Consequence-aware Sequential Counterfactual Generation

Philip Naumann[1,2] (✉) and Eirini Ntoutsi[1,2]

[1] Freie Universität Berlin, Germany
[2] L3S Research Center, Leibniz Universität Hannover, Germany
{philip.naumann, eirini.ntoutsi}@fu-berlin.de

Abstract. Counterfactuals have become a popular technique nowadays for interacting with black-box machine learning models and understanding how to change a particular instance to obtain a desired outcome from the model. However, most existing approaches assume instant materialization of these changes, ignoring that they may require effort and a specific order of application. Recently, methods have been proposed that also consider the order in which actions are applied, leading to the so-called sequential counterfactual generation problem.
In this work, we propose a model-agnostic method for sequential counterfactual generation. We formulate the task as a multi-objective optimization problem and present a genetic algorithm approach to find optimal sequences of actions leading to the counterfactuals. Our cost model considers not only the direct effect of an action, but also its consequences. Experimental results show that compared to state-of-the-art, our approach generates less costly solutions, is more efficient and provides the user with a diverse set of solutions to choose from.

Keywords: Sequential counterfactuals · Multi-objective optimization · Genetic algorithms · Model-agnostic.

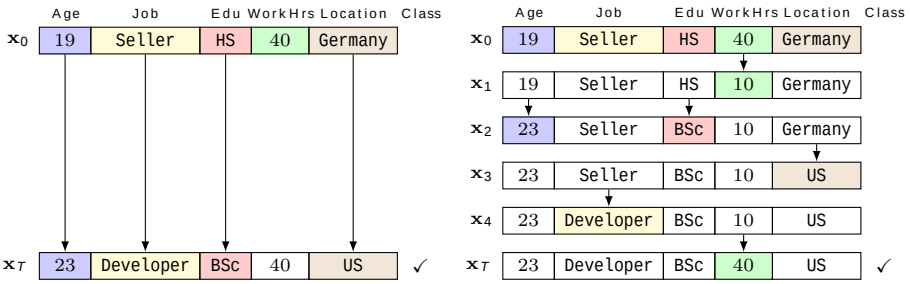
1 Introduction

Due to the increasing use of machine learning algorithms in sensitive areas such as law, finance or labor, there is an equally increased need for transparency and so-called *recourse* options [12, 24]. It is no longer sufficient to simply deliver a decision, but moreover to be able to explain it and, ideally, offer assistance if one feels unfairly treated by the algorithm. Hiding the decision-making algorithm behind a (virtual) wall like an API makes these issues especially intransparent and problematic in case of *black-box* models, as they are not able to communicate with the end user beyond the provided decision (e.g. `reject` or `accept`).

For this reason, algorithms emerged that aim to explain a (black-box) decision or even provide essential recourse information in order to change an undesired outcome in one's favor. The latter of these methods is of particular interest, since it has the capability to improve a bad decision for someone into a good one by giving explicit directions on what to change with respect to the provided information.

These methods are commonly referred to as *counterfactual explanations* [24]. The goal here is to change (tweak) characteristics (features) of a provided input so that the black-box decision turns out in favor afterwards (e.g. increase your level of education to get a higher salary). The result of applying these changes on the input is commonly referred to as a *counterfactual* [24].

Usually, those changes are atomic operations, meaning each feature is tweaked independently [4,15,17,24]. Thus, feature interrelationships, e.g. of causal nature, are not considered. Recent approaches propose to define changes in (multiple) features through so-called *actions*, which can be thought of as instructions on how to apply the modifications and their consequences (e.g. increasing the level of education has an impact on age because it takes time to obtain a degree). Actions further help to describe what features are actionable, mutable or immutable [12]. However, these approaches, like the traditional counterfactual methods, still assume that all these changes happen instantly and do not consider implications and consequences of the *order* of their application.



(a) Traditional counterfactual (b) Sequential counterfactual

Fig. 1. The difference between traditional counterfactual generation (a) and the sequential approach (b). Although the generated counterfactual $\mathbf{x}_T$ is the same, the process and implied knowledge/information is different.

For this reason, recent works regard the application of feature altering changes as a *sequential* process [19,20]. The problem is then shifted from simply computing the counterfactual, to finding an ordered *sequence* of actions that accomplishes it. In Fig. 1 we visualized this difference for a person $\mathbf{x}_0$ wishing to attain a higher salary (Class:) ✓) for which getting a higher education level, switching the job and changing the location is necessary. In this case, e.g., it was assumed that decreasing one's working hours is *beneficial* in order to obtain a higher degree, which is a relationship that traditional approaches do not model. Each state $\mathbf{x}_t$ in Fig. 1b describes the result of applying an action on the previous state $\mathbf{x}_{t-1}$ and $\mathbf{x}_T$ denotes the counterfactual. As we can see, the actions have different effects in the feature space and may alter more than one feature at a time. Moreover, the order makes a difference as increasing the level of education before changing the

job is usually more beneficial, as well as decreasing the working hours in order to attain the degree (representing a focus on the education). Later on, this change is reset to its original value through another action since an increased value is now more *plausible* again with respect to the job change. The whole process of Fig. 1b can be seen as a *sequential counterfactual* as each state is an important part of it, whereas in the traditional setting in Fig. 1a only the final result is important. Thus, a sequential counterfactual allows us to look beyond just flipping the class label and provides further information about the underlying process. It is not the absolute main goal anymore to only switch the class label, but to take consequential effects into account in order to improve the overall benefit of the actions. For the end user, this also means getting concrete information on the actions and their order. Our work is in the direction of sequential counterfactual generation and inspired by [19], which we will further elaborate in Sect. 2.

The rest of this paper is organized as follows: we will first give an overview on related work in Sect. 2. Then, we introduce the *consequence-aware counterfactual generation problem* in Sect. 3, for which our proposed method follows in Sect. 4. Lastly, we evaluate it in Sect. 5 to a state-of-the-art method and give final conclusions in Sect. 6. We want to note that the metaphorical examples used throughout do not always correspond to reality and are merely illustrative.

2 Related Work

Counterfactual explanations were first introduced in [24] as a mean towards algorithmic transparency. Motivated by the "closest possible world" [24] in which a favorable outcome is true (e.g. `accept`), most methods optimize on the *distance* between the original input $\mathbf{x}_0$ and the counterfactual $\mathbf{x}_T$ to keep the changes minimal. Since this notion alone was found to be insufficient, other popular objectives include the *plausibility* (often measured as the counterfactual being within the class distribution [10] or being close to instances from the dataset [4, 18, 23]) and *sparsity* [4, 17, 23] (measuring how many features had to change) of solutions. Desirable criteria regarding the algorithms are, e.g., being model-agnostic [4, 11, 14, 15] (e.g. by using genetic algorithms) or providing a diverse set of solutions [4, 5, 16, 17] (e.g. by using multi-objective optimization).

More recently, works [13, 19, 22] began to replace the distance function with a *cost* in order to express aspects such as the *effort* of a change. In alignment to this, the term *recourse* [5, 10, 12, 13, 18, 22] has attracted attention to describe the counterfactual generation. It can be defined as "the ability of a person to change the decision of a model by altering actionable input variables" [22] and thus emphasizes on the *actionability* of the features to provide comprehensible recommendations that can be acted upon [12]. In addition, more attention has been paid on the causal nature of feature interrelationships, e.g. by including causal models in the generation process to assess mutual effects [5, 13, 16].

Lastly, motivated by the fact that in reality most changes do not happen instantly, but are rather part of a *process*, there have been works that extend the formulation of actions and their consequences by incorporating them in a

sequential setting [19, 20]. In contrast to simply finding the counterfactual that switches the class label, the focus is on finding a subset of actions that, applied in a specific order, accomplishes the counterfactual while accounting for potential consequences of prior actions on subsequent ones (cf. Fig. 1).

In this work, we also focus on sequential counterfactual generation. The advantages of our method in comparison to [19] can be summarized as follows: our method is model-agnostic and not bound to differentiable (cost) functions. It finds diverse sequences instead of a single solution, thus giving more freedom of choice to the end user. Moreover, it is efficient in pruning and exploring the search space due to using already found knowledge (exploitation) and the ability to optimize all sub-problems (cf. Sect. 3.3) at once, while [19] breaks these down into separate problems. This efficiency allows us to find sequences of any length, whereas [19] requires multiple runs and more time for it (cf. Sect. 5). Another difference is our action-cost model (cf. Sect. 3.2). We regard consequences not only in the feature space (e.g. age increases as a consequence of obtaining a higher degree), but also explicitly model their effects in the objective or cost space (e.g. changing the job becomes *easier* with a higher degree). This way we extend the cost formulation of [19], which only proposes to model (feature) relationships through (boolean) pre- and post-requirement constraints (e.g. you *must* at least be 18 to get a driver's license). These constraints are also possible in our model. Finally, we note that the work of [20] also discusses consequential effects. However, since no cost function is optimized, but instead the target class likelihood of the counterfactual, we do not compare with [20] in this work.

3 Problem Statement

We assume a static *black-box classifier* $f : X \; ! \; Y$ where $X = X_1 \qquad X_d$ is the feature space and Y is the class space. The notation $\dddot{X}_h$ is used to refer to the feature itself (e.g. $\dddot{X}_{\texttt{Edu}}$ denotes `Education`, whereas $X_{\texttt{Edu}} = f\texttt{:::;HS;BSc;:::}g$ is the domain). For simplicity and without loss of generality, we assume f is a binary classifier with $Y = f\texttt{reject;accept}g$. Let $\mathbf{x}_0 \; 2 \; X$ be an instance of the problem, e.g. a person seeking to receive an annual salary of more than \$50k, and the current decision of f based on $\mathbf{x}_0$ is $f(\mathbf{x}_0) = \texttt{reject}$. The goal is to find a counterfactual example $\mathbf{x}_T$ for $\mathbf{x}_0$ such that $f(\mathbf{x}_T) = \texttt{accept}$. In other words, we want to change the original instance so that it will receive a positive decision from the black-box. The sort of changes we refer to are in the feature description of the instance, like increasing `Age` by 5 years or decreasing `Work Hours` to 20 per week. Our problem formulation builds upon [19]. We introduce actions in Sect. 3.1 along with their associated cost to implement the suggested changes in Sect. 3.2. Finally, we formulate the generation of sequential counterfactuals as a multi-objective optimization problem in Sect. 3.3.

3.1 Actions, Sequences of Actions and States

Let $A = fa_1;:::;a_ng$ be a problem-specific, manually-defined set of *actions*. Each action is a function $a_i : X \quad V \; ! \; X$ that modifies a subset of features

$I_{a_i} = \{\ddot{X}_h, \ddot{X}_k, \ldots\} \subseteq \ddot{X}$ in a given input instance $\mathbf{x}_{t-1} \in \mathcal{X}$ in order to realize a new instance (which we refer to as a *state*) $a_i(\mathbf{x}_{t-1}, v_i) = \mathbf{x}_t \in \mathcal{X}$. An action *directly* affects one feature $\ddot{X}_h \in I_{a_i}$ (e.g. `Education`) based on a tweaking value $v_i \in \mathcal{V}_h \subseteq \mathcal{X}_h$ and may have *indirect* effects on other features $\ddot{X}_{k \neq h} \in I_{a_i}$ as a consequence (e.g. `Age`). Here, $\mathcal{V} \subseteq \mathcal{X}$ describes the *feasible value space* that restricts the tweaking values based on the given $\mathbf{x}$ (e.g. `Age` may only increase).

For example, $\mathbf{x}_2$ in Fig. 1b is the result of applying $a_{\texttt{Edu}}(\mathbf{x}_1, \texttt{BSc}) = \mathbf{x}_2$ which changes the `Education` (`HS` $\rightarrow$ `BSc`) and affects the `Age` ($19 \rightarrow 23$) as a consequence. In this case, $\mathcal{V}_{\texttt{Edu}}$ and $\mathcal{V}_{\texttt{Age}}$ restrict the tweaking values so that they can only increase. It is possible to use a causal model as in [13] for evaluating how the indirectly affected features have to be changed. An action-value pair (a_i, v_i) thus represents a specification how I_{a_i} of $\mathbf{x}$ is affected with respect to the tweaking value v_i of feature $\ddot{X}_h$.

Additionally, each action a_i may be subject to boolean constraints $C_i : \mathcal{X} \times \mathcal{V} \rightarrow \mathbb{B}$ such as pre- and post-requirements as proposed in [19] (cf. Sect. 2), which can also be used to validate the feasibility of indirectly affected features. Each action-value pair is considered *valid*, if it satisfies the associated constraints and $\mathcal{V}$. An ordered sequence S of valid action-value pairs (a_i^t, v_i) leading to the counterfactual $\mathbf{x}_T$ is called a *sequential counterfactual*. Here, $t \in \{1, \ldots, T\}$ is the order of applying the actions and $T \in \{1, \ldots, |A|\}$ is the number of used actions in S, i.e. the sequence length.

3.2 Consequence-aware Cost Model

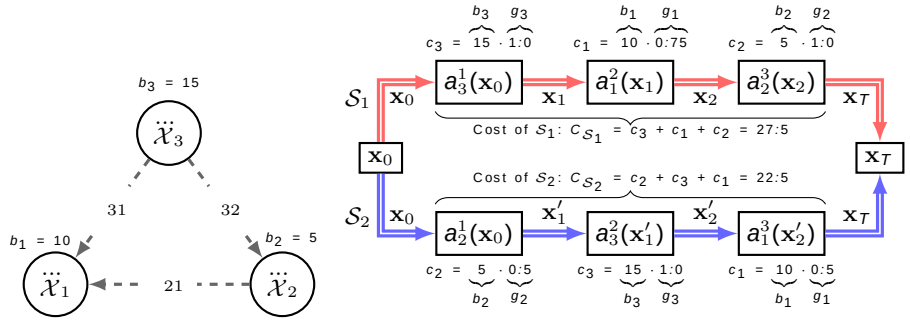
Our goal is to assess the direct effort (which can, e.g., be abstract, as based on personal preferences, or concrete, like money or time etc.) of an action while considering possible consequences. We define the cost of an action a_i as a function of two components: $c_i(\,) = b_i(\,) \cdot g_i(\,)$. Here, b_i represents the *direct effort*, whereas g_i acts as a discount of it in order to express (beneficial) *consequences* of prior actions. Please note that each action has its *own* cost function. Summing up all action costs of a sequence S yields the *sequence cost*: $C_S = \sum_{a_i \in S} c_i(\,)$. In the following we will explain the components in more detail.

Action Effort b_i: First, we introduce $b_i : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_+$, which is assumed to be an action-specific measure of the direct effort caused by an action a_i. Therefore, this is specified as a function between two consecutive states $\mathbf{x}_{t-1}, \mathbf{x}_t \in \mathcal{X}$, representing the direct effect of applying that action on $\mathbf{x}_{t-1}$. This function can be thought of as a typical cost function as in e.g. [19]. As an example, b_i could be specified linear and time based, whereby an effort caused by an action `addEdu` would be represented by the years required (e.g. four to progress from `HS` to `BSc`). Alternatively, monetary costs could be used (i.e. tuition costs), or a combination of both. Besides, there are no particular conditions on this function, so it can be defined arbitrarily (e.g. return a constant value).

Consequential Discount g_i: To assess a possible (beneficial) consequential effect of previous actions on applying the current one a_i^t, we introduce a so-called *consequential discount* $g_i : \mathcal{X} \rightarrow [0, 1]$ that affects the action effort b_i based on the current state $\mathbf{x}_{t-1}$ (i.e. before applying a_i^t). Such effects can be, e.g., "the higher

the `Education`, the easier it is to increase `Capital`" or "increasing `Education` in `Germany` is cheaper than in the `US` (due to lower tuition fees)". This discount therefore describes a value in $[0;1]$, where 0 would mean that the current state is so beneficial that the effort of the action to be applied is completely cancelled out, and 1 that there is no advantageous effect. We derive the aforementioned consequential effect on an action from consequential relationships between feature pairs. This is provided as a graph $G = (X; E)$ where the nodes X $\ddot{X}$ are a subset of the features and edges $e_{kh} \in E$ between each two nodes $\ddot{X}_k; \ddot{X}_h \in X$ describe a function $_{kh} : X \to [0;1]$ that models a consequential effect between one feature $\ddot{X}_k$ to another $\ddot{X}_h$. For the given features $\ddot{X}_1$:= `Job`, $\ddot{X}_2$:= `Education` and $\ddot{X}_3$:= `Location` we have exemplified G in Fig. 2a by the following relations:

1. The `Education` cost depends on the `Location` ($\ddot{X}_3$ $^{\tau_{32}}\!\!\to$ $\ddot{X}_2$). E.g., it is *cheaper* to get a degree in `Germany` than the `US` because of lower tuition fees.
2. The easiness of getting a `Job` depends on the `Location` ($\ddot{X}_3$ $^{\tau_{31}}\!\!\to$ $\ddot{X}_1$). E.g., it is *easier* to get a `Developer` job in the `US` than in other locations.
3. The higher the `Education`, the *easier* it is to change the `Job` ($\ddot{X}_2$ $^{\tau_{21}}\!\!\to$ $\ddot{X}_1$).



(a) Feature relationship graph $\mathcal{G}$ (b) Different sequences $\mathcal{S}_1$ (red) and $\mathcal{S}_2$ (blue)

Fig. 2. For simplicity, the $(\cdot)$ functions in (a) are based on binary conditions: $_{32}$ = 1:0 if $\mathcal{X}_3$:= `US`; else 0:5. $_{31}$ = 0:5 if $\mathcal{X}_3$:= `US`; else 1:0. $_{21}$ = 0:5 if $\mathcal{X}_2 \geq$ `BSc`; else 1:0. As a reference, the action efforts b_i are provided above each feature in (a).

Based on this modeling in G, we can then derive the consequential discount g_i of an action a_i (Eq. 2) by averaging the consequential effect $\hat{g}_h$ of *each affected feature* $\ddot{X}_h \in I_{a_i}$ (Eq. 1). It is assumed that g_i evaluates to 1:0 if no feature in I_{a_i} is influenced by another one in G (e.g. $\ddot{X}_3$ in Fig. 2a).

$$\hat{g}_h(\mathbf{x}_{t\ \ 1}) = \text{avg}(f_{\ kh}(\mathbf{x}_{t\ \ 1}) \ \forall \ \ddot{X}_k \in X \mid e_{kh} \in E) \tag{1}$$

$$g_i(\mathbf{x}_{t\ \ 1}) = \text{avg}(\hat{g}_h(\mathbf{x}_{t\ \ 1}) \ \forall \ \ddot{X}_h \in I_{a_i} \mid \ddot{X}_h \in X) \tag{2}$$

In order to understand the benefit of the consequential discount on the sequence order, we exemplify in Fig. 2b the situation from Fig. 1b (for simplicity

the working hours altering actions are omitted). The available actions are thus: "change `Job` to `Developer`" (a_1), "get a `BSc` degree" (a_2) and "change `Location` to `US`" (a_3) (notice all V_i are fixed to a single value). Each a_i alters their respective feature counterpart $\ddot{X}_i$ (e.g. $I_{a_1} = \{\ddot{X}_1\}$). We can see the cost computations in Fig. 2b for two differently ordered sequences $S_1 = \langle a_3^1; a_1^2; a_2^3 \rangle$ and $S_2 = \langle a_2^1; a_3^2; a_1^3 \rangle$ that achieve the same final outcome $\mathbf{x}_T$ (i.e. only the application order is different). To compute the consequential discount for action a_1^2 in S_1, e.g., we consider the relations $\ddot{X}_3 \overset{\tau_3}{\prec} \ddot{X}_1$ and $\ddot{X}_2 \overset{\tau_2}{\prec} \ddot{X}_1$ with respect to $\mathbf{x}_1$ to derive the *feature* discount (Eq. 1): $\hat{g}_1(\mathbf{x}_1) = \frac{0.5+1.0}{2} = 0.75$. Since no other feature is affected by a_1 according to I_{a_1}, the *action* discount (Eq. 2) evaluates to $g_1(\mathbf{x}_1) = \hat{g}_1(\mathbf{x}_1)$. After computing all action costs c_i, we can derive the sequence costs $C_{S_1} = 27.5$ and $C_{S_2} = 22.5$, which shows that S_2 would be preferred here as it benefits more from the consequential discount effects of G. Note, that if we leave out the consequential discounts completely, i.e. $c_i = b_i$, then there would be no notion of order here as each sequence would receive the same costs (assuming the same tweaking values). Furthermore, our consequence-aware formulation means, that additional actions are only used if their induced effort is lower than the consequential benefit they provide (as this would otherwise make C_S worse than if the action was not used).

3.3 Consequence-aware Sequential Counterfactual Generation

Based on the previous definitions, we now introduce the *consequence-aware sequential counterfactual generation* problem. Find the counterfactual $\mathbf{x}_T \in \mathcal{X}$ of an initial instance $\mathbf{x}_0 \in \mathcal{X}$ by taking valid action-value pairs $(a_i^t; v_i)$ of a sequence S according to the constraints C_i and sequence cost C_S such that $f(\mathbf{x}_T) = \texttt{accept}$. In order to solve this, we identify three sub-problems:

Problem 1 (Prob. 1): Find an **optimal subset of actions** $A \subseteq \mathcal{A}$.
Problem 2 (Prob. 2): Find **optimal values** $V \in \mathcal{V}$ for A.
Problem 3 (Prob. 3): Find the **optimal order** of S to apply the actions.

For an arbitrary set of actions A and feasible value space V many sequences can be generated, therefore it is important to assess their quality. For this purpose, we will use the sequence cost $o_1 := C_S$ as a *subjective* measure, as well as the Gower's distance [9] $o_2 := \texttt{dist}(\mathbf{x}_0; \mathbf{x}_T)$ to act as an *objective* assessment how much $\mathbf{x}_T$ differs from $\mathbf{x}_0$. The Gower's distance is able to combine numerical and categorical features and is thus an appropriate measure here [4]. The reason for using both is, that o_1 measures the *effort* of the *whole process*, whereas o_2 only considers the *difference* to the final counterfactual and is agnostic of the process.

In order to propose diverse solutions, we will formulate the problem as a multi-objective one and add, next to o_1 and o_2, the individual tweaking frequencies of each of the $1 \leq h \leq d$ features after unrolling the complete sequence, i.e. $o_{2+h} = \#(\ddot{X}_h \in I_{a_i} \forall a_i \in S)$. In other words, o_{2+h} measures how often a feature $\ddot{X}_h$ was affected by all actions of S combined. E.g., the frequencies for $o_3, \ldots, o_7$ would be $\{1, 1, 1, 2, 1\}$ in case of Fig. 1b. In a way this can be thought

of as the *sparsity* objective mentioned in Sect. 2, but aggregated individually per feature instead of a sum. The idea behind the diversity objectives is to keep the number of feature changes minimal and additionally force the optimization to seek alternative changes instead. This means, solutions mainly compete with those that change the same features with respect to o_1 and o_2, resulting in a diverse set of optimal options. Combining all the above yields the following multi-objective minimization problem:

$$\begin{aligned} &\min_S\big(\underbrace{o_1}_{\text{Sequence cost}}\ ;\ \underbrace{o_2}_{\text{Gower's distance}}\ ;\ \underbrace{o_{2+1};\dots;o_{2+h};\dots;o_{2+d}}_{\text{Feature tweaking frequencies}}\big) \\ &\text{s.t. } f(\mathbf{x}_T) = \texttt{accept} \text{ and } \bigwedge_{(a_i,v_i)\in S} C_i \end{aligned} \tag{3}$$

4 Consequence-aware Sequential Counterfactuals (CSCF)

In order to address the combinatorial (Prob. 1, Prob. 3) and continuous (Prob. 2) sub-problems with respect to Eq. 3, we used a *Biased Random-Key Genetic Algorithm* (BRKGA) [8] and adapted it for multi-objective optimization by using non-dominated sorting (NDS) [21]. NDS is preferred over a scalarization approach to avoid manual prioritization of the objectives and to address them equally. Moreover, by using BRKGA we avoid the manual definition of problem-specific operators, which is not trivial here. This choice allows to solve all sub-problems at once, is model-agnostic, derivative-free and provides multiple solutions.

BRKGA: The main idea behind BRKGA is that it optimizes on the genotype of the solutions and evaluates on their phenotype, making the optimization itself problem-independent [8]. A *genotype* is the internal representation of a solution, whereas the *phenotype* is the actual solution we wish to generate. The phenotype must always be deterministically derivable from the genotype through a *decoder* function [8]. Because of this, each solution genotype in BRKGA is encoded as a vector of real (random) values in the range of $[0; 1]$ (the so-called *random-keys*) [8].

In each generation (iteration), the decoded solution phenotypes are evaluated based on their fitness (given by the vector of evaluating each objective of Eq. 3 individually) and the population is divided into two subsets by applying NDS: the so-called *elites*, which are the feasible (valid), non-dominated (i.e. best) solutions in the Pareto-front [7], and the remaining ones, called *non-elites*. Then, genetic mating is performed by selecting two parents, one from the elite sub-population and one from the non-elites. A new solution is created by applying biased crossover [8] on the two parents, which favors selecting the value of the elite solution with a certain biased probability. This step is repeated until sufficient new solutions have been created. Additionally, a number of completely random solutions are generated in order to preserve the exploration of the search space and the diversity in the population. Finally, the different sub-populations (*elites*, *crossovered* and *random* solutions) are merged and evaluated and form the new population for the next generation. This loop continues until some termination criterion is reached. The Pareto-front of the last generation then represents our

final solution set. Note that the Pareto-front usually holds more than one solution (i.e. a diverse set of optimal sequences according to the objectives of Eq. 3).

Genotype: We wish to solve the three sub-problems from Sect. 3.3 at once. Inspired by similar, successful, encodings of problems (cf. [8]), we thus compose the genotype as $G = [A_1; \dots; A_N; V_{N+1}; \dots; V_{2N}] = [A; V]$, with $A_i; V_i \in [0; 1]$.

The first N values, A, in G encode the action subset A (cf. Prob. 1) and their ordering $t \in \{1; \dots; T\}$ in the sequence S (cf. Prob. 3). Each index position $i \in \{1; \dots; |A|\}$ corresponds to one of the actions in the action set $a_i \in A$ (i.e. $A_i \in A$ encodes $a_i \in A$). The other half, V, encodes the tweaking values V (cf. Prob. 2) of each action, which is also referred to by the index position $i \in \{1; \dots; |V|\}$ (i.e. $V_{N+i} \in V$ encodes $v_i \in V_h$). Fig. 3 visualizes this composition of the genotype.

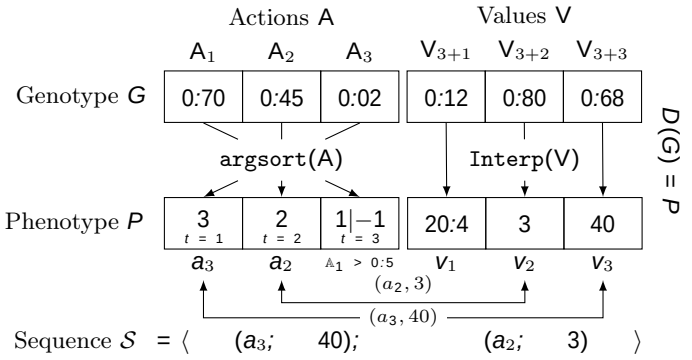


Fig. 3. Anatomy and representation of the solution decoding.

Decoder: Since BRKGA itself is problem-independent, we design a problem-specific *decoder* $D(G) = P$ to infer the phenotype P from G. Below we discuss its design, which is inspired by established concepts (cf. [8]).

The subset of actions (cf. Prob. 1) is decoded by identifying *inactive* actions in the actions part A. As a simple heuristic, we define an action in G as inactive (denoted by "-1"), if its genotype value is greater than 0.5. This follows from the idea that an action has an equal chance to be active or inactive when chosen randomly. To get the *active* actions and their *order* (cf. Prob. 3), we apply the commonly used $\texttt{argsort} : A \to A$ decoding [1, 8] on A and identify the inactive actions afterwards, which will always produce a non-repeating order. We find the actual actions by looking at the sorted index (cf. P in Fig. 3). Note that an action will be at an earlier position t in S the lower its genotype value is. Lastly, we decode the values part by applying a (linear) interpolation $\texttt{Interp} : V \to V$ on each of the genotype values in V. Therefore, only the original value ranges need to be provided (via V), or in case of categorical values a mapping between the interpolated value and the respective categorical value counterpart. The decoded value at position $i \in \{1; \dots; |V|\}$ then belongs to $a_i \in S$.

An example of the full decoding process (from a genotype solution G to the actual solution sequence S) is visualized in Fig. 3. As we can see, applying `argsort` on A *2* G realizes an order (3,2,1) for A via P. Since $A_1 > 0.5$, action a_1 is rendered inactive and the remaining, ordered, action set is $\langle a_3^1, a_2^2 \rangle$. The associated tweaking values are then decoded by interpolating V *2* G and assigned to their action counterparts, thus creating the sequence S. Note that decoding is necessary for all solutions in each iteration to evaluate the fitness and is repeated until the termination criterion (e.g. maximum number of iterations) is reached.

5 Experiments

The first goal of our experiments is to evaluate the costs of the generated sequences[1] in comparison to the state-of-the-art approach [19] (Sect. 5.1). Next, we analyze the diversity of the generated solutions in terms of the action space and the sequence orders (Sect. 5.2). Finally, we examine the effect of the action positions in a sequence for switching the class label (Sect. 5.3).

Datasets: We report on the datasets *Adult Census* [6] (for predicting whether income exceeds \$50k/year based on census data) and *German Credit* [6] (for predicting whether a loan application will be accepted).

Baselines: The following three methods were used for the evaluation. The first one acts as a direct competitor and the others are variations of our method.

`synth` [19]: The competitor[2] only optimizes for finding a *single* minimal cost sequence and solves two separate sub-problems independently. First, they generate candidate action subsets according to Prob. 1 with respect to one of their proposed heuristics. Then, they perform an adversarial attack based on [3] in order to find the tweaking values for each candidate sequence to solve Prob. 2. There is no explicit sequence order notion as per Prob. 3 apart from pre- and post-requirements, thus it only optimizes on their provided cost function which is equivalent to the action effort that we introduced in Sect. 3.2, i.e. $c_i = b_i$. To make the comparison to [19] fair, we use their exact same action-cost model and provided pre-trained black-box classifiers (which are neural networks here) for all methods. That means, all action behavior is identical in this comparison (i.e. tweaking effects, constraints, conditions and costs). Hence, we use the costs of their model for the action effort b_i.

`cscf`: Our method optimizes *all* sub-problems at once and provides *multiple* solutions. Regarding the cost, it considers the consequential discount and thus optimizes $c_i = b_i \ \ g_i$. For g_i, we provided a simple feature relationship graph G that models beneficial effects in *Adult Census* such as:

> The higher the `Education` level, the easier it gets to increase `Capital Gain`, change `Work-Class` and `Occupation`.
>
> The lower the `Work Hours`, the easier it is to increase the `Education`.

[1] Apart from the cost objective o_1, we will not report on the remaining objective space from Eq. 3 here, since the results were similar and thus not particularly informative.

[2] `https://github.com/goutham7r/synth-action-seq`

The higher the `Work Hours`, the easier it gets to increase `Capital Gain`. We only use `cscf` for *Adult Census* as it was not practical to create a meaningful graph based on the predefined actions from [19] for *German Credit*. Since the g_i part primarily affects the order of actions, we would generally expect `cscf` to behave similarly to `scf` in terms of b_i, though.

`scf`: This is a variation of our proposed `cscf`, leaving out the consequential discount and thus only optimizes on the action efforts from [19], i.e. $c_i = b_i$. When referring to findings that apply to both `cscf` and `scf`, we use "`(c)scf`".

Implementations: We implemented our method in Python[3] using the *pymoo* [2] implementation of BRKGA. The parameters for BRKGA are mostly based on recommendations from [8]: the population size was set to 500, the mutant and offspring fractions to 0.2 and 0.8, respectively, and the crossover bias to 0.7. As the termination criterion for `(c)scf`, we fixed the number of iterations to 150. From each dataset we chose 100 random instances that are currently classified by the black-box as the undesired class (i.e. `Salary` < \$50k and `Credit denied`) with the intention of generating a sequential counterfactual to flip their class label. Each instance represents an experiment. We ran all methods on the same 100 experiments and fixed the maximum sequence length of `synth` to $T = 2$ because of long runtimes for larger values which made the experiments of those unfeasible on our hardware[4]. The long runtimes of `synth` were already mentioned in their paper: "Time/iteration is 15s across instances" [19], which confirms our observations, since the algorithm may take up to a few 100 iterations according to [19] (running `synth` for T 2 took the same time as `(c)scf` needed for all sequence lengths simultaneously). Because of this, we used the "*Vanilla*" heuristic for `synth` as it was found to perform the best for shorter sequences based on [19]. Lastly, we had to filter out some experiments in the post-processing since `synth` produced constraint violating solutions or did not find a feasible one. Consequently, the number of experiments for *German Credit* was post-hoc decreased to 85, but for *Adult Census* it did not change.

5.1 Sequence Costs of Sequential Counterfactuals

We show the *undiscounted* (i.e. only using the action effort share b_i) sequence costs of the solutions (o_1 objective) in Fig. 4 for *Adult Census* and *German Credit*. In the x-axis we see the individual, pair-wise relative differences between the computed minimal cost sequences for two methods and each of the valid initial inputs/experiments (which are represented by the y-axis). The green color indicates that the method mentioned first in the title (A) performed better, whereas red indicates the other one (B) did. The blue line shows the point from which one method consistently outperforms the other.

Since our method finds *multiple* optimal sequences (on median 4 for *German Credit* and 7 for *Adult Census*) of different lengths per experiment, and `synth`

[3] `https://github.com/ppnaumann/CSCF`

[4] All experiments (competitor and our method) were executed on the free tier of *Google Colab* (`https://colab.research.google.com/`).

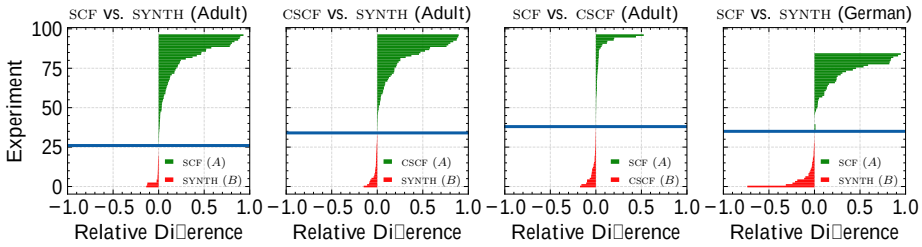


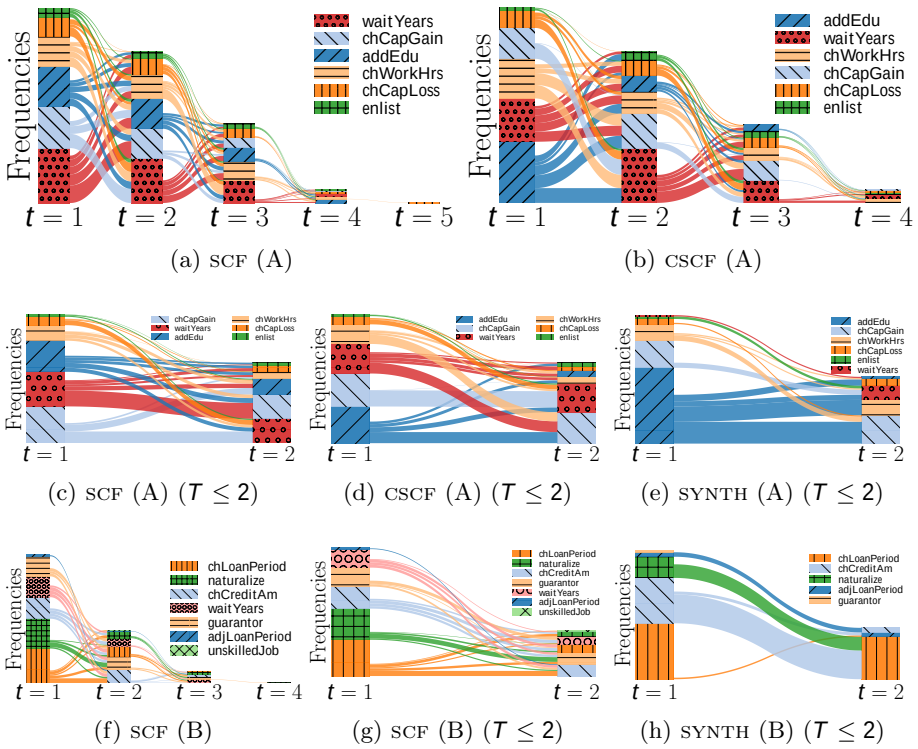
Fig. 4. Relative minimal sequence cost (o_1) differences between the three methods for both datasets and solutions with $T \leq 2$. It is computed as: $(B - A) / \max\{A; B\}$.

only finds a single one, we chose the least cost sequence in `(c)scf` per set that satisfies $T \quad 2$ in order to make the comparison fair. Note, that there could be a longer sequence with less overall cost that was found by `(c)scf` due to using more, but cheaper actions. Although `cscf` has optimized on the discounted cost, we only use the effort share, b_i, of it in this analysis to guarantee comparability.

As we can see in Fig. 4, `(c)scf` usually performed better in *Adult Census*. In *German Credit* it seems to be fairly even, but with a slight tendency towards `scf`. The overall larger relative differences in favor of `(c)scf` (green), with respect to `synth`, appear to be the result of `synth` selecting a different, but more expensive set of actions. By looking through the history trace, we identified that the same set of actions that `(c)scf` found to be optimal was evaluated by `synth`, although with different tweaking values. These values, however, seemed to produce a constraint-breaking solution that was either rendered invalid by `synth`, or had high costs, since constraints are enforced as a penalty in `synth`. The cases where `synth` outperforms `(c)scf` (red) show small cost differences only. Notably, the differences between `cscf` and `scf` are also minor, even though `cscf` optimizes on the discounted costs. Thus, this suggests that the augmentation by g_i does not significantly interfere with the goal of keeping C_S minimal. In general, we conclude that `(c)scf` is capable to find equivalent or better solutions in comparison to `synth` in terms of costs.

5.2 Diversity of Sequential Counterfactuals

In Fig. 5 we illustrated the prevalence of actions at different positions t in a sequence (indicated by the height of each action in the pillars) along with the frequency of how often one action followed on from another (indicated by the widths of the flows). The whitespace of an action in the pillar shows that a sequence stopped there (i.e. had no subsequent actions). For this purpose, we aggregated over *all* optimal solution sequences (i.e. the whole final Pareto-fronts) from each experiment per method and dataset, respectively. Furthermore, we additionally show `(c)scf` after filtering out all solutions with $T > 2$ (c,d,g in Fig. 5) for better comparability with `synth`. The plots (a,b,c,d,e) in Fig. 5 belong to the *Adult Census* (A) and (f,g,h) to the *German Credit* (B) dataset.



(a) SCF (A) (b) CSCF (A)

(c) SCF (A) ($T \leq 2$) (d) CSCF (A) ($T \leq 2$) (e) SYNTH (A) ($T \leq 2$)

(f) SCF (B) (g) SCF (B) ($T \leq 2$) (h) SYNTH (B) ($T \leq 2$)

Fig. 5. Sankey plots showing the sequence orders and flow between subsequent actions for differently aggregated solution (sub-)sets. The first two rows correspond to *Adult Census* (A) and the third to *German Credit* (B).

As we can see in Fig. 5 (a,b,c,d,f,g), `(c)scf` makes use of the whole available action space $\mathcal{A}$, and more evenly utilizes the actions in each step than `synth` (e,h). For *German Credit*, we notice that `synth` only used five different actions, whereas `scf` used all available (f,g). This observation is in alignment with our diversity goal and can be attributed to the tweaking frequency objectives (o_{2+h} from Eq. 3) which force the algorithm to seek alternative sequences that propose different changes while still providing minimal costs. Since `synth` only finds the single least cost solution, the same actions were chosen as they appear to be less expensive than their alternatives. Regarding the lengths of the found sequences, we see that the minimal cost ones were usually found up to length three according to (a,b,f). After that, only few sequences still provide some sort of minimal cost. Because of this, we can say that `(c)scf` does implicitly favor shorter sequences if they are in alignment with the costs. This behavior also follows from the tweaking frequency objectives, which minimize the number of times a feature was changed and thus the number of actions used (as these are directly related to another).

Looking at the particular differences between cscf (b,d) and scf (a,c), we can identify the distinct characteristics of discounting each action effort by g_i through G. The relationship "the higher the Education, the easier it gets to attain Capital Gain" is reflected in (b,d) as addEdu is the most frequent action at $t = 1$. Moreover, there is no single sequence where addEdu would appear *after* chCapGain, indicating that the beneficial consequence was always used by cscf. In comparison, scf in (a,c) has a more equal spread as it has no knowledge of the relationships. Lastly, the same peculiarity can be observed for chWorkHrs, which was more often favored to be placed before chCapGain in cscf than scf because of the beneficial relation in G. Even though it appears that the addEdu effect is also visible in synth according to (e), this is an artifact since there is no explicit mechanism that would enforce it. The most likely reason for this is the order in which the actions were processed in the *Vanilla* heuristic.

Finally, looking at the T 2 plots (c,d,e,g,h) specifically, we can see that some actions show a preferred co-occurrence. E.g., chCapGain and waitYears seem to appear more often subsequently than others (c,d). This is not visible in synth (e), which on the other hand shows a distinct co-occurrence of chCreditAm and chLoanPeriod. The reason for this can be traced back to the cost model, which values these combinations as least expensive for sequence lengths of T 2 (i.e. if we only use two actions). In case of (c)scf this effect is weaker though, as it seeks for alternatives by design (cf. diversity principle from Sect. 3.3).

5.3 Effect of the Action Positions on Achieving the Counterfactual

Lastly, looking at Fig. 6 we can see how each action affects the target class probability of accept in the *Adult Census* with respect to their positional occurrence in a sequence. We again aggregated over *all* computed solutions here. The x-axis denotes the position t in the sequence and the y-axis shows the median probability of the target class based on the black-box and the bootstrapped 95% confidence interval (i.e. 2.5 & 97.5 percentiles).

As we can see, there are some actions that are almost able to switch the class on their own when used (chCapGain, addEdu), whereas the remaining ones only do it later at position two (waitYears, enlist) or three (chWorkHrs, chCapLoss). Based on this, it suggests that some actions are only of supportive nature, whereas others can be seen as the main driver behind class label changes. Additionally, the actions that affect a feature, which was attributed a consequential effect through G (chCapGain, addEdu, enlist), are the only ones that show a significant difference here. Furthermore, the effect of G in cscf is visible. When G was used, chCapGain changed the class often on position one already, whereas in scf it was usually not quite possible. Based on this, we can infer that chCapGain at position one in cscf was typically used when it was able to change the class on its own. Moreover, it suggests that the Education level was already sufficiently high in the input instance, so that chCapGain was able to increase more while benefiting from the discount enough that the costs were kept low. The same might be the reason for enlist (i.e. joining the army), which was more able to change the class in cscf. Since this action affects the Occupation feature, the

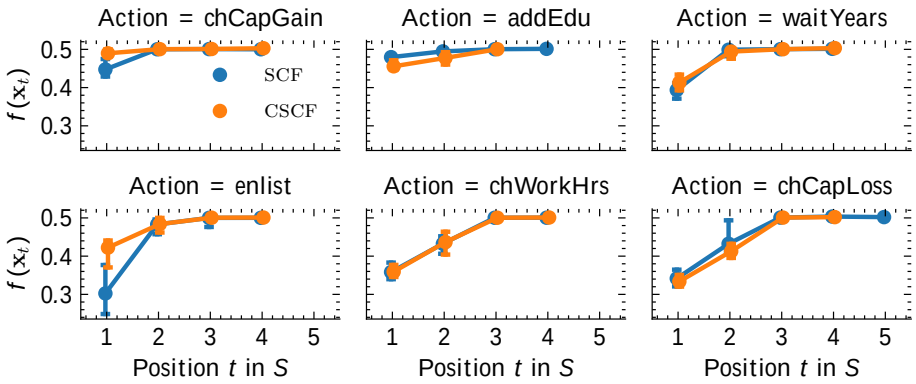


Fig. 6. Median effect with 95% confidence interval of each action at position t in a sequence on the target class (`accept`) probability for *Adult Census*. $f(\mathbf{x}_t) > 0.5$ indicates the class label switched at position t.

beneficial edge of `Education` might have discounted the cost here again so much that it became a minimal cost sequence, whereas in `scf` it might have been more expensive. The slightly lower probability of `addEdu` in `cscf` may further suggest that `Education` was more commonly used as a supporting action in order to discount future actions (hence its disappearance after $t = 3$).

In summary, we saw that `(c)scf` is able to find more diverse sequences while asserting the least cost objective with comparable or better performance to `synth` and being more efficient. Furthermore, we demonstrated that the usage of G in `cscf` provides the desired advantage of more meaningful sequence orders, according to the feature relationships, while maintaining minimal costs.

6 Conclusion and Future Work

We proposed a new method `cscf` for sequential counterfactual generation that is model-agnostic and capable of finding multiple optimal solution sequences of varying sequence lengths. Our variants, `cscf` and `scf`, yield better or equivalent results in comparison to `synth` [19] while being more efficient. Moreover, our extended consequence-aware cost model in `cscf`, that considers feature relationships, provides more meaningful sequence orders compared to `scf` and `synth` [19]. In future work, we aim to incorporate causal models to estimate consequential effects in the feature and cost space. Additionally, we want to investigate alternative measures and objectives for evaluating sequence orders and develop a final selection guide for the end user for choosing a sequence from the solution set.

References

1. Bean, J.C.: Genetic Algorithms and Random Keys for Sequencing and Optimization. ORSA Journal on Computing **6**(2), 154–160 (May 1994)

2. Blank, J., Deb, K.: Pymoo: Multi-objective Optimization in Python. IEEE Access **8**, 89497–89509 (2020)
3. Carlini, N., Wagner, D.: Towards Evaluating the Robustness of Neural Networks. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 39–57 (May 2017)
4. Dandl, S., Molnar, C., Binder, M., Bischl, B.: Multi-Objective Counterfactual Explanations. In: PPSN XVI. pp. 448–469. LNCS, Springer (2020)
5. Downs, M., Chu, J.L., Yacoby, Y., Doshi-Velez, F., Pan, W.: CRUDS: Counterfactual recourse using disentangled subspaces. ICML WHI 2020 pp. 1–23 (2020)
6. Dua, D., Graff, C.: UCI Machine Learning Repository. University of California, Irvine (2017), `http://archive.ics.uci.edu/ml`
7. Eiben, A., Smith, J.: Introduction to Evolutionary Computing. Natural Computing Series, Springer Berlin Heidelberg (2015)
8. Gonçalves, J.F., Resende, M.G.C.: Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics **17**(5), 487–525 (Oct 2011)
9. Gower, J.C.: A General Coefficient of Similarity and Some of Its Properties. Biometrics **27**(4), 857–871 (1971)
10. Joshi, S., Koyejo, O., Vijitbenjaronk, W., Kim, B., Ghosh, J.: Towards Realistic Individual Recourse and Actionable Explanations in Black-Box Decision Making Systems. arXiv:1907.09615 (Jul 2019)
11. Karimi, A.H., Barthe, G., Balle, B., Valera, I.: Model-Agnostic Counterfactual Explanations for Consequential Decisions. In: AISTATS. pp. 895–905. PMLR (2020)
12. Karimi, A.H., Barthe, G., Schölkopf, B., Valera, I.: A survey of algorithmic recourse: Definitions, formulations, solutions, and prospects. arXiv:2010.04050 (2020)
13. Karimi, A.H., von Kügelgen, B.J., Schölkopf, B., Valera, I.: Algorithmic recourse under imperfect causal knowledge: A probabilistic approach. NeurIPS **33** (2020)
14. Lash, M.T., Lin, Q., Street, N., Robinson, J.G., Ohlmann, J.: Generalized inverse classification. In: SDM17. pp. 162–170. SIAM (2017)
15. Laugel, T., Lesot, M.J., Marsala, C., Renard, X., Detyniecki, M.: Comparison-Based Inverse Classification for Interpretability in Machine Learning. In: IPMU 2018. pp. 100–111. CCIS, Springer (2018)
16. Mahajan, D., Tan, C., Sharma, A.: Preserving Causal Constraints in Counterfactual Explanations for Machine Learning Classifiers. arXiv:1912.03277 (Jun 2020)
17. Mothilal, R.K., Sharma, A., Tan, C.: Explaining machine learning classifiers through diverse counterfactual explanations. In: FAT* '20. pp. 607–617. ACM (Jan 2020)
18. Poyiadzi, R., Sokol, K., Santos-Rodriguez, R., De Bie, T., Flach, P.: FACE: Feasible and Actionable Counterfactual Explanations. In: Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society. pp. 344–350. AIES '20, ACM (Feb 2020)
19. Ramakrishnan, G., Lee, Y.C., Albarghouthi, A.: Synthesizing Action Sequences for Modifying Model Decisions. AAAI **34**(04), 5462–5469 (Apr 2020)
20. Shavit, Y., Moses, W.S.: Extracting Incentives from Black-Box Decisions. arXiv:1910.05664 (Oct 2019)
21. Srinivas, N., Deb, K.: Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. Evolutionary Computation **2**(3), 221–248 (1994)
22. Ustun, B., Spangher, A., Liu, Y.: Actionable Recourse in Linear Classification. In: Proceedings of the Conference on Fairness, Accountability, and Transparency. pp. 10–19. ACM (Jan 2019)
23. Van Looveren, A., Klaise, J.: Interpretable Counterfactual Explanations Guided by Prototypes. arXiv:1907.02584 (Jul 2019)
24. Wachter, S., Mittelstadt, B., Russell, C.: Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR. SSRN (2017)