Learning Fairness in Multi-agent Systems with Distributed-Evaluation, Centralized-Allocation

ASHWIN KUMAR, Washington University in St. Louis, USA
WILLIAM YEOH, Washington University in St. Louis, USA

A wide variety of resource allocation problems operate under resource constraints that are managed by a central arbitrator, with agents who evaluate and communicate preferences over these resources. We formulate this broad class of problems as *Distributed Evaluation, Centralized Allocation (DECA)* problems and propose methods to learn fair and efficient policies in centralized resource allocation. Our methods are applied to learning long-term fairness in a novel and general framework for fairness in multi-agent systems. We show three different methods based on Double Deep Q-Learning: (1) A joint weighted optimization of fairness and utility, (2) a split optimization, learning two separate Q-estimators for utility and fairness, and (3) an online policy perturbation to guide existing black-box utility functions toward fair solutions. Our methods outperform existing fair MARL approaches on multiple resource allocation domains, even when evaluated using diverse fairness functions, and allow for flexible online trade-offs between utility and fairness.

Additional Key Words and Phrases: Reinforcement Learning, Multi-agent Systems, Resource Allocation, Fairness

ACM Reference Format:
Ashwin Kumar and William Yeoh. 2026. Learning Fairness in Multi-agent Systems with Distributed-Evaluation, Centralized-Allocation. https://doi.org/10.1145/3805689.3806494

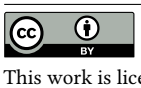
1 INTRODUCTION

Fairness is increasingly critical in multi-agent systems that make consequential decisions under resource constraints, where maximizing total system utility can lead to unfair outcomes. Yet most MARL approaches assume agents can act independently, optimizing only for system-wide utility [9, 22, 27]. This overlooks many real-world settings where actions must be coordinated to enforce global constraints like limited resources, rendering existing methods inapplicable and fairness difficult to enforce. We address this gap by studying fairness in a distinct paradigm: *Distributed Evaluation, Centralized Allocation (DECA)*.

DECA captures a broad class of problems where agents compute local utility estimates, but do not independently execute actions. Instead, a central allocator makes final decisions over which actions are taken, subject to global constraints. This is common in systems like ridesharing [23] and distributing social resources [11], where agent preferences are considered but resource constraints must be centrally enforced. While prior work has proposed domain-tailored solutions, we present a general framework for learning efficient policies in DECA settings. Importantly, DECA differs from the well-known Centralized Training with Decentralized Execution (CTDE) paradigm. While CTDE governs how agents learn, DECA describes how decisions are executed: agent policies evaluate candidate actions locally (Distributed Evaluation, DE), but a central module aggregates these evaluations and selects actions under resource constraints (Centralized Allocation, CA). MARL methods designed for decentralized execution struggle in this setting, motivating new approaches tailored to DECA.

Fairness in DECA-based decision-making is critical, as algorithmic biases can lead to disparities and decreased trust in automated systems [16]. Beyond ethical considerations, fair resource allocation may also be desirable from the perspective of the central controller. Standard DECA solutions rely on estimating agent utilities and solving a constrained optimization to compute the best action for each agent. To incorporate fairness, we enable agents to

Authors' addresses: Ashwin Kumar, ashwinkumar@wustl.edu, Washington University in St. Louis, Saint Louis, MO, USA; William Yeoh, wyeoh@wustl.edu, Washington University in St. Louis, Saint Louis, MO, USA.



learn long-term fairness effects of their actions, combining this with utility estimates to compute allocations. We introduce three strategies:

- **Joint Optimization (JO)**: A scalarized multi-objective learning approach that jointly optimizes for fairness and utility.
- **Split Optimization (SO)**: A method that learns separate fairness and utility estimators, enabling online trade-off adjustments for fairness and utility.
- **Fair-Only Optimization (FO)**: A fairness-focused approach that modifies an existing black-box utility function to incorporate fairness considerations.

To evaluate our methods, we develop a suite of DECA-specific environments by adapting fairness-aware MARL benchmarks. These new environments preserve the key coordination and fairness challenges of the originals while implementing the centralized allocation and constrained resources. We also adapt state-of-the-art MARL algorithms for fairness [9, 27] to operate in the DECA setting, showing how these approaches fail to work well in the face of centralized constraints. Our framework also accommodates a variety of fairness metrics, including variance, α-fairness, maximin fairness, and generalized Gini functions (GGFs), without requiring structural changes. Our primary results focus on variance, with additional results with other metrics included in the appendix.

This work addresses a critical gap in the literature by proposing DECAF, a unified framework for fairness in DECA problems, which encompass a wide range of multi-agent decision-making scenarios. Through our proposed optimization strategies (JO, SO, and FO), we offer a flexible framework for balancing fairness and efficiency in real time. Furthermore, we present the first general approach for integrating fairness into multi-agent resource allocation using Q-learning, paving the way for future advancements in fair AI decision-making.

The main contributions of this paper are as follows:

(1) We formalize and introduce *Distributed Evaluation, Centralized Allocation (DECA)*, a unifying framework for multi-agent multi-time resource allocation problems that generalizes a common structure across many different problems.
(2) We introduce DECAF, a general framework for learning fair policies in multi-agent resource allocation problems under the DECA paradigm.
(3) We propose three novel algorithms (JO, SO, FO) that enable agents to learn Q-functions balancing fairness and efficiency, accommodating various fairness metrics.
(4) We develop a suite of environments that mirror popular fairness-aware multi-agent RL benchmarks, demonstrating the effectiveness of our methods in achieving superior fairness-utility trade-offs compared to baselines.

2 RELATED WORK

While DECA has not been formalized in prior work, it has seen application in many domains. From optimizing passenger-driver matches in ridesharing [18, 23] to efficient allocation of homelessness resources [10, 11], many real-world applications follow this general structure. It is also analogous to the predict-then-optimize (P+O) approach [4, 26], where a predictive model estimates unknown parameters that are subsequently used in optimization. However, unlike P+O, which may not specifically address resource allocation or multi-agent systems, DECA explicitly focuses on these complexities. This distinction is crucial as it allows us to restrict the problem space and tailor our research towards enhancing fairness within multi-agent resource allocation.

Significant research has addressed algorithmic bias, where ML models, such as those used in hiring decisions [19], can exhibit harmful biases. We refer readers to an extensive survey by Mehrabi et al. [16] for a review of recent work. These studies typically focus on debiasing the outputs of predictive models to meet fairness criteria such as equalized odds [7] or demographic parity [3]. However, our work diverges from this approach. Instead of

correcting biases in predictions, we aim to develop algorithms that inherently promote fair decision-making via the actions they optimize.

Our focus is on learning fair policies in multi-agent RL with resource constraints. Prior work has explored fairness in RL broadly [5], but a few methods are especially relevant. FEN [9] uses a hierarchical network to optimize the coefficient of variation, learning when agents should act greedily or fairly. However, it lacks resource constraints and requires inter-agent communication. Other works model fairness as multi-objective optimization, treating each agent's utility as a separate objective and optimizing a social welfare function [24, 27]. These require learning over the full joint action space [24] or use decentralized policy gradients [27], which limits applicability under global constraints. SI [13] addresses fairness in rideshare-matching via fairness-aware post-processing of black-box utilities, but is myopic and domain-specific, and GIFF [15] extends this approach to other domains and metrics. Kumar [12] presents a cohesive theoretical and empirical foundation for this line of work.

The DECA approach allows us to consider global constraints while allocating resources, opening up the scope for better global solutions, which none of the prior approaches allow. The distributed evaluation allows each agent to only learn a local value function, which reduces the complexity when compared to learning a joint policy. Further, our SO and FO approaches allow changing the trade-offs between utility and fairness post-training, which provides additional flexibility that previous approaches lack.

3 PROBLEM FORMULATION

In the context of Distributed Evaluation, Centralized Allocation (DECA), our primary goal is to integrate fairness into the decision-making process of resource allocation in multi-agent systems. Formally, we seek to maximize a combined measure of system utility and fairness, represented as:

$$\max\ (1-\beta)\mathcal{U}_T + \beta\mathcal{F}_T \tag{1}$$

where $\mathcal{U}_T$ denotes the total utility at time T and $\mathcal{F}_T$ represents the fairness measure, weighted by β.

In this section, we describe the DECA optimization framework and how it can be used for resource allocation. In the next section, we will describe DECAF, our solution to learn to improve fairness in this framework.

3.1 Distributed Evaluation, Centralized Allocation (DECA)

We model DECA as a constrained multi-agent Markov decision process (CMMDP) [2] with partial observability, defined by the tuple:

$$\mathcal{M} = \langle \alpha, S, \boldsymbol{O}, \{A_i\}, T, R_u, \gamma, c \rangle \tag{2}$$

where α is the set of agents, S is the global state space, and $\boldsymbol{O} : S \to \prod_i O_i$ maps states to agent observations. Each agent i has action space A_i, and $T : S \times \prod_i A_i \times S \to [0, 1]$ defines joint transitions. The reward function $R_u : S \times \prod_i A_i \to \mathbb{R}^n$ returns a vector of agent utilities, and γ is the discount factor. The resource map $c : \bigcup_i A_i \to \mathbb{R}^K$ gives per-action consumption across K resource types.

In a DECA problem, illustrated by Figure 1, agents independently evaluate actions based on their local observations (DE), while a central controller aggregates these evaluations and optimizes resource allocation subject to constraints (CA). Agent actions include the null action and may have other effects apart from allocation of resources (e.g., moving in an environment), but only actions which consume resources are constrained.

3.2 Optimization Framework

The distributed evaluation (DE) step involves agents learning to predict the utilities of observation-action pairs using approaches such as Deep Q-learning [8, 17]. The utility estimates are computed using partially-observable post-decision states, which are estimated locally, ignoring other agents' actions due to the infeasibility of exploring the joint state space.

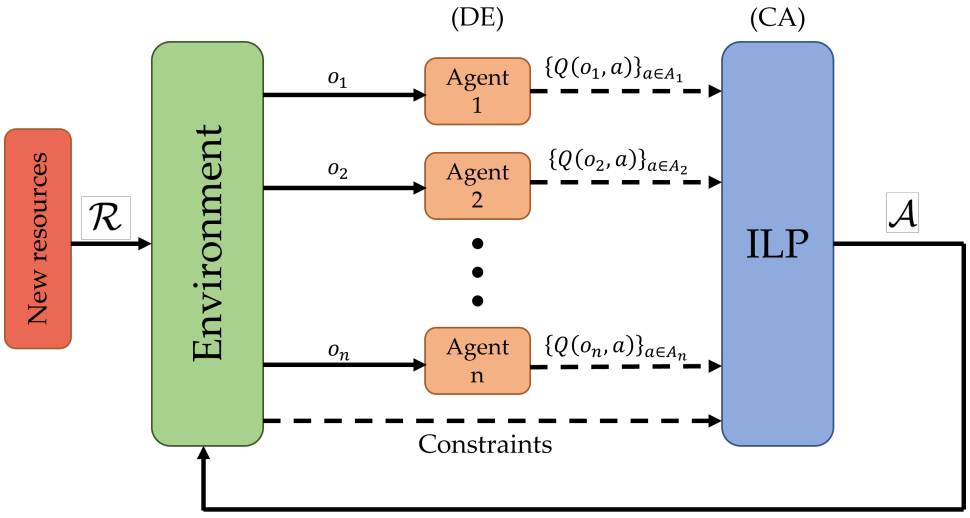


Fig. 1. An outline of the DECA pipeline. Each agent evaluates its available actions in a decentralized manner (DE), and the ILP finds the best joint action $\mathcal{A}$ using these evaluations and resource constraints (CA).

The central allocation (CA) step solves an integer linear program (ILP) that combines predicted utilities with resource constraints. Let $x_i(a) \in \{0, 1\}$ be a binary decision variable that indicates whether agent i is assigned action $a \in A_i$. Let $\mathcal{R} \in \mathbb{R}^K$ denote the vector of available resources, with $\mathcal{R}_k$ representing the quantity of resource type $k \in \{1, 2, \ldots, K\}$. Let $c(a) \in \mathbb{R}^K$ denote the resource consumption vector for action a, where $c(a)_k$ is the amount of resource k consumed by action a. The objective of the ILP is to select one action per agent that maximizes the total predicted utility $Q(o_i, a)$, subject to resource constraints. The optimization problem is defined as:

$$\max_{x_i(a)\in\{0,1\}} \quad \sum_{i\in\alpha}\sum_{a\in A_i} x_i(a)\cdot Q(o_i,a) \tag{3}$$

$$\text{subject to} \quad \sum_{a\in A_i} x_i(a) = 1, \quad \forall i \in \alpha \tag{4}$$

$$\sum_{i\in\alpha}\sum_{a\in A_i} x_i(a)\cdot c(a)_k \le \mathcal{R}_k, \quad \forall k \in \{1,\ldots,K\} \tag{5}$$

The first constraint ensures that exactly one action is selected for each agent. The second constraint ensures that the total resource consumption across all selected actions does not exceed the available amount of any resource. This ILP defines the central allocation mechanism, while the predicted Q-values $Q(o_i, a)$ are learned by each agent, enabling distributed evaluation. This offers benefits over completely distributed approaches by encapsulating resource constraints, and over completely centralized approaches by reducing the complexity of the learning objective. This setup is seen in many resource allocation problems [1, 10, 23].

4 DECAF: FAIRNESS IN DECAS

In this section, we describe DECAF, our framework for incorporating fairness into DECA problems using Q-Learning. Specifically, we detail how we can specify and learn the fairness objective in Eq. 1 through the use of

decomposed fairness rewards, and a modified Q-Learning algorithm to learn from these rewards using centralized training.

4.1 Fairness Reward

Previous work has considered learning to optimize a single social welfare function (SWF) that captures the notions of fairness and utility together, like the Coefficient of Variation used by FEN [9] or α-fairness and the Generalized Gini Function (GGF) used by SOTO [27]. With DECAF, we try to learn a class of objectives that trade off between system utility and fairness, characterized by a trade-off variable β (Eq. 1), with the aim of enabling flexible trade-offs between the two.

Let $\mathbf{Z} = \{z_i\}_{i\in\alpha}$ denote the vector of accumulated agent utilities (averaged or total). We interpret this utility as 'accumulated wealth,' and look to make allocations that can result in a fairer distribution of this wealth. Let $\mathbf{Z}_t^\pi$ represent the distribution of agent utility metrics at time t following policy π. Then, we define a fairness function $\mathbb{F} : \mathbb{R}^n \to \mathbb{R}$ as a mapping of vector $\mathbf{Z}$ to a real value, and our fairness objective is maximizing $\mathcal{F}_T = \mathbb{F}(\mathbf{Z}_{t=T}^\pi)$. Most popular SWFs and fairness functions can be cast into this form.

To realize this objective in DECAF, we compute a potential-based fairness reward based on the allocation made at each timestep. Let $\mathcal{A}^t = \{\mathcal{A}_i^t\}_{i\in\alpha}$ denote the action allocation at time t, where each $\mathcal{A}_i^t$ is the action assigned to agent i as a result of solving the ILP. The fairness reward $R_f(\mathbf{s}_t, \mathcal{A}^t)$ is a vector-valued signal that reflects how the selected allocation impacts system fairness. Specifically, it is computed as the per-step change in the fairness function:

$$\Delta\mathcal{F}|\mathcal{A}^t = \mathcal{F}_{t+1} - \mathcal{F}_t \tag{6}$$

$$= \mathbb{F}(\mathbf{Z}_{t+1}) - \mathbb{F}(\mathbf{Z}_t) \tag{7}$$

A naive way to decompose this reward is to evenly divide it among agents. This is commonly done in collaborative multi-agent RL when agents are optimizing a shared goal.

$$R_f(\mathbf{s}_t, \mathcal{A}^t) = \left[\frac{\Delta\mathcal{F}|\mathcal{A}^t}{n}\right]_{i\in\alpha} \tag{8}$$

Alternatively, specialized decompositions can be designed to give a more informative signal to each agent. For example, if variance is used as the fairness function ($\mathcal{F}_t = -\text{var}(\mathbf{Z}_t)$):

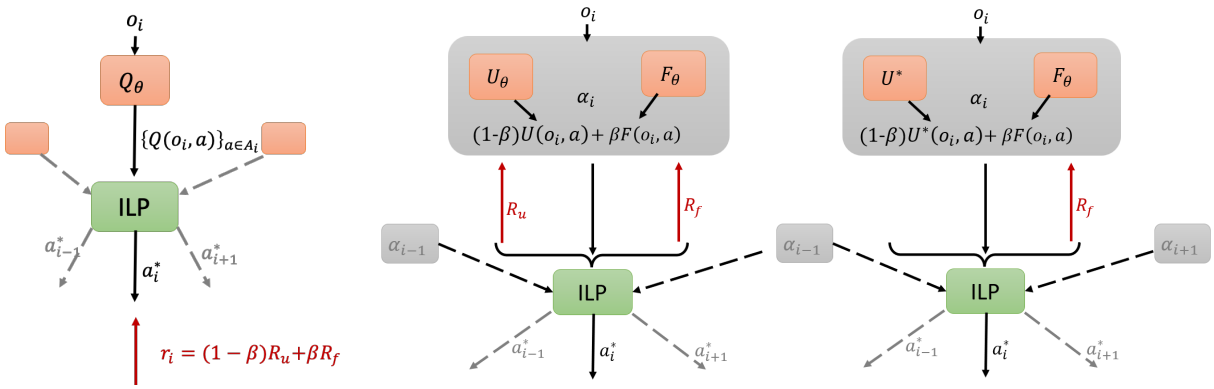
$$\Delta\mathcal{F}|\mathcal{A}^t = -\text{var}(\mathbf{Z}_{t+1}) + \text{var}(\mathbf{Z}_t) \tag{9}$$

$$= -\frac{1}{n}\sum_{i\in\alpha}\left(z_i^{t+1} - \bar{z}_{t+1}\right)^2 + \frac{1}{n}\sum_{i\in\alpha}\left(z_i^t - \bar{z}_t\right)^2 \tag{10}$$

$$R_f(\mathbf{s}, \mathcal{A}) = \left[-\frac{1}{n}\left(z_i^{t+1} - \bar{z}_{t+1}\right)^2 + \frac{1}{n}\left(z_i^t - \bar{z}_t\right)^2\right]_{i\in\alpha} \tag{11}$$

Observe that this reward only depends on the agent's own metric value and the average metric. Thus, each iteration, apart from the local observation, each agent only needs to be communicated information about the average utility of all agents to reliably predict this value. This could be done by the central agent, or by message passing between the agents.

For the main experiments of this paper, we use variance as our fairness function, with the reward function in Eq. 11 as the fair reward. However, our methods are not limited to using variance. We also provide reward decompositions and experiments with other fairness functions including α-fair, GGF, and maximin functions in the appendix, showing the generality of our approach.



(a) Joint Optimization (JO) (b) Split Optimization (SO) (c) Fair-Only (FO)

Fig. 2. Illustration of our three DECAF methods to learn fairness. Each subfigure shows how the values propagate for a single agent. The red lines and text denote the actual reward to the learning model, which is used to update weights using TD learning. (a) Joint Optimization learns to predict a single combined value. (b) Split Optimization learns two separate estimators for utility and fairness, and combines their output. (c) Fair-Only assumes a black-box utility model U^*, and learns a fairness estimator only, combining their outputs to make decisions.

4.2 DECAF Problem Formulation

With the fairness machinery in place, we define the DECAF problem $\mathcal{D}$ as the tuple:

$$\mathcal{D} \;=\; \big\langle \alpha,\, S^f,\, O,\, \{A_i\}_{i\in\alpha},\, T,\, R_u,\, \gamma,\, c,\, \mathbb{F},\, \beta,\, R_f \big\rangle, \tag{12}$$

where $\langle \alpha, O, \{A_i\}, T, R_u, \gamma, c\rangle$ are as in DECA, S^f is the fairness augmented state space including information from the accumulated utility vector $\mathbf{Z}_t = \{z_i^t\}_{i\in\alpha}$ (e.g. the current average utility or the entire vector $\mathbf{Z}_t$), $\mathbb{F} : \mathbb{R}^n \to \mathbb{R}$ is a fairness function evaluated over the accumulated utility vector $\mathbf{Z}_t = \{z_i^t\}_{i\in\alpha}$, $\beta \in [0,1]$ is a trade-off parameter, and $R_f : S^f \times \prod_i A_i \to \mathbb{R}^n$ is a decomposed fairness reward satisfying:

$$\sum_{i\in\alpha} \big[R_f(\mathbf{s}_t, \mathcal{A}^t)\big]_i \;=\; \mathbb{F}(\mathbf{Z}_{t+1}) - \mathbb{F}(\mathbf{Z}_t) \;\triangleq\; \Delta\mathcal{F}\big|\mathcal{A}^t. \tag{13}$$

At each time t, agents locally predict per-action scores $Q_f(o_i, a)$ that combine utility and fairness estimates. The centralized allocator then solves the fairness-aware ILP:

$$\max_{x_i(a)\in\{0,1\}} \quad \sum_{i\in\alpha}\sum_{a\in A_i} x_i(a)\, Q_f(o_i, a) \tag{14}$$

$$\text{s.t.} \quad \sum_{a\in A_i} x_i(a) = 1 \qquad \forall i \in \alpha \tag{15}$$

$$\sum_{i\in\alpha}\sum_{a\in A_i} x_i(a)\, c(a)_k \;\le\; \mathcal{R}_k \qquad \forall k \in \{1,\ldots,K\} \tag{16}$$

where $\mathcal{R} \in \mathbb{R}^K$ is the available resource vector and $c(a) \in \mathbb{R}^K$ is the per-action consumption. As in DECA, actions include a null action to ensure feasibility; only resource-consuming actions are constrained by (16).

$Q_f(o_i, a)$ is the key workhorse in DECAF, and understanding how to learn it is the focus of the next section. We present three approaches to learn Q_f that differ in how they estimate and combine utility and fairness. All three approaches use the same ILP formulation above to compute allocations, differing only in how Q_f is learned.[1]

4.3 Algorithms

Given the fair reward R_f described above, our approach targets the DE step to improve fairness, by changing the Q-values used in the ILP (Eq. 14) to also account for fairness. We do this modifying Q to be an estimator of the combined fair-efficient objective, with a weight $\beta \in [0, 1]$ used to regulate relative value of fairness and utility.

We use experience replay with centralized training to learn the Q-function, where an experience $\tau = \langle \mathbf{o}, \mathcal{A}, \mathbf{r}_u, \mathbf{r}_f, \mathbf{o}' \rangle$ stores a joint transition across all agents, with utility rewards $\mathbf{r}_u$ and fair rewards $\mathbf{r}_f$. Let θ denote the parameters of the Q-function. Given a replay buffer $\mathcal{D}$, we want to minimize the loss function $J_\theta = \mathbb{E}_{\tau \sim \mathcal{D}} L(\delta(\tau))$, where $\delta(\tau)$ is the Bellman error of the transition τ, and L is the MSE loss. We propose three approaches for integrating fairness, illustrated in Figure 2.[2]

- **Joint Optimization (JO):** A single estimator optimizes a weighted combination of fairness and utility.

$$\delta(\tau) = (1 - \beta)\mathbf{r}_u + \beta \mathbf{r}_f + \gamma Q_\theta(\mathbf{o}') - Q_\theta(\mathbf{o}, \mathcal{A}) \tag{17}$$

- **Split Optimization (SO):** Separate estimators for fairness ($F_\theta(\cdot)$) and utility ($U_\theta(\cdot)$) allow dynamic adjustment of their trade-off during policy execution.

$$\delta^f(\tau) = \mathbf{r}_f + \gamma F_\theta(\mathbf{o}') - F_\theta(\mathbf{o}, \mathcal{A}) \tag{18}$$

$$\delta^u(\tau) = \mathbf{r}_u + \gamma U_\theta(\mathbf{o}') - U_\theta(\mathbf{o}, \mathcal{A}) \tag{19}$$

$$Q(\mathbf{o}, \mathcal{A}) = (1 - \beta) U_\theta(\mathbf{o}, \mathcal{A}) + \beta F_\theta(\mathbf{o}, \mathcal{A}) \tag{20}$$

- **Fair-Only Optimization (FO):** A fairness estimator ($F_\theta(\cdot)$) adjusts a pre-existing utility function $U^*(\cdot)$ to incorporate fairness, useful when utility functions are provided externally.

$$\delta^f(\tau) = \mathbf{r}_f(s, a) + \gamma F_\theta(\mathbf{o}') - F_\theta(\mathbf{o}, \mathcal{A}) \tag{21}$$

$$Q(\mathbf{o}, A) = (1 - \beta) U^*(\mathbf{o}, \mathcal{A}) + \beta F_\theta(\mathbf{o}, \mathcal{A}) \tag{22}$$

Our learning algorithm is based on Double Deep Q-Learning [8], which uses a target network to stabilize updates. The key differentiating factor is in how the target values are computed. When learning from an experience, we compute the optimal action $\mathcal{A}^*$ in the successor state by solving the ILP (Eq. 14) using the online Q-network, and then compute the Q-value of the selected actions using the target network. The models are updated using the rewards (stored in the experience) obtained after the ILP allocation of the previous state, as shown in the red text and arrows in Figure 2. For SO and FO, we package the utility and fairness estimators into the Q-function, and compute the optimal successor action using both. Then, we independently update each estimator using the TD error of their respective objectives. We provide the pseudocode for the learning algorithms in the appendix. For FO, we skip training the utility estimator. SO and FO offer the additional benefits of interpretability, as during execution, we are able to discern how much of the decision was based on the utility gain and fairness improvement respectively.

SO also provides some useful properties described below.

Theorem 1. *Given perfect estimates for utility and fairness, increasing β always improves the one-step fairness gain for SO with $\gamma = 0$.*

[1] In the rest of the paper, we omit the subscript in Q_f, denoting it as Q.

[2] Unless stated, we use bold terms to denote vectors, and overload Q-functions to also operate on vectors to compute a vector of outputs. Further, we use $Q(\mathbf{o})$ as a shorthand for computing Q-values for all possible actions for each observation in $\mathbf{o}$.

Proof Sketch: We show this using the property of the ILP. The only way another action is selected when β is increased is if the objective value of that action is higher. Since $\beta \geq 0$, comparing the objective value of two allocations shows us that for $\beta' > \beta$, this only happens when the fairness gain for the new action is higher. □

Theorem 2. *For a large enough β, the fairest allocation will be selected with perfect utility and fairness estimators for SO with $\gamma = 0$.*

Proof Sketch: Since the fairest allocation has the largest fairness gain, we show that there exists a β_f such that the fairness gain's contribution to the objective outweighs any utility loss, making it the optimal allocation. □

We also provide the corollaries to these theorems for improving utility as β is reduced in the appendix, along with full proofs for these theorems. These properties also empirically hold when $\gamma \neq 0$, as our experiments demonstrate. This adaptability is a major strength of SO: It allows a degree of flexibility that other methods do not possess. Specifically, SO allows users to vary the trade-off weight β during runtime, and the behavior can be expected to be monotonic in the direction of change. For $\gamma = 0$, Theorem 1 and its corollary guarantee that the space of selected allocations is Pareto-efficient with changing β at each time step.

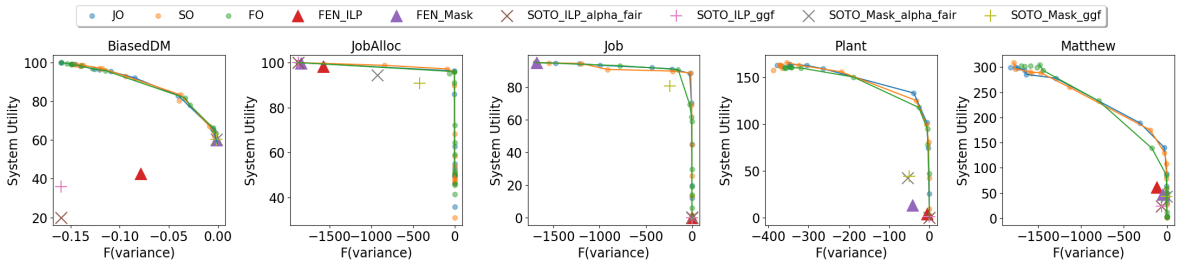


Fig. 3. Change in system utility and fairness as β is increased, with $\beta = 0$ at the top left $\beta = 1$ at the bottom-right. For all domains, we can see that split and joint optimization perform similarly, while learning only fairness can sometimes be slightly worse. All our methods Pareto-dominate SOTO and FEN. Each point depicts the average performance over five different models trained at that β value, and the lines show the Pareto front for each method.

5 EXPERIMENTAL SETUP

We conduct experiments for maximizing the objective in Eq. 1, where the system utility is the sum of all agent utilities at the end of an episode, and the fairness is measured as the negative of the variance of agent resources at the end of the episode. We perform experiments for a variety of β values, repeating each configuration five times for each of our three settings: **Joint Optimization (JO)**, **Split Optimization (SO)** and **Fair-Only Optimization (FO)**. We were unable to use off-the-shelf multi-agent RL libraries because of their lack of support for constrained central decision making. Thus, we implemented versions of the learning algorithm (DDQN with ϵ-greedy TD(0) learning), as described in Section 4. The network architecture has two hidden layers of dimension 20, and output of dimension 1. The utility model used for FO is randomly selected from the JO models trained with $\beta = 0$. We included features indicating the relative advantage of each agent as a signal for fairness, in addition to the features describing the local observation of each agent.[3]

[3]The code for our environments and experiments can be found at https://github.com/YODA-Lab/Fairness-in-Distributed-Evaluation-Centralized-Allocation

5.1 Environments

To evaluate our methods, we created new DECA environments of varying difficulty, based on existing fair MARL benchmarks [9]. We reformulate them as resource allocation problems, adding explicit resource constraints and creating action spaces corresponding to resources. Detailed environment descriptions are provided in the supplement. From simplest to most complex, we have the following:

- **BiasedDM:** This simple environment models an explicit bias in decision-making. Five agents compete for a single resource per timestep, where utility to the decision-maker increases with agent index ($0.2 \times i$ for agent i). Optimal utility is achieved by always allocating resources to agent 5. Fairness is assessed based on resource distribution over time.
- **JobAlloc:** This environment introduces basic coordination. Four agents directly compete to occupy an exclusive 'job', with actions limited to occupying or leaving it. The job can only be claimed if it is unoccupied. While a greedy agent can monopolize it, fairness requires agents to sometimes give up the job so others can benefit—making collaboration essential. This domain's challenge is overcoming the single-step suboptimality when no agent occupies the job.
- **Job:** This environment extends JobAlloc to a grid-based world. Four agents operate on a discretized grid with a fixed square containing a job. Agents receive rewards for occupying the job's location. Grid locations act as constraints, with only one agent allowed per location. Agents move in cardinal directions and communicate directional preferences to the decision-maker, who assigns final moves.
- **Plant:** This environment increases complexity by requiring agents to collect different combinations of resources to complete tasks, introducing combinatorial dependencies. Five agents operate on a discretized grid containing eight resources of three types. Agents must collect specific resource combinations to construct a 'unit' and earn rewards. Requirements vary in difficulty across agents. The decision-maker assigns resources based on agents' preferences, ensuring exclusivity. Agents deterministically move toward assigned resources.
- **Matthew:** This environment showcases the Matthew effect [6, 21], where the rich get richer. Ten agents move on a continuous unit grid with three resources available at a time. Consuming resources grants agents speed and size boosts, allowing faster access to future resources. Some agents start with inherent advantages. Actions involve assigning resources to agents, preventing others from accessing them, or taking a null action to move randomly. Agents provide utility estimates for each action, and the decision-maker allocates resources, ensuring no two agents share the same resource. Agents always move in straight lines toward their targets and are unable to target new resources while moving to collect a previously allocated resource.

It is important to note that while these environments have been designed to appear similar at a high-level as the versions used in the baselines [9, 27], they are completely different under the hood, incorporating agent and resource constraints and modifying the action spaces to align with the resource allocation tasks.

5.2 Baselines

As noted in our Related Work section, FEN [9] and SOTO [27] are the most relevant approaches that generalize to multiple domains; we thus include them as baselines in our experiments. However, both assume settings where agents act independently without resource constraints. To adapt them to the DECA framework, we evaluate two strategies for making constrained decisions using their per-agent policies:

- **Policy-as-Q ('_ILP' suffix):** We interpret action probabilities from a learned policy π as proxy Q-values $Q(o_i, a) = \pi(o_i, a)$ and input them into the ILP (Eq. 14) for centralized allocation.
- **Masked Sequential ('_Mask' suffix):** Agents select actions sequentially in a randomized order at each timestep. Each agent samples from its policy, with infeasible actions masked out based on remaining resource availability. The randomized agent order is used to mitigate ordering bias.

Table 1. Evaluation of all models on multiple metrics for each environment. For JO, SO, and FO, the values in the bracket denote the β value selected based on the model that maximizes $0.1 \cdot U$ - $0.9 \cdot \text{var}(\mathbf{Z})$. The selected β value is indicated in brackets. The values in bold are the best in each row.

Environment	Metric	JO(β)	SO(β)	FO(β)	FEN	SOTO(α-Fair)	SOTO(GGF)
BiasedDM	α-Fair	**-8.09(1)**	-8.3(0.999)	-8.19(0.9995)	-8.15	-8.15	-8.15
	GGF	**0.35(1)**	0.3(0.999)	0.33(0.9995)	0.33	0.33	0.33
	Maximin	**0.16(1)**	0.12(0.999)	0.15(0.9995)	0.15	0.15	0.15
	System Utility	58.31(1)	**63.65(0.999)**	63.38(0.9995)	59.95	60.22	60.24
	Variance	**-0.0007(1)**	-0.0033(0.999)	-0.0022(0.9995)	-0.0014	-0.0015	-0.0015
JobAlloc	α-Fair	**12.71(0.2)**	12.69(0.2)	**12.71(0.2)**	-35.31	-7.39	1.06
	GGF	43.11(0.2)	43.41(0.2)	**43.8(0.2)**	12.59	19.5	29.62
	Maximin	21.79(0.2)	22.21(0.2)	**22.71(0.2)**	0	3.49	10.62
	System Utility	96.28(0.2)	95.81(0.2)	96(0.2)	**99.89**	94.42	90.92
	Variance	-4.44(0.2)	-2.54(0.2)	-1.48(0.2)	-1839.76	-923.46	-421.93
Job	α-Fair	10.97(0.2)	**12.07(0.2)**	10.77(0.2)	-35.89	-55.21	5.03
	GGF	37.12(0.2)	**37.88(0.2)**	26.65(0.2)	11.87	0	22.17
	Maximin	16.99(0.2)	**18.13(0.2)**	13.13(0.2)	0	0	4.9
	System Utility	88.43(0.2)	88.63(0.2)	61.68(0.2)	**94.88**	0	80.57
	Variance	-25.14(0.2)	-13.38(0.2)	-4.59(0.2)	-1683.49	**0**	-242.3
Plant	α-Fair	**15(0.8)**	14.77(0.8)	14.54(0.8)	-35.48	-20.75	-20.62
	GGF	**35.67(0.8)**	34.63(0.8)	33.45(0.8)	1.4	10.28	10.89
	Maximin	**16.61(0.8)**	15.98(0.8)	15.74(0.8)	0	3.62	4.08
	System Utility	**101.72(0.8)**	99.38(0.8)	94.89(0.8)	13.31	42.57	43.84
	Variance	-6.34(0.8)	-6.75(0.8)	**-4.99(0.8)**	-41.64	-54.29	-49.91
Matthew	α-Fair	20.03(0.2)	**23.4(0.5)**	20.71(0.5)	-29.04	12.45	12.64
	GGF	14.41(0.2)	**19(0.5)**	11.92(0.5)	1.73	4.6	4.67
	Maximin	3.64(0.2)	**8.86(0.5)**	5.09(0.5)	0.36	1.66	1.69
	System Utility	**140(0.2)**	108.1(0.5)	85.5(0.5)	47.39	42.77	43.02
	Variance	-26.95(0.2)	**-1.28(0.5)**	-5.17(0.5)	-45.9	-3.33	-3.32

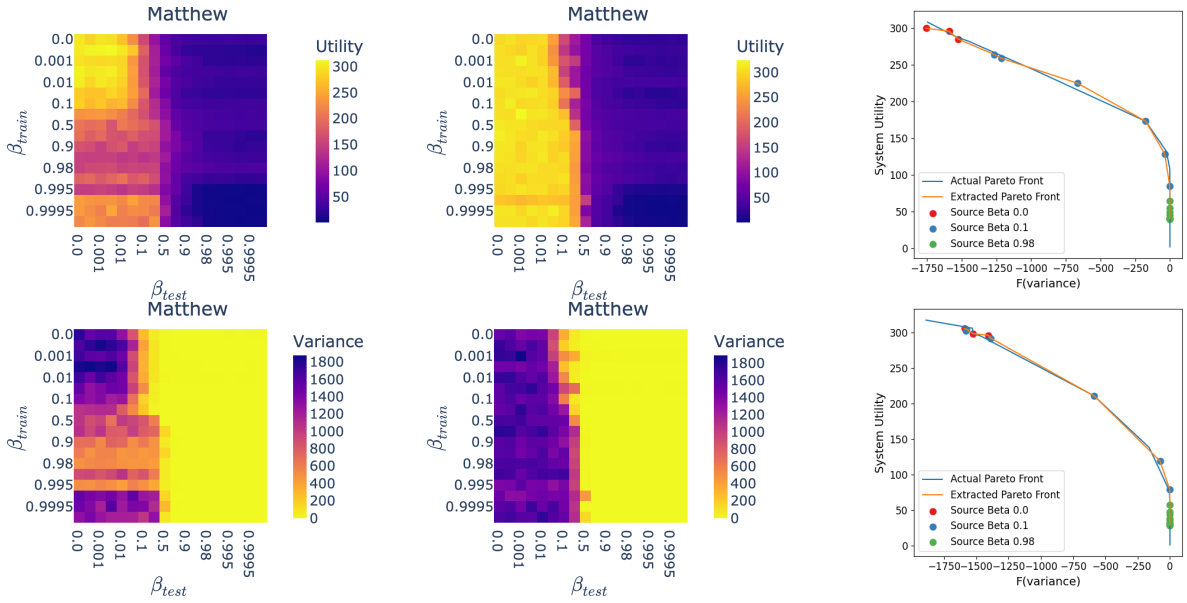
We add the extra features that SOTO requires (only for SOTO), and train SOTO with both the α-fair and *GGF* objective described in their paper [27]. We implement shared weights across agents.

6 RESULTS

Figure 3 shows the performance of all three DECAF methods (JO, SO, FO) on the five domains discussed above. For each method, we varied the hyperparameter β controlling the fairness-utility trade-off (Eqs. 17, 20, 22), starting with $\beta = 0$ (top-left) and increasing to $\beta = 1$ (bottom-right). As mentioned in Section 4, we present results where we optimize for variance as the fairness function here. Additional results on learning with different fairness functions are included in the supplement.

6.1 Efficacy of the Fairness-Utility Optimization

For all domains, all three methods are able to learn expressive policies which lie at various points close to the Pareto front. This shows that the optimization allows the model to trade off utility and fairness to show diverse behaviors as required by the user. This also confirms that the fairness reward proposed for minimizing variance is a good signal.



(a) SO models (b) FO models (c) Approximated Pareto fronts

Fig. 4. Generalization analysis on the Matthew environment. (a) and (b) show the generalization of SO and FO models trained on β_{train} and tested on β_{test}. Brighter colors indicate better outcomes for utility (top) and variance (bottom). (c) Approximated Pareto fronts using sparse β_{train} evaluated on other β values for the Matthew domain. Top: SO. Bottom: FO

6.2 Comparison Against Baselines

In DECA, centralized allocation under resource constraints makes the mapping from decentralized evaluations to realized outcomes: the ILP allocator can induce transitions that are not generated by any single decentralized on-policy behavior. On-policy methods like PPO rely on assumptions of on-policy transitions from decentralized policies (e.g., PPO-style baselines we adapt) and therefore degrade in DECA, particularly in complex domains requiring coordinated long-horizon strategies. This motivates and validates our use of off-policy value-based learning (Double DQN variants) in the DECA setting.

Although we adapt FEN and SOTO to operate in the DECA setting, their underlying learning algorithms assume on-policy transitions from decentralized policies. The introduction of resource constraints fundamentally alters the environment dynamics, breaking this assumption. As a result, their performance degrades—especially in complex domains like Matthew and Plant, where coordinated, long-term strategies are necessary. In simpler settings, where greedy decisions align with utilitarian outcomes, these methods can still recover reasonable policies. However, even when optimizing social welfare functions like GGF or α-fairness, they struggle to find high-performing strategies under constraints.

Among the adapted baseline variants, the masked versions outperform their ILP counterparts, as ILP-based selection produces trajectories that violate the assumptions of policy gradient methods. However, masking also limits coordination and suffers from order sensitivity. In contrast, DECAF handles global constraints and large, combinatorial action spaces more naturally, giving it a clear advantage in the DECA setting. As shown in Figure 3, DECAF consistently Pareto-dominates both FEN and SOTO across all domains, with SOTO_Mask (GGF) being the

most competitive baseline. Table 1 presents results for selected β values of JO, SO, and FO for each environment, showing DECAF's advantage across multiple metrics, even when trained on variance only.

6.3 Generalization Using Split Optimization (SO)

Figure 4a shows detailed results for the Matthew environment, when using SO. We evaluate each model trained on a particular β_{train} on all other β_{test}. This allows us to see how well the trained fairness and utility models generalize when the operating β is changed. Note that for all these models, β is not provided as a feature to the Q-function.

The diagonal elements show the behavior when training and testing is done on the same β value. From the plots for system utility (Figure 4a (top)) and variance (Figure 4a (bottom)), we can see that as β_{test} increases, variance improves, and as β_{test} decreases, utility improves. This is the expected behavior, and the major advantage of SO over JO. With JO, the model only predicts a single value, so we are unable to change the trade-off weight during evaluation, and we require a unique model for each β that we want the model to work for. However, with SO, selecting just a few spread out β values can allow us to extrapolate between them, providing online adaptability. This shows that SO has the flexibility to function well at operating points away from the β_{train} that it is trained for.

Figure 4c (top) shows how well a few selected models can generalize to the Pareto front. We pick β_{train} values evenly spaced across the search space, and evaluate the model on all β_{test}, picking the closest β_{train} in order of the search space. We can see that the approximated Pareto front closely matches the actual Pareto front, even with just 3 models, further demonstrating the strength of SO. These observations hold for other domains as well, and the results are included in the supplement.

6.4 Effectiveness of Fair-Only Optimization (FO)

Like SO, FO is also able to generalize well when different β_{test} are used to evaluate the learned models (Figure 4b). Because the utility model is fixed, all models achieve high utility as $\beta_{test} \to 0$ (Figure 4b (top)). Further, all models also improve fairness as β_{test} grows larger (Figure 4b (bottom)). The behavior change from utility-oriented to fairness-oriented is much sharper in FO when compared to SO. Looking at Figure 4c (bottom), we can again see that even FO has the ability to generalize from only a few models to cover the entire Pareto front. Despite being Pareto-dominated by SO and JO at intermediate β values in some domains, FO has the advantage of reliability: A trusted black-box utility model can be used in conjunction with a possibly smaller fairness model, with the guarantee to behave optimally as β_{test} is reduced. When such a model is available, FO is the best choice, given its competent performance and lower computational load.

6.5 Comparison of DECAF Methods

In our results, JO and SO generally exhibit similar performance characteristics, suggesting that simultaneous evolution of utility and fairness estimates is beneficial. FO is also very competetive, but in complicated environments (e.g., Matthew), it falls below the Pareto front. This underperformance is likely due to out-of-distribution transitions for the fixed utility model, which are more problematic in FO when a large fairness weight β is used, causing a larger shift in the state distribution and resulting in degraded utility estimates. We expect this to be a bigger issue in more complicated environments, or with poor black-box utility functions.

For practitioners, we recommend using SO when possible, as it provides the strongest intersection of flexibility and capability. Whenever a fixed desired tradeoff is known, JO is the best method, sacrificing flexibility for reduced computational cost. When the utility function is a black box (e.g. human preferences), FO is the optimal choice to improve fairness and enable flexible tradeoffs.

6.6 On the Importance of Past Discounting and Warm Starts

We also highlight two departures from prior work in our implementation of DECAF for learning fairness: (1) We discount the agent metrics $\mathbf{Z}$ over the past, which has been shown in recent work to improve ability to learn fairness [14], and (2) we implement warm starts for initializing $\mathbf{Z}$ instead of initializing at 0. The past discounts are necessary to account for the time dependence of various normalized fairness metrics, like the Gini coefficient and coefficient of variation, or the variance over normalized agent metrics. In these cases, actions taken early on can cause larger changes to the fairness metric while actions taken after a sufficient history has been established have an imperceptible effect. This is undesirable in long-horizon settings, but can be remedied by past discounts, effectively 'forgetting' events that happened far in the past. The warm starts function to counteract the other pitfall in some fairness metrics: The zero vector is perfectly fair, and any changes can incur a large fairness penalty. Adding randomized warm starts helps in preventing the algorithm from converging to this trivial solution. In practice, we keep the warm start values small, so that the past discounts effectively scale them down to zero over the course of an episode.

7 LIMITATIONS

Our work assumes a fully cooperative setting, where agents are self-interested, but still report truthful utilities, and the decision-maker is motivated by social welfare. In the presence of strategic agents, our approach might not yield the same results.

Our approach centers on an ILP optimization by the central decision-maker, which may not always be tractable. However, many resource allocation problems allow LP relaxations, and with certain assumptions, polynomial-time allocation algorithms like the Hungarian algorithm can be used instead. Additionally, message-passing and network based methods may also be used to arrive at a global solution in the absence of a central authority. We do not explore these avenues in our paper.

Our experiments use handcrafted domains based on prior work, which may not capture the full complexity of real-world systems. However, we show how existing approaches fail in all but the simplest tasks, while DECAF is able to generate strong fairness-efficiency tradeoffs even in complicated environments like Plant and Matthew.

8 CONCLUSIONS AND FUTURE WORK

In this work, we formalized the DECA class for resource allocation problems, and introduced DECAF, a framework for learning fair-efficient behavior in DECA problems. DECAF is among the first approaches to optimize fair resource allocation under resource constraints, supporting diverse problem settings by decoupling fairness and utility metrics. Split and Fair-Only optimizations enable online trade-offs between utility and fairness without retraining, enhancing interpretability. Our results demonstrate the flexibility and effectiveness of DECAF across various scenarios, while overcoming limitations of decentralized MARL approaches that rely on policy gradients. Our framework currently relies on Q-Learning, as deriving a policy gradient approach for DECA problems is challenging due to the dynamic state-action space and the indirect relationship between agent 'policies' and actions resulting from constrained action selection. Addressing this challenge is a promising direction for future research. Additionally, our fairness decomposition is modular; approaches like VDN [25] or QMIX [20] could be integrated to improve credit assignment for the fair reward, further strengthening our framework. Overall, DECAF provides a principled and practical foundation for fairness-aware multi-agent learning in constrained decision settings, and opens several promising avenues for future work.

9 GENERATIVE AI USAGE STATEMENT

ChatGPT (GPT-4o and GPT 5) was used in the preparation of this manuscript for the purpose of grammatical and style editing.

10 ACKNOWLEDGEMENTS

This work was supported in part by a seed grant from the Geospatial Research Initiative (GRI) at Washington University in St. Louis.

REFERENCES

[1] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. 2017. On-demand High-capacity Ride-sharing via Dynamic Trip-vehicle Assignment. *Proceedings of the National Academy of Sciences* 114 (2017), 462–467.

[2] Frits De Nijs, Erwin Walraven, Mathijs De Weerdt, and Matthijs Spaan. 2021. Constrained Multiagent Markov Decision Processes: A Taxonomy of Problems and Algorithms. *Journal of Artificial Intelligence Research* 70 (2021), 955–1001.

[3] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. 2012. Fairness through awareness. In *Proceedings of the Conference on Innovations in Theoretical Computer Science*. 214–226.

[4] Adam N. Elmachtoub and Paul Grigas. 2022. Smart "predict, then optimize". *Management Science* 68 (2022), 9–26.

[5] Pratik Gajane, Akrati Saxena, Maryam Tavakol, George Fletcher, and Mykola Pechenizkiy. 2022. Survey on Fair Reinforcement Learning: Theory and Practice. *arXiv preprint arXiv:2205.10032* (2022).

[6] Chongming Gao, Kexin Huang, Jiawei Chen, Yuan Zhang, Biao Li, Peng Jiang, Shiqi Wang, Zhong Zhang, and Xiangnan He. 2023. Alleviating Matthew Effect of Offline Reinforcement Learning in Interactive Recommendation. In *Proceedings of the International Conference on Research and Development in Information Retrieval*. 238–248.

[7] Moritz Hardt, Eric Price, and Nati Srebro. 2016. Equality of opportunity in supervised learning. In *Proceedings of the Conference on Neural Information Processing Systems*. 3323–3331.

[8] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double Q-Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 2094–2100.

[9] Jiechuan Jiang and Zongqing Lu. 2019. Learning fairness in multi-agent systems. In *Proceedings of the Conference on Neural Information Processing Systems*. 13854–13865.

[10] Amanda R. Kube, Sanmay Das, and Patrick J. Fowler. 2019. Allocating interventions based on predicted outcomes: A case study on homelessness services. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 622–629.

[11] Amanda R. Kube, Sanmay Das, and Patrick J. Fowler. 2023. Community-and data-driven homelessness prevention and service delivery: optimizing for equity. *Journal of the American Medical Informatics Association* 30, 6 (2023), 1032–1041.

[12] Ashwin Kumar. 2025. *Towards Fair Sequential Resource Allocation: Algorithmic Designs, Interventions, and Evaluations*. Ph.D. Dissertation. Washington University in St. Louis.

[13] Ashwin Kumar, Yevgeniy Vorobeychik, and William Yeoh. 2023. Using Simple Incentives to Improve Two-Sided Fairness in Ridesharing Systems. In *Proceedings of the International Conference on Automated Planning and Scheduling*. 227–235.

[14] Ashwin Kumar and William Yeoh. 2025. Past-Discounting is Key for Learning Markovian Fairness with Long Horizons. *arXiv preprint arXiv:2504.01154* (2025).

[15] Ashwin Kumar and William Yeoh. 2026. A General Incentives-Based Framework for Fairness in Multi-agent Resource Allocation. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*.

[16] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2021. A survey on bias and fairness in machine learning. *Comput. Surveys* 54, 6 (2021), 1–35.

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[18] Zhiwei Tony Qin, Hongtu Zhu, and Jieping Ye. 2022. Reinforcement learning for ridesharing: An extended survey. *Transportation Research Part C: Emerging Technologies* 144 (2022), 103852.

[19] Manish Raghavan, Solon Barocas, Jon Kleinberg, and Karen Levy. 2020. Mitigating bias in algorithmic hiring: Evaluating claims and practices. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*. 469–481.

[20] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2020. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research* 21, 178 (2020), 1–51.

[21] Daniel Rigney. 2010. *The Matthew Effect: How Advantage Begets Further Advantage*. Columbia University Press.

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[23] Sanket Shah, Meghna Lowalekar, and Pradeep Varakantham. 2020. Neural approximate dynamic programming for on-demand ride-pooling. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 507–515.

[24] Umer Siddique, Paul Weng, and Matthieu Zimmer. 2020. Learning fair policies in multi-objective (deep) reinforcement learning with average and discounted rewards. In *Proceedings of the International Conference on Machine Learning*. 8905–8915.

[25] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. 2018. Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward. In *Proceedings of the Conference on Autonomous Agents and Multiagent Systems*. 2085–2087.

[26] Kai Wang, Sanket Shah, Haipeng Chen, Andrew Perrault, Finale Doshi-Velez, and Milind Tambe. 2021. Learning MDPs from features: Predict-then-optimize for sequential decision making by reinforcement learning. In *Proceedings of the Conference on Neural Information Processing Systems*. 8795–8806.

[27] Matthieu Zimmer, Claire Glanois, Umer Siddique, and Paul Weng. 2021. Learning fair policies in decentralized cooperative multi-agent reinforcement learning. In *Proceedings of the International Conference on Machine Learning*. 12967–12978.

A THEORETICAL RESULTS

For this section, we use an alternate notation, replacing β with $\eta = \frac{\beta}{1-\beta}$ to make the equations easier to read, such that:

$$(1-\beta)U + \beta F \Leftrightarrow U + \eta F$$

This does not affect the allocation made by the ILP, as it only scales all Q-values by $1/(1-\beta)$. This would only be undefined when $\beta = 1$, but we avoid that condition in our proofs. As $\beta \to 1, \eta \to \infty$, and for any $\beta' > \beta$, $\eta' > \eta$. Note that in the theorem statements, we replace β with η, but the proofs are equivalent.

The following results hold for any fairness function used in the DECAF formulation.

Proposition 1. *As $\eta_{test} \to 0$, all fair-only models behave in a utility-maximizing manner.*

We state this without proof. It is easy to follow how this holds, as at $\eta = 0$, the fairness model does not play any role in the decision making.

Theorem 1. *Given perfect estimates for utility and fairness, increasing η always improves the one-step fairness gain for SO with $\gamma = 0$.*

Proof. We assume that the utility and fairness estimators are converged, i.e., the estimates of fairness and utility are correct. For the following discussion, assume the environment has evolved over some time t and is at a state s_t. We consider what changes when we change η at this state. Variables used henceforth are conditioned on s_t, wherever reasonable. We make the conditioning on s_t implicit and do not notate it, to make it easier to read.

With $\gamma = 0$, the optimal utility and fairness estimates equal the one-step return, i.e. the change in utility and fairness because of the resulting joint action. Note that these values are not known to the agents prior to the allocation as they depend on the joint actions of all agents, so computing these estimates is not trivial.

Let $U_{tot}(\mathcal{A})$ and $F_{tot}(\mathcal{A})$ be defined as follows, given an allocation $\mathcal{A}$:

$$U_{tot}(\mathcal{A}) = \sum_{i \in \alpha} U(\mathcal{A}_i) \tag{23}$$

$$F_{tot}(\mathcal{A}) = \sum_{i \in \alpha} F(\mathcal{A}_i) \tag{24}$$

We remind the reader that $\mathcal{A}_i$ refers to the action assigned to agent i in the allocation $\mathcal{A}$.

Let $\mathbf{Z}_t$ represent the agent metrics at time t. Further, let $\mathcal{A}^*$ represent the optimal allocation from the ILP with η as the trade-off weight. Since $\mathcal{A}^*$ is optimal, it follows that for all other possible allocations $\mathcal{A}_o$:

$$U_{tot}(\mathcal{A}^*) + \eta F_{tot}(\mathcal{A}^*) \geq U_{tot}(\mathcal{A}_o) + \eta F_{tot}(\mathcal{A}_o) \tag{25}$$

$$U_{tot}(\mathcal{A}^*) - U_{tot}(\mathcal{A}_o) \geq \eta(F_{tot}(\mathcal{A}_o) - F_{tot}(\mathcal{A}^*)) \tag{26}$$

We are interested in finding what happens to the allocation when we increase η. For $\eta' > \eta$, note that the left side of Eq. 26 remains the same. Since utility estimates are not affected by changing η, any other allocation $\mathcal{A}_o$ can only be selected over $\mathcal{A}^*$ if the following condition holds:

$$U_{tot}(\mathcal{A}^*) + \eta' F_{tot}(\mathcal{A}^*) \leq U_{tot}(\mathcal{A}_o) + \eta' F_{tot}(\mathcal{A}_o) \tag{27}$$

$$U_{tot}(\mathcal{A}^*) - U_{tot}(\mathcal{A}_o) \leq \eta'(F_{tot}(\mathcal{A}_o) - F_{tot}(\mathcal{A}^*)) \tag{28}$$

Combining Eqs.26 and 28, we get the following:

$$\eta(F_{tot}(\mathcal{A}_o) - F_{tot}(\mathcal{A}^*)) \leq \eta'(F_{tot}(\mathcal{A}_o) - F_{tot}(\mathcal{A}^*)) \tag{29}$$

$$F_{tot}(\mathcal{A}^*)(\eta' - \eta) \leq F_{tot}(\mathcal{A}_o)(\eta' - \eta) \tag{30}$$

Since $\eta \geq 0$ and $\eta' > \eta$, Eq. 30 can only be true if $F_{tot}(\mathcal{A}_o) > F_{tot}(\mathcal{A}_o)$.

Thus, any allocation $\mathcal{A}_o$ that is optimal (and thus selected by the ILP) for $\eta' > \eta$ is guaranteed to have equal or better fairness than the allocation $\mathcal{A}^*$ at η. □

We also state the corollary to Theorem 1.

Corollary 1. *Given perfect estimates for utility and fairness, decreasing η always improves the one-step utility gain for SO with $\gamma = 0$.*

The proof follows a similar structure to Theorem 1.

While we only prove the behavior for $\gamma = 0$, our empirical results show that we can expect similar behavior for long-horizon estimates. For any state, we will select actions that improve fairness in the long run starting from that state as η is increased.

We also show the following useful property:

Theorem 2. *For a large enough η, the fairest allocation will be selected with perfect utility and fairness estimators for SO with $\gamma = 0$.*

Proof. Let $\mathcal{A}_f$ denote the allocation with the largest fairness gain:

$$\mathcal{A}_f = \operatorname*{argmax}_{\mathcal{A}} F_{tot}(\mathcal{A})$$

For simplicity, let us assume no two allocations have the same $F_{tot}(\mathcal{A})$. For any other allocation $\mathcal{A}_o$, we have:

$$F_{tot}(\mathcal{A}_f) > F_{tot}(\mathcal{A}_o) \tag{31}$$

Then, $\mathcal{A}_f$ will be optimal and selected by the ILP if the following condition holds:

$$U_{tot}(\mathcal{A}_f) + \eta_f F_{tot}(\mathcal{A}_f) \geq U_{tot}(\mathcal{A}_o) + \eta_f F_{tot}(\mathcal{A}_o) \tag{32}$$

$$\eta_f \geq \frac{U_{tot}(\mathcal{A}_o) - U_{tot}(\mathcal{A}_f)}{F_{tot}(\mathcal{A}_f) - F_{tot}(\mathcal{A}_o)} \tag{33}$$

We can compute an upper bound for η_f by considering the range of values that U_{tot} and F_{tot} can take. Let $U_{max} = \max_{\mathcal{A}} U_{tot}(\mathcal{A})$, and $F_{max} = \max_{\mathcal{A}, \mathcal{A} \neq \mathcal{A}_f} F_{tot}(\mathcal{A})$.

Then, we have the following:

$$\eta_f \geq \frac{U_{tot}(\mathcal{A}_o) - U_{tot}(\mathcal{A}_f)}{F_{tot}(\mathcal{A}_f) - F_{tot}(\mathcal{A}_o)} \tag{34}$$

$$\leq \frac{U_{max} - U_{tot}(\mathcal{A}_f)}{F_{tot}(\mathcal{A}_f) - F_{tot}(\mathcal{A}_o)} \tag{35}$$

$$\leq \frac{U_{max} - U_{tot}(\mathcal{A}_f)}{F_{tot}(\mathcal{A}_f) - F_{max}} = \eta_f^u \tag{36}$$

Eq. 36 gives us an upper bound for η_f. Thus, for all $\eta > \eta_f^u$, $\mathcal{A}_f$ will be the optimal allocation. □

Corollary 2. *For a small enough η, the most utilitarian allocation will be selected with perfect utility and fairness estimators for SO with $\gamma = 0$.*

The proof follows a similar structure to the proof for the previous theorem.

Algorithm 1 DECAF Algorithm

Initialize: agent network Q_θ, target network $Q_{\theta'}$
Initialize: ϵ (exploration rate)
Initialize: replay buffer $\mathcal{D}$
for episode = 1 to N_{eps} **do**
  Decay ϵ according to decay schedule
  RunEpisode(Q_θ, ϵ, T, env, $\mathcal{D}$)
  if episode % k == 0 **then** ▷ Run validation with $\epsilon = 0$
    RunEpisode(Q_θ, 0, ∞, env, $\mathcal{D}$) ▷ Save validation objective
  end if
  if episode % τ == 0 **then**
    $Q_{\theta'} \leftarrow Q_\theta$ ▷ Update target weights
  end if
end for
Load model with best validation objective value
Run 50 validation episodes using RunEpisode(Q_θ, 0, ∞, env, $\mathcal{D}$)
Save validation results

Algorithm 2 RunEpisode (Executes a single episode)

function RunEpisode(Q_θ, ϵ, T, env, $\mathcal{D}$)
  Reset environment, get initial observation $\mathbf{o}_0$
  for t = 1 to N_{steps} **do**
    Sample a random number $r \in [0, 1]$
    if $r < \epsilon$ **then**
      $\mathbf{Q}_t = \mathbf{Q}_{\text{random}}$ ▷ Random Q-values for exploration
    else
      $\mathbf{Q}_t = Q_\theta(\mathbf{o}_t)$ ▷ Q-values from the agent
    end if
    Use ILP to compute optimal action $\mathbf{a}_t$:
      $\mathbf{a}_t$ = solve_ILP($\mathbf{Q}_t$, env.constraints)
    Take step in environment:
      $(\mathbf{R}_{f,t}, \mathbf{R}_{u,t}, \mathbf{o}_{t+1})$ = env.step($\mathbf{a}_t$)
    Store transition $(\mathbf{o}_t, \mathbf{a}_t, \mathbf{R}_{u,t}, \mathbf{R}_{f,t}, \mathbf{o}_{t+1})$ in replay buffer $\mathcal{D}$
    if $t\%T == 0$ **then**
      update(Q_θ, Q'_θ, $\mathcal{D}$, env)
    end if
  end for
end function

B LEARNING ALGORITHM

Algorithm 1 shows the overall training loop used for our experiments, with Algorithm 2 showing how each episode is executed. We decay epsilon to 0.05 over half of the total number of episodes. T, k, τ decide how frequently we learn, validate and update the target model respectively. Algorithm 3 and Algorithm 4 detail how

Algorithm 3 Update for Joint Optimization

function UPDATE(Q_θ, $Q_{\theta'}$, $\mathcal{D}$, env)
  Sample a mini-batch of n experiences from replay buffer $\mathcal{D}$
  for each experience $\langle \mathbf{o}, \mathcal{A}, \mathbf{r}_u, \mathbf{r}_f, \mathbf{o}' \rangle$ in the mini-batch **do**
    Compute Q-values for the successor observation $Q_\theta(\mathbf{o}')$
    Solve ILP to get the optimal allocation $\mathcal{A}^*$ for the next observation $\mathbf{o}'$:
      $\mathcal{A}^*$ = solve_ILP($Q_\theta(\mathbf{o}')$, env.constraints)
    Compute Q-values for $\mathcal{A}^*$ using the target network:
      $Q_{\theta'}(\mathbf{o}', \mathcal{A}^*)$
    Compute the target for the TD update:
      target = $(1-\beta)\mathbf{r}_u + \beta\mathbf{r}_f + \gamma Q_{\theta'}(\mathbf{o}', \mathcal{A}^*)$
    Compute the TD loss:
      loss = $(Q_\theta(\mathbf{o}, \mathcal{A}) - \text{target})^2$
    Perform gradient descent on the TD loss to update Q_θ
  end for
end function

Algorithm 4 Update for Split Optimization

function UPDATE(Q_θ, $Q_{\theta'}$, $\mathcal{D}$, env)
  Sample a mini-batch of n experiences from replay buffer $\mathcal{D}$
  Unpack Q_θ into U_θ and F_θ
  Unpack $Q_{\theta'}$ into $U_{\theta'}$ and $F_{\theta'}$
  for each experience $\langle \mathbf{o}, \mathcal{A}, \mathbf{r}_u, \mathbf{r}_f, \mathbf{o}' \rangle$ in the mini-batch **do**
    Compute the combined Q-values for the successor observation:
      $Q_\theta(\mathbf{o}') = (1-\beta)U_\theta(\mathbf{o}') + \beta F_\theta(\mathbf{o}')$
    Solve ILP to get the optimal action $\mathcal{A}^*$ for the next observation $\mathbf{o}'$:
      $\mathcal{A}^*$ = solve_ILP($Q_\theta(\mathbf{o}')$, env.constraints)
    for model in $\{U, F\}$ **do**
      if model is U **then**
        Set $M_\theta = U_\theta$, $M_{\theta'} = U_{\theta'}$, and $r = \mathbf{r}_u$
      else
        Set $M_\theta = F_\theta$, $M_{\theta'} = F_{\theta'}$, and $r = \mathbf{r}_f$
      end if
      Compute the target for the TD update:
        target = $r + \gamma M_{\theta'}(\mathbf{o}', \mathcal{A}^*)$
      Compute the TD loss:
        loss = $(M_\theta(\mathbf{o}, \mathcal{A}) - \text{target})^2$
      Perform gradient descent on the TD loss to update M_θ
    end for
  end for
end function

the update step is performed for joint and split models. The update for FO is identical to SO, except omitting the update for the utility model.

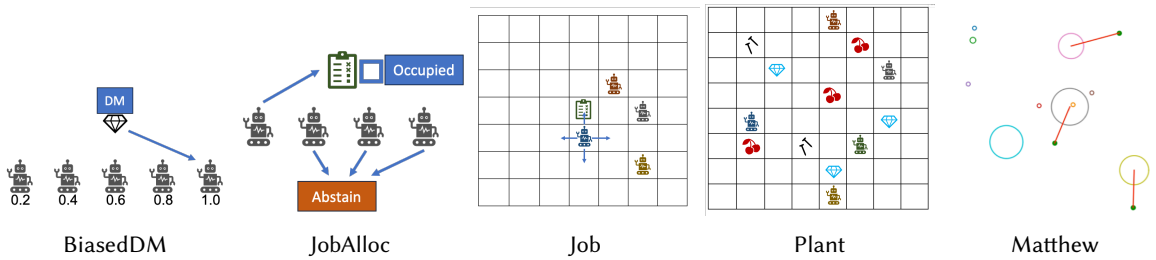


Fig. 5. Illustration of all five environments

B.1 Model Architecture and Training

All our models use a learning rate of 0.0003 with the Adam optimizer. For all environments except BiasedDM, we train for 1000 episodes, and run validation every 50 steps for model selection. For BiasedDM, we train for 200 episodes, and validate every 20 episodes.

The neural network architecture for all models is the same, with two fully connected hidden layers, of dimension 20, with ReLU activations. The output (1-dimensional) does not have any activation function. We implement our networks using pytorch.

We use a replay buffer of size 250000, where one experience is a joint transition across all agents. During training, we sample experiences from the replay buffer, and for each experience, we evaluate actions for all agents using the current online network, solving the ILP to get the best joint action. Then, we score the post-decision state for each agent using the target network, and compute the MSE loss between the target value and value estimates of the selected action from the online network.

We ran all our main experiments on a university compute cluster, with each experiment running on a single CPU node with 12GB RAM. Experiment runtime varied with environment choice. Training a single model with one β value took between 30 minutes (BiasedDM) and 2 hours (Matthew). Evaluation, as for the generalization experiments, was performed on a 2019 MacBook Pro, where one episode took 2-5 seconds to run, and we bootstrap over 5 runs and report the average performance.

C ENVIRONMENT DETAILS

Here, we provide further details about the environments for reproducibility. It is important to note that while these environments have been designed to have similar high-level as the versions used in the baselines [9, 27], they are completely different under the hood, incorporating agent and resource constraints and modifying the action spaces to align with the resource allocation tasks. The training code and environment implementations are included in the supplement attachment.

C.1 JobAlloc

JobAlloc focuses on coordination under exclusivity. A single job (resource) is available, and four agents must learn to take turns occupying it to ensure fair access. A greedy strategy allows one agent to monopolize the job, but a fair outcome requires agents to sometimes relinquish the job so others can benefit, which involves temporary self-sacrifice.

Action dynamics: At each timestep, all agents evaluate two actions: attempting to occupy the job or forfeiting it. However, only one agent may occupy it at a time, and a transition can only happen if no agent is on the job at the beginning of the timestep. This means successful transfer requires coordination: the current occupant must

vacate, so that another agent may attempt to claim it at the next time step. This constraint introduces a "gap step" where no one receives a reward, making fair transitions more costly and difficult.

There are 4 agents, and one episode lasts 100 steps.

C.2 Job

Job builds on JobAlloc by introducing a spatial grid, where agents must navigate toward a fixed job location. The job remains in the center of the grid, and only one agent can occupy it at a time. A fair outcome requires agents to not only reach the job but also time their arrivals to avoid collisions and take turns accessing the reward. This emulates the Job environment in Jiang and Lu [9] with added constraints.

Action dynamics: Each agent occupies a position on a 7×7 grid and can move in the four cardinal directions or remain still. The job location is fixed at the center. Agents independently score their movement preferences each timestep, and the centralized decision-maker resolves conflicts and determines the final set of moves. Movement is constrained—agents cannot leave the grid or move into an occupied location—so timing and coordination are essential to sharing the job fairly.

There are 4 agents on the grid, and one episode for this environment lasts 100 steps. Agents are able to observe nearby grid cells in a 3×3 area centered on the agent.

C.3 Plant

Plant is a multi-resource, multi-goal environment where five agents must collect combinations of three resource types to construct units and earn rewards. Each agent has a unique requirement profile—some easier than others—leading to natural inequality in unit completion rates. A fair allocation in this environment involves compensating for these differences to balance agent-level success over time.

Action dynamics: 5 agents operate on an 8×8 grid containing eight randomly placed resources of three types. Each agent submits preferences for which resource to pursue, and a centralized decision-maker allocates resources to avoid conflict. Agents move deterministically toward assigned resources. When an agent collects the needed combination of resources, they build a unit and receive a reward. Each agent has a requirement of a set of resources it must collect so that it can construct this unit. The requirements are:

$$\{(2, 1, 0), (1, 0, 1), (1, 0, 0), (1, 3, 0), (0, 1, 2)\}$$

For example, agent 1 requires two resources of type 1 and one resource of type 2 following which it can get a reward. Some agents are given easier requirements to fulfill, which creates a bias in the number of units agents produce. The resources and agent locations are randomly initialized at the beginning of each episode. The differing resource requirements add a combinatorial challenge to both planning and fairness. An action corresponds to an agent 'claiming' a resource, with other agents unable to pursue resources already allocated to other agents. Upon collection of a resource, another resource of the same type appears in a random location on the map.

There are 5 agents, and one episode for this environment lasts 200 steps. Agents are able to observe nearby grid cells in a 5×5 area centered on the agent.

C.4 Matthew

Matthew is designed to showcase the Matthew effect: early advantages amplify over time, leading to persistent inequality. Ten agents compete for three resources per timestep on a continuous 2D plane. Agent speeds grow proportionally to their size, and a size ceiling exists to prevent agents from growing too large for the environment bounds. Agent and resource positions are 2-D coordinates in $[0, 1]$. At the beginning of each episode, agent and resource positions are randomly initialized. To show the Matthew effect, 4 agents are initialized to have a larger initial size than other agents, making them more likely to secure early rewards—which further increase their

speed and size. Fairness here means interrupting this compounding feedback loop to ensure long-term balance in access and growth.

Action dynamics: At each step, agents estimate the utility of pursuing each available resource. The centralized decision-maker assigns which agent gets to pursue which resources. Once a resource is assigned, the agent moves in a straight line toward it and cannot switch targets mid-motion. Unassigned agents perform a null action resulting in random movement. Agents receive a unit reward when they collect a resource, in addition to a small increase in size and speed. Resources allocated to agents are reserved, and other agents cannot pick them up. A new resource only spawns when an agent reaches its allocated resource, not when it is allocated. The dynamics encourage cumulative advantage, as resource collection leads to further growth, making early interventions crucial to promoting fairness.

There are 10 agents, and one episode for this environment lasts 200 steps.

C.5 BiasedDM

BiasedDM is a minimal environment that isolates the trade-off between fairness and utility. In each timestep, a single resource is available, and the decision-maker must choose which agent among five will receive it. The twist is that the decision-maker's utility is biased: higher-indexed agents yield more utility. A fair policy in this setting must avoid allocating all resources to the most "valuable" agent and instead distribute allocations more evenly over time.

Action dynamics: At each step, agents communicate utilities for getting the resource or not getting it, and the decision-maker selects one of the five agents to receive the resource. Because the optimal (utility-maximizing) strategy is to always choose the same agent, fairness requires intentional deviation from that utilitarian solution to give other agents a share of the resource. This could entail the advantaged agent learning to have a smaller difference in its valuation of getting versus not getting the resource, allowing the decision-maker to hand it to another agent.

There are 5 agents, and one episode for this environment lasts 100 steps. In other environments, fairness is computed as the variance over the accumulated rewards for each agent. In this environment, however, fairness is computed over the resource rate, which is the fraction of steps in which an agent received the resource ($z_i \in [0, 1]$).

$$z_i = \frac{\text{Num. resources}}{time} \tag{37}$$

This also results in much smaller variances, thus our hyperparameter search for this domain explores the higher range of β values more.

D LEARNING WITH OTHER FAIRNESS FUNCTIONS

As mentioned in the main text, we are not restricted to using variance and the given decomposition. We show here results for three more functions: α-fairness, *GGF*, and maximin.

1. ***α-fairness.*** The α-fair function can be stated as:

$$F_\alpha(\mathbf{Z}) = \sum_{z_i \in \mathbf{Z}} \begin{cases} \frac{z_i^{1-\alpha}}{1-\alpha} & \alpha \neq 1 \\ \log z_i & \alpha = 1 \end{cases} \tag{38}$$

With $\alpha = 1$, this is equivalent to proportional fairness or log Nash Welfare, both popular notions of fairness, while at $\alpha = 0$ it represents the utilitarian objective. In our experiments, we use $\alpha = 1$.

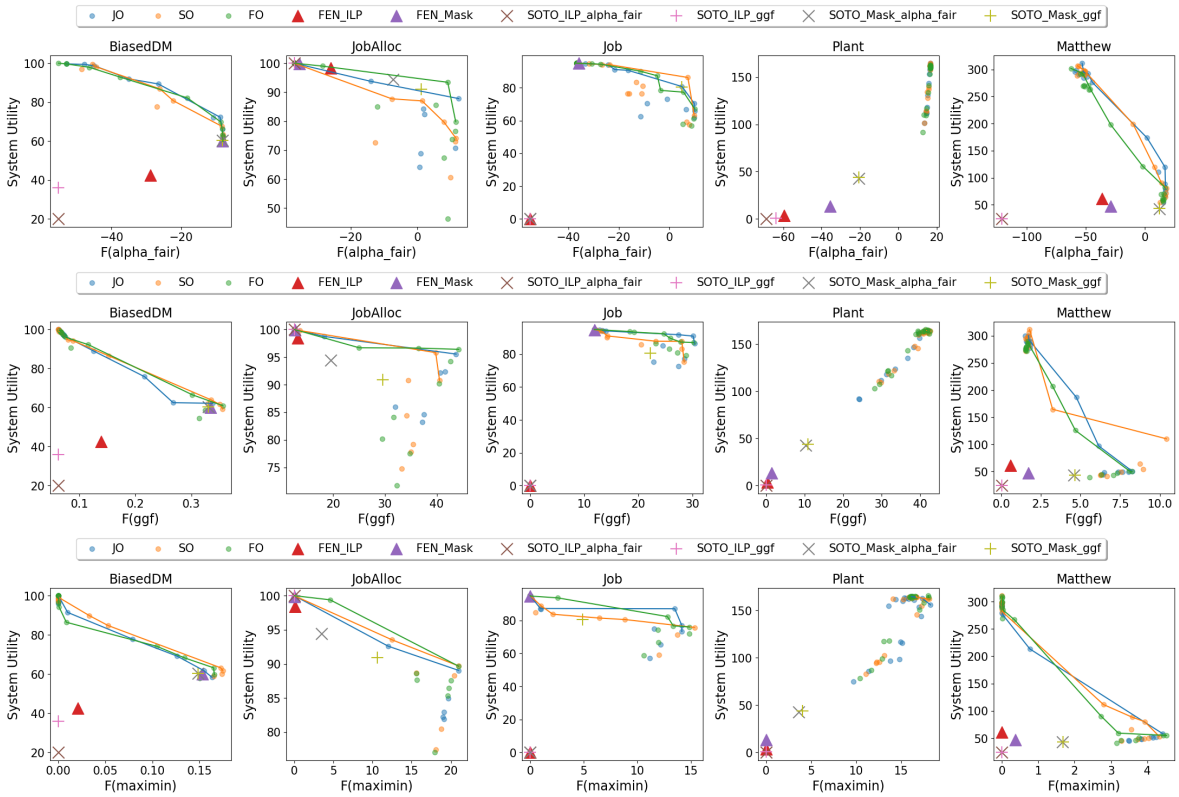


Fig. 6. Results training DECAF with α-fairness, GGF and maximin fairness functions. The lines show the Pareto fronts for each model type.

2. ***Generalized Gini Function (GGF).*** Given a sequence of positive, fixed, strictly decreasing weights $\mathbf{w}$, the GGF function can be stated as:

$$G_w(\mathbf{Z}) = \sum_{i \in \alpha} \mathbf{w}_i z_i^{\uparrow} \tag{39}$$

Here, $\mathbf{z}^{\uparrow}$ represents the vector obtained by sorting the $\mathbf{Z}$ vector. This function can also represent a diverse set of SWFs including utilitarian and maximin fairness. In our experiments, we use decreasing negative powers of 2 as the weights (i.e. $\mathbf{w} = [1, 2^{-1}, 2^{-2}, \ldots 2^{-(n-1)}]$).

For both of the above objectives, we use the equal decomposition, where the fairness reward is computed as an equal division of the change in the metric value across all agents.

$$R_f(\mathbf{s}_t, \mathcal{A}^t) = \left[\frac{\Delta \mathcal{F} | \mathcal{A}^t}{n} \right]_{i \in \alpha} \tag{40}$$

3. ***Maximin Fairness.*** This function captures the worst off utility of any agent:

$$F_{MMF}(\mathbf{Z}) = \min(\mathbf{Z}) \tag{41}$$

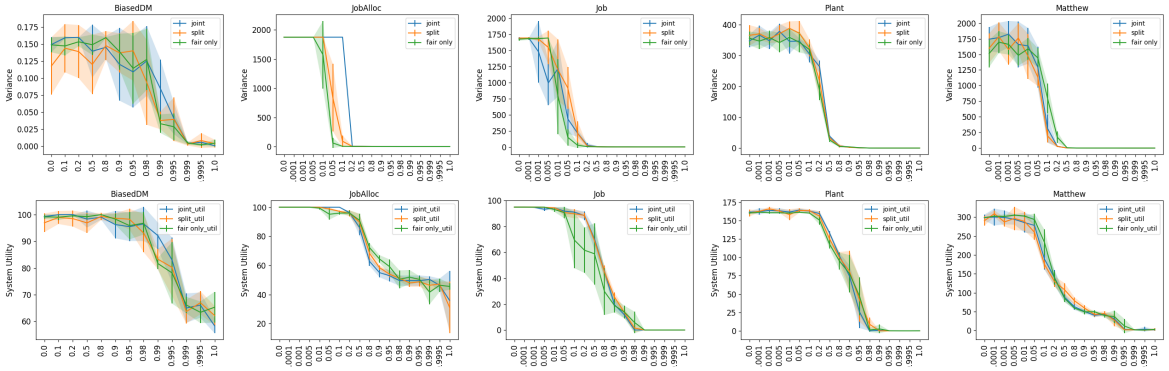


Fig. 7. Effect of changing β on variance (top row) and utility (bottom row) for all three methods on all five environments. The shaded area shows the 1-σ error bar. We observe that the performance of FO has large variation for intermediate β values.

This is a hard objective to learn, as the maximin objective changes only when the worst-off agent is improved. We decompose this reward by combining the global signal with the per-agent contribution towards improving the minimum. Intuitively, each agent receives a reward for a joint action that improves the minimum, but the agents that were at the minimum (and improved) receive a larger reward.

$$r_{f,i} = \frac{\min(\mathbf{Z}') - \min(\mathbf{Z})}{n} \tag{42}$$

$$r_{f,i} = r_{f,i} + \begin{cases} z'_i - z_i & \text{if } z_i = \min(\mathbf{Z}) \\ 0 & \text{otherwise} \end{cases} \tag{43}$$

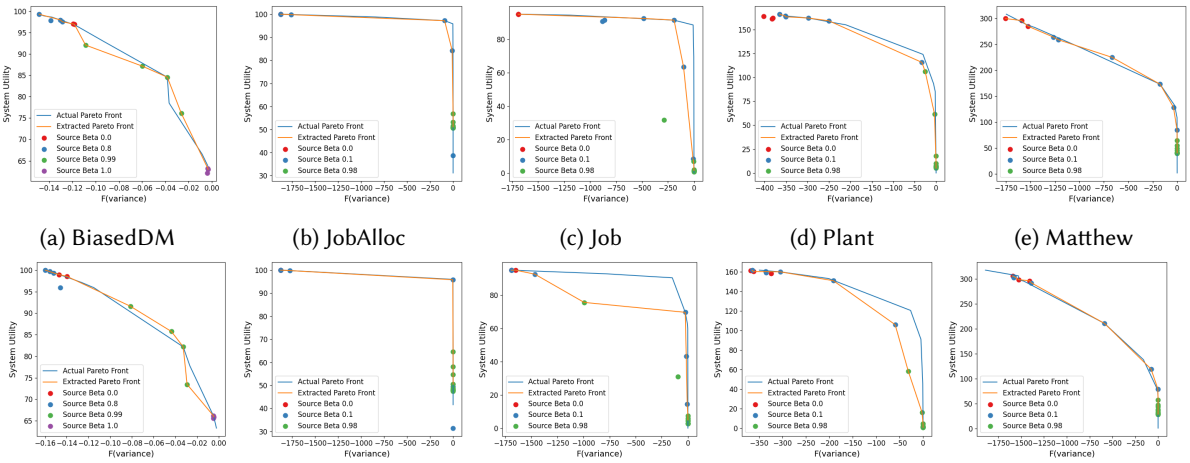
$$r_{f,i} = r_{f,i} + \begin{cases} z'_i - z_i & \text{if } z'_i = \min(\mathbf{Z}') \\ 0 & \text{otherwise} \end{cases} \tag{44}$$

$$R_{f,i} = \frac{r_{f,i}}{\sum_j r_{f,j}} \left(\min(\mathbf{Z}') - \min(\mathbf{Z})\right) \tag{45}$$

D.1 DECAF Results with Other Fairness Metrics

Figure 6 shows the results of our approaches when using different fairness functions, with the decompositions as described above. In general, we observe our methods offer a good range of trade-offs in all environments, often Pareto-dominating SOTO and FEN. We describe some additional details about these results in this section.

In almost all environments, it is possible to get significant fairness improvement starting from the utilitarian model. The only exception is the Plant environment for α-fair and GGF, where the utilitarian solution is almost optimal for fairness as well. In Job and JobAlloc, SOTO masked models are competitive, especially for the α-fair fairness function, while some DECAF models are not as performant. This may suggest the existence of a more informative decomposition for this function which can lead to better learning. In BiasedDM, SOTO and FEN face a significant disadvantage, as they can not contend with the misaligned fairness signal. Thus, they can either know the system utility, or the payoffs on which fairness is based. In the prior case, the learned fair policy performs poorly for both utility and fairness, as it tries to equalize the accumulated value from the decision maker's perspective, or, as in the latter case (selected for the results shown), only looks at the resource distribution and has no idea of the utility to the decision maker.



(a) BiasedDM (b) JobAlloc (c) Job (d) Plant (e) Matthew

Fig. 8. Approximate Pareto fronts for Split Optimization and Fair-Only Optimization models for variance across different environments. The top row represents SO models, while the bottom row represents FO models.

α-fair and GGF could also be decomposed to compute per-agent contributions, but since the metrics treat all agents indepependently, the interaction between agent utilities is harder to learn, especially for the Job and JobAlloc environment, where being fair requires a globally suboptimal decision in terms of utility. In other environments, the pressure for fairness is better captured in the ILP optimization, by the valuations of other agents that could benefit more from getting certain resources.

D.2 On the Importance of Past Discounting and Warm Starts

We also highlight two departures from prior work in our implementation of DECAF for learning fairness: (1) We discount the agent metrics $\mathbf{Z}$ over the past, and (2) we implement warm starts for initializing $\mathbf{Z}$ instead of initializing at 0. The past discounts are necessary to account for the time dependence of various normalized fairness metrics, like the Gini coefficient and coefficient of variation, or the variance over normalized agent metrics. In these cases, actions taken early on can cause larger changes to the fairness metric while actions taken after a sufficient history has been established have an imperceptible effect. This is undesirable in long-horizon settings, but can be remedied by past discounts, effectively 'forgetting' events that happened far in the past. The warm starts function to counteract the other pitfall in some fairness metrics: The zero vector is perfectly fair, and any changes can incur a large fairness penalty. Adding randomized warm starts helps in preventing the algorithm from converging to this trivial solution. In practice, we keep the warm start values small, so that the past discounts effectively scale them down to zero over the course of an episode.

D.2.1 Past Discounts and Warm Starts. Past discounts and warm starts significantly helped in improving the stability of learning in our experiments, especially with variance as the fairness function. Small initial perturbations (that are smoothed out over time using past discounts) help in exploration of different states, as well as in avoiding the cold start problem where any action would lead to a worse state and hence agents learn to avoid taking beneficial actions. For variance, we used warm starts based on the size of the maximum rewards possible in an episode in each environment. For BiasedDM, the warm starts are used to create an initial 'resource rate' by normalizing based on the number of total warm start resources, as this environment uses resource rates as the payoff vector $\mathbf{Z}$.

Table 2. Warm start (w) and past discount (γ_p) values for different environments and fairness functions used for DECAF.

		Matthew	Plant	Job	JobAlloc	BiasedDM
α-fair	w	0	0	0	0	0
	γ_p	1	1	1	1	1
GGF	w	0.1	0.1	0.1	0.1	0.1
	γ_p	1	1	1	1	1
Maximin	w	5	1	3	3	2
	γ_p	0.995	0.995	0.995	0.995	0.999
Variance	w	5	1	3	3	2
	γ_p	0.995	0.995	0.995	0.995	0.999

For α-fair, we used $\alpha = 1$ for the experiments, and hence we avoided using warm starts for this metric, as the log function is sensitive to small perturbations especially with near-zero utilities. For GGF, we used a warm start of 0.1 for each environment. For both of these functions, we used no past discounting, as the metrics are additive functions over agent utilities, so any past discounting would cause negative fairness gains. For maximin, we used the same past discounting and warm start values as variance. The warm start and past discount values for all environments are given in Table 2.

Given a warm start value of w, we compute an initial distribution of pseudo-resources by uniformly sampling from a region of width $w/4$ centered around w. For additive utilities, we use past discounts to decay the accumulated payoffs in $\mathbf{Z}$ before adding the current time-step's reward. If γ_p is the past discount factor, and $R_{t,i}$ is the resource value allocated to agent i at time t, then we compute:

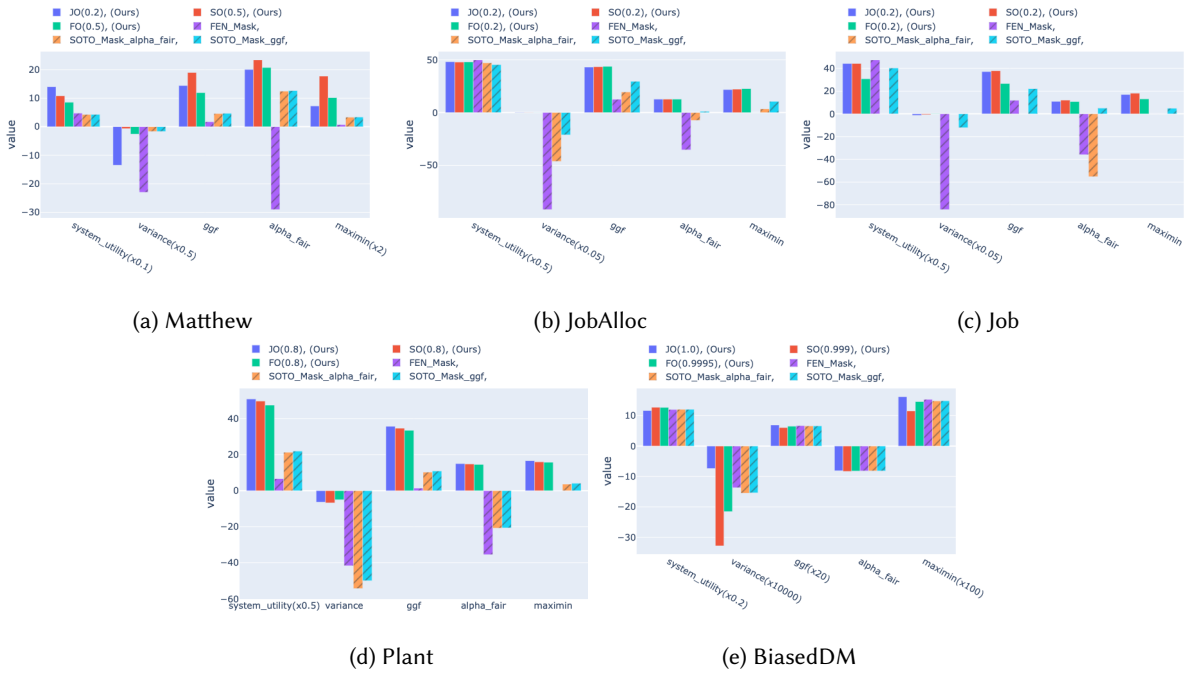
$$z_i^{t+1} = \gamma_p z_i + R_{t,i}$$

For averaged rewards (as in BiasedDM), where the payoff $z_i = \#resources/\#timesteps$, we compute the discount by reweighting both the numerator and denominator. Note that this can be easily extended to act as a 'resource rate' where the denominator is the number of potential resources the agent could have received, instead of the time.

$$\begin{aligned} z_i^t &= \frac{res_i}{t_i} \\ res_i &= \gamma_p res_i + R_{t,i} \\ t_i &= \gamma_p t_i + 1 \\ z_i^{t+1} &= \frac{res_i}{t_i} \end{aligned}$$

E FEN AND SOTO IMPLEMENTATION DETAILS

For FEN and SOTO, we use 5 times the number of training steps as used for DECAF, to allow the PPO based approaches sufficient trajectories to learn from, and to better match the experiments in the respective papers. We do not use past discounts and warm starts. For BiasedDM, we chose the reward vector to be the number of resources each agent received (1 for each agent), instead of the biased utility to the decision-maker ($0.2 \cdot i$ for agent i). Since our fairness metrics operate on the vector of resources as well, and since FEN and SOTO aim



(a) Matthew (b) JobAlloc (c) Job

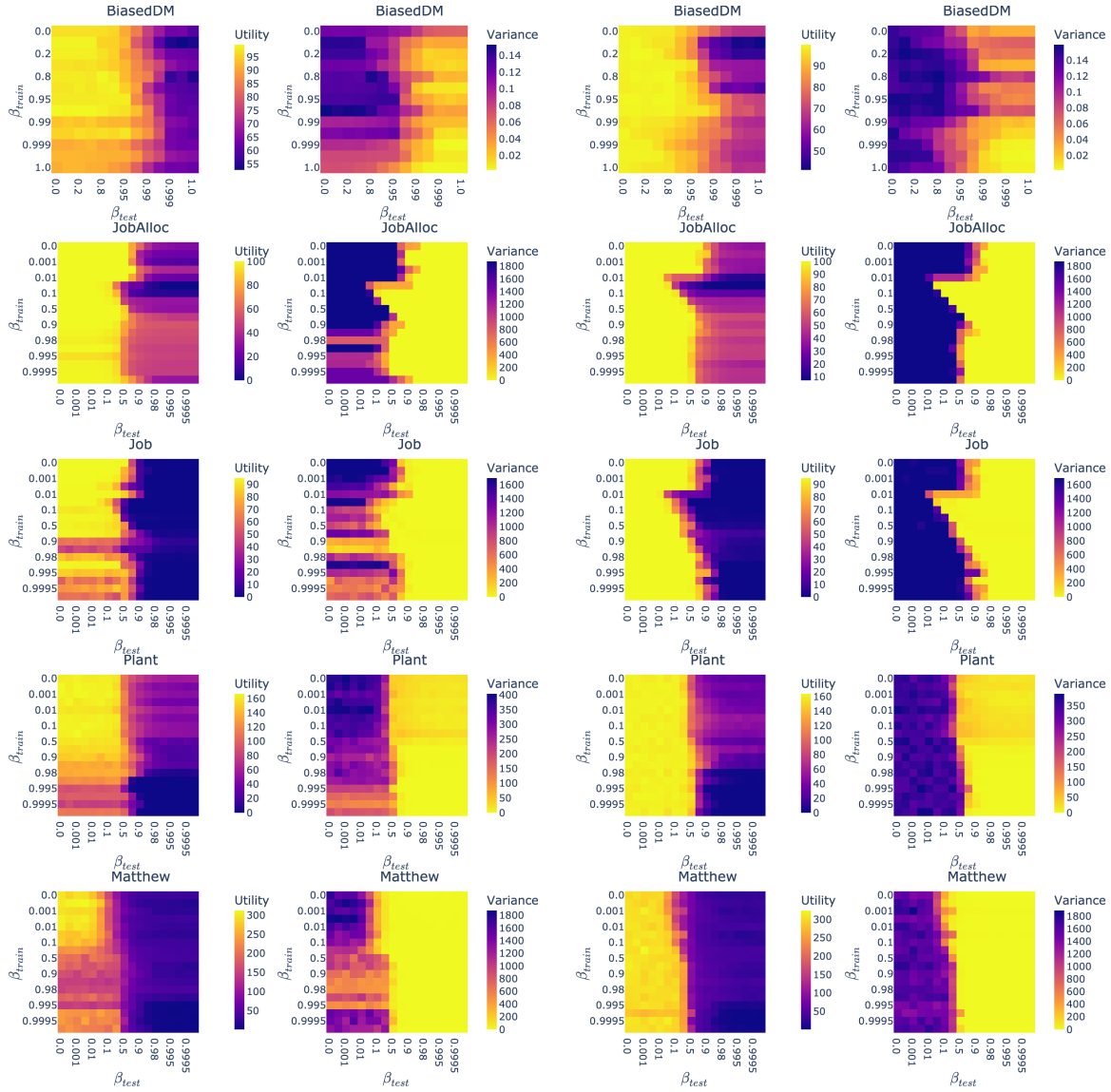
(d) Plant (e) BiasedDM

Fig. 9. Comparisons of selected DECAF models against the three baselines, scaled to fit on the same axes. We omit results for ILP versions of FEN and SOTO due to their poor performance. We selected DECAF models (trained on variance) that maximized $0.1U - 0.9\text{var}(\mathbf{Z})$. The numbers in brackets denote the selected β value for our models.

to learn fairness, this was the obvious choice. We also ran experiments where the reward vector was based on the decision-maker's biased utility, and found the solutions to be poor for both utility and fairness based on resources, so we omit them in our results. However, the greedy self-oriented model in SOTO was trained using the decision-maker's reward.

We used FEN without gossip, where the agent is directly communicated the distribution information, without need for inter-agent communication rounds. This is expected to be the stronger version of FEN. For all models, we used the same features from the DECA environments, containing the agent's relative advantage as a feature. In addition to this, SOTO also required the entire payoff distribution $\mathbf{Z}$ and information about nearby neighbors for the tiered team-oriented network.

The Job environment uses a shaping reward for the distance to the job as a scaled penalty. For the Job environment for SOTO, we reduced the weight of the shaping reward to 0.01 to minimize its effect on the optimization. We found this to be the best setting for learning in this environment. The ILP version of SOTO was not able to learn at all in this setting, even when we completely removed the shaping reward. Again, the inability to use shaping rewards is a significant handicap for methods like SOTO, since they optimize fairness over the rewards. Our methods explicitly decouple learning utility and fairness, so they are more expressive, and able to learn better especially at intermediate values of β.



(a) SO models (b) FO models

Fig. 10. Evaluation of Split Optimization (left) and Fair-Only Optimization (right) models trained on β_{train} (for variance) and evaluated on β_{test} across different environments. Brighter colors indicate better outcomes.

F EXTENDED RESULTS FOR VARIANCE

Here we provide additional results for our methods, including providing confidence intervals for our main results, additional results for generalization, and more comparison to baselines.

F.1 Confidence Intervals for Results

We provide confidence intervals for system utility and fairness separately as we vary β, as plotting this on a Pareto plot (as in the main results) would be hard to read (Figure 7). An interesting thing to note here is that we see a phase lag between when variance starts to reduce, and when utility starts to drop. This shows there is a range of solutions where fairness can be improved without significantly harming the utility.

F.2 Approximating the Pareto Front

We also show generalization results by generating approximate Pareto fronts for all methods (Figure 8) based on a limited set of β_{train} models, showing that the SO and FO methods generalize very well even without training for all intermediate β values. One interesting thing to note here is that FO falls under the Pareto front with intermediate β values, showing how a tuned utility model also helps in guiding agents towards better decisions.

F.3 Comparison of Selected Models

Figure 9 shows the performance of selected DECAF models trained on variance when evaluated on different metrics on all environments, compared to the baselines. We can see DECAF models perform much better on all metrics. For BiasedDM, all methods were able to converge to approximately the same fairest policy, so the differences in the figure are small. The variance for SO might appear large, but we point out that this variance is scaled 10000 times, and the actual values are all very close to zero.

F.4 Generalization Heatmaps

We also provide the generalization results for varying β_{test} for both Split (Figure 10a) and Fair Only (Figure 10b) models for all environments. We note that the general trends noted in the main experimental results hold, with each model able to improve fairness as β_{test} is increased. Further, in all cases, FO maximizes utility at $\beta_{test} = 0$, and has a sharper transition between utility-maximizing and fairness-maximizing behavior.

G POTENTIAL SOCIAL IMPACT

This paper presents a general formulation for improving fairness in multi-agent resource allocation with a centralized arbitrator. Care should be taken when deploying our algorithms to real systems with human stakeholders, as the selection of fairness criteria can strongly affect resource allocation dynamics in ways that may not favor all stakeholders. For example, better-off stakeholders may lose some of their utility. The fairness function should be selected keeping the interests of all parties in mind. Mis-specified objectives can lead to undesirable and unfair outcomes.