

# Learning Actionable Counterfactual Explanations in Large State Spaces

Keziah Naggita    Matthew R. Walter    Avrim Blum  
Toyota Technological Institute at Chicago  
{knaggita, mwalter, avrim}@ttic.edu

## Abstract

Counterfactual explanations (CFEs) are sets of actions that an agent with a negative classification could take to achieve a (desired) positive classification, for consequential decisions such as loan applications, hiring, admissions, etc. In this work, we consider settings where optimal CFEs correspond to solutions of weighted set cover problems. In particular, there is a collection of actions that agents can perform that each have their own cost and each provide the agent with different sets of capabilities. The agent wants to perform the cheapest subset of actions that together provide all the needed capabilities to achieve a positive classification. Since this is an NP-hard optimization problem, we are interested in the question: can we, from training data (instances of agents and their optimal CFEs) learn a CFE generator that will quickly provide optimal sets of actions for new agents?

In this work, we provide a deep-network learning procedure that we show experimentally is able to achieve strong performance at this task. We consider several problem formulations, including formulations in which the underlying “capabilities” and effects of actions are not explicitly provided, and so there is an informational challenge in addition to the computational challenge. Our problem can also be viewed as one of learning an optimal policy in a family of large but deterministic Markov Decision Processes (MDPs).

## 1 Introduction

Machine learning models are increasingly being used to guide consequential decision-making, e.g., for hiring and school admissions. Since these decisions can significantly impact people’s lives, society demands the right to explanation, as stated in Articles 13–15 of the European Parliament and Council of the EU [9] General Data Protection Regulation. One of the most needed explanations is of how people (agents) can modify their input features (i.e., their state) to get a (desired) positive classification from these models. Counterfactual explanations (CFEs) provide insights into which input features to change and to what extent to achieve a desirable model outcome [26, 6, 19]. CFEs are transformed into actionable feedback to make it easy for agents to act on the insights, that is to say, negatively classified agents receive recommendations as to how to optimally act to get desirable model outcomes [23, 13, 12, 14].

We consider settings where an optimal CFE corresponds to the solution of weighted set cover problem. Specifically, given a set of actions with known costs and known effects on state features, the problem is to find the lowest-cost subset of actions that achieve all capabilities (state features greater than or equal to a given threshold) needed to be positively classified. This formulation is, in principle, similar to search-based optimization CFE generation frameworks [21], user-specific integer linear program recourse approaches [23, 5, 11], and logic and answer-set programs based counterfactual generation [4, 17, 18]. For example, a bank loan officer with access to a binary

	CFE name	Action(s) name	Action(s) cost	Action(s) effects	Classifier query-access
Weighted set cover problem	×	✓	✓	✓	✓
Named-action	×	✓	×	×	×
Full-action	×	✓	×	✓	×
Named	✓	×	×	×	×

Table 1: We propose three kinds of data-driven CFE generators: named-action, full-action and named CFE generators, each with a specific set of informational challenges it works with.

classifier that determines loan eligibility and a set of actions, their underlying effect on the agent features, and their costs can use the solution of the weighted set cover problem to suggest the least expensive subset of actions that can help an ineligible borrower become eligible for a loan.

However, this formulation, like similar contemporary approaches to CFE generation heavily relies on information access assumptions, for example, at the very least, query access to the classifier and knowledge of the actions, their costs, and underlying effects on agent features. These assumptions rarely hold in practice which limits the usage of the CFE generators. We are, therefore interested in generating CFEs for settings where different information challenges might exist (see Table 1).

Consider, for example, unlike the previous setup, the loan officer assistant only knows the name of actions but not their costs and underlying effects on agent state features or even query access to the loan eligibility classifier. In this setting, the assistant recommends a *named-action* CFE, an optimal set of named-actions to the agent, e.g., increase bank credit (named-action), without information on the cost of the named-actions, or how the actions might affect the agent’s state, but only relying on how agents that had a similar profile (current agent state) became loan eligible after increasing their bank credit.

In another setting, the loan officer, even with knowledge of the actions and their underlying effects on the agent’s state, might not know the cost of those actions. Relying on the history of which CFEs worked well for agents with a similar state, they then recommend a *full-action* CFE, an optimal set of full-actions, to help a new agent in a similar state become loan-worthy, e.g., increase credit score by 20% (full-action). This setting would preclude the use of contemporary CFE generators that require knowledge of the action costs and at least query access to the classifier.

Alternatively, there are instances where there is very limited information access, where the decision-makers have no query access to the classifier and the actions, their costs, and underlying effects on the features is unknown. In this case, the decision-maker might have access to *named* CFEs, where the name of a CFE holds significant implicit information, and there is an expectation the agent taking the CFE possesses some domain expertise. For example, a mortgage broker without query access to the bank loan eligibility classifier, actions, their costs, and underlying effects on the agent features might recommend that an agent wanting to take out a mortgage first takes a personal loan x years prior (named CFE), without specifying what sub-actions the agent needs to take.

Other than the aforementioned informational challenges to CFE generation, generating an optimal CFE as a solution of weighted set cover problem is an NP-hard optimization problem [15]. Additionally, like other similar CFE generation methods, this CFE generator requires solving an optimization problem for each new agent which is not scalable and is significantly expensive. Given all these informational and computational challenges, we are interested in the question: *can we, from training data (instances of agents and their optimal CFEs), learn a CFE generator that will*

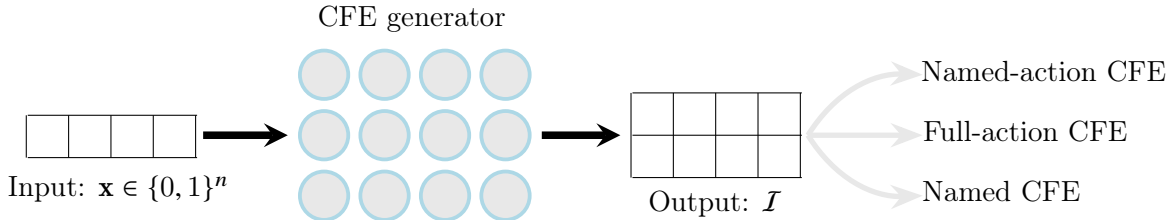


Figure 1: Data-driven CFE generators trained on agent-CFE data generate information specific CFEs ( $\mathcal{I}$ ) for new agents given their state ( $\mathbf{x}$ ) without re-optimizing the CFE generator.

*quickly provide optimal sets of actions for most new agents?*

In this work, we provide a neural network learning procedures (see Figure 1) that we experimentally show to achieve strong performance on this task. We consider several problem formulations, including those in which the classifier and grouped access to actions are not explicitly known, and the underlying effects of the actions are not explicitly provided.

## 1.1 Related Work

Closest to our work is that of Naumann and Ntoutsis [20] and Verma et al. [25]. While our formulation is synonymous with learning an optimal policy in a family of large but deterministic Markov decision processes (MDPs), Verma et al. [25] learns an optimal policy in stochastic, but relatively small MDP settings. While Naumann and Ntoutsis [20] optimizes their CFE generation method for each agent to find an optimal ordered sequence of actions to recommend, in our generalized CFE generation approach, we assume that the train-set CFE is optimal, and we don't re-optimize the model to generate a CFE for each new agent.

Our approach to generating CFEs is similar to that of CFE generation tools that use reinforcement learning [7, 22]. However, due to computational complexity, these tools only provide approximate solutions, while our generators produce exact solutions.

Our main contribution is CFE generators that are computationally efficient and work under varied information settings, thus providing a solution to the informational and computational challenges of CFE generation in settings where actions correspond to solving a weighted set cover problem. Unlike other methods that need, at the very least, query access to the classifier as well as knowledge of the cost of each action and its effect on state features [20, 7, 22, 25], our formulation effectively generates CFEs in the absence of explicit access to this information.

Lastly, our work is also related to data-driven algorithm design [10, 3, 2] in which models trained on training data instances perform well on the training data and generalize to the testing data. Specifically, given testing data: instances of agents and their optimal CFEs (of varying information), we train neural-network models that we empirically show achieve strong performance at generating optimal CFEs for new instances.

## 2 Motivation

Consider a binary classifier  $\mathbf{t} = \{t_1, t_2, \dots, t_n\}$  over  $n$  features that classifies an agent as positive if its state  $\mathbf{x}$  satisfies the condition:  $x_j \geq t_j, \forall j \in [n]$ , and negative otherwise. We consider scenarios in which agents that are classified as negative can take a set of actions to acquire capabilities required to receive a positive classification.

An action  $\mathbf{a} \in \mathcal{A}$  provides the agent with different sets of capabilities to transform features of the agent’s state. An action transforms the feature  $j$  of the agent state,  $\mathbf{x}$ , if after performing the action, the new agent state  $\mathbf{x} + \mathbf{a} = \mathbf{x}'$ , is such that  $x'_j > x_j$  and  $x'_j \geq t_j$ .

An action provides a negatively classified agent with all necessary capabilities, if after performing the action, the state of the agent changes from the initial state  $\mathbf{x}$  to a counterfactual state  $\mathbf{x}^*$ , such that  $x_j^* \geq t_j, \forall j \in [n]$ , where  $\mathbf{x}^*$  results in a positive classification. A counterfactual state is the final state with which the agent has all the needed capabilities. If the agent requires more than one action to achieve these capabilities, actions are performed sequentially to transform the agent’s initial state to the counterfactual state.

## 2.1 Weighted Set Cover Problem

Let there be a collection of actions  $\mathcal{A}$  that an agent can perform, where associated with each action  $a_i \in \mathcal{A}$  is its cost  $c_i \in \mathbf{c}$  and a specific set of effects that that action has on the agent’s state. A negatively classified agent,  $\mathbf{x}$ , wants to perform the cheapest subset of actions  $I \subseteq \mathcal{A}$  that together provide all the needed capabilities to achieve a positive classification, i.e.,  $\mathbf{x} + I = \mathbf{x}^*$ , s.t.  $x_j^* \geq t_j, \forall j \in [n]$ .

**Example** Given an agent  $\mathbf{x} = [0, 0, 0, 0, 1]$ , a binary classifier  $\mathbf{t} = [1, 1, 1, 1, 1]$ , and actions  $\mathcal{A} = \{[0, 0, 1, 0, 1], [1, 0, 0, 0, 1], [0, 0, 1, 1, 0], [0, 0, 0, 0, 1], [0, 1, 0, 0, 0], [1, 1, 1, 1, 1], [0, 0, 0, 0, 0], [1, 0, 0, 0, 0], [0, 1, 1, 0, 0]\}$  with their associated costs  $\mathbf{c} = [6, 5, 9, 2, 1, 15, 0, 3, 5]$ , the goal is to find the lowest-cost subset of actions that together provide the agent all the needed capabilities.

Formulating this as a weighted set cover problem, we find the best (lowest-cost) subset of actions for the agent to take as  $I = \{[0, 0, 1, 1, 0], [0, 1, 0, 0, 0], [1, 0, 0, 0, 0]\}$ , at a total cost of 13.0. Sequentially performing the actions in  $I$  changes the agent state from the initial state  $\mathbf{x} = [0, 0, 0, 0, 1]$  to the counterfactual state  $\mathbf{x}^* = [1, 1, 1, 1, 1]$ .

**Integer Linear Program (ILP)** We mathematically formulate the weighted set cover problem as an integer linear program [24].

Given a set of actions  $\mathcal{A} \in \{0, 1\}^{A \times n}$  and their costs  $\mathbf{c} \subseteq \mathbb{R}^A$ , and a negatively classified agent with initial state  $\mathbf{x} \in \{0, 1\}^n$ , the problem is to find the lowest-cost subset of actions  $I \subseteq \mathcal{A}$  that the agent can take to reach the counterfactual state.

Let the binary matrix  $\mathcal{D} \in \{0, 1\}^{A \times n}$  denote the relationship between the features to transform and actions the agent can take, where  $d_{ij} \in \mathcal{D} = 1$  if action  $i$  transforms feature  $j$ , and 0 otherwise. Therefore, we define  $\{i \in \mathcal{A} : d_{ij} = 1\}, j \in [n]$  as the number of actions that can transform feature  $j$ , and  $\{j \in [n] : d_{ij} = 1\}, i \in \mathcal{A}$  as the number of features action  $i$  can transform. If we choose action  $i$ , then  $a_i = 1$  and 0 otherwise. Formally, we define the integer linear program as follows:

$$\begin{aligned} & \text{minimize} && \sum_{i \in \mathcal{A}} c_i a_i \\ & \text{subject to} && \sum_{i \in \mathcal{A}} d_{ij} a_i \geq t_j, \forall j \in [n], \\ & && a_i \in \{0, 1\}, d_{ij} \in \{0, 1\} \end{aligned}$$

Therefore, for each agent with a negative classification, the lowest-cost (optimal) CFE required to achieve the capabilities needed for a positive classification is given by  $I = \{i : a_i = 1\}$ . The cost of the optimal CFE is given by the total cost of the actions in the CFE,  $\sum_{i \in I} c_i$ .

**Challenges of using the weighted set cover problem CFE generator** In principle, as described above, the weighted set cover problem can be solved by exhaustive search over all the combinations of the actions. In our empirical experiments, an agent can use one action or a set of two/three actions to get all the necessary capabilities needed to achieve a positive classification. It’s inexpensive to try all the  $(A/3)$  ways of finding the most optimal combination (set) of actions. However, in the worst-case scenario, a search through all the combinations of actions,  $2^A - 1$  is needed to find the optimal set of actions the agent needs to acquire all the needed capabilities. Therefore, exhaustive search would have an exponential time complexity  $O(n2^A)$ . Although removing already covered/satisfied features of an agent could reduce the complexity by a factor, the CFE generator would still be computationally expensive.

In addition to being computationally expensive, this nature of CFE generation requires solving an optimization problem for each new agent, and needs access to the classifier. Also, given real-world settings, like health care, especially counterfactual explanations in medical settings, where health professionals recommend interventional actions based on the current state of the patient and similarity to other patients who were in a similar state, CFE generation as a solution to the weighted set cover problem doesn’t adequately leverage advantages of historical counterfactual explanations.

### 3 CFE Generation as a Learning Problem

Due to computational and informational challenges of CFE generation in settings where actions correspond to solving a weighted set cover problem, we propose various CFE generators based on the limited and specific information accessible to the decision-maker, and without having CFE generation as an optimization problem for every single agent. Below are three kinds of proposed data-driven CFE generators.

#### 3.1 Named-action CFE Generators

The named-action (NA) CFE generator addresses the *informational challenge* where only the names of the actions in the CFE are known. Explicit information regarding the cost of actions or their explicit effects on features is unknown, and decision-makers have no query access to the classifier. The named-action CFE generator is considerably cheaper than the weighted set cover problem CFE generator, and the number of CFEs is upper bounded by the number of training examples.

The named-action CFE generator is trained on instances of agents and their optimal CFEs for which the names of the actions in the CFE are known, but not their effects on features. Each action in the collection of actions  $\mathcal{A}_{\text{name}}$  is represented by its name. Each negatively classified train-set agent  $\mathbf{x}_i \in \mathcal{X}_T$  has an optimal named-action CFE that is a subset of the actions,  $I_i \subseteq \mathcal{A}_{\text{name}}$ . The CFE  $I_i$  is represented by a binary vector, where  $I_i^{(a)} = 1$  if the  $a^{\text{th}}$  action is in the optimal CFE vector  $I_i$  and 0 otherwise. For each agent  $\mathbf{x}_i$ , the named-action CFE generator computes a vector score for each action using a linear combination of the input features and CFE generator weights  $\mathbf{z}_i = \mathbf{x}_i W$ . The probability that a generated action,  $\hat{I}_i^{(a)}$  is among the actions in the optimal CFE for the agent  $\mathbf{x}_i$  is given by the sigmoid function:

$$\hat{I}_i^{(a)} = \sigma(\mathbf{z}_i^{(a)})$$

To ensure that the named-action CFE generator performs well on the train-set,  $\mathcal{X}_T \in \{0, 1\}^{m \times n}$  and generalizes to the test-set, we optimize the binary cross entropy loss function given by:

$$\mathcal{L}_{\text{NA}} = -\frac{1}{m} \sum_{i=1}^m [I_i \log(\hat{I}_i) + (1 - I_i) \log(1 - \hat{I}_i)]$$

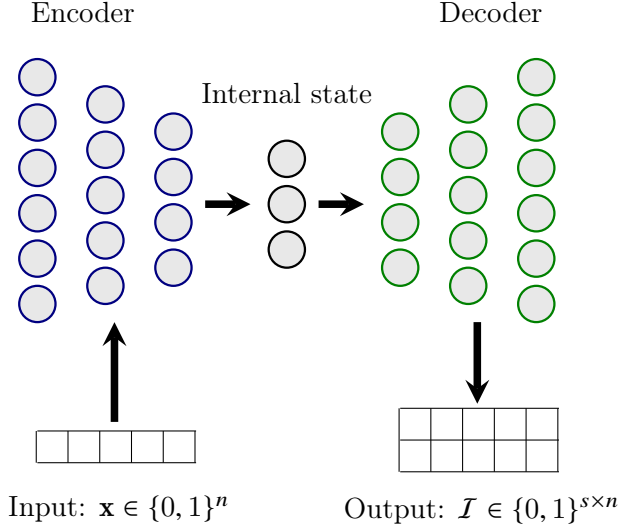


Figure 2: An encoder-decoder full-action CFE generator.

The binary vectored named-action CFEs were sparse because agents had either 1, 2, or 3 named-actions in their CFEs, and there were at most 100 possible named-actions. As a result, the named-action CFE generator was susceptible to overfitting. Additionally, some named-actions were more common than others especially the second and third named-actions in three-action CFEs were less common than those in one-action CFEs.

To address named-action imbalance and overfitting, we weight and regularize the loss function,  $\mathcal{L}_{\text{NA}}$ , as follows:

$$\mathcal{L}_{\text{NA}}^w = p_w \mathcal{L}_{\text{NA}} + \alpha \frac{1}{m} \sum_{i=1}^m |\hat{I}_i - I_i|$$

The weighting factor  $p_w$  weights the loss function  $\mathcal{L}_{\text{NA}}$  to help with named-action imbalance by scaling the contribution of each sample to the loss function. The sparsity term  $\alpha \frac{1}{m} \sum_{i=1}^m |\hat{I}_i - I_i|$  regularizes the model, thus preventing overfitting, where  $\frac{1}{m} \sum_{i=1}^m |\hat{I}_i - I_i|$  encourages the model to produce named-action CFEs closer to  $I_i$ 's distribution.

### 3.2 Full-action CFE Generators

The full-action (FA) CFE generator addresses the *informational challenge* where all the exhaustive space of actions is unknown and the decision-maker has no query access to the classifier. Only actions in the CFE, their effects on features are known.

Given instances of agents  $\mathbf{x}_i \in \mathcal{X}_T$  and their optimal CFEs  $I_i \in \{0, 1\}^{s \times n}$  for which the underlying and effects of the  $s$  full-actions on the agent features are explicitly provided, the goal is to learn a full-action CFE generator that will quickly provide optimal CFEs for new agents. We model the generator using two kinds of neural networks: a sequential encoder-decoder network (see Figure 2) and a multi-label classifier. In the case of the multi-label classifier, we convert the CFEs into padded binary vectors and apply the same loss function as described in the named-action CFE generators in Section 3.1.

### 3.3 Named CFE Generators

The named CFE generator addresses the *informational challenge* where only the CFE names and costs are known. Explicit information about the actions in each CFE, including actions’ costs and explicit effects on features is unknown, and decision-makers have no query access to the classifier.

In this setting, the decision-maker generates CFEs with only the names and costs of the CFEs at hand. Each negatively classified train-set agent  $\mathbf{x}_i \in \mathcal{X}_T$  has an optimal CFE  $I_i \in \mathcal{I}_T$ , where  $I_i$  is the optimal CFE name. To generate the optimal CFE for a test-set agent, we use a named CFE generator trained on the train-set to learn the patterns between agents and their optimal named CFEs. Following are the two proposed named CFE generators.

**Hamming distance** As a baseline, we consider the hamming distance named CFE generator. In this case, given a negatively classified test-set agent  $\mathbf{x}_v$ , we compute the Hamming distance (see Figure 3) between it and each of the train-set agents  $\mathbf{x}_i, \forall i \in [m]$ . Based on the computed distances, we choose the  $k$  nearest train-set agents and their associated optimal CFEs. We then use the most common CFE as the CFE for the test-set agent  $\mathbf{x}_v$ .

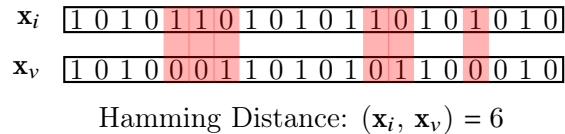


Figure 3: Hamming distance between the train-set agent  $\mathbf{x}_i$  and test-set agent  $\mathbf{x}_v$ .

**Multi-class classification** We consider a more sophisticated model, a neural network-based named CFE generator, i.e., a multi-class classification (MC) named CFE generator. First, we one-hot encode the CFE names to  $\mathcal{I}_T \subseteq \{0, 1\}^K$ , where  $K$  is number of unique named CFEs. For each agent  $\mathbf{x}_i \in \mathcal{X}_T$ , the CFE generator computes a vector score for each CFE using a linear combination of the input features and CFE generator weights  $\mathbf{z}_i = \mathbf{x}_i W$ . The probability that a generated CFE  $\hat{I}_i^{(k)}$  is the lowest-cost CFE for the agent  $\mathbf{x}_i$  is given by the softmax function:

$$\hat{I}_i^{(k)} = \frac{e^{\mathbf{z}_i^{(k)}}}{\sum_{j=1}^K e^{\mathbf{z}_i^{(j)}}}$$

To ensure that the CFE generator performs well on the train-set,  $\mathcal{X}_T \in \{0, 1\}^{m \times n}$  and generalizes to the test-set, we optimize the cross entropy loss:

$$\mathcal{L}_{\text{MC}} = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K I_i^{(k)} \log(\hat{I}_i^{(k)})$$

## 4 Experimental Setup

In this section, we describe the setup used to empirically demonstrate the strong performance with which our method generates optimal CFEs for new instances of negatively classified agents.

## 4.1 Data generation

We generate the agents by activating each feature with probability  $p_f$ . We then create a collection of actions  $\mathcal{A} \in \{0, 1\}^{A \times n}$  by setting each action  $\mathbf{a} \in \mathcal{A}$  to 1 with probability  $p_a$ . Each action coordinate has its own pre-generated cost, and the cost of active action coordinates (those with capabilities) determines the cost of the action. Using solutions of the weighted set cover problem based on the generated actions and the agent datasets, we create datasets of instances of agents and their associated optimal CFEs. We generate three kinds of agent-CFE datasets that exhibit variations in their dimension, binary classifier sparsity, and actions access:

- Varied data dimensions** We created three datasets, varying the number of features ( $n = 20, 50, 100$ ) and keeping  $p_f = 0.68$  the same for all datasets. In all three cases, the binary classifier was a unit vector of length  $n$ . We created the actions dataset with  $p_a = 0.5$ . Each action coordinate cost was randomly assigned in the range  $n$ . The cost of each action equaled the sum of the cost of action coordinates.
- Varied binary classifiers sparsity** We generated five,  $n = 20$  datasets with varied binary classifier sparsity. The first dataset (**Last5**) has all zeros except for the last five features. The second dataset (**First5**) has all zeros except for the first five features. The third dataset (**First10**) has all zeros except for the first ten features. The fourth dataset (**Last10**) has all zeros except for the last ten features. Finally, the fifth dataset (**Mid5**) has a binary classifier for which the five middle features are one and the rest are zero.
- Varied access to actions** We also consider a setting where grouped agents have restricted access to a set of actions. We consider two main settings: 1) actions generated with the same probability  $p_a = 0.5$  and agents are randomly assigned a subset of actions; and 2) agents assigned to groups and each group has its own actions generated by different probabilities  $p_a = [0.4, 0.5, 0.6, 0.7, 0.8]$ . In what follows, we refer to the first setting as **manual groups** and the second as **probabilistic groups**. See Figure 4a and Figure 4 for the statistics of the datasets.

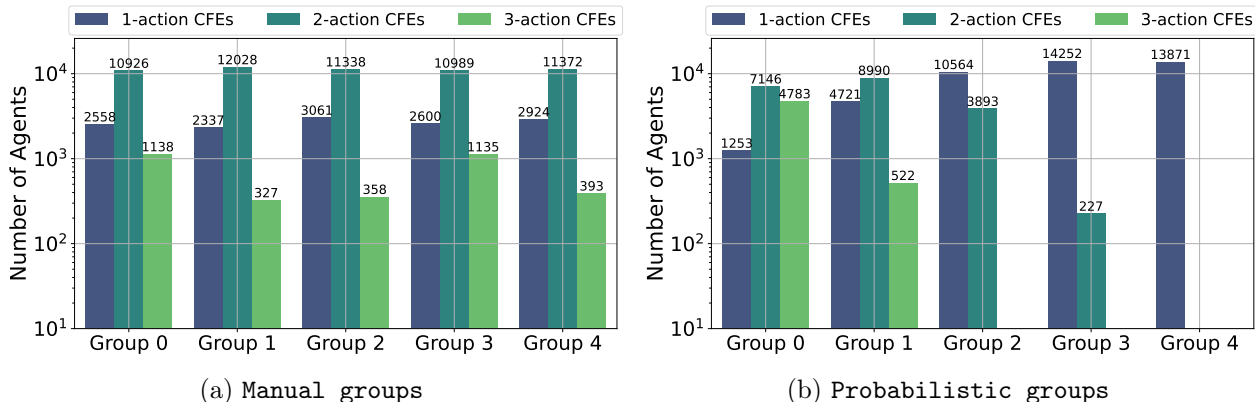


Figure 4: Statistics of the varied actions access datasets for **Manual groups** and **Probabilistic groups**. For the **Probabilistic groups**, group 0 is  $p_a = 0.4$ , group 1 is  $p_a = 0.5$ , group 2 is  $p_a = 0.6$ , group 3 is  $p_a = 0.7$ , and group 4 is  $p_a = 0.8$ .

Each agent had exactly one optimal CFE, i.e., there were no equally cheap alternate CFEs. We excluded four-action CFE agents because they were very limited in number (e.g., only 4 agents of



the 20-feature dataset), and even after the data split, they were in train-set. In the case of *varied binary classifiers sparsity* datasets, due to reduced thresholds required for validity, some agents did not need CFEs and were therefore excluded.

For each dataset of agents and their associated optimal CFEs, we created three datasets: `all` where all data is included, `>10` where a frequency of more than 10 agents per CFE is ensured, and `>40` where a frequency of more than 40 agents per CFE is ensured. We then encode the CFEs into named-action, full-action and named CFEs, fit for training and testing each specific proposed CFE generator. We create 80%/20% train-set/test-set splits for each of the datasets. In the case of the `>40` dataset, each CFE has a frequency of at least 20 in the train-set. We generate a different data split for every experiment and all methods use the same data split.

## 4.2 Data observations

Although most actions have effects on multiple features of the agent state, most agents require more than one action to achieve all the capabilities needed for a positive classification, which increases with the data dimension,  $n$ . Additionally, the uniqueness of CFEs increases with the data dimension. The average frequency of the CFEs for the `all` train-set for the 20, 50, and 100 feature datasets is 46.64%, 21.75%, and 8.09%, respectively. Additionally, 18.115%, 20.797%, and 31.072% of the CFEs of the train-sets of the 20, 50, and 100 feature datasets, respectively have a frequency of one.

In `Manual groups` datasets, agents are more balanced with respect to the number of actions needed (see Figure 4a) because the agents have access to the same distribution of agents, i.e., although agents in each group have access to only a selected group of actions, all actions for all groups were generated with the same probability,  $p_a = 0.5$ . However, for the `Probabilistic groups` datasets, as seen from Figure 4b, the lower the probability of action capabilities  $p_a$ , the more actions agents need to acquire all the capabilities needed for a positive classification. In other words, some agents in some groups only have access to more expensive and limited capability actions compared to agents in other groups. Agents in group 0 of `Probabilistic groups` have a harder time (limited capability and more expensive actions), getting positively classified than those in group 4.

## 4.3 CFE generator architectures

The Hamming distance named CFE generator had varied the nearest neighbors: 5, 10, and 15, for the 20-, 50-, and 100-dimensional datasets, respectively.

We mainly based the architectures for the multi-class classification (MC) named CFE generator on neural network models. Typically, each model has two hidden layers, each consisting of 2000 neurons,  $\ell_2$  regularization, dropout, and batch normalization. The model architecture was optimal for the 20-, 50-, and 100-dimensional datasets. We used the Adam optimizer [16] for all datasets and implemented early stopping with the restoration of the best weights after a patience level of 360. On average, we set the batch size to 2000, and the number of epochs set to 3000.

The named-action (NA) CFE generator, like the MC CFE generator, relied on neural network models. The model has three hidden layers, each with 2000 neurons,  $\ell_2$  regularization, dropout, and batch normalization. We used the Adam optimizer [16] and implemented early stopping with best weights restored after a patience level of 300. We, on average, set the batch size to 6000 and the number of epochs to 5000. We choose the values of  $\alpha$  from the set  $\{0.05, 0.1, 0.07\}$  and  $p_w$  from  $\{0.05, 0.1, 0.07\}$ .

We utilized two kinds of neural network models for the full-action CFE generator: sequential encoder-decoder and NA CFE generator-like networks. To fit the NA model setting for the latter model, we added padding to the CFE vectors. The architecture and hyperparameters for

both networks varied greatly depending on the context. In general, the models were shallow and implemented on CPU machines.

#### 4.4 Evaluation of generated CFEs

For all the CFE generators, we evaluate the generated CFEs with the zero-one loss that compares the generated CFE  $\hat{I}_i$  to the true CFE  $I_i$ .

$$\mathcal{L}_{\text{eval}}(I_i, \hat{I}_i) = \begin{cases} 0 & \text{if } I_i = \hat{I}_i \\ 1 & \text{if } I_i \neq \hat{I}_i \end{cases}$$

### 5 Experimental Results

In this section, we empirically demonstrate the strong performance with which our method generates optimal CFEs for new instances of negatively classified agents.

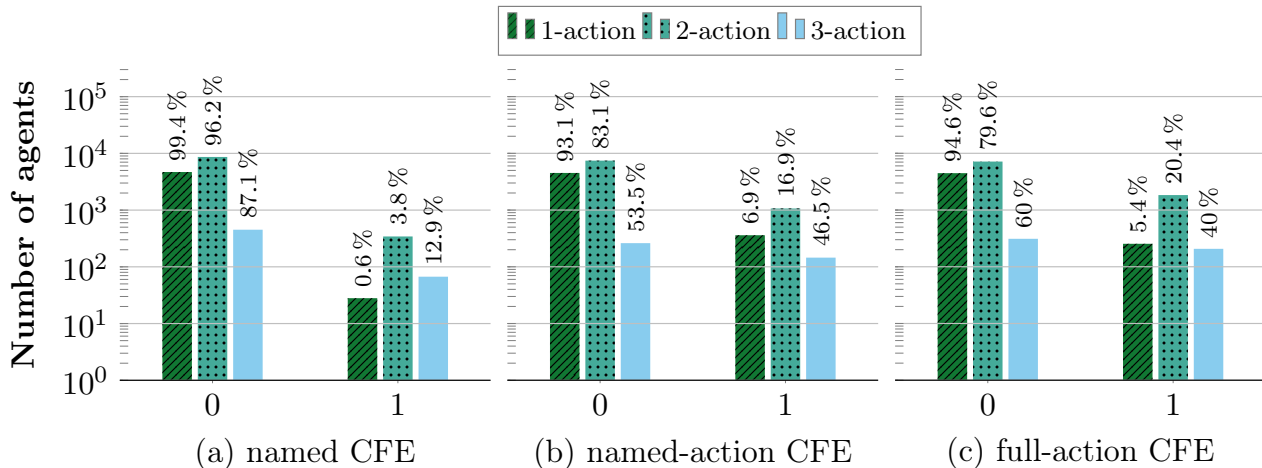


Figure 5: Accuracy of various CFE generators on varied datasets. Accuracy of the named (a), named-action (b), and full-action (c) CFE generators on 20-dimensional, `all` dataset. The named CFE generator is the most accurate, and in all cases, single CFEs were the most accurately generated.

#### 5.1 Accuracy and Confidence of CFE Generators

CFE generators generally performed well on new agents. On the 20-dimensional dataset, the named-action CFE generator generated CFEs for new agents at an accuracy of 96.9% on `all`, 98.4% on `>10`, and 99.3% on `>40` datasets (see Table 2). Similarly, as shown in Figure 5, the named and full-action CFE generators, do equally well.

The CFE generators are confident when they are correct. As seen from Figure 7a, the confidence scores are close to 1 when the named-action CFE generator correctly generates CFEs for the test-set agents. Additionally, the accuracy of CFE generators has very small margin of error, therefore results are precise and reproducible (see Table 2).

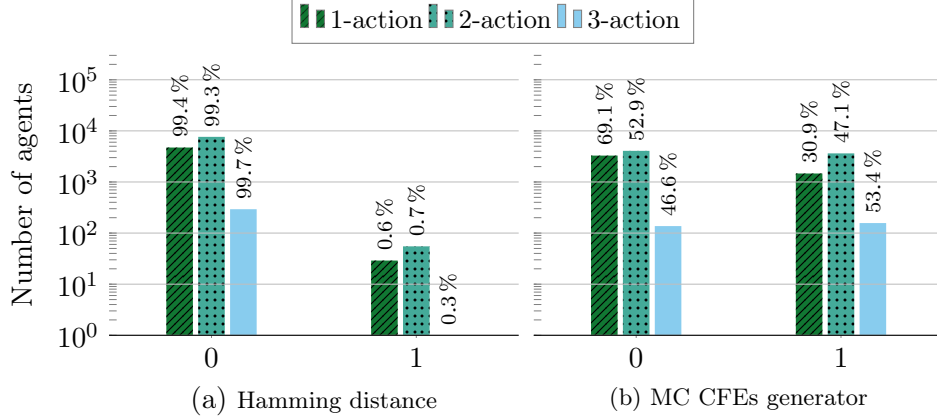


Figure 6: Accuracy of named action CFEs generators on the 20-dimensional, >40 dataset.

CFE Generators on 20-dimensional Datasets			
	all	>10	>40
MC	$0.969 \pm 0.00284$	$0.984 \pm 0.00208$	$0.993 \pm 0.00141$
NA	$0.854 \pm 0.00581$	$0.886 \pm 0.00531$	$0.940 \pm 0.00411$
FA	$0.839 \pm 0.00605$	$0.892 \pm 0.00518$	$0.937 \pm 0.00420$

Table 2: Accuracy of CFE generators: named (MC), named-action (NA) and full-action (FA), on 20 feature: all, >10, and >40 datasets.

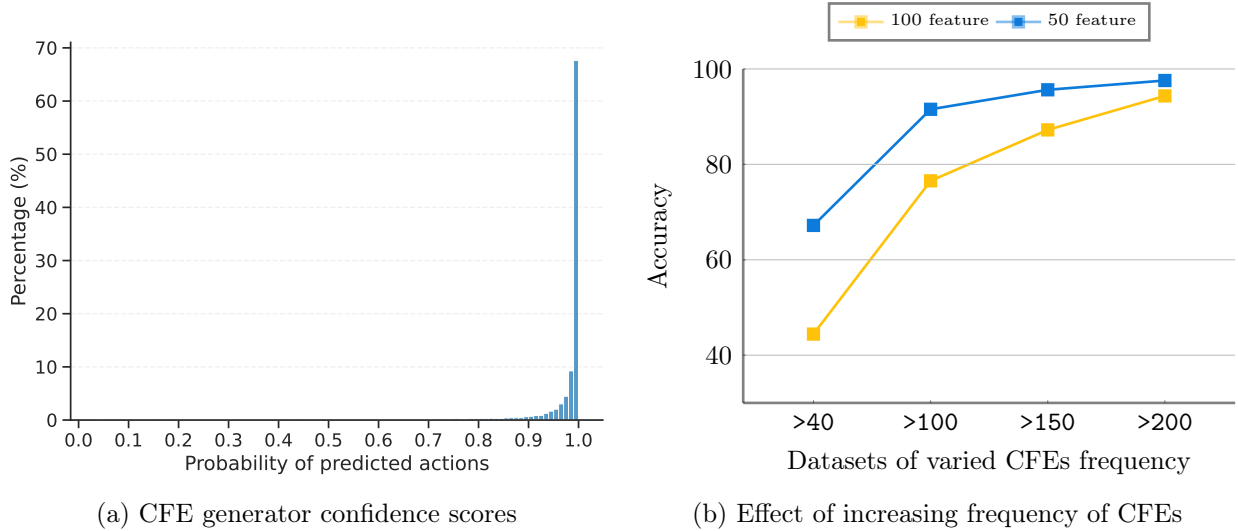


Figure 7: (a) Confidence scores of the named-action CFE generator, and (b) increasing frequency of CFEs in train-set on named-action CFE generator on 100 and 50 feature datasets increases the generator accuracy

## 5.2 Effect of Data Dimension and CFEs Frequency

Due to the uniqueness of CFEs, after data partitioning, some CFEs appeared in one data split and not the other. For example, for the 20, 50, and 100 features: `all` datasets, some CFEs appeared in the test-set but not the train-set. There were 52, 154 and 708 unique CFEs in the test-set that were not seen before in the train-set for 20, 50, and 100 features: `all` datasets, respectively.

The CFE generators become less accurate as the frequency of CFEs in the train set decreases. In Table 2, the accuracy of all CFE generators for the 20 feature dataset was highest when CFEs had a frequency of at least 20 in the train-set (`>40`) and lowest for all data (`all`), in which case, some CFEs appeared in the test-set but not the train-set. For example, the accuracy of the named CFE generator decreased from an accuracy of 99.3% on 20-dimensional `>40` dataset to 96.9% on 20-dimensional `all` dataset.

Since the frequency of CFEs in train-set decreases with increases in data dimension, the generalization of CFE generators decreases with dimension. As seen in Table 3 (top), the named CFE generator had the lowest accuracy on the 100-dimensional dataset and the highest on the 20-dimensional dataset. For example, while the named CFE generator had an accuracy of 74.4% on the 50-dimensional `all` dataset, it had an accuracy of 96.9% on the 20-dimensional `all` dataset.

Additionally, the minimum frequency of CFEs required for a generalizable CFE generator increases with information in the CFE and the number of features in the agent state. While the frequency of 20 is sufficient for all 20 features CFE generators (see Table 2), a frequency of at least 20 CFEs in the train-set is sufficient for named CFE generators for the 50 and 100 feature datasets, and a higher CFE frequency is needed for named-action CFE generators (see Figure 7b).

CFE Generators on 20-, 50-, and 100-dimensional Datasets				
		<code>all</code>	<code>&gt;10</code>	<code>&gt;40</code>
MC	20-dim	0.969 ± 0.00284	0.984 ± 0.00208	0.993 ± 0.00141
NA	50-dim	0.744 ± 0.00608	0.838 ± 0.00534	0.915 ± 0.00458
FA	100-dim	0.354 ± 0.00664	0.630 ± 0.00778	0.856 ± 0.00772

Before & After Augmentation				
		20 features	50 features	100 features
B4A		0.969 ± 0.00284	0.744 ± 0.00608	0.354 ± 0.00664
AG1		0.965 ± 0.00303	0.760 ± 0.00595	0.505 ± 0.00694
AG2		0.982 ± 0.00218	0.845 ± 0.00504	0.790 ± 0.00565

Table 3: On the top, accuracy of the named-action (NA), full-action (FA), and named (MC) CFE generators for 20 (20-dim), 50 (50-dim) and 100 (100-dim) feature: `all`, `>10`, and `>40` datasets. The bottom table shows that data augmentation improves accuracy of CFE generators, as seen for named CFE generators on 20, 50, and 100 features: `all` datasets.

**Data augmentation** To increase the frequency of CFEs, we employ two kinds of data augmentation using Algorithm 1. The principle we follow for Algorithm 1 is to make the agent worse enough, such that the current CFE is still the best CFE for the worse-off agent. Worse-off agents are those such that the features where the CFE is doing more than required are made worse, i.e., for  $j$  such that  $x_{ij}^* > t_j, aug_j < x_{ij}$ . A CFE is doing more than required to feature  $j$  of  $\mathbf{x}_i$ , if by

---

**Algorithm 1:** Data Augmentation

---

**Input:** agent:  $\mathbf{x}_i$ , its CFE:  $I_i$ , and binary classifier:  $\mathbf{t}$   
**Output:** valid derived augments of  $\mathbf{x}_i$ :  $\mathcal{X}_{\text{aug}}^i$   
**Data:** features where CFE is doing more than required:  $ids$ , length of  $ids$ :  $num$

- 1  $\text{aug} \leftarrow 2^{num}$  derived worse-off agents;
- 2 **for**  $\text{aug}$  *in*  $\text{aug}$  **do**
- 3     **if**  $\text{aug}$  *is valid* **then**
- 4          $\mathcal{X}_{\text{aug}}^i \cup \text{aug}$ ;

---

applying it to  $x_{ij}$ ,  $x_{ij}^* > t_j$ . A derived worse-off augment  $\text{aug}$  is valid if  $\mathbf{x}_i$ 's CFE is the best CFE for it,  $\text{aug}_j^* \geq t_j, \forall j \in [n]$ .

With Algorithm 1, we augmented the train-set to ensure that more CFEs are included and representation of CFEs is balanced (**AG1**). As seen from Table 3 (bottom), applying **AG1** improves the accuracy of the CFE generators. For the 100 feature dataset with the named CFE generator, accuracy increases from 35.37% before data augmentation to 50.54% after **AG1**.

After applying **AG1**, we apply another level of data augmentation, **AG2**, aimed at increasing the number of frequency of CFEs in the train-set whose current frequency is less than 20. We reduce the number of CFEs with less than 20 agents from 813 to 638, 2676 to 2005, and 9043 to 7144 for the 20- 50- and 100-dimensional datasets, respectively. **AG2** improves the accuracy of the CFE generators as shown in Table 3 (bottom). For example, for the 100-dimensional dataset with named CFE generator, accuracy increases from 35.37% to 78.99%.

### 5.3 Differential Validity & Grouped Actions Access

In addition to the action-specific and classifier access *informational challenges*, we also examine cases where grouped information about agents is unknown. We examine two additional *information challenges*: agents verified by different thresholds (differential validity) and agents having restricted access to a group of actions (**manual groups** and **probabilistic groups**). Refer to Section 4.1 for information on how we created these groups for the experimental setup.

Without explicit knowledge of the differential validity, when given test-set agents, the CFE generators successfully generate the right CFEs. The full-action CFE generator achieves an accuracy of 99.683% on **First10**, 99.496% on **Last10**, 100% on **First5**, 100% on **Mid5**, and 100% on **Last5**, varied validity groups. Similar to differential validity, we study the case in which the agents in a group only have access to a selected set of actions. Without explicit knowledge of groups and the actions available to the agents in each group, the CFE generators correctly generate the CFEs for test-set agents in both **manual groups** and **probabilistic groups** (see Table 4).

Since agents in the **Manual groups** had more balanced access to actions (Figure 4a), CFE generators had almost similar accuracy (~87%) in the generation of CFEs across all agents in different groups (as shown in Table 4 (left)). On the other hand, agents in the **Probabilistic groups** had access to varied actions (see Table 4 (right)), which led to varied accuracy in generating their CFEs. For instance, as expected, the CFEs for **Probabilistic groups** 4 agents with one-action CFEs were more accurately generated with an accuracy of 93.06% as compared to group 0 and 1 agents, generated at an accuracy of 88.04% and 77.09%, respectively.

Manual Groups		Probabilistic Groups	
Group	Accuracy	Group	Accuracy
group 0	$0.881 \pm 0.01200$	group 0 (0.4)	$0.880 \pm 0.04400$
group 1	$0.871 \pm 0.01260$	group 1 (0.5)	$0.771 \pm 0.02081$
group 2	$0.875 \pm 0.01249$	group 2 (0.6)	$0.802 \pm 0.01571$
group 3	$0.847 \pm 0.01359$	group 3 (0.7)	$0.873 \pm 0.01241$
group 4	$0.886 \pm 0.01212$	group 4 (0.8)	$0.931 \pm 0.00947$

Table 4: Group-wise accuracy of the CFE generator on manual groups & probabilistic groups.

#### 5.4 Single-action CFEs vs. Multiple-action CFEs

In general, all CFE generators, regardless of information entailed in the CFEs, generate single-action CFEs more accurately than multiple-action CFEs. As can be seen in Figure 5a, one-action CFEs were generated at higher accuracy of 99.4%, compared to 96.2% for two-action CFEs and 87.1% for three-action CFEs. Similarly, as seen in Figure 5c, full-action CFE generators generated one-action CFEs at an accuracy of 94.4%, compared to 79.6% for two-action CFEs and 60.0% for three-action CFEs. This might be partly because each CFE regardless of length has the first action, therefore there is more data on the first action than any other action, so in cases of single action, they are more accurately generated.

#### 5.5 Comparison of CFE Generators

Figure 5a to 5c and Table 2 show that named CFE generators are more accurate and need less CFE frequency in the train-set, than named-action and full-action CFE generators. As can be seen in Table 2, the named CFE generators consistently generated CFEs at a higher accuracy than the named-action and full-action CFE generators. For the 20 feature, `a11` dataset, MC had an accuracy of 96.9%, compared to 85.4% with NA and 83.9% with FA. Additionally, as shown in Figure 5a and 5b, the multi-class classification named CFE generator performs better than the baseline Hamming distance CFE generator.

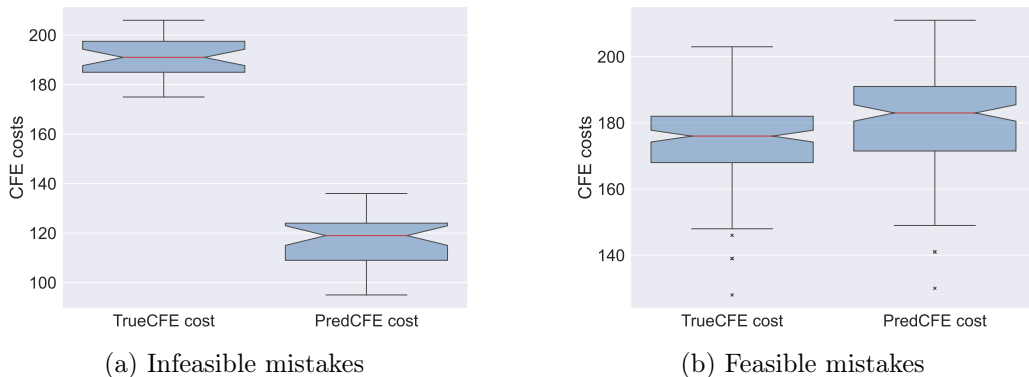


Figure 8: Distribution of costs of generated and true CFEs for infeasible (a) and feasible (b) mistakes the named CFE generator on 20 feature, `a11` dataset.

## 5.6 Feasibility of CFE Generators’ Mistakes

A generated CFE is a mistake when it doesn’t match the true CFE. A CFE mistake is said to be feasible if transforms the agent’s initial state to a counterfactual state but is not the lowest-cost CFE. CFE mistakes are labeled as infeasible if they do not transform the agent to a counterfactual state. Feasible CFE mistakes are by definition, more expensive than the true CFEs, while infeasible CFE mistakes are cheaper than the true CFEs (see Figure 8).

Of the 1.58%, 16.23% and 37.00% mistakes the named CFE generator makes on the 20-, 50-, and 100-dimensional >10 datasets, 100%, 99.23%, and 87.29%, respectively, were feasible. Similarly, the majority of the mistakes of the full-action CFE generators were feasible, e.g., for the 20-dimensional >10 dataset, of the 10.8% of the mistakes, 63.10% were feasible.

Additionally, the percentage of feasible mistakes decreases with the frequency of CFEs in the train-set, e.g., the percentage of feasible mistakes decreases from 87.29% on the 100-dimensional, >10 dataset to 57.83% on 100-dimensional, >all dataset.

## 6 Conclusion

In this work, we provide a solution to the informational and computational challenges of CFE generation in settings where selecting a set of actions corresponds to solving a weighted set cover problem. We propose three CFE generators: named-action, full-action, and named CFE generators. The CFE generators are computationally cheaper than the single agent optimization-based ILP CFE generator, and each addresses a unique informational challenge specific to the CFE, its action, and its costs. We empirically show that the CFE generators generate CFEs for new agents at high accuracy, which increases with the frequency of CFEs in the training set. Lastly, we explore two other potential informational challenges: agents with varied access to actions and different agents classified by different classification thresholds. Without explicit information on which agents were classified by which threshold or which agents had access to which actions, given the mappings of agents to CFEs, the CFE generator accurately generates CFEs for new agents.

**Limitations** We focus on learning to generate CFEs in settings where actions provide agents with various “capabilities” and a CFE is a solution to a weighted set cover problem. This limits CFEs to being characterized as a set of actions to perform. More generally, one could consider settings where the order of actions matters, such as where a CFE corresponds to an optimal policy for an agent in a deterministic Markov decision process (MDP). Even more generally, one could consider actions whose effects are stochastic, and a CFE then corresponds to an optimal policy for the agent in a general MDP.

In our work, we investigate two grouped agent settings. These settings involve agents having different access to different actions and agents classified by different thresholds. Promising future works could extend these settings to more complex scenarios and explore potential issues such as survivorship bias and its effect on generating CFEs. Additionally, future research could investigate the ethical implications of generating CFEs using data-driven algorithms.

Lastly, it is important to note that our approach to generating CFEs is subject to certain assumptions. We assume that variables are discrete, that generalized CFE generators have access to labeled historical agent-CFE data and that agents have only one optimal CFE. These are all potential directions for future work.

**Ethical statement** Our proposed approaches to CFE generation are closely related to data-driven algorithm design. As a result, ethical concerns related to data-driven algorithms, for example,

potentially propagating and exacerbating biases in historical agent-CFE data, might apply to our proposed CFE generators. Additionally, our CFE generators are not 100% accurate, and the CFE generator mistakes could lead to the wastage of the agent's effort and other undesirable downstream effects like wrong allocation of resources. The experimentation code is at this [link](#).

## Acknowledgements

We thank Alexandra DeLucia for the helpful feedback on the manuscript. This work was supported in part by the National Science Foundation under grants CCF-2212968 and ECCS-2216899, and by the Simons Foundation under the Simons Collaboration on the Theory of Algorithmic Fairness.



## References

- [1] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018. 19
- [2] Maria-Florina Balcan. Data-driven algorithm design. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 626–645. Cambridge University Press, 2020. doi: 10.1017/9781108637435.036. URL <https://doi.org/10.1017/9781108637435.036>. 3
- [3] Maria-Florina Balcan, Travis Dick, and Colin White. Data-driven clustering via parameterized Lloyd’s families. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, pages 10664–10674, 2018. URL <https://proceedings.neurips.cc/paper/2018/hash/128ac9c427302b7a64314fc4593430b2-Abstract.html>. 3
- [4] Leopoldo E. Bertossi. An asp-based approach to counterfactual explanations for classification. *CoRR*, abs/2004.13237, 2020. URL <https://arxiv.org/abs/2004.13237>. 1
- [5] Zhicheng Cui, Wenlin Chen, Yujie He, and Yixin Chen. Optimal action extraction for random forests and boosted trees. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 179–188, 2015. ISBN 9781450336642. doi: 10.1145/2783258.2783281. URL <https://doi.org/10.1145/2783258.2783281>. 1
- [6] Susanne Dandl, Christoph Molnar, Martin Binder, and Bernd Bischl. Multi-objective counterfactual explanations. In *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 448–469, 2020. ISBN 978-3-030-58111-4. doi: 10.1007/978-3-030-58112-1\textunderscore31. URL [https://doi.org/10.1007/978-3-030-58112-1\\_31](https://doi.org/10.1007/978-3-030-58112-1_31). 1
- [7] Giovanni De Toni, Bruno Lepri, and Andrea Passerini. Synthesizing explainable counterfactual policies for algorithmic recourse with program synthesis. *Machine Learning*, 112(4):1389–1409, 2023. ISSN 0885-6125. doi: 10.1007/s10994-022-06293-7. URL <https://doi.org/10.1007/s10994-022-06293-7>. 3
- [8] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016. 19
- [9] European Parliament and Council of the EU. Regulation (EU) 2016/679 of the European Parliament and of the Council, 2016. URL <https://data.europa.eu/eli/reg/2016/679/oj>. 1
- [10] Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. In *Proceedings of the ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 123–134, 2016. ISBN 9781450340571. doi: 10.1145/2840728.2840766. URL <https://doi.org/10.1145/2840728.2840766>. 3
- [11] Vivek Gupta, Pegah Nokhiz, Chitradeep Dutta Roy, and Suresh Venkatasubramanian. Equalizing recourse across groups. *arXiv preprint arXiv:1909.03166*, 2019. URL <http://arxiv.org/abs/1909.03166>. 1
- [12] Shalmali Joshi, Oluwasanmi Koyejo, Warut Vijitbenjaronk, Been Kim, and Joydeep Ghosh. Towards realistic individual recourse and actionable explanations in black-box decision making systems. *arXiv preprint arXiv:1907.09615*, 2019. URL <http://arxiv.org/abs/1907.09615>. 1
- [13] Amir-Hossein Karimi, Bernhard Schölkopf, and Isabel Valera. Algorithmic recourse: From counterfactual explanations to interventions. In *Proceedings of the ACM Conference on Fairness, Accountability, and Transparency (ACM FAccT)*, pages 353–362, 2021. ISBN 9781450383097. doi: 10.1145/3442188.3445899. URL <https://doi.org/10.1145/3442188.3445899>. 1
- [14] Amir-Hossein Karimi, Gilles Barthe, Bernhard Schölkopf, and Isabel Valera. A survey of algorithmic recourse: Contrastive explanations and consequential recommendations. *ACM Computing Surveys*, 55(5), 2022. ISSN 0360-0300. doi: 10.1145/3527848. URL <https://doi.org/10.1145/3527848>. 1
- [15] Richard M. Karp. *Reducibility Among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. ISBN 978-1-4684-2001-2. doi: 10.1007/978-1-4684-2001-2\textunderscore9. URL [https://doi.org/10.1007/978-1-4684-2001-2\\_9](https://doi.org/10.1007/978-1-4684-2001-2_9). 2
- [16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 9

- [17] Xinghan Liu and Emiliano Lorini. A unified logical framework for explanations in classifier systems. *Journal of Logic and Computation*, 33(2):485–515, 2023. doi: 10.1093/logcom/exac102. 1
- [18] Joao Marques-Silva. Logic-based explainability in machine learning, 2023. 1
- [19] Ramaravind K. Mothilal, Amit Sharma, and Chenhao Tan. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the ACM Conference on Fairness, Accountability, and Transparency (ACM FAccT)*, pages 607–617, 2020. ISBN 9781450369367. doi: 10.1145/3351095.3372850. URL <https://doi.org/10.1145/3351095.3372850>. 1
- [20] Philip Naumann and Eirini Ntoutsi. Consequence-aware sequential counterfactual generation. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 682–698, 2021. ISBN 978-3-030-86519-1. doi: 10.1007/978-3-030-86520-7\textunderscore42. URL [https://doi.org/10.1007/978-3-030-86520-7\\_42](https://doi.org/10.1007/978-3-030-86520-7_42). 3
- [21] Goutham Ramakrishnan, Yun Chan Lee, and Aws Albarghouthi. Synthesizing action sequences for modifying model decisions. *arXiv preprint arXiv:1910.00057*, 2019. URL <http://arxiv.org/abs/1910.00057>. 1
- [22] Yonadav Shavit and William S. Moses. Extracting incentives from black-box decisions. *arXiv preprint arXiv:1910.05664*, 2019. URL <http://arxiv.org/abs/1910.05664>. 3
- [23] Berk Ustun, Alexander Spangher, and Yang Liu. Actionable recourse in linear classification. In *Proceedings of the ACM Conference on Fairness, Accountability, and Transparency (ACM FAccT)*, pages 10–19, 2019. ISBN 9781450361255. doi: 10.1145/3287560.3287566. URL <https://doi.org/10.1145/3287560.3287566>. 1
- [24] V.V. Vazirani. *Approximation Algorithms*. Springer Berlin Heidelberg, 2013. ISBN 9783662045657. URL <https://books.google.com/books?id=bJmqCAAAQBAJ>. 4
- [25] Sahil Verma, Keegan Hines, and John P. Dickerson. Amortized generation of sequential algorithmic recourses for black-box models. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 8512–8519, 2022. doi: 10.1609/aaai.v36i8.20828. URL <https://ojs.aaai.org/index.php/AAAI/article/view/20828>. 3
- [26] Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *arXiv preprint arXiv:1711.00399*, 2017. URL <http://arxiv.org/abs/1711.00399>. 1

## A Supplemental Data

We created agent-CFE datasets using solutions of the weighted set cover problem, implemented with CVXPY Python package [8, 1]. Below, we provide information on missing dataset statistics on varied binary classifier sparsity and action access.

dataset	dataset size	one-action CFEs	two-action CFEs	three-action CFEs
20-dimensional dataset	71125	23687	44858	2576
50-dimensional dataset	98966	1262	96770	934
100-dimensional dataset	99728	0	45515	54213
Manual groups	73484	13480	56653	3351
Probabilistic groups	70226	44661	20258	5307
First10	74524	61794	12046	39
First5	74594	60656	6005	0
Last10	74401	53822	19952	1
Last5	74565	66068	644	0
Mid5	74594	63530	3010	0

Table 5: Statistics of the datasets used in the experiments.