CREATED: Generating Viable Counterfactual Sequences for Predictive Process Analytics

Olusanmi Hundogan, Xixi Lu(✉), Yupei Du, and Hajo A. Reijers

Utrecht University, Utrecht, The Netherlands
{o.a.hundogan,x.lu,y.du,h.a.reijers}@uu.nl

Abstract. Predictive process analytics focuses on predicting future states, such as the outcome of running process instances. These techniques often use machine learning models or deep learning models (such as LSTM) to make such predictions. However, these deep models are complex and difficult for users to understand. Counterfactuals answer "what-if" questions, which are used to understand the reasoning behind the predictions. For example, what if instead of emailing customers, customers are being called? Would this alternative lead to a different outcome? Current methods to generate counterfactual sequences either do not take the process behavior into account, leading to generating invalid or infeasible counterfactual process instances, or heavily rely on domain knowledge. In this work, we propose a general framework that uses evolutionary methods to generate counterfactual sequences. Our framework does not require domain knowledge. Instead, we propose to train a Markov model to compute the *feasibility* of generated counterfactual sequences and adapt three other measures (*delta* in outcome prediction, *similarity*, and *sparsity*) to ensure their overall viability. The evaluation shows that we generate viable counterfactual sequences, outperform baseline methods in viability, and yield similar results when compared to the state-of-the-art method that requires domain knowledge.

Keywords: Counterfactual · Explainable AI · Predictive Process Analytics · Evolutionary Algorithm

1 Introduction

Predictive process analytics is an emerging research field in the process mining discipline that focuses on predicting the future states or outcome of running cases of business processes. The proposed techniques often use Machine Learning (ML) models or deep learning models (such as LSTM). These predictive models are trained on historical executions of business processes (i.e., *event logs*) to make predictions of future states or outcomes. Studies have shown that predictive models can forecast the outcome of processes from various domains well [12,20]. For instance, in the medical domain, predictive models are applied to predict the outcome or trajectory of a patient's condition [13].

M. Indulska et al. (Eds.): CAiSE 2023, LNCS 13901, pp. 541–557, 2023.
https://doi.org/10.1007/978-3-031-34560-9_32

While these predictive models are very powerful, they are usually complex and difficult to comprehend. Therefore, they are also known as *blackbox models*. A lack of comprehension is undesirable for many application domains. For example, not knowing why a mortgage application was denied makes it impossible to rule out possible biases. In critical domains like medicine, the reasoning behind decisions becomes more crucial. For instance, if we know that a treatment process of a patient reduces the chances of survival, we want to know which treatment step is the critical factor we ought to avoid. For the engineering of fair and effective information systems, it is essential to comprehend and explain the reasoning behind predictions.

The Explainable AI (XAI) discipline proposes *counterfactuals* as a human-friendly approach to understanding the underlying reasoning of ML models [14, p. 221]. Counterfactuals can help us answer hypothetical "what-if" questions. In other words, assuming we know *what* would happen *if* we changed the execution of a process instance, we could change it for the better. For example, what if instead of emailing customers, customers are contacted by phone? Would this alternative sequence have led to a different outcome (e.g., instead of rejecting the offer, the customer accepts the offer)?

Existing methods can be divided into two categories: traditional and process-aware. The *traditional* counterfactual methods focus on static, tabular data, such as DICE [15]. These methods aim to minimize the feature changes while maximize the flip in the outcome prediction. These methods do not take the process behavior into account. Applying them directly to event logs may lead to generating *invalid* or *infeasible* counterfactual sequences. The *process-aware* methods adapt the traditional methods for counterfactual generations of event logs [8]. While taking normative process behavior into account, these state-of-the-art methods, however, heavily rely on domain knowledge (e.g., users need to know the flows between milestones of a process) [8].

In this paper, we approach the problem of generating counterfactual sequences for process outcome prediction without domain knowledge. In particular, we propose a general framework that uses evolutionary algorithms to generate sequences. The framework contains three components. The first component is a pre-trained predictive model, which we require to explain using counterfactuals. We assume that the prediction model *accurately* predicts the outcome of a process at any step[1]. The second component implements the evolutionary algorithm, which generates counterfactual sequences that should be of high quality. To quantify the quality of counterfactual sequences and select the best ones, we define a *viability* measure as our third component, which takes four measures into account, namely (1) feasibility of a counterfactual sequence, (2) the delta flipped in the outcome prediction, (3) the similarity between factual and counterfactual, and (4) the sparsity counting the number of changes. As we use evolutionary algorithms to generate our counterfactuals, we refer to

[1] The accuracy-condition is favorable, but not necessary. If the component is accurately modelling the real world, we can draw real-world conclusions from the explanations generated. If the component is inaccurate, the counterfactuals only explain the prediction decisions and not the real world.

this framework as CREATED: the **C**ounte**R**factual Sequence generation with **E**volutionary **A**lgori**T**hms on **E**vent **D**ata. The name reflects how our model CREATEs new counterfactual sequences.

To evaluate the CREATED framework, we used ten event logs from three real-life processes and performed two experiments. First, we examined 54 configurations of the CREATED framework to obtain optimal configurations and compared our results with three baseline methods (case-based, sample-based, and random). The results show that we outperform the baseline methods in viability. In the second experiment, we compared our counterfactual sequences to the ones generated by a state-of-the-art method, showing that we yield similar counterfactuals without requiring domain knowledge.

The remainder of the paper is structured as follows. Sections 2 and 3 respectively discuss the related work and preliminary concepts. Section 4 presents our approach. Section 5 explains the evaluation set-up. Section 6 discusses the results, and Sect. 7 concludes the paper.

2 Related Work

As stated before, We divide the existing methods for counterfactual generation into two categories: *traditional* methods and *process-aware* methods. The *traditional* methods concern the classical ML models, and the topic of counterfactual generation as an explanation method was first introduced by [22]. The authors defined a loss function that incorporates the criteria to generate a counterfactual that maximizes the likelihood of a predefined outcome and minimizes the distance to the original instance. A more recent approach by [4] incorporates four main criteria for counterfactuals by applying a genetic algorithm with a multi-objective fitness function [4]. This approach strongly differs from gradient-based methods, as it does not require a differentiable objective function. However, the above traditional methods focus on static data. They do not take process behaviors into account. Applying these methods directly on event logs may result in generating infeasible counterfactual sequences.

Within process mining, the *process-aware* methods for counterfactuals have followed two streams. The first steam uses the Causal Inference techniques to analyse and model business processes, as the causal relationships can be used to understand the effect of decisions in a process on its outcome. However, early work has often attempted to incorporate domain-knowledge about the causality of processes in order to improve the process model itself [2,7,19,23]. Among these, the approach in [16] is one of the first to include counterfactual reasoning for process optimization [16]. Later, the work by [17] uses counterfactuals to generate alternative solutions to treatments, which lead to a desired outcome. However, the authors do not attempt to provide an explanation of the model's outcome and therefore, disregard multiple viability criteria for counterfactuals in XAI. [18] published the most recent paper on the counterfactual generation of explanations. The authors use a known Structural Causal Model (SCM) to guide the generation of their counterfactuals. However, this approach requires a process model which is as close as possible to the *true* process model. Our approach assumes no knowledge of such a normative process model.

The second stream in *process-aware* methods adapts the *traditional* counterfactual methods for process-aware counterfactuals. The DICE4EL approach [8] extends the DICE method [15] to generate counterfactuals for event logs while building on the same notion of incremental generation. The authors recognised that some processes have critical events (mile-stones) which govern the overall outcome. Hence, by simply avoiding the undesired outcome from critical event to critical event, it is possible to limit the search space and compute viable counterfactuals. However, their approach requires concrete domain knowledge about these critical points. We propose a framework that avoids this constraint and does not require domain knowledge. The LORELEY approach [9] extends the LORE method [5] and also uses an evolutionary algorithm. However, this approach focuses on mutating the case/event attributes. More specifically, the approach treats the encoded features representing the control flow as a single attribute in the crossover and mutation steps; thus, no unseen counterfactual sequences are created. In contrast, we generate unseen process sequences. Furthermore, we propose to automatically train a Markov model from the input event log to capture the likelihood of a process sequence. This Markov model is then used to derive the feasibility of counterfactual sequences.

3 Background

We start by formalising the event log and its elements.

Definition 1. Case, Event and Log. Let $\mathcal{E}$ be the universe of the event identifiers and $E \subseteq \mathcal{E}$ a set of events. An event log $L \subseteq \mathcal{E}^*$ is a set of sequences of events. Let C be a set of case identifiers and $\pi_c : E \mapsto C$ a surjective function that links every element in E to a case $c \in C$ in which c signifies a specific case. For a set of events $E \subseteq \mathcal{E}$, the shorthand s^c denotes a particular sequence $s^c = \langle e_1, e_2, \ldots, e_t \rangle$ with c as case identifier and a length of t. Each s is a trace of the process log $s \in L$. Let $\mathcal{T}$ be the time domain and $\pi_t : E \mapsto \mathcal{T}$ a non-surjective linking function which strictly orders a set of events. Each event e_t consists of a set $e_t = \{a_1 \in A_1, a_2 \in A_2, \ldots, a_I \in A_I\}$ with the size $I = |A|$, in which A_i is an attribute and a_i represents a possible value of that attribute.

Definition 2. Attribute Representation. Let $\pi_d : A_i \mapsto \mathbb{N}$ be a surjective function, which determines the dimensionality of a_i, and let F be a set of size I containing a representation function for every attribute. Let $f_i \in F$ be mapping functions to a vector space $f_i : a_i \mapsto \mathbb{R}_i^d$, in which d represents the dimensionality of an attribute value $d = \pi_d(A_i)$. We denote any event $e_t \in s^c$ of a specific case c as a vector, which concatenates every attribute representation f_i as $\mathbf{e}_t^c = [f_1; f_2; \ldots; f_I]$. Therefore, $\mathbf{e}_t^c$ is embedded in a vector space of size D which is the sum of each individual attribute dimension $D = \sum_i \pi_d(A_i)$. In other words, we concatenate all representations, whether they are scalars or vectors to one final vector representing the event. Furthermore, if we refer to a specific attribute A_i, we use the shorthand $\overline{a}_i$.

4 Methods

4.1 Methodological Framework: CREATED

To generate counterfactuals, we need to establish a conceptual framework consisting of three main components. The three components are shown in Fig. 1.

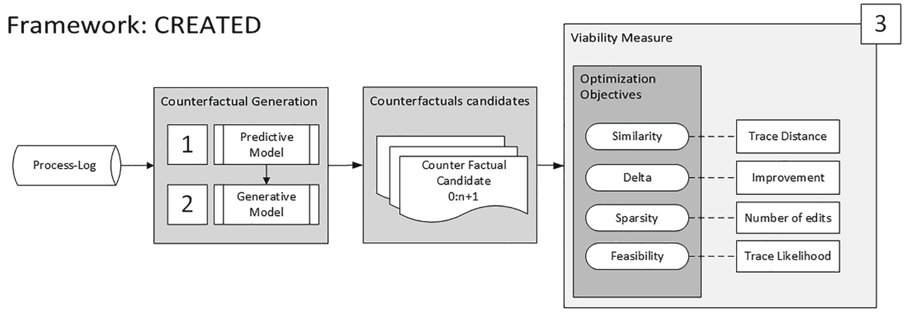

Fig. 1. The CREATED framework: the input is the process log; the log is used to train a predictive model (Component 1) and the generative model (Component 2). This process produces a set of candidates which are subject to evaluation via the validity metric (Component 3).

The first component is a *predictive model*. As we attempt to explain model decisions with counterfactuals, the predictive model needs to be pretrained. We can use any model that can predict the probability of a sequence. The prediction model in this paper is a simple LSTM model using the process log as an input. The architecture is inspired by [8]. The model is trained to predict the outcome given a sequence.

The second component is a *generative model*. The generative model produces counterfactuals given a factual sequence. We implement an evolutionary generator that takes a factual as input and yields counterfactuals candidates as output.

The generated candidates are subject to the third major component. To select the most *viable* counterfactual candidate, we evaluate their viability score using a custom metric. The metric incorporates four criteria for viable counterfactuals. We measure the **similarity** between two sequences using a multivariate sequence distance metric. The **outcome-delta** is the difference between the likelihood of the factual and the counterfactual. For this purpose, we require the predictive model, which computes a prediction score reflecting the likelihood. We measure **sparsity** by counting the number of changes in the features and computing the edit distance. Lastly, we need to determine the **feasibility** of a counterfactual. We measure the feasibility by estimating the probability of a counterfactual. Note that our method was developed for outcome prediction but can be adapted to the next activity prediction task.

4.2 Counterfactual Generators

Generative Model: Evolutionary Algorithm. In this section, we describe the concrete set of operators and select a subset that we want to explore further.

For our purposes, the *gene* of a sequence consists of the sequence of events within a process instance. Hence, if an offspring inherits one parent gene, it inherits the activity associated with the event and its event attributes. Our goal is to generate candidates by evaluating the sequence based on our viability measure. Our measure acts as the fitness function. The candidates that are deemed fit enough are subsequently selected to reproduce offspring. This process is explained in Fig. 2.

$$\underset{\text{Genes passed on}}{\overset{\text{Parent 1}}{\begin{pmatrix} a & b & a \\ 0.6 & 0.25 & 0.70 \\ 0 & 0 & 1 \\ 1.2 & 4.5 & 2.3 \end{pmatrix}}} + \underset{\text{Genes passed on}}{\overset{\text{Parent 2}}{\begin{pmatrix} a & b & a & c \\ 0.6 & 0.75 & 0.64 & 0.57 \\ 0 & 0 & 1 & 0 \\ 1.2 & 4.5 & 3.3 & 3.0 \end{pmatrix}}} = \underset{\text{Inherited} \quad \text{Inherited}}{\overset{\text{Offspring}}{\begin{pmatrix} a & b & a & c \\ 0.6 & 0.25 & 0.64 & 0.57 \\ 0 & 0 & 1 & 0 \\ 1.2 & 4.5 & 3.3 & 3.0 \end{pmatrix}}}$$

Fig. 2. A newly generated offspring inheriting genes in the form of activities and event attributes from both parents.

The offspring is subject to mutations. We evaluate the new population and repeat the procedure until a termination condition is reached. We can optimise the viability measure established in Sect. 4.3.

Operators. We implemented several different evolutionary operators. Each one belongs to one of five categories. The categories are initiation, selection, crossing, mutation, and recombination. Table 1 contains a complete list of the operators.

Naming-Conventions. We use abbreviations to refer to each model configuration. For instance, *CBI-RWS-OPC-RM-RR* refers to an evolutionary operator

Algorithm 1. The basic structure of an evolutionary algorithm.

Require: factual, configuration, sample-size, population-size, mutation-rate, termination-point
Ensure: The result is the final counterfactual sequences
  $counterfactuals \leftarrow initialize(\text{factual})$
  while not *termination* **do**
    cf-parents $\leftarrow$ *select*(counterfactuals, sample-size)
    cf-offsprings $\leftarrow$ *crossover*(cf-parents)
    cf-mutants $\leftarrow$ *mutate*(cf-offsprings, mutation-rate)
    cf-survivors $\leftarrow$ *recombine*(counterfactuals, cf-mutants, population-size)
    termination $\leftarrow$ *determine*(cf-survivors, termination-point)
    counterfactuals $\leftarrow$ cf-survivors
  end while

Table 1. An overview of all evolutionary operators used in this paper and a short description.

Label	Name	Description
Initiation		
RI	Random Initialisation	Generates an initial population in which the event sequence was chosen at random based on the log. The event attributes were drawn from a normal distribution
SBI	Sampling-Based Initialisation	Generates an initial population by sampling from a data distribution estimated from the data directly. The event sequence was sampled using the event transition probabilities. The attributes were sampled using distributions conditioned on the emitted events
CBI	Case-Based Initialisation	Samples initial population directly from the Log
Selection		
RWS	Roulette-Wheel-Selection	Selects individuals randomly in proportion to their fitness value
TS	Tournament-Selection	Selects pairs of individuals and compares each pair. The better individual between both pairs has a higher chance of being selected
ES	Elitism-Selection	Selects individual with the highest fitness
Crossover		
UCx	Uniform Crossover	Uniformly choose a fraction of genes of one individual (*Parent 1*) and overwrite the respective genes of another individual (*Parent 2*)
OPC	One-Point Crossover	Chooses a point in the sequence and overwrites the genes of *Parent 2* by the genes *Parent 2* from that point onward
TPC	Two-Point Crossover	Chooses two points in the sequence and overwrites the sequence in between the two points from *Parent 2* with the sequence from *Parent 1*
Mutation		
RM	Random-Mutation	Inserts, changes or deletes activities randomly. Event attributes are drawn from a normal distribution
SBM	Sampling-Based Mutation	Inserts, changes or deletes activities randomly. Event attributes are drawn from an estimated data distribution
Recombination		
FSR	Fittest-Survivor Recombination	Strictly determines the survivors among the mutated offsprings and the current population by sorting them in terms of viability
BBR	Best-of-Breed Recombination	Determines offsprings that are better than the average within their generation and adds them to survivors of past generations
RR	Ranked Recombination	Selects the new population differently than the former recombination operators. Instead of using the viability directly, we sort each individuum by every viability component separately. This approach allows us to select individuals regardless of the scales of every individual viability measure

configuration that samples its initial population from the data (CBI), probabilistically samples parents based on their fitness (RWS), crosses them on one point (OPC), and so on. For the *Uniform-Crossing* (UCx) operator, we additionally indicate its crossing rate using a number. For instance, *CBI-RWS-UC3-RM-RR* uses the *Uniform-Crossing* (UC3) operator. The child receives roughly 30% of the genome of one parent and 70% of another parent.

Hyperparameters. The evolutionary approach comes with a number of hyperparameters. We first discuss the *model configuration*. As shown in this section, there are a 135 ways to combine all operators. Depending on each operator combination, we might see very different behaviours. The decision of the appropriate set of operators is by far the most important in terms of convergence speed and result quality. The next hyperparameter is the *termination point* which determines the duration of the search. Optimally, we find a termination point, which is not too early but not too late, too. The *mutation rate* is another hyperparameter. It signifies how much a child can differ from its parent.

4.3 Viability Measure

Feasibility-Measure. To determine the feasibility of a counterfactual trace, it is important to consider two aspects. First, we have to compute the probability of the sequence of event transitions. This is a difficult task, given the *Open World assumption*[2]. Therefore, we have to assume the data is representative and the underlying process is static. This assumption allows us to estimate first-order transition probabilities by counting event transitions.

Second, we have to compute the feasibility of the individual feature values given the sequence. We can relax the computation of this probability using the *Markov Assumption*. In other words, we assume that each event vector depends on the current activity but on none of the previous events and features. This means that we can model density estimators for every event and use them to determine the likelihood of a set of features.

We define the feasibility measure in Eq. 1, where e_t represents the current event, transited from the previous event e_{t-1}. Likewise, f represents the emission of the feature attributes. Hence, the probability of a particular sequence is the product of the transition probability multiplied by the state emission probability for each step.

$$p\left(e_{0:T}, f_{0:T}\right) = p\left(e_0\right) p\left(f_0 \mid e_0\right) \prod_1^T p\left(e_t \mid e_{t-1}\right) p\left(f_t \mid e_t\right) \tag{1}$$

Delta-Outcome. For the delta measure, we evaluate the likelihood of a counterfactual trace by determining whether a counterfactual leads to the desired outcome or not. For this purpose, we use the predictive model, which returns a prediction for each counterfactual sequence. As we are predicting process outcomes, we typically predict a class. However, forcing a deterministic model to produce a different class prediction is often difficult. Therefore, we can relax the condition by maximising the prediction score of the desired counterfactual outcome [14]. If we compare the difference between the counterfactual prediction score with the factual prediction score, we can determine an increase or decrease. Ideally, we want to increase the likelihood of the desired outcome. We refer to this value as *delta*. For the binary outcome prediction case, we define the function as shown in Eq. 2.

$$delta = \begin{cases} |p(o|s^*) - p(o|s)| & \text{if } p(o|s) > 0.5 \& p(o|s) > p(o|s^*) \\ -|p(o|s^*) - p(o|s)| & \text{if } p(o|s) > 0.5 \& p(o|s) \leq p(o|s^*) \\ |p(o|s^*) - p(o|s)| & \text{if } p(o|s) \leq 0.5 \& p(o|s) > p(o|s^*) \\ -|p(o|s^*) - p(o|s)| & \text{if } p(o|s) \leq 0.5 \& p(o|s) \leq p(o|s^*) \end{cases} \tag{2}$$

Similarity Measure. We use a function to compute the **similarity** between the factual sequence and the counterfactual candidates. To incorporate differences in length between both sequences, we use a weighted version of the

[2] In theory, we cannot know whether or not any event *can* follow after another event.

Table 2. All datasets used within the evaluation. DiCE4EL is used for the qualitative evaluation, and the remaining are used for quantitative evaluation purposes.

Dataset	#Cases	Min Len	Max Len	% Unique Traces	#Unique Ev	#Data Columns	#Event Attr	#Regular	#Deviant
DiCE4EL	3 051	12	25	0.000328	23	9	7	1 853	1 198
BPIC12-25	3 051	12	25	0.000328	23	23	21	1 853	1 198
BPIC12-50	4 587	12	50	0.000218	23	23	21	2 405	2 182
BPIC12-75	4 677	12	75	0.000214	23	23	21	2 436	2 241
BPIC12-100	4 685	12	96	0.000213	23	23	21	2 442	2 243
Sepsis-25	707	5	25	0.001414	15	75	73	610	97
Sepsis-50	770	5	47	0.001299	15	76	74	662	108
Sepsis-75	777	5	66	0.001287	15	76	74	667	110
Sepsis-100	779	5	88	0.001284	15	76	74	669	110
TrafficFines	129 615	2	20	0.000008	10	40	38	70 602	59 013

Damerau-Levenshtein distance [3]. The Damerau-Levenstein distance applies a cost constant of 1 for each sequential difference. However, as process instances differ not only in event sequences but also in their event attribute values, we use a distance function to weigh the cost. In the case of **similarity**, we apply the euclidian distance. For formal definitions, we refer to [11, p. 42].

Sparsity Measure. For measuring the sparsity, we use the same weighted version of the Damerau-Levenshtein distance. However, to measure the distance, we count the number of differences between event attributes. For formal definitions, we refer to [11, p. 42].

Viability-Measure. We combine the feasibility measure, the outcome delta, the normalised sparsity, and normalised similarity measure by summation. As each measure can have values between 0 and 1, the viability measure ranges between 0 and 4. For more details on the viability measure, we refer to [11, Chap. 3.3].

5 Evaluation

5.1 Datasets

For our evaluation, we use ten event logs of three real-life processes, which were also used in [21]. Each dataset consists of events and contains labels that signify a process instance's outcome. We focus on binary outcome predictions. We include a variation of the BPIC dataset. This dataset was used in [8]. The difference between Hsieh et al.'s dataset and the original dataset is two-fold. First, the authors focus on the generation of two event attributes. Second, the dataset is primarily designed for next-activity prediction, not outcome prediction. We modified the dataset to fit the outcome prediction model. For more information about these datasets we refer to the comparative study by [21]. We list the important descriptive statistics in Table 2.

We list the predictions of our prediction component in Table 3. The F1-Scores on the test sets are generally higher for the BPIC dataset. Furthermore, in the case of the BPIC datasets, the prediction model always predicts the correct outcome if the max-length of the sequence exceeds 25. It is fair to assume that the length of a loan application process determines the chance of getting rejected or not.

Table 3. The evaluation metrics for the prediction component on all datasets. Includes precision, recall and f1 score for test, training and validation data.

Subset Dataset	Precision			Recall			f1-score			Support		
	Test	Training	Validation	Test	Training	Validation	Test	Training	Validation	Test	Training	Validation
BPIC12-100	1.000	0.999	0.999	1.000	0.999	0.999	1.000	0.999	0.999	60.000	1000.000	841.000
BPIC12-25	0.808	0.770	0.765	0.750	0.742	0.733	0.738	0.733	0.723	60.000	1000.000	1000.000
BPIC12-50	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	60.000	1000.000	819.000
BPIC12-75	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	60.000	1000.000	841.000
DiCE4EL	0.780	0.806	0.821	0.700	0.755	0.749	0.677	0.744	0.739	60.000	1000.000	1000.000
Sepsis-100	0.259	0.246	0.250	0.509	0.496	0.500	0.343	0.329	0.333	55.000	123.000	42.000
Sepsis-25	0.478	0.511	0.528	0.483	0.508	0.519	0.449	0.482	0.495	60.000	1000.000	873.000
Sepsis-50	0.250	0.240	0.261	0.500	0.490	0.511	0.333	0.322	0.346	60.000	1000.000	1000.000
Sepsis-75	0.207	0.254	0.300	0.455	0.504	0.548	0.284	0.338	0.388	55.000	123.000	42.000
TrafficFines	1.000	0.987	0.984	1.000	0.987	0.983	1.000	0.987	0.983	60.000	1000.000	1000.000

5.2 Preprocessing

To prepare the data for our experiments, we employed basic tactics for preprocessing. First, we split the log into a training and a test set. Then, we filter out every case whose sequence length exceeds 25. We keep this maximum threshold for most experiments focusing on the evolutionary algorithm. The reason is the polynomial computation time of the viability measure. The similarity and sparsity components of the proposed viability measure have a runtime complexity of at least N^2. Hence, limiting the sequence length saves a substantial amount of temporal resources. Next, we extract time variables if they are provided in the log. Then, we normalise the values. Each categorical variable is converted using binary encoding. The activity is label-encoded. As a result, every category is assigned to a unique integer. The label column is binary encoded, as we focus on outcome prediction. Lastly, we pad each sequence towards the longest sequence in the dataset.

5.3 Baseline Models

We use three baseline models and compare them to the evolutionary models. The first baseline generates a random sequence of events and event attributes. Hence, we refer to this approach as **Random baseline** (RGW). We expect most models to perform better than this baseline. Otherwise, it would indicate that a random search would generate better counterfactuals than a guided one. The second baseline resembles the random baseline. However, we use the data likelihood to guide the random search for the generation of counterfactuals. We first generate a random seed of possible starting events $(p\,(e_0))$. Afterwards, we randomly sample subsequent events by iteratively sampling new activities according to the transition probabilities we gathered from the data ($\prod_1^T p(e_t \mid e_{t-1})$). Given the sequence, we simply sample the features per event from $p(f_t \mid e_t)$. We call this baseline **Sample-Based** (SBGW). In contrast to both sampling-based baselines, the last baseline leverages actual examples of the data. We refer to this case-based approach as **Case-Based baseline** (CBGW). The idea is to randomly pick traces from the log and evaluate them using the viability measure.

5.4 Experimental Setup

All the experiments were run on a Windows machine with 12 processor cores (Intel Core i7-9750H CPU 2.60 GHz) and 32 GB Ram. The code is written in Python version 3.8. The models were developed with Tensorflow [1] and NumPy [6]. We provide the full code and instructions on Github [10].

In terms of operators, we introduced three initiators, three selectors, five crossers, two mutators, and three recombiners. For the experiments, we exclude the random mutator as preliminary experiments showed that it often leads to results with a feasibility of 0. To reduce the number of model configurations, we initially compare all 135 evolutionary operator combinations. We select the best three models and compare them to the three baseline models. Afterwards, we assess the viability of all the chosen evolutionary and baseline generators. We sample 10 factuals from the BPIC-25 dataset and use our models as well as the baselines to generate 50 counterfactuals for each factual. We determine the mean viability across the counterfactuals. We expect the evolutionary algorithms to outperform the baselines when it comes to viability. In the end, we assess the quality of the generated counterfactuals. In line with [8], we aim to answer the question *what would one have had to change in order to flip the outcome of a process*. The goal is to show that the counterfactuals our models generate are viable without having to rely on domain-specific knowledge. In the current paper, we did not include any results of the individual viability components. Furthermore, we refer to [11, p.64] for more specific and extensive observations.

6 Results

6.1 Experiment 1: Comparing with Baseline Generators

We examined a set of model-configurations containing 135 elements. We choose to run each model configuration for 100 evolution cycles. We randomly sample four factual process instance from the test set. Afterwards, we use the average viability across the instances to evaluate all model configurations. Fig. 3 shows the bottom and top-5 model configurations based on the viability after the final iterative cycle. The figure also shows how the viability evolves for each iteration.

According to Fig. 3, *CBI-ES-UC3-SBM-RR*, *CBI-RWS-OPC-SBM-BBR*, and *CBI-RWS-OPC-SBM-FSR* are the best model configurations. As all best-performing model-configurations use the *Case-Based Inititiation*-operator, we identify it as the most important configuration. The results suggest that the initiation operator governs the starting point of the optimisation. For the following

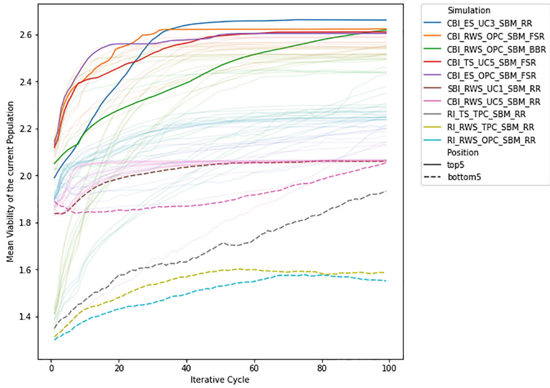

Fig. 3. This figure shows the average viability of the five best and worst model configurations. The x-axis shows how the viability evolves for each evolutionary cycle. The semi-transparent lines are the model configurations that are neither in the best five nor worst five groups. They show the general trend of the viability improvement.

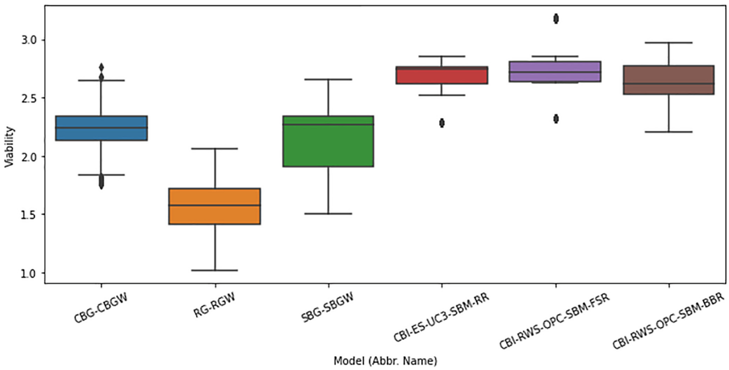

Fig. 4. This figure shows boxplots of the viability of each model's generated counterfactuals.

experiment, we ran each evolutionary algorithm for 200 iterative cycles and set the mutation rate to 0.01.

Next, we employed the baseline models mentioned in Sect. 4.2 and examined their results across all datasets. We randomly sampled 20 factuals from the test set and used the same factuals for every generator. We ensured that the outcomes are evenly divided. The remaining procedure followed the established practice of previous experiments. The results in Fig. 4 show that the evolutionary algorithm *CBI-ES-UC3-SBM-RR* returns better results when it comes to the mean viability. The worst model is the randomly generated model. The Case-Based model appears to be evenly and normally distributed at a viability of *2.25*. The *CBI-RWS-OPC-SBM-FSR* has outliers that far exceed and underperform against other evolutionary algorithms on both ends.

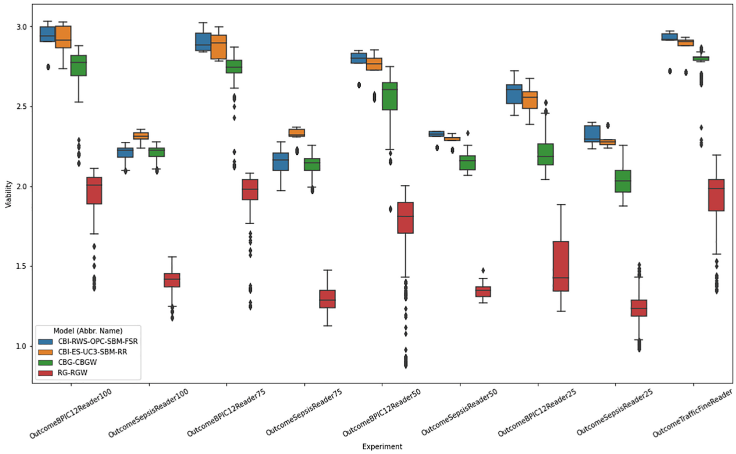

Fig. 5. Boxplots of the viability of each model's generated counterfactuals across a heterogeneous collection of datasets.

Figure 5 displays the results of running each algorithm on a set of different datasets. The figure shows a clear dominance of the evolutionary models across all datasets. Here, *CBI-ES-UC3-SBM-RR* and *CBI-RWS-OPC-SBM-FSR* display a higher median of viability across all datasets. This is unsurprising as the evolutionary algorithm uses initiators based on the baselines. However, it is surprising that the evolutionary models consistently outperform the Casebased-Search Generator (green) across all datasets. In six out of nine datasets, we see an improvement of at least 0.15. The highest median is reached for *CBI-RWS-OPC-SBM-FSR* at 2.94. The Random-Search Generator never manages to come even close to the case-based model. Except for the BPIC12-100 dataset, the Random-Search Generator has a median below 2.

The results for Fig. 5 show that both evolutionary algorithms outperform the competition across *all* datasets and against *all* baselines. This result shows that the algorithm can outperform baselines regardless of the process log and its length. The baseline comparison also shows that we can optimise towards viability successfully. Recall that we defined four criteria for the viability of counterfactuals (similarity, sparsity, feasibility, and delta in likelihood); a model optimising towards those criteria can apparently return superior results.

Table 4. A comparison between the CBI-RWS-OPC-SBM-FSR and D4EL.

Factual Seq.				Our CF Seq.				DiCE4EL CF Seq.		
Amount	Activity	Outcome	Resource	Amount	Activity	Outcome	Resource	Activity	Resource	Amount
5 000	A-SUBMITTED	0	112	7 000	A-SUBMITTED	1	112			
5 000	A-PARTLYSUBMITTED	0	112	7 000	A-PARTLYSUBMITTED	1	112			
5 000	A-PREACCEPTED	0	101	7 000	A-PREACCEPTED	1	112			
5 000	W-Afhandelen leads	0	101					A-SUBMITTED	112	5 000
5 000	A-ACCEPTED	0	111					A-PARTLYSUBMITTED	112	5 000
5 000	O-SELECTED	0	111	7 000	A-ACCEPTED	1	111	A-PREACCEPTED	112	5 000
5 000	A-FINALIZED	0	111	7 000	O-SELECTED	1	111	A-ACCEPTED	1	5 000
5 000	O-CREATED	0	111	7 000	A-FINALIZED	1	111	O-SELECTED	1	5 000
5 000	O-SENT	0	111	7 000	O-CREATED	1	111	A-FINALIZED	1	5 000
5 000	W-Completeren aanvraag	0	111	7 000	O-SENT	1	111	O-CREATED	1	5 000
5 000	W-Nabellen offertes	0	111	7 000	W-Completeren aanvraag	1	111	O-SENT	1	5 000
5 000	O-CANCELLED	0	111					W-Completeren aanvraag	1	5 000
5 000	A-CANCELLED	0	111	7 000	W-Nabellen offertes	1	111	O-SENT-BACK	11259	5 000
5 000	W-Nabellen offertes	0	111	7 000	W-Nabellen offertes	1	111	W-Nabellen offertes	11259	5 000
				7 000	O-ACCEPTED	1	629	O-ACCEPTED	9	5 000

6.2 Experiment 2: Qualitative Assessment

Figure 4 shows the generation of the model-configuration *CBI-RWS-OPC-SBM-FSR* and the model of [8]. Both models also return reasonable counterfactuals. The counterfactual sequence of events of both approaches are almost identical. For instance, our counterfactual and the D4EL counterfactual recognize that after O-SENT, there appears at least one *W-Completeren aanvraag* and one *W-Nabellen offertes* that eventually leads to an acceptance of the counterfactual. We also see that both evolutionary algorithms start the process with the correct sequence of A-SUBMITTED, A-PARTLYSUBMITTED and A-PREACCEPTED. These are strictly the same across all cases. If our generative model had not recognised these, one could question its utility.

In Table 5 we applied the same approach on a different dataset. The generator generates a counterfactual that is close to the original factual and only modifies the number of open cases. Here, we can conclude that a sudden increase in open cases during the *Add penalty* step results in a change of outcome.

The examples show that our generative approach does not rely on domain knowledge, such as milestones. In contrast, the approach by [8] only applies to datasets with clear milestones such as *BPIC-12*.

Table 5. A counterfactual for the Traffic-Fines dataset by the CBI-RWS-OPC-SBM-FSR model.

Factual Seq.				Our CF Seq.			
Open Cases	Activity	Outcome	Resource	Open Cases	Activity	Outcome	Resource
16 318	Create Fine	1	537	15 742	Create Fine	0	537
16 612	Send Fine	1	537	16 504	Send Fine	0	537
16 693	Insert Fine Notification	1	537	16 693	Insert Fine Notification	0	537
16 972	Add penalty	1	537				

6.3 Discussion and Limitations

All models successfully flip the outcome of the prediction model and are close to the factual. In contrast, the model by [8] proposes more changes to the sequence. It is important to recall that the generated counterfactuals focus on explaining the prediction model rather than the true process. More specifically, our generative model shows which events and attributes have to be present or omitted to flip the outcome of the prediction model.

In contrast to [8], we show that we can create these counterfactuals without incorporating domain-specific knowledge, such as an understanding of milestone patterns. Domain knowledge can help to improve or evaluate our solutions. However, they are not strictly required. Furthermore, our models can generate sequences not present within the input event log. Case-based solutions often overlook this aspect, as they are heavily biased toward the input data.

It is worthwhile to discuss that *counterfactual sequences* differ from *counterfactual rules* or *explanations*. To obtain explicit explanations or rules, the generated counterfactuals should be compared to the factual. Our framework enables some alignments between the generated counterfactuals with the factual sequence (see Fig. 4), which may act as an explanation. We consider deriving rules as a post-prior analysis, which is interesting for future work.

Our viability components showed that they can lead to an optimised solution. However, there are most likely other ways to operationalise viability criteria. In addition, what makes an excellent counterfactual and how we can quantify that is still a subject of debate. Currently, there is a lack of standardized evaluation protocols, benchmark techniques, and datasets. As a result, many researchers fall back on defining their custom evaluation methods. In fact, this is still an open research question [8,15]. Therefore, we often have to evaluate the counterfactuals in some subjective and qualitative way. In this paper, we decided to compare the counterfactuals with another approach in the literature and the factual themselves. Because our counterfactuals produced reasonable results, we deemed them viable. As future work, we also see value in incorporating experts to evaluate such an approach.

7 Conclusion

In this paper, we proposed CREATED, a modular framework to generate viable counterfactuals. The framework incorporates an evolutionary algorithm to generate counterfactual sequences while not requiring any domain knowledge other than the log itself. In addition, we proposed a viability measure to quantify and assess the quality of counterfactual sequences when compared to a factual sequence. The viability measure takes four aspects into account: feasibility, the delta in flipping the outcome prediction, similarity, and sparsity. The approach is capable of generating counterfactuals without explicit knowledge about the domain, as we only require the log. We achieve this by incorporating a Markov model trained on the event log. Our evaluation shows that our framework can generate counterfactual sequences which are higher than our naive baselines (i.e., case-based, sample-based, and random baselines). With these results, we demonstrate that optimizing a viability measure does generate higher-quality counterfactuals. We also compared the generated counterfactuals to the state-of-the-art method in the literature and show that our framework can generate similar counterfactuals, without using domain knowledge. The current feasibility measure tends to return lower values than other viability components as it is very sensitive to trace length. In the future, we aim to investigate better feasibility measures.

References

1. Abadi, M., et al.: Tensorflow: a system for large-scale machine learning. In: OSDI. pp. 265–283. USENIX Association (2016)
2. Baker, J., Song, J., Jones, D.R.: Closing the loop: empirical evidence for a positive feedback model of IT business value creation. J. Strateg. Inf. Syst. **26**(2), 142–160 (2017)
3. Damerau, F.: A technique for computer detection and correction of spelling errors. Commun. ACM **7**(3), 171–176 (1964)
4. Dandl, S., Molnar, C., Binder, M., Bischl, B.: Multi-objective counterfactual explanations. In: Bäck, T., Preuss, M., Deutz, A., Wang, H., Doerr, C., Emmerich, M., Trautmann, H. (eds.) PPSN 2020. LNCS, vol. 12269, pp. 448–469. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58112-1_31
5. Guidotti, R., Monreale, A., Giannotti, F., Pedreschi, D., Ruggieri, S., Turini, F.: Factual and counterfactual explanations for black box decision making. IEEE Intell. Syst. **34**(6), 14–23 (2019)
6. Harris, C.R., et al.: Array programming with numpy. Nat. **585**, 357–362 (2020)
7. Hompes, B.F.A., Maaradji, A., La Rosa, M., Dumas, M., Buijs, J.C.A.M., van der Aalst, W.M.P.: Discovering causal factors explaining business process performance variation. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 177–192. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_12
8. Hsieh, C., Moreira, C., Ouyang, C.: Dice4el: Interpreting process predictions using a milestone-aware counterfactual approach. In: ICPM. pp. 88–95. IEEE (2021)
9. Huang, T., Metzger, A., Pohl, K.: Counterfactual explanations for predictive business process monitoring. In: EMCIS. Lecture Notes in Business Information Processing, vol. 437, pp. 399–413. Springer (2021)
10. Hundogan, O.: CREATED, https://github.com/Olu93/project_CREATED/blob/a376a41ac51018c43af29a5add7aed6504a37277/README.md
11. Hundogan, O.: CREATED: the generation of viable counterfactual sequences using an evolutionary algorithm for event data of complex processes. Master's thesis, Utrecht University (2022). https://studenttheses.uu.nl/handle/20.500.12932/43117
12. Klímek, J., Klimek, J., Kraskiewicz, W., Topolewski, M.: Long-term series forecasting with query selector - efficient model of sparse attention. CoRR abs/2107.08687 (2021)
13. Mannhardt, F., Blinde, D.: Analyzing the trajectories of patients with sepsis using process mining. In: RADAR+EMISA@CAiSE. CEUR Workshop Proceedings, vol. 1859, pp. 72–80. CEUR-WS.org (2017)
14. Molnar, C.: Interpretable machine learning. Lulu.com (2020), https://christophm.github.io/interpretable-ml-book/

15. Mothilal, R.K., Sharma, A., Tan, C.: Explaining machine learning classifiers through diverse counterfactual explanations. In: FAT*, pp. 607–617. ACM (2020)
16. Narendra, T., Agarwal, P., Gupta, M., Dechu, S.: Counterfactual reasoning for process optimization using structural causal models. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNBIP, vol. 360, pp. 91–106. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26643-1_6
17. Oberst, M., Sontag, D.A.: Counterfactual off-policy evaluation with gumbel-max structural causal models. In: ICML. Proceedings of Machine Learning Research, vol. 97, pp. 4881–4890. PMLR (2019)
18. Qafari, M.S., van der Aalst, W.M.P.: Case level counterfactual reasoning in process mining. In: Nurcan, S., Korthaus, A. (eds.) CAiSE 2021. LNBIP, vol. 424, pp. 55–63. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79108-7_7
19. Shook, C.L., Ketchen, D.J., Jr., Hult, G.T.M., Kacmar, K.M.: An assessment of the use of structural equation modeling in strategic management research. Strateg. Manag. J. **25**(4), 397–404 (2004)
20. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with lstm neural networks. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 477–492. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_30
21. Teinemaa, I., Dumas, M., Rosa, M.L., Maggi, F.M.: Outcome-oriented predictive process monitoring: Review and benchmark. ACM Trans. Knowl. Discov. Data **13**(2), 17:1–17:57 (2019)
22. Wachter, S., Mittelstadt, B., Russell, C.: Counterfactual explanations without opening the black box: Automated decisions and the GDPR. Harv. JL Tech. **31**, 841 (2017)
23. Wang, Z., Zhang, J., Xu, H., Chen, X., Zhang, Y., Zhao, W.X., Wen, J.: Counterfactual data-augmented sequential recommendation. In: SIGIR, pp. 347–356. ACM (2021)