

Synthesizing explainable counterfactual policies for algorithmic recourse with program synthesis

Giovanni De Toni^{1,2*}, Bruno Lepri², Andrea Passerini¹

¹Structured Machine Learning Group (SML), University of Trento, Italy

²Mobile and Social Computing Lab (MobS), Fondazione Bruno Kessler, Italy

{giovanni.detoni, andrea.passerini}@unitn.it, lepri@fbk.eu

Abstract

Being able to provide counterfactual interventions – sequences of actions we would have had to take for a desirable outcome to happen – is essential to explain how to change an unfavourable decision by a black-box machine learning model (e.g., being denied a loan request). Existing solutions have mainly focused on generating feasible interventions without providing explanations on their rationale. Moreover, they need to solve a separate optimization problem for each user. In this paper, we take a different approach and learn a program that outputs a sequence of explainable counterfactual actions given a user description and a causal graph. We leverage program synthesis techniques, reinforcement learning coupled with Monte Carlo Tree Search for efficient exploration, and rule learning to extract explanations for each recommended action. An experimental evaluation on synthetic and real-world datasets shows how our approach generates effective interventions by making orders of magnitude fewer queries to the black-box classifier with respect to existing solutions, with the additional benefit of complementing them with interpretable explanations.

1 Introduction

Counterfactual explanations are very powerful tools to explain the decision process of machine learning models [Wachter *et al.*, 2017; Karimi *et al.*, 2020b]. They give us the intuition of what could have happened if the state of the world was different (e.g., if you had taken the umbrella, you would not have gotten soaked). Researchers have developed many methods that can generate counterfactual explanations given a trained model [Wachter *et al.*, 2017; Dandl *et al.*, 2020; Mothilal *et al.*, 2020; Karimi *et al.*, 2020a; Guidotti *et al.*, 2018; Stepin *et al.*, 2021]. However, these methods do not provide any actionable information about which steps are required to obtain the given counterfactual. Thus, most of these methods do not enable algorithmic recourse. Algorithmic recourse describes the ability to pro-

vide “explanations and recommendations to individuals who are unfavourably treated by automated decision-making systems” [Karimi *et al.*, 2021]. For instance, algorithmic recourse can answer questions such as: what actions does a user have to perform to be granted a loan? Recently, providing feasible algorithmic recourse has also become a legal necessity [Voigt and Bussche, 2017]. Some research works address this problem by developing ways to generate counterfactual interventions [Karimi *et al.*, 2021], i.e., sequences of actions that, if followed, can overturn a decision made by a machine learning model, thus guaranteeing recourse. While being quite successful, these methods have several limitations. First, they are purely optimization methods that must be rerun from scratch for each new user. As a consequence, this requirement prevents their use for real-time interventions’ generation. Second, they are expensive in terms of queries to the black-box classifier and computing time. Last but not least, they fail to explain their recommendations (e.g., why does the model suggest getting a better degree rather than changing jobs?). On the contrary, explainability has been pointed out as a major requirement for methods generating counterfactual interventions [Barocas *et al.*, 2020].

In this paper, we cast the problem of providing explainable counterfactual interventions as a program synthesis task [De Toni *et al.*, 2021; Pierrot *et al.*, 2019; Bunel *et al.*, 2018; Balog *et al.*, 2017]: we want to generate a “program” that provides all the steps needed to overturn a bad decision made by a machine learning model. We propose a novel reinforcement learning (RL) method coupled with a discrete search procedure, Monte Carlo Tree Search [Coulom, 2006], to generate counterfactual interventions in an efficient data-driven manner. As done by [Naumann and Ntoutsis, 2021], we assume a causal model encoding relationships between user features and consequences of potential interventions. We also provide a solution to distil an explainable deterministic program from the learned policy in the form of an automaton. Figure 1 provides an overview of the architecture and the learning strategy, and an example of an explainable intervention generated by the extracted automaton. Our approach addresses the three main limitations that characterize existing solutions:

- It learns a general policy that can be used to generate interventions for multiple users, rather than running separate user-specific optimizations.

*Contact Author

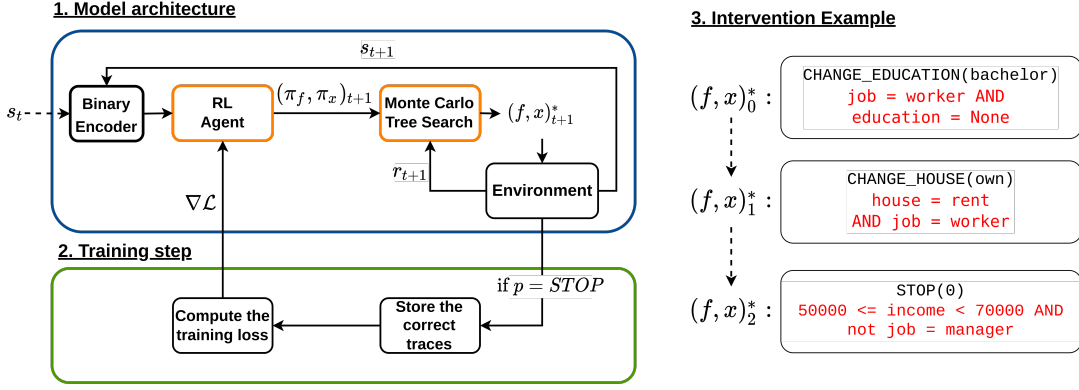


Figure 1: **1. Model architecture.** Given the state s_t representing the features of the user, the agent generates candidate intervention policies π_f and π_x for functions and arguments, respectively (an action is a function-argument pair). MCTS uses these policies as a prior, and it extracts the best next action $(f, x)_{t+1}^*$. Once found, the reward received upon making the action is used to improve the MCTS estimates, and correct traces (i.e., those leading to the desired outcome change) are saved in a replay buffer. **2. Training step.** The buffer is used to sample a subset of correct traces to be used to train the RL agent to mimic the behaviour of MCTS. **3. Explainable intervention.** Example of an explainable intervention generated by the automaton extracted from the learned agent. Actions are in black, while explanations for each action are in red.

- By coupling reinforcement learning with Monte Carlo Tree Search, it can efficiently explore the search space, requiring massively fewer queries to the black-box classifier than the best evolutionary algorithm (EA) model available, especially in settings with many features and (relatively) long interventions.
- By extracting a program from the learned policy, it can complement the intervention with explanations motivating each action from contextual information. Furthermore, the program can be executed in real-time without accessing the black-box classifier.

Our experimental results on synthetic and real-world datasets confirm the advantages of the proposed solution over existing alternatives in terms of generality, scalability and interpretability.

2 Related Work

Recent research has shown how to generate counterfactual interventions for algorithmic recourse via various techniques [Karimi *et al.*, 2020b], such as probabilistic models [Karimi *et al.*, 2020c], integer programming [Ustun *et al.*, 2019], reinforcement learning [Yonadav and Moses, 2019], program synthesis [Ramakrishnan *et al.*, 2020] and genetic algorithms [Naumann and Ntoutsis, 2021]. Methods with (approximated) convergence guarantees on the optimal counterfactual policies have also been proposed [Tsirtsis and Rodriguez, 2020]. However, most of these methods ignore the causal relationships between user features [Tsirtsis and Rodriguez, 2020; Ustun *et al.*, 2019; Yonadav and Moses, 2019; Ramakrishnan *et al.*, 2020]. It has been recently shown that optimal algorithmic recourse is impossible to achieve without a causal model of the interactions between the features [Karimi *et al.*, 2020c]. The work by Karimi *et al.* [2020c] does provide algorithmic recourse following a causal model but optimizes for interventions that can operate on multiple user’s attributes simultaneously, which is unrealistic. CSCF [Naumann and Ntoutsis,

2021] is the only model-agnostic method capable of producing *consequence-aware* sequential interventions by exploiting causal relationships between features represented by a causal graph. However, CSCF is still purely an (evolutionary-based) optimization method, so it has to be run from scratch for each new user. Furthermore, the approach is opaque with respect to the reasons behind a suggested intervention. In this work, we show how our approach improves over CSCF in terms of generality, efficiency and interpretability.

3 Methods

3.1 Problem setting

The state of a user is represented as a vector of attributes $s \in \mathcal{S}$ (e.g., age, sex, monthly income, job). A black-box classifier $h : \mathcal{S} \rightarrow \{\text{True}, \text{False}\}$ predicts an outcome given a user state, with *True* being favourable to the user and *False* being unfavourable. The setting can be easily extended to multiclass classification by either grouping outcomes in favourable and unfavourable ones or learning separate programs converting from one class to the other. A *counterfactual intervention* I is a sequence of actions. Each action is represented as a tuple, $(f, x) \in \mathcal{A}$, composed by a *function*, f , and its argument, $x \in \mathcal{X}_f$ (e.g., (change_income, 500)). When an action is performed for a certain user, it modifies their state by altering one of their attributes according to its argument. A library \mathcal{F} contains all the possible functions which can be called. This library is typically defined as a-priori by domain experts to prevent changes to protected attributes (e.g., age, sex, etc.). Moreover, each function possesses *pre-conditions* in the form of Boolean predicates over its arguments which describe the conditions that a user state must meet in order for a function to be called. The end of an intervention I is always specified by the STOP action. We also define a cost function, $C : \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ which mimics the effort made by a given user to perform an action given the current state. The cost is computed by looking at a causal graph \mathcal{G} ,

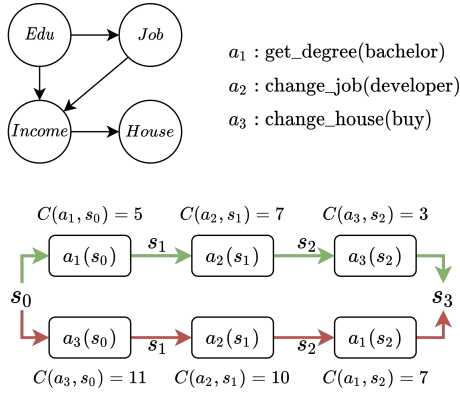


Figure 2: **Examples of interventions on a causal graph.** (Top) A causal graph and a set of candidate actions. (Bottom) Examples of interventions together with their costs. Note that the green line ($\sum C = 15$) has a lower cost than the red line ($\sum C = 28$) thanks to a better ordering of the actions making up the intervention.

where the nodes of the graph are the user’s features. This assumption encodes the concept of *consequences* and it ensures a notion of order for the intervention’s actions. For example, it might be easier to get first a degree and then a better salary, rather than doing the opposite. Figure 2 shows an example of a causal graph \mathcal{G} and of the corresponding costs. Our goal is to train an agent that, given a user with an unfavourable outcome, generates counterfactual interventions that overturn it. Given a black-box classifier h , a user s_0 for whom the prediction by h is unfavourable (i.e., $h(s_0) = \text{False}$), a causal graph \mathcal{G} and a set of possible actions \mathcal{A} (implicitly represented by the functions in \mathcal{F} and their arguments in \mathcal{X}), we want to generate a sequence I^* , that, if applied to s_0 , produces a new state, $s^* = I(s_0)$, such that $h(s^*) = \text{True}$. This sequence must be *actionable*, which means that the user has to be able to perform those actions, and minimize the user’s *cost*. More formally:

$$\begin{aligned}
 I^* = \min_I \quad & \sum_{t=0}^T C(a_t, s_t) \\
 \text{s.t.} \quad & I = \{a_t\}_{t=0}^T \quad a_t \in \mathcal{A} \quad \forall t \\
 & s_t = I_{t-1}(s_{t-1}) \quad \forall t > 0 \\
 & h(I(s_0)) \neq h(s_0)
 \end{aligned} \tag{1}$$

3.2 Model Architecture

Overall structure. Figure 1 shows the complete model architecture. It is composed of a binary encoder and an RL agent coupled with the Monte Carlo Tree Search procedure. The binary encoder converts the user’s features into a binary representation. The conversion is done by one-hot-encoding the categorical features and discretizing the numerical features into ranges. In the following sections, we will use s_t to directly indicate the user’s state binary version. Given a state s_t , the RL agent generates candidate policies, π_f and π_x , for the function and argument generation respectively. MCTS uses these policies as priors for its exploration of the action

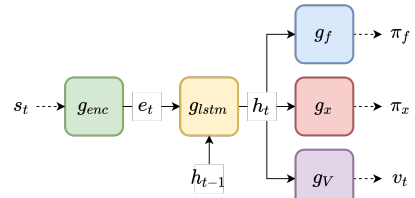


Figure 3: **Agent architecture.** Given the user’s state s_t , it outputs a function policy, π_f , an argument policy π_x and an estimate of the expected reward from the state v_t . These outputs are used to select the next best action $(f, x)_{t+1}$.

space and extracts the best next action $(f, x)_{t+1}^*$. The action is then applied to the environment. The procedure ends when the STOP action is chosen (i.e., the intervention was successful) or when the maximum intervention length is reached, in which case the result is marked as a failure. During training, the reward is used to improve the MCTS estimates of the policies. Moreover, correct traces (i.e., traces of interventions leading to the desired outcome change) are stored in a replay buffer, and a sample of traces from the buffer is used to refine the RL agent.

RL agent structure. The agent structure is inspired by previous program synthesis works [De Toni *et al.*, 2021; Pierrot *et al.*, 2019]. It is composed by 5 components: a state encoder, g_{enc} , an LSTM controller, g_{lstm} , a function network g_f , an argument network g_x and a value network g_v . We use simple feedforward networks to implement g_f , g_x and g_v .

$$g_{enc}(s_t) = e_t \quad f_{lstm}(e_t, h_{t-1}) = h_t \tag{2}$$

$$g_f(h_t) = \pi_f \quad g_x(h_t) = \pi_x \quad g_v(h_t) = v_t \tag{3}$$

g_{enc} encodes the user’s state in a latent representation which is fed to the controller, g_{lstm} . The controller, g_{lstm} learns an implicit representation of the program to generate the interventions. The function and argument networks are then used to extract the corresponding policies, π_f and π_x , by taking as input the hidden state h_t from g_{lstm} . g_v represents the value function V and it outputs the expected reward from the state s_t . Here, we omit the state s_t when defining the policies and the value function output, since s_t is already embedded into the h_t representation. In our settings, we try to learn a single program, which we call INTERVENE.

Policy. A policy is a distribution over the available actions (i.e., functions and their arguments) such that $\sum_{i=0}^N \pi(i) = 1$. Our agent produces two policies: π_f on the function space, and π_x on the argument space. The next action, $(f, x)_{t+1}$, is chosen by taking the argmax over the policies:

$$f_{t+1} = \operatorname{argmax}_{f \in \mathcal{F}} \pi_f(f) \quad x_{t+1} = \operatorname{argmax}_{x \in \mathcal{X}_{f_{t+1}}} \pi_x(x|f_{t+1}) \tag{4}$$

Each program starts by calling the program INTERVENE, and it ends when the action STOP is called.

Reward. Once we have applied the intervention I , given the black-box classifier h , the reward, r , is computed as:

$$r = \lambda^T R \quad \lambda \in (0, 1), \quad R = \begin{cases} 1 & h(I(s)) \neq h(s) \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

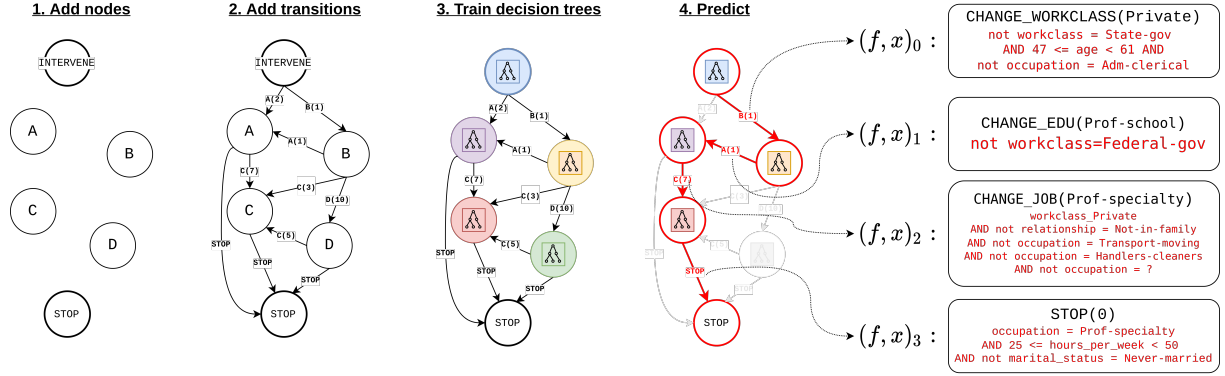


Figure 4: **Procedure to generate the explainable program from intervention traces.** 1. For all $f \in \mathcal{F}$, we add a new node. 2. Given the samples traces, we add the transitions, and we store $(s_i, (f_i, x_i))$ in each node. 3. We train a decision tree for each node to predict the next action (consistently with the sampled traces). 4. We execute the program on the new instance at prediction time, using the decision trees to decide the next action at each node. We extract a Boolean rule explaining it from the corresponding decision tree for each action. On the right, an example of generated intervention. The actions (f, x) are black, while the explanations are red.

where λ is a regularization coefficient and T is the length of the intervention. The λ^T penalizes longer interventions in favour of shorter ones.

3.3 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a discrete heuristic search procedure that can successfully solve combinatorial optimization problems with large action spaces [Silver *et al.*, 2018; Silver *et al.*, 2016]. MCTS explores the most promising nodes by expanding the search space based on a random sampling of the possible actions. In our setting, each tree node represents the user’s state at a time t , and each arc represents a possible action determining a transition to a new state. MCTS searches for the correct sequence of interventions that minimize the user effort and changes the prediction of the black-box model. We use the agent policies, π_f and π_x , as a prior to explore the program space. Then, the newly found sequence of interventions is used to train the RL agent. To select the next node, we maximize the UCT criterion [Kocsis and Szepesvári, 2006]:

$$(f, x)_{t+1} = \underset{f \in \mathcal{F}, x \in \mathcal{X}_f}{\operatorname{argmax}} Q(s, (f, x)) + U(s, (f, x)) + L(s, (f, x)) \quad (6)$$

Here $Q(s, (f, x))$ returns the expected reward by taking action (f, x) . $U(s, (f, x))$ is a term which trades-off exploration and exploitation, and it is based on how many times we visited node s in the tree. $L(s, (f, x))$ is a scoring term which is defined as follow:

$$L(s, (f, x)) = e^{-(l_{\text{cost}}((f, x), s) + l_{\text{count}}(f))} \quad (7)$$

where $l_{\text{cost}} = C(a, s) \in \mathbb{R}$ represents the *effort* needed to perform the $a = (f, x) \in \mathcal{A}$ action, and $l_{\text{count}} \in \mathbb{R}$ penalizes interventions that call multiple times the same function f . MCTS uses the simulation results to return an improved version of the agent policies π_f^{mcts} and π_x^{mcts} .

From the found intervention, we build an *intervention trace*, which is a sequence of tuples that stores, for each time

step t : the input state, the output state, the reward, the hidden state of the controller and the improved policies. The traces are stored in the replay buffer, to be used to train the RL agent.

3.4 Training the agent

The agent has to learn to replicate the interventions provided by MCTS at each step t . Given the replay buffer, we sample a batch of *intervention traces* and we minimize the cross-entropy \mathcal{L} between the MCTS policies and the agent policies for each time step t :

$$\underset{\theta}{\operatorname{argmin}} \sum_{\text{batch}} (V - r)^2 - (\pi_f^{\text{mcts}})^T \log(\pi_f) - (\pi_x^{\text{mcts}})^T \log(\pi_x) \quad (8)$$

where θ represents the agent’s parameters and V is the value function evaluation computed by the agent.

3.5 Explainable Intervention Program

We now show how we can build a deterministic program given the agent. Figure 4 shows the complete procedure and an example of the produced trace. First, we sample M intervention traces from the trained agent and extract a sequence of $\{(s_i, (f, x)_i)\}_{i=0}^T$ for each trace. Then, we construct an automaton graph, \mathcal{P} , in the following way:

1. Given the function library \mathcal{F} , we create a node for each function f available. We also add a starting node called INTERVENE and a “sink” node called STOP;
2. We connect each node by unrolling the sampled traces. Starting from INTERVENE, we treat each action $(f, x)_t$ as a transition. We label the transition with (f, x) and we connect the current node to the one representing the function f ;
3. Lastly, for each node f , we store a collection of outgoing state-action pairs $(s_i, (f, x)_i)$. Namely, we store all the states s and the corresponding outward transitions which were decided by the model while at the node f ;
4. For each node, $f \in \mathcal{P}$, we train a decision tree on the tuples $(s_i, (f, x)_i)$ stored in the node to predict the transition $(f, x)_i$ given a user’s state s_i .

Dataset	$ D $	$h(s) = 1$	$h(s) = 0$	$ s $	$ B(s) $	$ \mathcal{F} $
<i>german</i>	1002	301	701	10	44	7
<i>adult</i>	48845	11691	37154	15	125	6
<i>syn</i>	10004	5002	5002	10	40	6
<i>syn_long</i>	10004	5002	5002	14	64	10

Table 1: **Description of the datasets.** $|D|$ is the size of the dataset. $|s|$ the number of features for an instance. $|B(s)|$ shows how many binary features the agent sees after the conversion with the binary converter. $|\mathcal{F}|$ is the size of the agent program library. $h(s)$ indicates the number of favourable (1) and unfavourable (0) samples.

Generate explainable interventions. The intervention generation is done by traversing the graph \mathcal{P} , starting from the node INTERVENE, until we reach the STOP node or we reach the maximum intervention length. In the last case, the program is marked as a failure. Given the node $f \in \mathcal{P}$ and given the state s_t , we use the decision tree of that node to predict the next transition (f', x') . Moreover, we can extract from the decision tree interpretable rules which tell us why the next action was chosen. Then, we follow (f', x') , which is an arc going from f to the next node f' , and we apply the action to s_t to get s_{t+1} .

4 Experiments

Our experimental evaluation aims at answering the following research questions: (1) Does our method provide better performances than the competitors in terms of the accuracy of the algorithmic recourse? (2) Does our approach allow us to complement interventions with action-by-action explanations in most cases? (3) Does our method minimize the interaction with the black-box classifier to provide interventions?

The code and the dataset of the experiments are available on Github to ensure reproducibility¹. The software exploit parallelization through `mpi4python` [Dalcin and Fang, 2021] to improve inference and training time. We compared the performance of our algorithm with CSCF [Naumann and Ntoutsis, 2021], to the best of our knowledge the only existing model-agnostic approach that can generate consequence-aware interventions following a causal graph. However, note that earlier solutions still perform user-specific optimization, so that our results in terms of generality, interpretability and cost (number of queries to the black-box classifier and computational cost) carry over to these alternatives. For the sake of a fair comparison, we built our own parallelized version of the CSCF model based on the original code. We developed the project to make it easily extendable and reusable by the research community. The experiments were performed using a Linux distribution on an Intel(R) Xeon(R) CPU E5-2660 2.20GHz with 8 cores and 100 GB of RAM (only 4 cores were used).

4.1 Dataset and black-box classifiers

Table 1 shows a brief description of the datasets. They all represent binary (favourable/unfavourable) classification problems. The two real world datasets, *German Credit* (*german*)

¹<http://github.com/xxx/syn-counterfactual-interventions> - it will be made available at a later time.

and *Adult Score* (*adult*) [Dua and Graff, 2017], are taken from the relevant literature. Given that in these datasets a couple of actions is usually sufficient to overturn the outcome of the black-box classifier, we also developed two synthetic datasets, *syn* and *syn_long*, where longer interventions are required, so as to evaluate the models in more challenging scenarios. The datasets are made by both categorical and numerical features (e.g., monthly income, job type, etc.). Each dataset was randomly split into 80% train and 20% test. For each dataset, we manually define a causal graph, \mathcal{G} , by looking at the features available. For the synthetic datasets, we sampled instances directly from the causal graph. The black-box classifier for *german* and *adult* was obtained by training a 5-layers MLP with ReLU activations. The trained classifiers are reasonably accurate (~ 0.9 test-set accuracy for *german*, ~ 0.8 for *adult*). The synthetic datasets (*syn* and *syn_long*) do not require any training since we directly use our manually defined decision function.

4.2 Models

We evaluate four different models: the agent coupled with MCTS (M_{mcts}), the deterministic program distilled from the agent (M_{prog}), and two versions of CSCF, one (M_{cscf}) with a large budget of generation, n , and population size, p , ($n = 50, p = 200$) and one (M_{cscf}^{small}) with a smaller budget ($n = 25, p = 100$).

4.3 Evaluation

The left plot in Figure 5 shows the average accuracy of the different models, namely the fraction of instances for which a model manages to generate a successful intervention. We can see how M_{mcts} outperforms or is on-par with the M_{cscf} and M_{cscf}^{small} models on both the real-word and synthetic datasets. The performance difference is more evident in the synthetic datasets, because the evolutionary algorithm struggles in generating interventions that require more than a couple of actions. The accuracy loss incurred in distilling M_{mcts} into a program (M_{prog}) is rather limited. This implies that we are able to provide interventions with explanations for 94% (*german*), 66% (*adult*), 99% (*syn*) and 87% (*syn_long*) of the test users². Moreover, M_{prog} generates similar interventions to M_{mcts} . The sequence similarity between their respective interventions for the same user are 0.89 (*german*), 0.72 (*adult*), 0.80 (*syn*) and 0.71 (*syn_long*), where 1.0 indicates identical interventions.

The main reason for the accuracy gains of our model is the ability to generate long interventions, something evolutionary-based algorithms struggle with. This effect can be clearly seen from the middle plot of Figure 5. Both M_{cscf} and M_{cscf}^{small} rarely generate interventions with more than two actions, while our approach can easily generate interventions with up to five actions. A drawback of this ability is that intervention costs are, on average, higher (right plot of Figure 5).

²Note that the accuracy loss observed on *adult* is due to the limited sampling budget we allocated for M_{prog} (250 traces for all datasets). Adapting this budget to the feature space size (considerably larger for *adult*) can help boosting the performance, at the cost of generating longer explanations (see Supplementary Material).

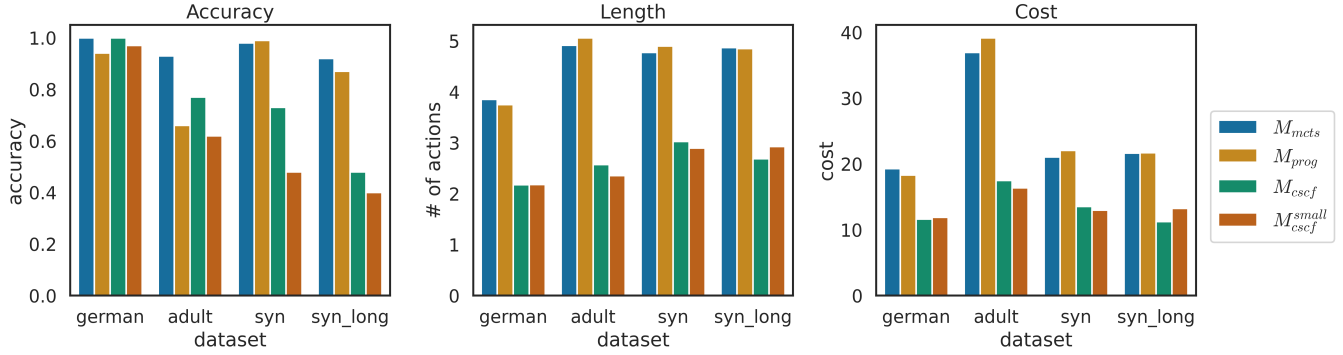


Figure 5: **Experimental results.** (Left) Accuracy (fraction of successful interventions); (Middle) Average length of a successful intervention; (Right) Average cost of a successful intervention. Results are averaged over 100 test examples.

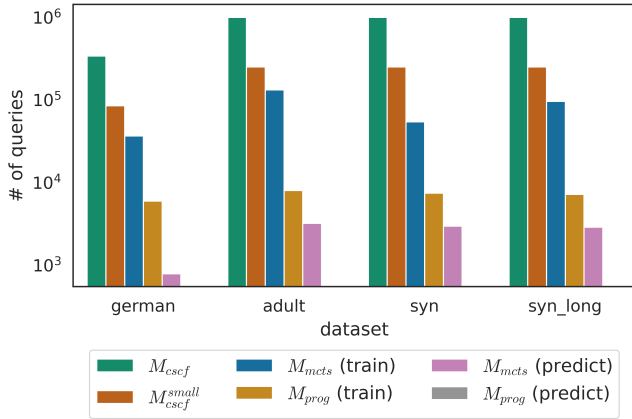


Figure 6: **Number of queries.** Total number of queries to the black-box classifier made by the models. $M_{prog}(predict)$ is not visible, as the automaton does not query the black-box classifier to generate interventions. Note that the number of queries is in logscale.

On the one hand, this is due to the fact that our model is capable of finding interventions for more complex instances, while M_{cscf} and M_{small_cscf} fail. Indeed, if we compute lengths and costs on the subset of instances for which all models find a successful intervention, the difference between the approaches is less pronounced (see Supplementary Material). On the other hand, there is a clear trade-off between solving a new optimization problem from scratch for each new user, and learning a general model that, once trained, can generate interventions for new users in real-time and without accessing the black-box classifier.

Figure 6 reports the average number of queries to the black-box classifier. Our approach requires far fewer queries than M_{cscf} (note that the plot is in logscale), and even substantially less than M_{small_cscf} (that is anyhow not competitive in terms of accuracy). Furthermore, most queries are made for training the agent ($M_{mcts}(train)$), which is only done once for all users. Once the model is trained, generating interventions for a single user requires around two orders of magnitude fewer queries than the competitors. Note that MCTS is crucial to allow the RL agent to learn a successful policy with

a low budget of queries. Indeed, training an RL agent without the support of MCTS fails to converge in the given budget, leading to a completely useless policy (see Supplementary Materials for the details). When turning to the program, building the automaton ($M_{prog}(train)$) requires a negligible number of queries to extract the intervention traces used as supervision. Using the automaton to generate interventions *does not* require to query the black-box classifier. This characteristic can substantially increase the usability of the system, as M_{prog} can be employed directly by the user even if they have no access to the classifier. Computationally speaking, the advantage of a two-step phase is also quite dramatic. M_{cscf} takes an average of $\sim 693s$ for each user to provide a solution (the same order of magnitude of training a model for all users with M_{mcts}), while M_{mcts} inference time is under 1s, allowing real-time interaction with the user.

Overall, our experimental evaluation allows us to affirmatively answer the research questions stated above.

5 Conclusion

This work improves the state-of-the-art on algorithmic recourse by providing a method that can generate effective and interpretable counterfactual interventions in real-time. Our experimental evaluation confirms the advantages of our solution with respect to alternative consequence-aware approaches in terms of accuracy, interpretability and number of queries to the black-box classifier. Our work unlocks many new research directions, which could be explored to solve some of its limitations. First, following previous work on causal-aware intervention generation, we use manually-crafted causal graphs and action costs. Learning them from the available data directly, minimizing the human intervention, would allow applying the approach in settings where this information is not available or unreliable. Second, we showed how our method learns a general program by optimizing over multiple users. It would be interesting to investigate additional RL methods to optimize the interventions globally and locally to provide more personalized sequences to the users. Such methods could be coupled with interactive approaches eliciting preferences and constraints directly from the user, thus maximizing the chance to generate the most appropriate intervention for a given user.

Ethical Impact

The research field of algorithmic recourse aims at improving fairness, by providing unfairly treated users with tools to overturn unfavourable outcomes. By providing real-time, explainable interventions, our work makes a step further in making these tools widely accessible. As for other approaches providing counterfactual interventions, our model could in principle be adapted by malicious users to “hack” a fair system. Research on adversarial training can help in mitigating this risk.

References

- [Balog *et al.*, 2017] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. Deepcoder: Learning to write programs. In *ICLR*, April 2017.
- [Barocas *et al.*, 2020] S. Barocas, A. D. Selbst, and M. Raghavan. The hidden assumptions behind counterfactual explanations and principal reasons. In *FAT**, pages 80–89, 2020.
- [Bunel *et al.*, 2018] R. Bunel, M. Hausknecht, J. Devlin, R. Singh, and P. Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. In *ICLR*, 2018.
- [Coulom, 2006] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *In: Proceedings Computers and Games 2006*. Springer-Verlag, 2006.
- [Dalcin and Fang, 2021] L. Dalcin and Y. Fang. mpi4py: Status update after 12 years of development. *Computing in Science Engineering*, 23(4):47–54, 2021.
- [Dandl *et al.*, 2020] S. Dandl, C. Molnar, M. Binder, and B. Bischl. Multi-objective counterfactual explanations. In *PPSN*, pages 448–469. Springer, 2020.
- [De Toni *et al.*, 2021] G. De Toni, L. Erculiani, and A. Passerini. Learning compositional programs with arguments and sampling. In *AIPLANS*, 2021.
- [Dua and Graff, 2017] D. Dua and C. Graff. UCI machine learning repository, 2017.
- [Guidotti *et al.*, 2018] R. Guidotti, A. Monreale, S. Ruggieri, D. Pedreschi, F. Turini, and F. Giannotti. Local rule-based explanations of black box decision systems. *CoRR*, abs/1805.10820, 2018.
- [Karimi *et al.*, 2020a] A. Karimi, G. Barthe, B. Balle, and I. Valera. Model-agnostic counterfactual explanations for consequential decisions. In *AISTATS*, pages 895–905. PMLR, 2020.
- [Karimi *et al.*, 2020b] A. Karimi, G. Barthe, B. Schölkopf, and I. Valera. A survey of algorithmic recourse: definitions, formulations, solutions, and prospects. *arXiv preprint arXiv:2010.04050*, 2020.
- [Karimi *et al.*, 2020c] A. Karimi, J. von Kügelgen, B. Schölkopf, and I. Valera. Algorithmic recourse under imperfect causal knowledge: a probabilistic approach. In *NeurIPS*, pages 265–277. Curran Associates, Inc., December 2020.
- [Karimi *et al.*, 2021] A. Karimi, B. Schölkopf, and I. Valera. Algorithmic recourse: from counterfactual explanations to interventions. In *FaccT*, pages 353–362, 2021.
- [Kocsis and Szepesvári, 2006] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *ECML*, page 282–293, Berlin, Heidelberg, 2006. Springer-Verlag.
- [Mothilal *et al.*, 2020] R. K. Mothilal, A. Sharma, and C. Tan. Explaining machine learning classifiers through diverse counterfactual explanations. In *FAT**, pages 607–617, 2020.
- [Naumann and Ntoutsis, 2021] P. Naumann and E. Ntoutsis. Consequence-Aware Sequential Counterfactual Generation. In *Machine Learning and Knowledge Discovery in Databases. Research Track*, pages 682–698. Springer International Publishing, 2021.
- [Pierrot *et al.*, 2019] T. Pierrot, G. Ligner, S.E. Reed, O. Sigaud, N. Perrin, A. Laterre, D. Kas, K. Beguir, and N. de Freitas. Learning compositional neural programs with recursive tree search and planning. *NeurIPS*, 32:14673–14683, 2019.
- [Ramakrishnan *et al.*, 2020] G. Ramakrishnan, Y. C. Lee, and A. Albarghouthi. Synthesizing action sequences for modifying model decisions. In *AAAI*, volume 34, pages 5462–5469, 2020.
- [Silver *et al.*, 2016] D. Silver, C. J. Maddison A. Huang, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.
- [Silver *et al.*, 2018] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.
- [Stepin *et al.*, 2021] I. Stepin, J. M. Alonso, and A. Catalaand M. Pereira-Fariña. A survey of contrastive and counterfactual explanation generation methods for explainable artificial intelligence. *IEEE Access*, 9:11974–12001, 2021.
- [Tsiptsis and Rodriguez, 2020] S. Tsiptsis and M. Rodriguez. Decisions, counterfactual explanations and strategic behavior. In *NeurIPS*, 2020.
- [Ustun *et al.*, 2019] B. Ustun, A. Spangler, and Y. Liu. Actionable recourse in linear classification. In *FAT**, pages 10–19, 2019.
- [Voigt and Bussche, 2017] P. Voigt and A. Bussche. *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Springer Publishing Company, Incorporated, 1st edition, 2017.
- [Wachter *et al.*, 2017] S. Wachter, B. Mittelstadt, and C. Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harv. JL & Tech.*, 31:841, 2017.

[Yonadav and Moses, 2019] S. Yonadav and W. S. Moses.
Extracting incentives from black-box decisions. *CoRR*,
abs/1910.05664, 2019.