareasoner*, monitors and controls the agents' performance (object-level process) [26], [32], by providing a penalty if agents' actions produce side effects. The NSE penalty is determined by a function R_N which is known to the metareasoner (either by design or learned using human feedback) but unknown to the agents.

We assume the following about the NSEs: (1) agents acting in isolation produce no (or negligible) NSEs, but their joint actions produce NSEs that must be mitigated; (2) agents have no prior knowledge about NSEs of the joint actions except for the penalty assigned by the metareasoner; and (3) NSEs are undesirable but do not hinder task completion. We address the problem of mitigating NSEs by augmenting the agents' models with secondary reward functions that correspond to NSE penalties. We target settings where the completion of the agents' assigned tasks (o_1) are *prioritized* over minimizing NSEs (o_2), $o_1 \succ o_2$.

MASE-MDP. The problem of mitigating NSEs in a cooperative multi-agent system is formulated as a Dec-MDP with two objectives and a lexicographic ordering over them.

Definition 1. *A multi-agent side effects MDP (MASE-MDP) is a bi-objective Dec-MDP with lexicographic ordering over the objectives, denoted by* $M = \langle \mathcal{A}, S, A, T, \boldsymbol{R}, o \rangle$, *where:*

- $\mathcal{A} = \{1, \ldots, m\}$ *is a finite set of agents in the system;*
- $S = \hat{S}_1 \times \ldots \times \hat{S}_m$ *is the joint state space;*
- $A = \hat{A}_1 \times \ldots \times \hat{A}_m$ *is the joint action space;*
- $T : S \times A \times S \to [0, 1]$ *is the transition function;*
- $\boldsymbol{R} = [R_1, R_N]$ *is the reward function with* $R_1 : S \times A \to \mathbb{R}$ *denoting the reward for task performance and* $R_N : S \times A \to \mathbb{R}$ *denoting the penalty function for NSEs; and*
- $o = [o_1, o_2]$ *denotes the objectives, where* o_1 *is the primary objective denoting agents' assigned tasks and* o_2 *is minimizing NSEs, with* $o_1 \succ o_2$.

A MASE-MDP is characterized by independence of transition function and task completion reward R_1, meaning each agent's transitions and task reward depend only on its local state and action and is independent of other agents. This allows each agent to independently compute a policy for its assigned task [2], [9]. However, the agents incur a *joint penalty* for the NSEs, which introduces a form of inter-agent dependence and prevents decentralized computation of individual policies. Section IV describes an approach to decompose joint penalties into individual penalties, thereby facilitating decentralized planning.

Local and Global State Features. We consider a factored state representation. Let F denote the set of features in the environment, which are categorized into *local features* F_l and *global features* F_g, $F = F_l \cup F_g$. The local features of an agent i are denoted by F_l^i. Local features are agent-specific features that are controlled by the agent's actions and affect its performance (e.g. x, y location). Global features are shared among agents and denote the overall state of the system. They are further divided into static and dynamic global features, denoted by F_{gs} and F_{gd} respectively, based on whether they are immutable or can be modified by agent actions. An agent's state $\hat{s}$ is described by $\vec{f} = \vec{f_l^i} \cup \vec{f_{gd}} \cup \vec{f_{gs}}$.

Definition 2. ***Static global features*** F_{gs} *are exogenous factors that affect all agents and are observable to all agents but not changed by the agents' actions (e.g. ocean currents).*

Algorithm 1 RECON

Input: NSE Tolerance η, $\vec{\pi}$ for primary assigned task

1: Initialize $R_N^i(\hat{s}) = 0, \forall \hat{s} \in \hat{S}_i, \forall i \in \mathcal{A}$
2: Calculate joint penalty $r_N^{\vec{\pi}}$ for $\vec{\pi}$
3: **if** $r_N^{\vec{\pi}} > \eta$ **then**
4: Compute blame $B_i(s)$ using Eqn. 1, $\forall i \in \mathcal{A}, \forall s \in S$
5: $R_N^i(\hat{s}_i) \leftarrow B_i(s)$, $\forall i \in \mathcal{A}, \forall \hat{s}_i \in s, \forall s \in S$
6: **end if**
7: **return** $[R_N^1, ..., R_N^m]$

Definition 3. ***Dynamic global features*** F_{gd} *describe properties of the environment that affect agent operation directly or indirectly, and can be modified by agent's actions (e.g. locations and sizes of the shelves moved by the agents).*

We model the NSE penalty as a function of *dynamic global features* of the joint state, $R_N(s) = \Omega(\vec{f}_{gd})$, where Ω is a mapping from $\vec{f}_{gd}$ to the penalty value. This can also be written in the common state-action notation as $R_N(s, a) = R_N(s), \forall a \in A$.

IV. PENALTY DECOMPOSITION

Our algorithm for NSE penalty decomposition, *Reward Estimation using Counterfactual Neighbors (RECON)*, is outlined in Algorithm 1. RECON first initializes each agent's penalty function R_N^i to zero (Line 1). After the agents have calculated their naive policies independently, the NSE Monitor component in the metareasoner calculates the joint penalty $\vec{r}_N$ for NSEs incurred from joint policy $\vec{\pi}$ (Line 2). If the penalty exceeds a pre-defined NSE tolerance threshold η, then the Blame Resolver decomposes the joint penalty into local penalties for each agent, based on their relative contribution to NSE, calculated using blame (credit) assignment (Lines 3-5). The penalty decomposition resolves the dependency induced by the joint penalty and enables decentralized policy computation to optimize task completion while minimizing NSEs.

A. Blame Estimation Using Counterfactual Evaluation

A blame value is calculated corresponding to each agent's contribution to the global penalty, using counterfactual information. Since the NSE occurrence is determined only by the dynamic global features ($\vec{f}_{gd}$), counterfactuals must be calculated only over $\vec{f}_{gd}$ to avoid incorrect attribution. For example, consider multiple warehouse robots navigating in a narrow corridor, with some carrying large shelves. The NSE of blocked paths for human access is determined by number of agents carrying large shelves. Generating counterfactuals by considering all state features, including changing agent location, does not provide information on NSE associated with multiple agents carrying large shelves. We therefore estimate blame using *counterfactual neighbors*, which are a subset of counterfactual states.

Definition 4. *The* ***counterfactual neighbors*** *of a joint state s, denoted by s_c, are the set of all states that differ only in the values of dynamic global features ($\vec{f}_{gd}$) while all other feature values are same as in s.*

The set of counterfactual neighbors that are reachable from the start state in the environment are referred to as *valid counterfactual neighbors* and are denoted by s_c^v.

Definition 5. ***Agent-specific counterfactual neighbors****, denoted by $s_c^i \subset s_c$, are states that differ in those dynamic global features that can be controlled by agent i, while other feature values are same as in the current joint state s (e.g. a warehouse robot evaluated with a different sized box).*

Agent-specific counterfactuals estimate an agent's contribution to NSE, similar to credit assignment where the agent contribution is estimated by either removing it or replacing its actions, while keeping others' behaviors fixed [15], [17]. Let $s_c^{i,v} \subset s_c^v$ be the set of valid (reachable) counterfactual neighbors for agent i. The blame B_i for agent i in the joint state s is calculated based on the difference between the current NSE penalty and the minimum NSE penalty that could have been achieved by the agent:

$$B_i(s) = \frac{b_i(s)}{\sum_{i \in \mathcal{A}} b_i(s)} \cdot R_N(s), \text{ with} \tag{1}$$

$$b_i(s) = \frac{1}{2}\left(R_N^* + \epsilon + \left(R_N(s) - \min_{s' \in s_c^{i,v}} R_N(s')\right)\right), \tag{2}$$

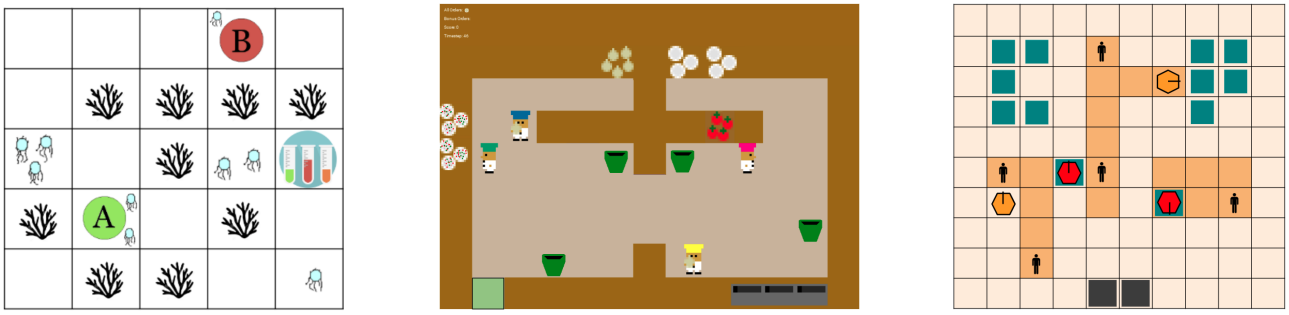
where R_N is the joint NSE penalty function, $R_N^* = \max_{s \in S} R_N(s)$ denotes the maximum joint NSE penalty possible which is used to rescale blame in the range $[0, R_N^*]$ to avoid impractical values that might be negative or exceed the joint NSE penalty itself. ϵ is a small fixed value to avoid singularities during normalization and rescaling, and $b_i(s)$ is an intermediate value used to calculate $B_i(s)$. Equation 1 assigns blame proportional to the agent's ability to mitigate NSEs. This ensures that agents already making their best efforts are penalized less compared to other agents.

The metareasoner's Blame Resolver compiles a local penalty function R_N^i for each agent i as follows:

$$R_N^i(\hat{s}_i) = B_i(s), \forall \hat{s}_i \in s, \forall s \in S, i \in \mathcal{A}. \tag{3}$$

The agents then solve the MASE-MDP using lexicographic value iteration (LVI) [27], with the prescribed R_1^i and R_N^i provided by the metareasoner.

Generalizing R_N^i The penalty function R_N^i, calculated using Equation 3, is based on the blame values corresponding to $\vec{\pi}$ and does not provide any information about potential NSEs that may occur if the agents followed a *different joint policy*. As a result, NSEs may persist when agents update their policies by solving the MASE-MDP with R_N^i. To overcome this limitation, we consider a supervised learning approach to generalize the NSE penalty to unseen situations by using the R_N^i, based on the initial $\vec{\pi}$, as training data. The prediction accuracy can further be improved by including the counterfactuals as part of the training [28]. Generalization helps scale the algorithm to handle many agents, eliminating the need for an iterative RECON, which incurs a high computational overhead.



(a) Salp agents, inspired from [24], [25], are tasked with collecting physical samples from sites A, B, and depositing them at lab facility surrounded by corals prone to damage.

(b) Overcooked environment, inspired from [5], shows agents cooking and cleaning in kitchen with tomatoes, onions, cooking pots, clean dishes, serving counter, dirty dishes, and waste bins.

(c) Warehouse environment, inspired from [16], shows agents tasked with processing shelves, at counter. The narrow corridors with human workers.

Fig. 3: Instances of environments from (a) salp, (b) overcooked, and (c) warehouse domains used in our experiments.

V. EXPERIMENTS

We evaluate in simulation and using mobile robots. Code will be made public after paper acceptance. The MASE-MDP problem is solved using LVI [27], with zero NSE tolerance $\eta = 0$ and using $\epsilon = 10^{-4}$ for rescaling the blame values.

Baselines We compare the performance of RECON with:

1) *Naive Policy* that is optimal for the assigned task and does not optimize NSE mitigation, providing an upper bound on NSE penalty;
2) *Difference Reward* [17] that has been modified to perform blame assignment only considering dynamic global features, to be consistent with our approach, $B_i(s) = R_N(s) - \max_{s' \in s_c^{i,v}} R_N(s')$;
3) *Considerate Reward* approach [1] that has been modified to support multiple agents acting simultaneously, $R_r^i(\hat{s}_i) = \alpha_1 \frac{R_1^i(\hat{s}_i)}{R_1^*} + \alpha_2 \frac{R_N(s) - B_i(s)}{R_N^*}$ where $R_1^* = \max_{\hat{s} \in \hat{S}_i} R_1(\hat{s})$ is the maximum possible reward for assigned task for agent i, α_1 and α_2 are the selfish and care coefficients respectively. R_1^* and R_N^* are used for normalization so that their relative scales are not an inherent factor, but a controlled one (using α_1, α_2);
4) *Generalized RECON w/o counterfactual data* which generalizes the estimated R_N^i to unseen states, using supervised learning; and
5) *Generalized RECON w/ counterfactual (cf) data* which generalizes R_N^i by including the counterfactuals in the training data, using supervised learning.

NSE Penalty Calculation Our experiments use a logarithmic NSE penalty function to model scenarios where the NSE impact plateaus with a certain number of robots involved [8], [31], such as negligible difference in penalty between 10 and 11 robots blocking a corridor. The penalty is calculated as,

$$R_N(s) = \sum_{d \in F_{gd}} \sum_{k \in d} \beta_k \cdot \log\left(\alpha_d N_k + 1\right) \tag{4}$$

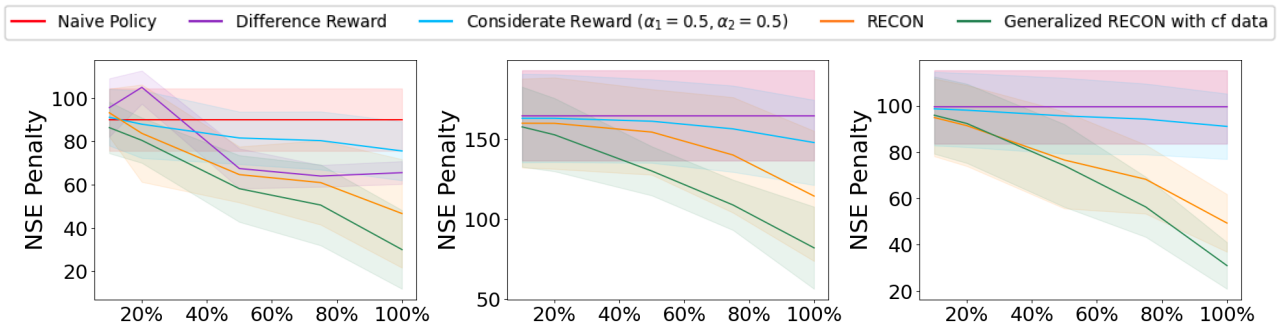
where d is a feature in the set of dynamic global features F_{gd}, N_k is the number of robots with feature value $d = k$ in the joint state s, and β_k is a scaling factor for penalty based on a specific feature value $d = k$. Essentially, higher NSE penalties correspond to higher values of β_k, reflecting that addressing key feature values yields a more substantial reduction in NSE penalty. $\alpha_d\,(> 0)$ is a sensitivity parameter for NSE associated with feature d. A larger α_d denotes that the NSE penalty is more sensitive to an increasing number of robots with a specific feature value.

A. Evaluation in simulation

Experiments are conducted using the following three domains in simulation and the results are averaged over five instances in each domain.

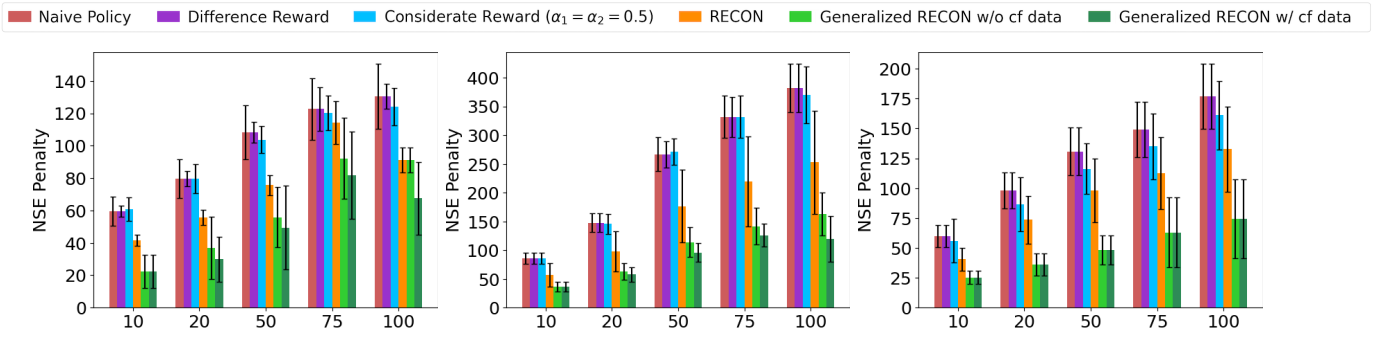
Sample Collection using Salps: Salp-inspired [24] underwater robots are tasked with collecting chemical samples of type A or B from different locations in the seabed as illustrated in Fig. 3a. A robot's state is denoted by $\langle x, y, sample, coral, status \rangle$, where x, y denote robot's location, $sample$ indicates the sample type with X indicating no sample, $coral$ indicates presence of coral at x, y, and $status$ indicates if the sample has been deposited at the destination. Features used to generalize R_N^i are $\langle sample, coral \rangle$. We test with five 20×20 grids that vary in coral locations. **NSE:** The salp-like robots may have chemical residues floating around them when transporting samples. A joint NSE occurs when multiple robots carrying chemical samples are in immediate vicinity of corals, potentially damaging it.

Overcooked: As shown in Fig. 3b, robots are tasked with preparing a fixed number of tomato and onion soup orders, while keeping the kitchen clean [30]. In each problem instance, 20% agents are assigned to cleaning and the rest are assigned cooking tasks. An agent's state is represented as $\langle x, y, dir, object, bin, status \rangle$, where x, y denote its location, dir is its orientation, bin indicates the presence of garbage bins at x, y, and task completion status is denoted by $status$. Agents can move forward in all directions, and $interact$ with objects. Interactions include picking up and putting down objects, cooking, and dumping garbage in bins. Features used for generalizing R_N^i are $\langle object, bin \rangle$. We test with five 15×15 grids with varying locations of the garbage bins. **NSE:** The garbage bins emit bad odors and attract flies. Any object involved in soup preparation must therefore be kept away from the garbage bins or the waste must be disposed at a farther bin.



(a) Salp domain (b) Overcooked domain (c) Warehouse domain

Fig. 4: Average NSE penalty and standard deviation for varying % of agents undergoing policy update in each domain with 25 agents.



(a) Salp domain (b) Overcooked domain (c) Warehouse domain

Fig. 5: Average NSE penalties and standard deviations, averaged over five problem instances with varying number of agents.

TABLE II: Time taken (in minutes) to solve problems with 100 agents, averaged over five problem instances in each domain.

Domains	Considerate Reward	Difference Reward	RECON	Generalized RECON without CF	Generalized RECON with CF
Salp	1.1 ± 0.1	2.9 ± 0.3	4.0 ± 0.4	5.5 ± 0.8	5.7 ± 0.8
Overcooked	121.3 ± 3.8	174.4 ± 3.4	205.1 ± 4.1	254.4 ± 3.2	288 ± 6.3
Warehouse	9.1 ± 0.3	10.0 ± 0.4	12.4 ± 0.2	16.1 ± 0.1	19.0 ± 0.4

Warehouse: Robots are assigned specific shelves of different sizes (*big* or *small*) that they need to locate, pick up, transport, process (drop at the counter), and bring back, in order to complete their task [12], [13]. A robot's state is represented as $\langle x, y, \textit{shelf size}, \textit{shelf status}, \textit{corridor}, \textit{done} \rangle$, where x, y is its location, *shelf size* is the size of the shelf transported by the robot and can be one of *big*, *small*, or *X* for no shelf. *shelf status* denotes the processing stage of the assigned shelf which could be one of {*picked up, processed, delivered*}, *corridor* denotes presence of a narrow corridor at a given location x, y, and *done* is a flag indicating task completion. Features used to generalize R_N^i are $\langle \textit{shelf size}, \textit{shelf status}, \textit{corridor} \rangle$. Test instances, similar to Fig. 3c, are generated by varying the locations of corridors across the warehouse. **NSE:** When multiple robots carry large shelves simultaneously through the same narrow corridor, it inconveniences human workers trying to access the area. The NSE penalty depends on shelf size and the number of agents carrying shelves in the same corridor.

B. Results and Discussion

Number of agents undergoing policy update For each domain in simulation, we consider 25 agents in the environment and vary the percentage of agents undergoing policy update to minimize NSEs, from 10% to 100% (Fig. 4). We select agents to update policies by ranking them in the decreasing order of their blame values. Note that in some cases, NSE may not be avoided even when we update the policies of 100% of the agents in the system. This is because we prioritize completing the task optimally over minimizing NSEs, and it may be impossible for the agents to avoid NSEs while optimally completing their tasks. This is a problem characteristic and not a limitation of RECON. Overall, the results show that RECON and its variants with generalizations can successfully mitigate NSEs, without updating the policies of a large number of agents.

NSE mitigation and scalability Fig. 5 shows that RECON and both version of generalized RECON outperform other methods, consistently reducing NSE penalty across domains and with varying number of agents in the system. The results also show that generalization is useful and can mitigate NSEs considerably even when counterfactual data is not used for training. The above results are with 50% of agents undergoing policy update for each technique, based on the trend in Fig. 4. While updating 100% agents reduced the

NSE penalty in our experiments, it is practically infeasible to implement it in practice for large systems.

Table II shows the run time (in minutes) of various techniques to solve problems with 100 agents. The results indicate an approximately linear increase in the run time with increase in the number of agents. The considerate reward baseline takes the least time as it solves a single-agent problem with other agents treated as part of the environment.

C. Evaluation using mobile robots

We conduct experiments in the salp domain with two Turtlebots to validate real-time usability and effectiveness of our approach in mitigating NSEs, using the map in Fig. 1. The map layout and operation zone of each robot make it inevitable to fully avoid NSEs. Table III shows the percentage of NSE states encountered, with standard deviation. We report results only with Generalized RECON w/ counterfactual data as it consistently performs similar to or better than the other RECON versions (Fig. 5). Considerate Reward with fewer agents is worse than Naive approach, as each agent is less cautious about avoiding NSEs due to reduced joint penalties. Generalized RECON w/ counterfactual data produces the least NSEs, outperforming the baselines.

TABLE III: Average NSE encounters and standard deviation from our experiments with two mobile robots as shown in Fig. 1.

Approach	Average NSE Encounters
Naive	$40.33\% \pm 5.42$
Difference Reward	$38.86\% \pm 6.73$
Considerate Reward	$46.83\% \pm 4.77$
Gen. RECON w/ cf data	$10.33\% \pm 5.33$

VI. SUMMARY AND FUTURE WORK

This paper formalizes the problem of mitigating NSEs in cooperative multi-agent settings as a decentralized, bi-objective problem. The agents' assigned tasks follow transition and reward independence. The agents produce no NSE when operating in isolation but their joint actions produce NSE and incur a joint penalty. We present a metareasoning approach to detect NSEs and update agent policies in a decentralized manner, by decomposing the joint NSE penalty into individual penalties. Our algorithm, RECON, uses a counterfactual-based blame attribution to estimate each agent's contribution towards the joint penalty. Our experiments demonstrate the effectiveness of our approach in mitigating NSEs. Our framework currently supports Dec-MDPs with transition and reward independence. In the future, we aim to relax this assumption and extend our approach to settings with tightly coupled task assignment.

ACKNOWLEDGMENTS

This work was supported in part by ONR grant number N00014-23-1-2171.

REFERENCES

[1] P. A. Alamdari, T. Q. Klassen, R. T. Icarte, and S. A. McIlraith, "Be considerate: Avoiding negative side effects in reinforcement learning," in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2022, pp. 18–26.

[2] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman, "Solving transition independent decentralized markov decision processes," *Journal of Artificial Intelligence Research (JAIR)*, vol. 22, pp. 423–455, 2004.

[3] J. M. Bilbao and P. H. Edelman, "The shapley value on convex geometries," *Discrete Applied Mathematics*, vol. 103, no. 1-3, pp. 33–40, 2000.

[4] A. Carlin and S. Zilberstein, "Bounded rationality in multiagent systems using decentralized metareasoning," in *Decision Making with Imperfect Decision Makers*. Springer, 2012, pp. 1–28.

[5] M. Carroll, R. Shah, M. K. Ho, T. Griffiths, S. Seshia, P. Abbeel, and A. Dragan, "On the utility of learning about humans for human-ai coordination," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[6] M. Choudhury, S. Saisubramanian, H. Zhang, and S. Zilberstein, "Minimizing negative side effects in cooperative multi-agent systems using distributed coordination," in *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, 2024, pp. 2213–2215.

[7] F. Fioretto, E. Pontelli, and W. Yeoh, "Distributed constraint optimization problems and applications: A survey," *Journal of Artificial Intelligence Research (JAIR)*, vol. 61, pp. 623–698, 2018.

[8] I. Gemp, T. Anthony, J. Kramar, T. Eccles, A. Tacchetti, and Y. Bachrach, "Designing all-pay auctions using deep learning and multi-agent simulation," *Scientific Reports*, vol. 12, no. 1, p. 16937, 2022.

[9] C. V. Goldman and S. Zilberstein, "Decentralized control of cooperative systems: Categorization and complexity analysis," *Journal of Artificial Intelligence Research (JAIR)*, vol. 22, pp. 143–174, 2004.

[10] T. Q. Klassen, S. A. McIlraith, C. Muise, and J. Xu, "Planning to avoid side effects," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 36, no. 9, 2022, pp. 9830–9839.

[11] V. Krakovna, L. Orseau, R. Ngo, M. Martic, and S. Legg, "Avoiding side effects by considering future tasks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 19 064–19 074, 2020.

[12] B. Li and H. Ma, "Double-deck multi-agent pickup and delivery: Multi-robot rearrangement in large-scale warehouses," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3701–3708, 2023.

[13] M. Logothetis, G. Karras, K. Alevizos, C. Verginis, P. Roque, K. Roditakis, A. Makris, S. Garcia, P. Schillinger, A. Di Fava, *et al.*, "Efficient cooperation of heterogeneous robotic agents: A decentralized framework," *IEEE Robotics & Automation Magazine*, vol. 28, no. 2, pp. 74–87, 2021.

[14] F.-Y. Meng, "The core and shapley function for games on augmenting systems with a coalition structure," *International Journal of Mathematical and Computational Sciences*, vol. 6, no. 8, pp. 813–818, 2012.

[15] D. T. Nguyen, A. Kumar, and H. C. Lau, "Credit assignment for collective multiagent RL with global rewards," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[16] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht, "Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS)*, 2021.

[17] S. Proper and K. Tumer, "Modeling difference rewards for multiagent learning." in *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2012, pp. 1397–1398.

[18] A. Rahmattalabi, J. J. Chung, M. Colby, and K. Tumer, "D++: Structural credit assignment in tightly coupled multiagent domains," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4424–4429.

[19] M. J. Rentmeesters, W. K. Tsai, and K.-J. Lin, "A theory of lexicographic multi-criteria optimization," in *Proceedings of the 2nd International Conference on Engineering of Complex Computer Systems*. IEEE, 1996, pp. 76–79.

[20] S. Saisubramanian, E. Kamar, and S. Zilberstein, "A multi-objective approach to mitigate negative side effects," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence (JAIR)*, 2020.

[21] ——, "Avoiding negative side effects of autonomous systems in the open world," *Journal of Artificial Intelligence Research (JAIR)*, vol. 74, pp. 143–177, 2022.

[22] L. S. Shapley, "A value for n-person games," 1953.
[23] M. Sundararajan and A. Najmi, "The many shapley values for model explanation," in *International Conference on Machine Learning (ICML)*. PMLR, 2020, pp. 9269–9278.
[24] K. R. Sutherland and L. P. Madin, "Comparative jet wake structure and swimming performance of salps," *Journal of Experimental Biology*, vol. 213, no. 17, pp. 2967–2975, 2010.
[25] K. R. Sutherland and D. Weihs, "Hydrodynamic advantages of swimming by salp chains," *Journal of The Royal Society Interface*, vol. 14, no. 133, p. 20170298, 2017.
[26] J. Svegliato, C. Basich, S. Saisubramanian, and S. Zilberstein, "Metareasoning for safe decision making in autonomous systems," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 11 073–11 079.
[27] K. Wray, S. Zilberstein, and A.-I. Mouaddib, "Multi-objective MDPs with conditional lexicographic reward preferences," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 29, no. 1, 2015.
[28] N. Zerbel and K. Tumer, "Counterfactual focused learning," 2023.
[29] S. Zhang, E. H. Durfee, and S. Singh, "Minimax-regret querying on side effects for safe optimality in factored markov decision processes," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.
[30] M. Zhao, R. Simmons, and H. Admoni, "Coordination with humans via strategy matching," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 9116–9123.
[31] M. Zhu, X.-Y. Liu, and X. Wang, "Joint transportation and charging scheduling in public vehicle systems—a game theoretic approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 8, pp. 2407–2419, 2018.
[32] S. Zilberstein, "Metareasoning and bounded rationality," in *Metareasoning: Thinking about Thinking*. Cambridge, MA, USA: MIT Press, 2011, pp. 27–40.