Contribution Title*

First Author[1][0000−1111−2222−3333], Second Author[2,3][1111−2222−3333−4444], and Third Author[3][2222−−3333−4444−5555]

[1] Princeton University, Princeton NJ 08544, USA
[2] Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany
lncs@springer.com
http://www.springer.com/gp/computer-science/lncs
[3] ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany
{abc,lncs}@uni-heidelberg.de

Abstract The abstract should briefly summarize the contents of the paper in 15–250 words.

Keywords: First keyword · Second keyword · Another keyword.

1 Introduction

1.1 Motivation

Many processes, often medical, economical, or administrative in nature, are governed by sequential events and their contextual environment. Many of these events and their order of appearance play a crucial part in the determination of every possible outcome**?**. With the rise of AI and the increased abundance of data in recent years, several techniques emerged that help to predict the outcomes of complex processes in the real world. A field that focuses on modelling processes is Process Mining (PM).

Research in the Process Mining discipline has shown that it is possible to predict the outcome of a particular process fairly well**??**.

For instance, in the medical domain, models have been shown to predict the outcome or trajectory of a patient's condition**?**. In the private sector, process models can be used to detect faults or outliers. The research discipline Deep Learning has shown promising results within domains that have been considered difficult for decades. The Moravex Paradox**?**, which postulates that machines are capable of doing complex computations easily while failing in tasks that seem easy to humans such as object detection or language comprehension, does not hold anymore. Meaning that with enough data to learn, machines are capable of learning highly sophisticated tasks better than any human. The same holds for predictive tasks. However, while many prediction models can predict certain outcomes, it remains a difficult challenge to understand their reasoning.

This difficulty arises from models, like neural networks, that are so-called *blackbox models*. Meaning, that their inference is incomprehensible, due to the

* Supported by organization x.

vast amount of parameters involved. This lack of comprehension is undesirable for many fields like IT or finance. Not knowing why a loan was given, makes it impossible to rule out possible biases. Knowing what will lead to a system failure will help us knowing how to avoid it. In critical domains like medicine, the reasoning behind decisions becomes crucial. For instance, if we know that a treatment process of a patient reduces the chances for survival, we want to know which treatment step is the critical factor we ought to avoid. To summarise, knowing the outcome of a process often leads us to questions on how to change it. Formally, we want to change the outcome of a process instance by making it maximally likely with as little interventions as possible**?**. Figure 1 is a visual representation of the desired goal.

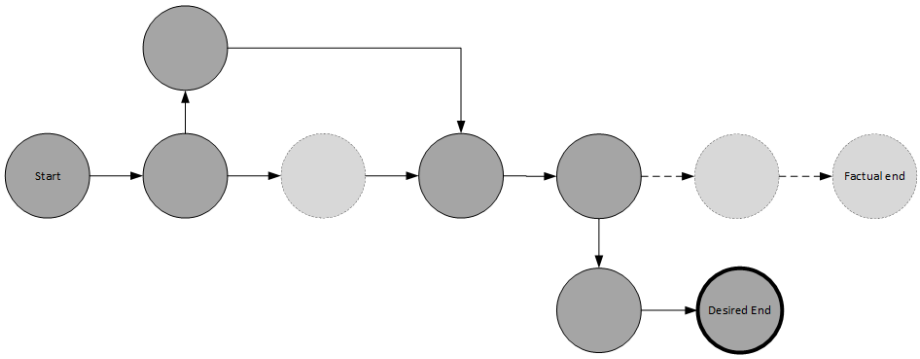


Figure 1: This figure illustrates a model, that predicts a certain trajectory of the process. However, we want to change the process steps in such a way, that it changes the outcome.

One way to better understand the Machine Learning (ML) models lies within the eXplainable AI (XAI) discipline. XAI focuses the developments of theories, methods, and techniques that help explaining blackbox models models to humans. Most of the discipline's techniques produce explanations that guide our understanding. Explanations can come in various forms, such as IF-THEN rules(**?**, p.90) or feature importance(**?**, p.45), but some are more comprehensible for humans than others.

A prominent and human-friendly approach are *counterfactuals*(**?**, p. 221). Counterfactuals within the AI framework help us to answer hypothetical "what-if" questions. Basically, if we know *what* would happen *if* we changed the execution of a process instance, we could change it for the better. In this thesis, we raise the question how we can use counterfactuals to change the trajectory of a process models' prediction towards a desired outcome. Knowing the answers not only increases the understanding of blackbox models, but also help us avoid or enforce certain outcomes.

1.2 Problem Space

In this thesis, we approach the problem of generating counterfactuals for processes. The literature has provided a multitude of techniques to generate counterfactuals for AI models, that are derived from static data[4]. However, little research has focussed on counterfactuals for dynamic data[5].

For process data, the literature often uses terms like structured and semi-structured, as they are related to the staticity and dynamicity. Both, structuredness and semi-structuredness, often relate to the data model, in which we structure the information at hand. As static data neither changes over time nor changes its structure, we can use structured data-formats such as tables to capture the information where each data point is an independent entity. We can take the MNIST dataset**?** or Iris dataset**??** as examples for structured and static data. In both datasets, all data points are independent and have the same amount of attributes. In contrast, semi-structured data does not have to follow these strict characteristics. Here, data points often belong to a group of data points which constitutes the full entity. Furthermore, the attributes of each data point may vary. The grouping mechanism could take the form of associative links, class associations or temporal cause-effect relationships. Examples of these are Part-of-Speech datasets like Penn Treebank set**?**. Here, we often associate each data point with a sentence. However, the temporal relationship between words is debatable and hence, whether the data is *dynamic*, as well. So, not all semi-structured datasets are dynamic and vice versa. However, structured data will almost always be static, with the exception of time-series. Lastly, there is also unstructured data, which does not incorporate any specific data model. Corpora like the Brown dataset**?**, for instance, are collections of text heavy unstructured information. In Figure 2, we show various examples of data.

A major reason, why there has not been much research on counterfactuals for dynamic semi-structured data, emerges from a multitude of challenges, when dealing with counterfactuals and sequences. Three of these challenges are particularly important.

First, counterfactuals within AI attempt to explain outcomes which never occured. *What-if* questions often refer to hypothetical scenarios. Therefore, there is no evidential data from which we can infer predictions. Subsequently, this lack of evidence further complicates the evaluation of generated counterfactuals. In other words, you cannot validate the correctness of a theoretical outcome that has never occured.

Second, sequential data is highly variable in length, but process steps have complicated factors, too. The sequential nature of the data impedes the tractability of many problems due to the combinatorial explosion of possible sequences. Furthermore, the data generated is seldomly one-dimensional or discrete. Henceforth, each dimension's contribution can vary in dependance of its context, time and magnitude.

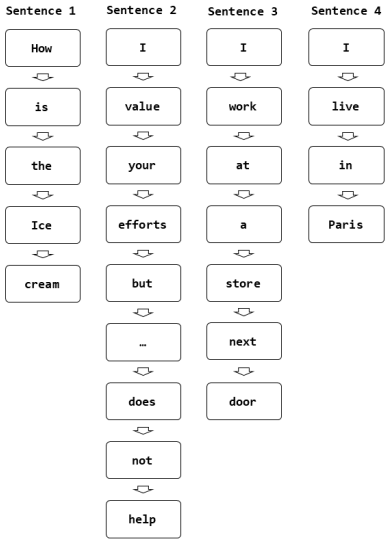
[4] With static data, we refer to data that does not change over a time dimension.

[5] With dynamic data, we refer to data that has a temporal relationship as a major component, which is also inherently sequential

s.length	s.width	p.length	p.width	variety
6.5	2.8	4.6	1.5	Versicolor
5.8	2.7	4.1	1.0	Versicolor
6.7	3.3	5.7	2.5	Virginica
4.6	3.4	1.4	0.3	Setosa
6.4	3.2	5.3	2.3	Virginica
5.9	3.0	4.2	1.5	Versicolor
7.4	2.8	6.1	1.9	Virginica
5.5	2.4	3.8	1.1	Versicolor
5.6	2.5	3.9	1.1	Versicolor
5.0	3.4	1.5	0.2	Setosa
6.9	3.1	5.4	2.1	Virginica
5.5	2.5	4.0	1.3	Versicolor
5.7	2.6	3.5	1.0	Versicolor
5.8	2.7	3.9	1.2	Versicolor
7.6	3.0	6.6	2.1	Virginica
6.7	3.3	5.7	2.1	Virginica
5.0	3.5	1.6	0.6	Setosa
7.7	2.8	6.7	2.0	Virginica
6.4	2.7	5.3	1.9	Virginica
7.7	3.8	6.7	2.2	Virginica
5.2	3.5	1.5	0.2	Setosa
5.7	3.8	1.7	0.3	Setosa

(a) An excerpt of the MNIST dataset. This is a structured dataset.

(b) A number of heterogenous documents. A dataset like this is unstructured.



(c) Multiple seqeuences of words. Each word forms a sentence of different lengths. Therefore, this data is semi-structured.

Figure 2: Schematic examples of static structured, dynamic semi-structured data and unstructured data.

Third, process data often requires knowledge of the causal structures that produced the data in the first place. However, these structures are often hidden and it is a NP-hard problem to elicit them**?**.

These challenges make the field, in which we can contribute, a vast endeavor.

1.3 Related Literature

Many researchers have worked on counterfactuals and PM. Here, we combine the important concepts and discuss the various contributions to this thesis.

Generating Counterfactuals The topic of counterfactual generation as explanation method was introduced by **?** in **??**. The authors defined a loss function which incorporates the criteria to generate a counterfactual which maximizes the likelihood for a predefined outcome and minimizes the distance to the original instance. However, the solution of **?** did not account for the minimisation of feature changes and does not penalize unrealistic features. Furthermore, their solution cannot incorporate categorical variables.

A newer approach by **?** incorporates four main criteria for counterfactuals (see subsection 2.8) by applying a genetic algorithm with a multi-objective fitness function**?**. This approach strongly differs from gradient-based methods, as it does not require a differentiable objective function. However, their solution was only tested on static data.

Generating Counterfactual Sequences When it comes to sequential data most researchers work on ways to generate counterfactuals for natural language. This often entails generating univariate discrete counterfactuals with the use of Deep Learning techniques. **?** and later **?** are early examples of counterfactual NLP research**??**. Their approach strongly focuses on the manipulation of sentences to achieve the desired outcome. However, as **?** puts it, their counterfactuals do not comply with *realisticness***?**.

Instead, **?** showed that it is possible to generate realistic counterfactuals with a Generative Adversarial Model (GAN)**?**. They use the model to implicitly capture a latent state space and sample counterfactuals from it. Apart from implicitly modelling the latent space with GANs, it is possible to sample data from an explicit latent space. Examples of these approaches often use an encoder-decoder pattern in which the encoder encodes a data instance into a latent vector, which will be peturbed and then decoded into a similar instance**??**. By modelling the latent space, we can simply sample from a distribution conditioned on the original instance. **?** provide an overview of the strengths and weaknesses of common generative models.

Even though, a single latent vector model can theoretically produce multivariate sequences, it may still be too restrictive to capture the combinatorial space of multivariate sequences. Hence, most of the models within Natural Language Processing (NLP) were not used to produce a sequence of vectors, but a sequence of discrete symbols. For process instances, we can assume a causal

relation between state vectors in a sequential latent space. We call models that capture a sequential latent state-space, which has causal relations, *dynamic*?. Early models of this type of dynamic latent state-space models are the well-known *Kalman-Filter* for continuous states and Hidden Markov Model (HMM) for discrete states. In recent literature, many techniques use Deep Learning to model complex state-spaces. The first models of this type were developed by ???. Their Deep Kalman Filter (DKF) and subsequent Deep Markov Model (DMM) approximate the dynamic latent state-space by modelling the latent space given the data sequence and all previous latent vectors in the sequence. There are many variations??? of ?'s model, but most use Evidence Lower-Bound (ELBO) of the posterior for the current Z_t given all previous $\{Z_{t-1}, \ldots, Z_1\}$ and X_t?.

Generating Counterfactual Time-Series Within the *multivariate time-series* literature two recent approaches yield ideas worth discussing.

First, ? introduce a case-based reasoning to generate counterfactuals?. Their method uses existing counterfactual instances, or *prototypes*, in the dataset. Therefore, it ensures, that the proposed counterfactuals are *realistic*. However, case-based approaches strongly depend on the *representativeness* of the prototypes(?, p. 192). In other words, if the model displays behaviour, which is not captured within the set of prototypical instances, most case-based techniques will fail to provide viable counterfactuals. The likelihood of such a break-down increases due to the combinatorial explosion of possible behaviours if the *true* process model has cycles or continuous event attributes. Cycles may cause infinite possible sequences and continuous attributes can take values on a domain within infinite negative and positive bounds. These issues have not been explored in the paper of ?, as it mainly deals with time series classification?. However, despite these shortcomings, case-based approaches may act as a valuable baseline against other sophisticated approaches.

The second paper within the multivariate time series field by ? also uses a case-based approach?. However, it contrasts from other approaches, as it does not specify a particular model but proposes a general framework instead. Hence, within this framework, individual components could be substituted by better performing components. Describing a framework, rather than specifying a particular model, allows to adapt the framework, due to the heterogeneous process dataset landscape. In this paper, we also introduce a framework that allows for flexibility depending on the dataset.

Generating Counterfactuals for Business Processes So far, none of the techniques have been applied to process data.

Within PM, Causal Inference has long been used to analyse and model business processes. Mainly, due to the causal relationships underlying each process. However, early work has often attempted to incorporate domain-knowledge about the causality of processes in order to improve the process model itself????. Among these, ? approach is one of the first to include counterfactual reasoning

for process optimization**?**. **?** use counterfactuals to generate alternative solutions to treatments, which lead to a desired outcome**?**. Again, the authors do not attempt to provide an explanation of the models outcome and therefore, disregard multiple viability criteria for counterfactuals in XAI. **?** 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. For our approach, we assume that no knowledge about the dependencies are known.

Within the XAI context, **?** develop the first explanation method for process data**?**. However, their work closely resembles the work of **?** and treat the task as Markov Decision Process (MDP)**?**. This extension of a regular Markov Process (MP) assumes that an actor influences the outcome of a process given the state. This formalisation allows the use of Reinforcement Learning (RL) methods like Q-learning or SARSA. However, this often requires additional assumptions such as a given reward function and an action-space. For counterfactual sequence generation, there is no obvious choice for the reward function or the action-space.

Nonetheless, both **?** and **?** contribute an important idea. The idea of incrementally generating the counterfactual instead of the full sequence. **?** has recently published an approach that builds on the same notion of incremental generation. Their approach has a very similar structure to our approach and appears to be the only one that we can compare our counterfactuals against.

For this reason, this thesis highlights some key differences and similarities. However, to understand the differences and similarities, we first have to establish some core concepts. In this section, we only discuss their approach, briefly.

The authors recognised that some processes have critical events 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. They use an extension of DiCE**?** to generate counterfactuals. However, their approach requires concrete knowledge about these critical points. We propose a Framework that avoids this constraint.

To our knowledge, the authors are also the first authors that try to optimize their counterfactual process generation based on criteria that ensure their viability. However, in our approach, we use different operationalisations to quantify the criteria.

1.4 Research Question

As we seek to make data-driven process models interpretable, we have to understand the exact purpose of this thesis. Hence, we establish the open challenges and how this thesis attempts to solve them.

Having discussed the previous work on counterfactual sequence generation, a couple of challenges emerge. First, we need to generate on a set of criteria and therefore, require complex loss and evaluation metrics, that may or may not be differentiable. Second, they cannot to be logically impossible, given the

dataset. Hence, we have to restrict the space to counterfactuals of viable solutions, while being flexible enough to not just copy existing data instances. Third, using domain knowledge of the process significantly reduces the practicality of any solution. Therefore, we have to develop an approach, which requires only the given log as input while not relying on process specific domain knowledge. This begs the question, whether there is a method to generate sequential counterfactuals that are viable, without relying on process specific domain knowledge. In terms of specific research questions we try to answer:

RQ: How do we generate counterfactual sequences while incorporating structural differences between the factual sequence and the counterfactual sequence?
- RQ1: How can we employ existing methods to compute viability so that its optimization incorporates information about the structure of the sequence?
- RQ2: To what extent can we generate counterfactuals that fulfill the criteria to be viable?
- RQ3: How does an algorithm, which optimizes multiple viability quality metrics, perform against other approaches?

We approach these questions, by proposing a schematic framework which allows the exploration of several independent components. Figure 3 shows the conceptual framework of the base approach visually.

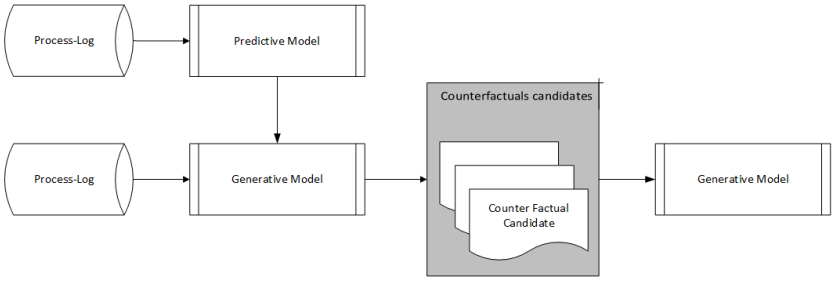


Figure 3: A simplified schematic representation of the framework which is explored in this thesis.

The framework contains three parts. First, we need a pre-trained predictive component, which we aspire to explain. The component should *accurately* predict the outcome of a process at any step. 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. The second part requires a generative component. The generative component needs to generate viable sequential counterfactuals which are logically *plausible*. A plausible counterfactual is one whose outcome can be predicted by the predictive component. If the predictive component cannot predict the counterfactual sequence, we can assume that the generative model is *unfaithful* to the predictive component that we want to explain. The third component

is the evaluation metric upon which we decide the viability of the counterfactual candidates.

For the evaluation, we have to show the following:

RQ2-H1: If we use a viability function which incorporates multiple criteria to determine counterfactuals, we consistently retrieve more viable counterfactuals, than choosing the counterfactuals the at random.

RQ2-H2: The generated counterfactuals consistently outperform the most viable counterfactuals among examples in the dataset.

RQ3-H1: The results of the counterfactual are comparable to other existing literature.

1.5 Outline

The remainder of the thesis is outlined as follows: In section 2, we introduce all of the important concepts that are crucial to this thesis. Most importantly, we introduce the main research discipline PM and the subject of our research: *Counterfactuals*. Furthermore we cover some necessary background required to understand the methods, we employ.

The section 3, introduces our methodological framework in further detail. The chapter explains all the important components and methods, we apply, to answer the research question. Among these methods, we introduce the methodological architecture, a modified version of the Damerau-Levenshstein distance.

section 4 covers the main approach behind our experimental setup. We discuss how we attempt to answer our research questions and introduce the datasets we are using and how we conduct the preprocessing.

In section 5 we report on the results and insights we gain from executing our research approach.

All the results are summarised in section 6. Here, we summarize and interpret our results. We discuss limitations and possible improvements. We also discuss implications for future research endeavors.

The section 7 summarizes the thesis and the implications for the PM research field.

2 Background

This chapter explores the most important concepts for this work. Hence, we focus on the problem domain, starting with an overview about PM. Afterwards, we discuss the nature of the data, we handle in this thesis by discussing *Multivariate Discrete Time-Series*. Next, we introduce counterfactuals and establish how we characterise *viable* counterfactuals.

2.1 Process Mining

This thesis focuses on processes and the modelling of process generated data. Hence, it is important to establish a common understanding for this field.

2.2 A definition for Business Processes

Before elaborating on Process Mining, we have to establish the meaning of the term *process*. The term is widely-used and therefore, has a rich semantic volume. A process generally refers to something that advances and changes over time**?**. Despite, legal or biological processes being valid interpretations, too, we focus on *business processes*.

An example is a loan application process in which an applicant may request a loan. The case would then be assessed and reviewed by multiple examiners and end in a final decision. The loan might end up in an approval or denial. The *business* part is misleading as these processes are not confined to commercial settings alone. For instance, a medical business process may cover a patients admission to a hospital, followed by a series of diagnostics and treatments and ending with the recovery or death of a patient. Another example from a Human Computer Interaction (HCI) perspective would be an order process for an online retail service like Amazon. The buyer might start the process by adding articles to the shopping cart and proceeding with specifying their bank account details. This order process would end with the submission or receival of the order.

All of these examples have a number of common characteristics. They have a clear starting point which is followed by numerous intermediary steps and end in one of the possible sets of outcomes. For this work, we mainly follow the understanding outlined in **??**. Each step, including start- and end-points, is a process event which was caused by an *activity*. Often, both terms, *event* and *activity*, are used interchangeably. However, there are subtle differences. We understand an event as something that happens at a specific point in time. The driving question is *when* the event happens. In contrast, an activity is related to the content of an event. Here, we ask *what* happens at a point in time. For instance, if we apply for a loan that requires an approval by one person and afterwards a second approval, we can call both activities **APPROVAL**. Although both activities are fundamentally the *same*, they happen at different points in time. Henceforth, both events remain *different*. Mainly, because one can argue that both events have varying time dependent contexts. For instance, an approval at daytime might be caused by different reasons, than an event caused at night-time.

Each process event may contain additional information in the form of event attributes. If a collection of events *sequentially* relate to a single run through a process, we call them *process instance* or *trace*. These instances do not have to be completed. Meaning, the trace might end prematurely. In line with the aforementioned examples, these process instances could be understood as a single loan application, a medical case or a buy order. We can also attach process instance related information to each instance. Examples would be the applicants location, a patients age or the buyers budget. In its entirety, a business process can be summarised as a *graph*, a *flowchart* or another kind of visual representation. Figure 4's graphical representation is an example of such a *process map***?**.

In conclusion, in this thesis a *business process* refers to

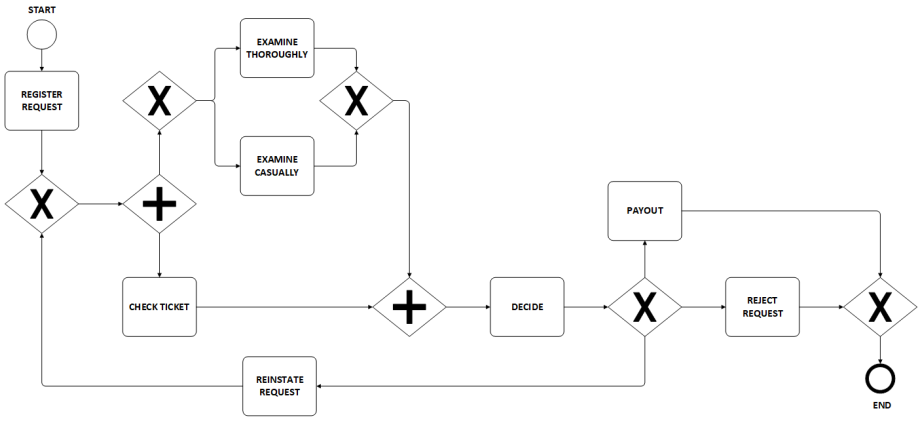


Figure 4: This graph shows an example of a Business Process Modell Notation (BPMN) process map.

> *A finite series of discrete events with one or more starting points, intermediary steps and end points. Each intermediate step has at least one precedent and at least one antecedent step.*

However, we have to address a number of issues with this definition.

First, it excludes infinite processes like solar system movements or continuous processes such as weather changes. There may be valid arguments to include processes with these characteristics, but they are not relevant for this thesis.

Second, in each example, we deliberately used words that accentuate modality such as *may*, *can* or *would*. It is important to understand that each process anchors its definition within an application context. Hence, what defines a business process is indisputably subjective. For instance, while an online marketplace like Amazon might be interested in the process starting from the customers first visit until the successful shipment, an Amazon vendor might only be interested in the delivery process of a product.

Third, the example provided in Figure 4 may not relate to the *real* underlying data generating process. As process *models* are inherently simplified, they may or may not be accurate. The *true* process is often unknown. Therefore, we distinguish between the *true process* and a *modelled process*. The *true process* is a hypothetical concept whose *true* structure remains unknown. In, contrast, a process *model* simplifies and approximates the characteristics of the *true process*.

2.3 What is Process Mining?

Having established a definition for a process, we next discuss *Process Mining*. This young discipline has many connections to other fields that focus on the modelling and analysis of processes such as Continuous Process Improvement (CPI) or Business Process Management (BPM)**?**. However, its data-centric approaches originate in Data Mining. The authors **?** describe this field as a discipline "to discover, monitor and improve real processes (i.e., not assumed processes) by

extracting knowledge from event logs readily available in today's (information) systems"**?**. The discipline revolves around the analysis of event logs. An event log is a collection of process instances, which are retrieved from various sources like an Information System (IS) or database. Logs are often stored in data formats such as Comma Seperated Values (CSV) or eXtensible Event Stream (XES)**?**.

2.4 The Challenges of Process Mining

As mentioned in section 1, process data modelling and analysis is a challenging task. **?** mentions a number of issues that arise from processes**?**.

The first issue arises from the quality of the dataset. Process logs are seldomly collected with the primary goal of mining information and hence, often appear to be of subpar quality for information mining purposes. The information is often incomplete, due to a lack of context information, the ommision of logged process steps, or wrong levels of granularity**?**.

This issue is exacerbated by the second major issue with process data. Mainly, its complexity. Not only does a process logs' complexity arise from the variety of data sources and differing levels of complexity, but also from the datas' characteristics. The data can often be viewed as multivariate sequence with discrete and continuous features and variable length. This characteristic alone creates problems explored in subsection 2.5. However, the data is also just a *sample* of the process. Hence, it may not reflect the real process in its entirety. In fact, mining techniques need to incorporate the *open world assumption* as the original process may generate unseen process instances**?**.

A third issue which contributes to the datasets' incompleteness and complexity is a phenomenon called *concept drift***?**. This phenomenon relates to the possibility of changes in the *true* process. The change may occur suddenly or gradually and can appear in isolation or periodically. An expression of such a drift may be a sudden inclusion of a new process step or domain changes of certain features. These changes are not uncommon and their likelihood increases with the temporal coverage and level of granularity of the dataset**?**. In other words, the more *time* the dataset covers and the higher its detail, the more likely a change might have occurred over the time.

All three issues relate to the *representativeness* of the data with regards to the unknown *true* process that generated the data. However, they also represent open challenges that require research on their own. For our purpose, we have to assume that the data is representative and its underlying process is static. These assumptions are widely applied in the body of process mining literature**??**.

2.5 Multivariate Time-Series Modeling

The temporal and multivariate nature of process instance often turns PM into a Multivariate Time-Series Modeling problem. Therefore, it is necessary to establish an understanding for this type of data structure.

The data which is mined in Process Mining is typically a multivariate time-series. It is important to establish the characteristics of time-series.

2.6 What are Time Series Models?

A time series can be understood as a series of observable values and depend on previous values. The causal dependence turns time-series into a special case of sequence models. Sequences do not *have to* depend on previous values. They might depend on previous and future values or not be interdependent at all. An example of a sequence model would be a language model. Results in NLP, that the words in a sentences for many languages do not seem to only depend on prior words but also on future words**?**. Hence, we can assume that a human has formulated his sentence in the brain before expressing it in a sequence of words. In contrast to sequences, time series cannot depend on future values. The general understanding of *time* is causal and forward directed. The notion of time relates to our understanding of *cause and effect*. Hence, we can decompose any time series in a precedent (causal) and an antecedent (effect) part**?**. A time series model attempts to capture the relationship between precedent and antecedent.

2.7 The Challenges of Time Series Modelling

The analysis of unrestricted sequential opens up a myriad of challenges. First, sequential data introduces a combinatorial set of possible realisations (often called *productions*). For instance, a set of two objects $\{A, B\}$ *produces* 7 theoretical combinations ($\{\emptyset\}$, $\{A\}$, $\{B\}$, $\{A, B\}$, $\{B, A\}$, $\{A, A\}$, $\{B, B\}$). Just by adding C and then D to the object set increases the number of combinations to 40 and 341, respectively. Second, sequential data may contain cyclical patterns which increase the number of possible productions to infinity**?**. Both, the combinatorial increase and cycles, yield a set of a countable infinite number of possible productions. However, as processes may also contain additional information a third obstacle arises. Including additional information extends the set to an uncountable number of possible productions. With these obstacles in mind, it often becomes intractable to compute an exact model.

Hence, we have to include restrictive assumptions to reduce the solution space to a tractable number. A common way to counter this combinatorial explosion is the inclusion of the *Granger Causality* assumption**?**. This idea postulates the predictive capability of a sequence given its preceding sequence. In other words, if we know that C must be followed by D, then 341 the number of possible combinations reduces to 156. All of these possible 156 combinations are now temporally-related and hence, we speak of a *time-series*.

However, the prediction of sequences recontextualises the issue to two new questions: First, if we know the precedence of a time-series, what is the antecedent? And second, if we can predict the antecedent accurately, what caused it? We often use data-driven AI-methods like Hidden-Markov-Models or Deep Learning to solve the first question. However, the second question is more subtle. At first glance, it is easy to believe that both questions are quite similar, because we could assume that the precedent causes the antecedent. Meaning, that we can use the data available to elicit sequential correlative patterns. In reality,

the latter question is much more difficult as data often does not include any information about the inter-relationships. To illustrate this difficulty, we could say that the presence of C causes D. But if D also appears to be valid in a sequence 'AABD', it cannot be caused by the presence of C alone.

Answering this question requires additional tools within the XAI framework. One such method is the focus of this thesis and is further explored in subsection 2.8.

2.8 Counterfactuals

Counterfactuals are an important explanatory tool to understand a models' cause for decisions. Generating counterfactuals is main focus of this thesis. Hence, we establish the most important chateristics of counterfactuals in this section.

What are Counterfactuals? Counterfactuals have various definitions. However, their semantic meaning refers to "*a conditional whose antecedent is false*"**?**. A simpler definition from **?** states that counterfactual modality concerns itself with "*what is not, but could or would have been*". Both definitions are related to linguistics and philosophy. Within AI and the mathematical framework various formal definitions can be found in the causal inference**?** literature. A prominent figure within the causal inference discipline is **?**, who postulates that a "*kind of statement – an 'if' statement in which the 'if' portion is untrue or unrealised – is known as a counterfactual*"**?**. What binds all of these definitions is the notion of causality within *what-if* scenarios.

For this paper, we use the understanding established within the XAI context. Within XAI, counterfactuals act as a prediction which "*describes the smallest change to the feature values that changes the prediction to a predefined output*" according to **?**(**?**, p. 212). Note that XAI mainly concerns itself with the explanation of *models*, which are always subject to inductive biases and therefore, inherently subjective. The idea behind counterfactuals as explanatory tool[6] is simple. We understand the outcome of a model, if we know *what* outcome would occur *if* we changed its input.

Let us assume, a student is approaching an important deadline, which she desires to meet. Every day, she has a multitude of options to choose from. Either, continue with the report (option A), focus on learning more about the topic (option B), pursue her hobby as a break (option C), meet up with friends (option D), or procrastinate (option E). Furthermore, we assume, there are 7 days left and she can either miss (0) the deadline or meet it (1). The approach she follwos is $ABABD\mathbf{E}A$ and she misses the deadline. Let us refer to this sequence of actions as the factual *sequence 1*. Then, a counterfactual $ABABD\mathbf{B}A$ that meets the deadline tells us that **E** (probably) caused missing the deadline. In other words, if the student had not procastinated two days before the deadline she could have make it on time.

[6] There are other explanatory techniques in XAI like *feature importances* but counterfactuals are considered the most human-understandable

As counterfactuals only address explanations of one model result and not the model as a whole, they are *local* explanations(**?**, p. 212). According to **?** *Valid* counterfactuals satisfy **four** criteria(**?**, p. 212):

Similarity: A counterfactual should be similar to the original instance. If a sucsessful counterfactual to sequence 1 was $ABAB\boldsymbol{EE}A$, we would already have difficulties to discern whether meeting with friends D, procrastinating E or both caused the outcome of missing the deadline 0. Hence, we want to be able to easily compare the counterfactual with the original. We can archive this by minimizing their mutual distance.

Sparcity: In line with the notion of similarity, we want to change the original instance only minimally. If the sequence had many changes, it would similarly impede the understanding of causal relationships in sequence 1.

Feasibility: Each counterfactual should be feasible. In other words, impossible values are not allowed. As an example, if the student followed a strict $A\boldsymbol{A}A\boldsymbol{A}A\boldsymbol{A}EA$ would not be feasible if we consider students could burn-out. Typically, we can use data to ensure this property. However, the *open-world assumption* impedes this solution. With *open-world*, we mean that processes may change and introduce behaviour that has not been measured before. A student might only attempt a Bachelor's thesis once. Especially, for long and cyclical sequences, we have to expect previously unseen sequences.

Likelihood: A counterfactual should produce the desired outcome if possible. This characteristic is ingrained in **?**'s definition. However, as the model might not be persuaded to change its prediction, we relax this condition. We say that we want to increase the likelihood of the outcome as much as possbile. If the counterfactual $ABABDXA$ hinges on X as in an earthquake occuring that postpones the deadline, the sequence would be highly unrealistic. Hence, we cannot be certain of our conclusion for sequence 1. Therefore, we want the counterfactual's likelihood to be at least more likely than the factual outcome.

All four criteria allow us to assess the viability of each generated counterfactual and thus, help us to define an evaluation metric for each individual counterfactual. However, we also seek to optimise certain qualities on the population level of the counterfactual candidates.

Diversity: We typically desire multiple diverse counterfactuals. One counterfactual might not be enough to understand the causal relationships in a sequence. In the example above, we might have a clue that E causes an outcome of 0, but what if outcome 0 is by more than E? If we are able to find counterfactuals all counterfactuals that involve E and that lead to missing the deadline, we get a better understanding of what caused outcome 0.

Realism: For a real world application, we still have to evaluate their *reasonability* within the applied domain. This is a characteristic that can only be evaluated by a domain expert.

We refer to both sets of viability criteria as *individual viability* and *population viability*. However, to remains concise, we use *viability* to refer to the individual

criteria only. We explicitly mention *population viability* if we refer to criteria that concern the population.

The Challenges of Counterfactual Sequence Generation The current literature surounding counterfactuals exposes a number of challenges when dealing with counterfactuals.

The most important disadvantage of counterfactuals is the Rashommon Effect(**?**, ch.9.3). If all of the counterfactuals are viable, but contradict each other, we have to decide which of the *truths* are worth considering.

This decision reveals the next challenge of evaluation. Although, the criteria can support us with the decision, it remains an open research question *how* to evaluate counterfactuals according to **??**. So far, no one was able to establish a standardised evaluation protocol**?**. Every automated measure comes with implicit assumptions and they cannot guarantee a realistic explanations. Furthermore, we attempt to explain something with – in simple terms – *experiences* that never actually occured. We still need domain experts to assess their *plausibility*.

The generation of counterfactual sequences contribute to both former challenges, due to the combinatorial expansion of the solution space. This problem is common for counterfactual sentence generation and has been adressed within the NLP. However, as process mining data not only consist of discrete objects like *words*, but also event and case features, the problem remains a daunting task. So far, little work has gone into the generation of multivariate counterfactual sequences like process instances.

2.9 Formal Definitions

Before diving into the rest of this thesis, we have to establish preliminary definitions, we use in this work. With this definitions, we share a common formal understanding of mathematical descriptions of every concept used within this thesis.

2.10 Process Logs, Cases and Instance Sequences

We start by formalising the event log and its elements. We use a medical process as an example to provide a better semantic understanding. An event log is denoted as L. Here, L could be as database which logs the medical histories of all patients in a hospital.

We assume the database logs all interactions, be it therapeutic or diagnostic and store them as an event with a unique identifier. Let $\mathcal{E}$ be the universe of these event identifiers and $E \subseteq \mathcal{E}$ a set of events. The set E could consist, for instance, of a patients first session with a medical professional, then a diagnostic scan, followed by therapy sessions, surgery and more.

All of these interactions with one patient make up a case, which has a unique identifier, too. Let C be a set of case identifiers and $\pi_\sigma : E \mapsto C$ a surjective

function that links every element in E to a case $c \in C$ in which c signifies a specific case. The function allows us to associate every event within the database to a single patient. The function's surjective property ensures for each case there exists at least one event.

For a set of events $E \subseteq \mathcal{E}$, we use a shorthand s^c being 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 $s \in L$. To understand the difference between c and s, we can say, that c is the ID for the case of patient X. Henceforth, s^c reflects all interactions that the database has logged for patient X.

These events are ordered in the sequence, in which they occurred for patient X. Therefore, 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. In other words, every event in the database maps to one point in time. If the database logs every event on a daily basis, then all possible dates in history constitute $\mathcal{T}$. However, not every day has to be linked to a case as π_t is non-surjective.

Let $\mathcal{A}$ be a universe of attribute identifiers, in which each identifier maps to a set of attribute values $\overline{a}_i \in \mathcal{A}$. An attribute identifier describes everything the database might store for a patient, such as heart-rate or blood sugar level. If the database logs the heart-rate, then heart-rates of -42 beats-per-minute are not possible. Hence, $\overline{a}_i$ can per definition only map to positive integers.

Let $\overline{a}_i$ correspond to a set of possible attribute values by using a surjective mapping function $\pi_A : \mathcal{A} \mapsto A$. Then, 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 $I = |\mathcal{A}|$, in which each a_i refers to a value within its respective set of possible attribute values. In other words, every event consists of a set of values. If the event was recorded after a physio therapeutic session, then a_1 might be the specific degree to which you can move your ligaments and a_2 a description for the type of activity. If the event was recorded after a breast-cancer scan, the a_1, a_2 and a_3 might relate to the specific diameter, the threat-level and again an indicator for the activity type. Conversely, we define a mapping from an attribute value to its respective attribute identifier $\pi_{\overline{a}} : A \mapsto \mathcal{A}$. Hence, we can map every event attribute value back to its attribute identifier.

The following part is not necessarily connected with *what* is stored within the database symbolically, but rather *how* it is represented in the database or during processing.

We require a set of functions F to map every attribute value to a representation which can be processed by a machine. Let $\pi_d : A_i \mapsto \mathbb{N}$ be a surjective function, which determines the dimensionality of a_i and also F be a set of size I containing a representation function for every named attribute set. We denote each function $f_i \in F$ as a mapper to a vector space $f_i : a_i \mapsto \mathbb{R}_i^d$, in which d represents the dimensionality of an attribute value $d = \pi_d(A_i)$. For instance categorical variables will can map to a one-hot-encoded vector. Numerical values like heart-beat might be recorded in scalar form.

With these definitions, 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 named attribute set A_i, we use the shorthand $\overline{a}_i$.

Figure 7 shows a schematic representation of a log L, a case c and an event e.

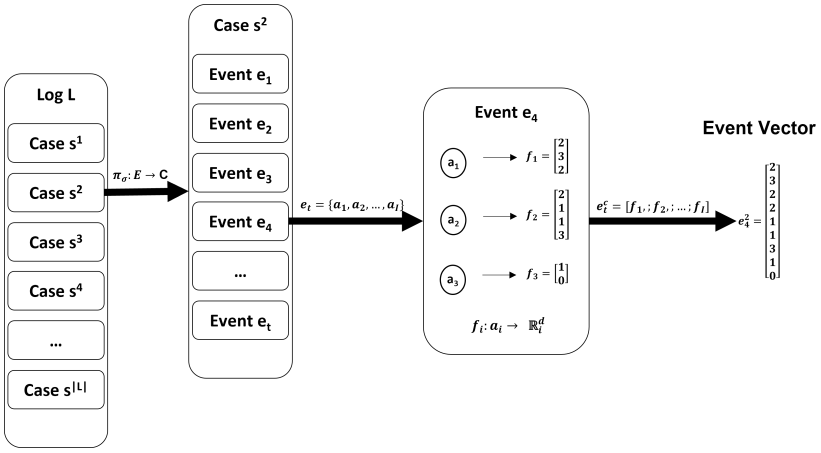


Figure 5: This figure shows the representation of a log L which contains a number of cases s. Case s^2 contains a number of events e_t. Each events has attribute values a_i, which are mapped to vector spaces of varying dimensions. At last, all of the vectors are concatenated.

2.11 State-Space Models

Generally speaking, every time-series can be represented as a state-space model**?**. Within this framework the system consists of *input states* for *subsequent states* and *subsequent outputs*. A mathematical form of such a system is shown in Equation 1.

$$\begin{aligned} \mathbf{z}_{t+1} &= h(t, \mathbf{z}_t, \mathbf{u}_t) \\ \mathbf{e}_t &= g(t, \mathbf{z}_t, \mathbf{u}_t) \\ \mathbf{z}_{t+1} &:= \frac{d}{dt}\mathbf{z}_t \end{aligned} \tag{1}$$

Here, $\mathbf{u}_t$ represents the input, $\mathbf{z}_t$ the state at time t. The function h maps t, $\mathbf{z}_t$ and $\mathbf{u}_t$ to the next state $\mathbf{z}_{t+1}$. The event $\mathbf{e}_t$ acts as an output computed by function g which takes the same input as h. The variables $\mathbf{z}_t$, $\mathbf{u}_t$ and $\mathbf{e}_t$ are vectors with discrete or continuous features. The distinction of $\mathbf{z}_{t+1}$ and

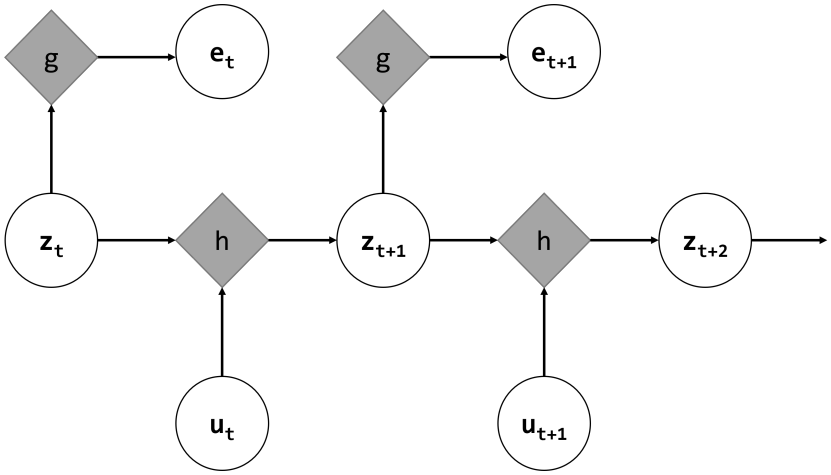


Figure 6: This figure shows a simplified graphical representation of a state-space model. Each arrow represents the flow of information.

$\mathbf{e}_t$ decouples *hidden*[7] states, from *observable* system outputs. Figure 6 shows a graphical representation of these equations.

The body of literature for state-space models is too vast to discuss them in detail. However, for process mining, we can use this representation to discuss the necessary assumptions for process mining. In line with the process-definition in subsection 2.1, we can understand the event log as a collection of the observable outputs of a state-space model. The state of the process is hidden as the *true* process which generated the data cannot be observed as well. The time t is a step within the process. Hence, we treat t as a discrete scalar value to denote discrete sequential time steps. Hence, if we have $\sigma = \{a, b, b, c\}$, then t, describes the index of each element in σ. The input $\mathbf{u}_t$ represents all context information of the process. Here, $\mathbf{u}_t$ subsumes observable information such as the starting point and process instance-related features. The functions h and g determine the transition of a process' state to another state and its output over time. Note, that this formulation disregards any effects of future timesteps on the current timestep. Meaning, that the state transitions are causal and therefore, ignorant of the future. As we establish in subsection 2.1, we can assume that a process is a discrete sequence, whose transitions are time-variant. In this framework, we try to identify the parameters of the functions h and g. Knowing the functions, it becomes simple to infer viable counterfactuals. However, the function parameters are often unknown and therefore, we require probablistic approaches.

We can formulate Equation 1 probablistically as shown in Equation 2.

[7] A state does not have to be hidden. Especially, if we know the process and the transition rules. However, those are often inaccessible if we only use log data. Instead, many techniques try to approximate the hidden state given the data instead. For an introduction to state-space models see:**?**

$$\mathbb{E}[p\left(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h\right)] = \int z_{t+1} \cdot p\left(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h\right) \quad (2)$$

$$\mathbb{E}[p\left(x_t \mid t, z_{1:T}, u_{1:T}, \theta_g\right)] = \int x_t \cdot p\left(x_t \mid t, z_{1:T}, u_{1:T}, \theta_g\right)$$

Note, that h and g are substitued with probability density functions parametrised with θ_h and θ_g. T signifies the full sequence including future timesteps. Both expectations are intractable as they require integrating over n-dimensional vectors. To solve the intractability, we characterize the system as a *Hidden Markov Process* and Probalistic Graphical Model (PGM). This framework allows us to leverage simplifying assumptions such as the independece from future values and *d-seperation*.

These characteristics change the probabilities in Equation 2 to Equation 3:

$$p\left(z_{t+1} \mid z_{1:t}, u_{1:t}, \theta_h\right) = \prod_{1}^{t} p\left(z_t \mid z_{1:t}, u_t, \theta_h\right) \quad (3)$$

$$p\left(x_t \mid z_{1:t}, \theta_g\right) = \prod_{1}^{t} p\left(x_{t-1} \mid z_{1:t}, \theta_f\right) \quad (4)$$

For $p\left(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h\right)$, we ignore future timesteps, as T changes into t. *d-seperation* allows us to ignore all $\mathbf{e}_t$ of previous timesteps. The graphical form also decomposes the probability into a product of probabilities that each depend on all previous states and its current inputs. Previous $\mathbf{e}_t$ are ignored due to *d-seperation*. $p\left(x_t \mid t, z_{1:T}, u_{1:T}, \theta_g\right)$ only depends on its current state, which is in line with HMMs. Note, that we deliberately not assume a *strong Markov Property*, as the Deep Learning-Framework allows us to take all previous states into account. The *strong Markov Property* would assume that only the previous state suffices. At last, we assume that we do not model automatic or any other process whose state changes without a change in the input or previous states. Hence, we remove the dependency on the independent t variable. Only the previous states $z_{1:T}$ and the input information $\mathbf{u}_t$ remain time-dependent.

In this probabilistic setting, the generation of counterfactuals, amounts to drawing samples from the likelihood of Equation 3. We then use the samples to reconstruct the most-likely a counterfactual $e^*_{1:t}$. Hence, our goal is to maximize both likelihoods.

2.12 Representation

To process the data in subsequent processing steps, we have to discuss the way we encode the data. There are a multitude of ways to represent a log. We introduce four ways and the reason we choose the *hybrid-vector-representation*. Figure 7 shows schematically, how we can represent process data.

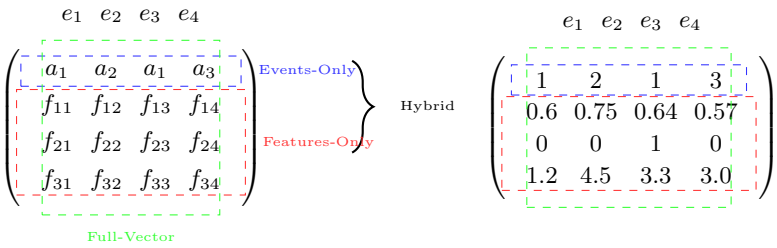


Figure 7: All four possible representations on an exemplary process instance.

First, we can choose to concentrate on *event-only-representation* and ignore feature attributes entirely. However, feature attributes hold significant amount of information. Especially in the context of using counterfactuals for explaining models as the path of a process instance might strongly depend on the event attributes. Similar holds for a *feature-only-representation*.

The first is a *single-vector-representation* with this representation we can simply concatenate each individual representation of every original column. This results in a matrix with dimensions (case-index, max-sequence-length, feature-attributes). The advantage of having one vector is the simplicity with which it can be constructed and used for many common frameworks. Here, the entire log can be represented as one large matrix. However, even though, it is simple to construct, it is quite complicated to reconstruct the former values. It is possible to do so by keeping a dictionary which holds the mapping between original state and transformed state. However, that requires every subsequent procedure to be aware of this mapping. Furthermore, we use methods, that treat events and their associated features (event attributes) separately. For instance, if we want to sample from a *Markov Model* with transition probabilities and emission probabilities, then it is much easier to first sample the event trajectory and then, the conditional feature attributes. Or, if we attempt to compute an edit distance between two sequences, it is easier to compute those, if we keep events and event attributes separate.

Therefore, we decide to keep the original sequence structure of events as a separate matrix and complementary to the remaining event attributes. If required, we turn the label encoded activities ad-hoc to one-hot encoded vectors. Thus, this *hybrid-vector-representation* grants us greater flexibility. However, we now need to process two matrices. The first matrix has the dimensions (case-index, max-sequence-length) and the latter (case-index, max-sequence-length, feature-attributes).

2.13 Long-Short-Term Memory Models

In order to explain the decisions of a prediction we have to introduce a predictive model, which needs to be explained. Any sequence model suffices. Additionally, the model's prediction do not have to be accurate. However, the more accurate

the model can capture the dynamics of the process, the better the counterfactual functions as an explanation of these dynamics. This becomes particularly important if the counterfactuals are assessed by a domain expert.

In this thesis, the predictive model is an Long Short-Term Memory (LSTM) model. LSTMs are well-known models within Deep Learning, that use their structure to process sequences of variable lengths**?**. LSTMs are an extension of Recurrent Neural Networks (RNNs). We choose this model as it is simple to implement and can handle long-term dependencies well.

Generally, RNNs are Neural Networks (NNs) that maintain a state h_{t+1}. The state is computed and then propagated to act as an additional input alongside the next sequential input of the instance x_{t+1}. The hidden state h is also used to compute the prediction o_t for the current step. The formulas attached to this model are shown in

$$h_{t+1} = \sigma(Vh_t + Ux_t + b) \tag{5}$$

$$o_t = \sigma(Wh_t + b) \tag{6}$$

Here, W, U and V are weight matrices that are multiplied with their respective input vectors h_t, x_t. b is a bias vector and σ is a non-linearity function. LSTM fundamentally work similarly, but have a more complex structure that allows to handle long-term dependencies better. They manage this behaviour by introducing additional state vectors, that are also propagated to the following step. We omit discussing these specifics in detail, as their explanation is not further relevant for this thesis. For our understanding it is enough to know that h_t holds all the necessary state information. Figure 8 shows a schematic representation of an RNN.

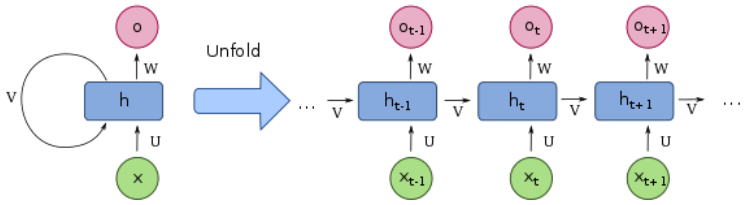


Figure 8: A schematic representation of an RNN viewed in compact and unfolded form.

2.14 Damerau-Levenshtein

The Damerau-Levenshstein distance function is a modified version of the Levenshstein distance**?**, which is a widely used to compute the edit-distance of two

discrete sequences**??**. The most important applications are within the NLP discipline and the Biomedical Sciences. Within these areas, we often use the Levenshtein distance to compute the edit-distance between two words, two sentences or two DNA sequences. Note, that the elements of these sequences are often atomic symbols instead of multidimensional vectors. Generally, the distance accounts for inserts, deletions and substitutions of elements between the two serquences. **?** modified the distance function to allow for transposition operations. For Process Mining, transpositions are important as one event can transition into two events that are processed in parallel and may have varying processing times**?**. In Figure 9, we schematically show two sequences and their distance.

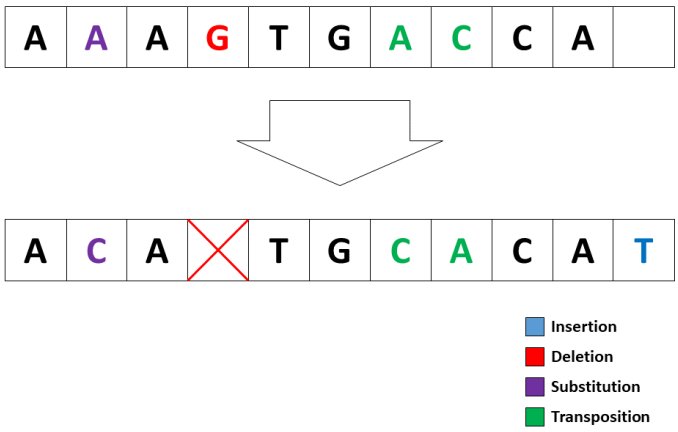


Figure 9: Two arbitrary sequences and their edit difference according to Damerau. The edit distance is the sum of each operation necessary to transform the sequence to another sequence. Blue shows an insert, red a deletion, purple a substitution and green a transposition. Therefore, the edit distance is 4.

Equation 7 depicts the recursive formulation of the distance. The distance computes the costs of transforming the sequence a to b, by computing the minimum

of five seperate terms.

$$d_{a,b}(i,j) = \min \begin{cases} d_{a,b}(i-1,j)+1 & \text{if } i > 0 \\ d_{a,b}(i,j-1)+1 & \text{if } j > 0 \\ d_{a,b}(i-1,j-1)+1 & \text{if } i,j > 0 \\ d_{a,b}(i-2,j-2)+1 & \text{if } i,j > 1 \wedge a_i = b_{j-1} \wedge a_{i-1} = b_j \\ 0 & \text{if } i = j = 0 \end{cases} \tag{7}$$

The recursive form $d_{a,b}(i,j)$ for sequences a and b with respective elements i and j takes the minimum of each allowed edit operation. In particular, no change, deletion, insertion, substitution and transposition. For each operation, the algorithm adds an edit cost of 1.

We cannot use the Damerau-Levenshstein distance for process mining if the process carries additional information about event attributes. Mainly, because two events may be emitted by the same activity, but they may still carry different event attributes.

To illustrate the issue, we explore a couple of examples. Let us assume, we have two strings $s^1 = aaba$ and $s^2 = acba$. Using the Damerau-Levenshstein distance, the edit distance between both sequences is 1, as we can recognise a substitution at the second position in both strings. However, this representation is insufficient for process instances. Therefore, we now characterise the two sequences as process events rather than strings in Equation 8.

$$s^1 = \{a, a, b, a\} \tag{8}$$

$$s^2 = \{a, a^*, b, a\} \tag{9}$$

$$s^3 = \{a, c, b, a\} \tag{10}$$

$$s^4 = \{a, a, b\} \qquad a, b, c \in \mathbb{R}^3 \tag{11}$$

$$a = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix} a^* = \begin{bmatrix} 3 \\ 3 \\ 4 \end{bmatrix} b = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} c = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix} \tag{12}$$

If we do not consider attribute values, it becomes clear that s^2, s^3 and s^4 have an edit-distance to s^1 of 0, 1 and 1. However, with attribute values in mind, s^1 and s^2 display clear differences. Similarly, s^1 and s^3 not only differ in terms of activity but also attribute value. Lastly, s^1 and s^4 are the same in attribute values, but one element still misses entirely. It appears unintuitive that each of these differences are associated with the same cost. The examples show that we can neither disregard attribute values nor events, while computing the edit distance of two process instances.

Instead, we have to define a cost function which takes attribute variables into account. Therefore, we modify the Damerau-Levenshstein distance by introducing a cost function instead of a static cost. Here, the cost of each edit-type is determined by a distance-function, which considers the difference between

event-attributes. Therefore, we propose an edit-function, which captures structural sequence differences, as well as, content related differences. Going back to our example, if assume our cost function to only count differences in attributes, then the difference between s^1 and s^2 shall be 2 as their activities are the same, but the first two event attributes are different. To illustrate the structural elements, the difference between s^1 and s^3 shall be 3 instead of 2. Even if both a and c have two common event attributes, the activities they represent are still different. For instance, if both s^1 and s^3 were medical processes and a and c represented taking a cancer drug or a placebo, anyone would understand both activities are different even if the patient took the same dosage.

2.15 Evolutionary Algorithms

Many of our generative models are based on Evolutionary Algorithms. This section provides a small overview about this optimization technique.

All evolutionary algorithms use ideas that resemble the process of evolution. There are four broad categories: A Genetic Algorithm (GA) uses bit-string representations of genes, while Genetic Programming (GP) uses binary codes to represent programs or instruction sets. Evolutionary Strategy (ES) require the use of vectors. Lastly, Evolutionary Programming (EP), which closely resembles ES, without imposing a specific data structure type**??**. Our approach falls into the category of GA. We refer to the literature review of **?** for more insights into the field. The most vital concept in this category is the *gene* representation.

For the algorithm, we follow a rigid structure of the operations as outlined in 1. As 1 shows, we define 5 fundamental operations. Initiation, Selection, Crossover, Mutation and Recombination. The core idea is to generate new individuals every generation while discarding those who are not deemed fit enough for the next iteration cycle. This optimization method differs from gradient-based methods such as Deep Learning, because it does not require us to use differentiable functions. This makes evolutionary algorithms tremendously useful but also highly dependent on the composition of the fitness function.

Algorithm 1 The basic structure of an evolutionary algorithm.

Require: Hyperparameters
Ensure: The result is the final population
  $population \leftarrow$ INIT $population$;
  while not $termination$ **do**
    $parents \leftarrow$ SELECT $population$;
    $offspring \leftarrow$ CROSSOVER $parents$;
    $mutants \leftarrow$ MUTATE $offspring$;
    $survivors \leftarrow$ RECOMBINE $population \cup mutants$;
    $termination \leftarrow$ DETERMINE $termination$
    $population \leftarrow survivors$
  end while

The initiation operator refers to the creation of the initial set of candidates for the selection process in the first iteration of the algorithm. Often, this amounts to the random generation of individuals. However, choosing among a subset of the search space can allow for a faster convergence.

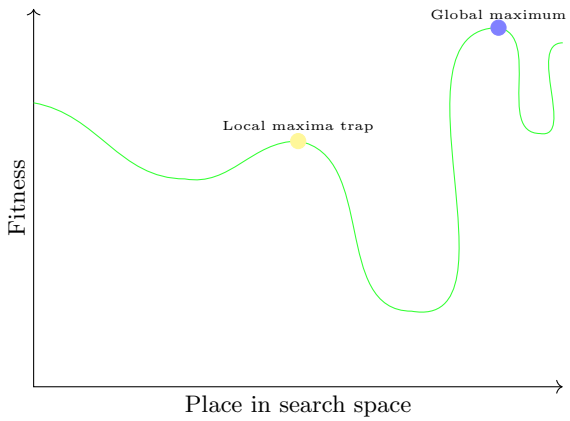


Figure 10: Schematic example showing various local optima.

The selection operator chooses a set of individuals among the population according to a selection procedure. These individuals go on to act as material to generate new individuals. This operator can strongly influence the level with which the algorithm explores the search space. For instance, if an algorithm only selects the best individuals it is easy to get stuck within an local maximum. Figure 10 displays why local maxima should be avoided. Mainly, if we get stuck, we may not be able to find the absolute best solution.

Within the crossover procedure, we select random pairing of individuals to pass on their characteristics. We can often generate at least two additional offsprings, if we have two parents and just reverse the operation.

Mutations introduce random pertubations to the offsprings. This extends the search space beyond what is available by the parents.

The Recombination operation decides which individuals remain in the population for the next iteration[1]. This operator-type is a second source that determines the exploration space of the evolutionary algorithm. For instance, if we only allow the best 10 individuals to move on to the next iteration and only select the top 3 individuals for the crossover phase, we quickly converge towards one solution. Hence, both operators interact, which is why the literature often treat these operations as identical. However, splitting them allows us to control the number of sampled off spring and the population size separately.

We name the strict selection of the best individuals among the offsprings and the previous population *Fittest-Survivor-Recombination*. This recombiner strictly optimizes the population and is susceptible to getting stuck in local

minima. In contrast, we name the addition of the top-k best offsprings to the initial population *Best-of-Breed-Recombination*. The former will guarantee, that the population size remains the same across all iterations but is prone to local optima. Furthermore, we propose one additional recombination operator. The operator selects the new population in a different way than the former recombination operators. Instead of using the viability directly, we sort each individual by every viability component, separately. This approach allows us to select individuals regardless of the scales of every individual viability measure. We refer to this method as *Ranked-Recombination*.

3 Methods

In this chapter, we describe details of our framework and discuss advantages and limitations. Therefore, we provide a more detailed overview and additionally describe all components. As the framework resembles the work of **??**, we also discuss differences and similarities between both solutions.

3.1 Methodological Framework: CREATED

Architecture To generate counterfactuals, we need to establish a conceptual framework consisting of three main components. The three components are shown in Figure 11.

The first component is a predictive model. As we attempt to explain model decisions with counterfactuals, the model needs to be pretrained. We can use any model that can predict the probability of a sequence. This condition holds for models trained for process outcome classification and next-activity prediction. The model used in this thesis is a simple LSTM model using the process log as an input. The model is trained to predict the next action given a sequence.

The second component is a generative model. The generative model produces counterfactuals given a factual sequence. In our approach, each generative model should be able to generate a set of counterfactual candidates given one factual sequence. Specifically, we compare an evolutionary approach against 3 different generative baseline approaches. The baselines do not iteratively optimise towards viability criteria. All approaches allow us to use a factual sequence as a starting point for the generative production of counterfactuals. Furthermore, they also generate multiple variations of the final solution.

The generated candidates are subject to the third major component's scrutiny. To select the most *viable* counterfactual candidate, we evaluate their viability score using a custom metric. The metric incorporates all main criteria for viable counterfactuals mentioned in subsection 2.8. We measure the *similarity* between

[1] We have to point out that in the literature, recombination is often synonymous with crossover. Both steps are similar in their filtering purpose. However, the selector filters potential parents while the recombiner filters the population. However, in this thesis recombination refers to the update process which generates the next population.

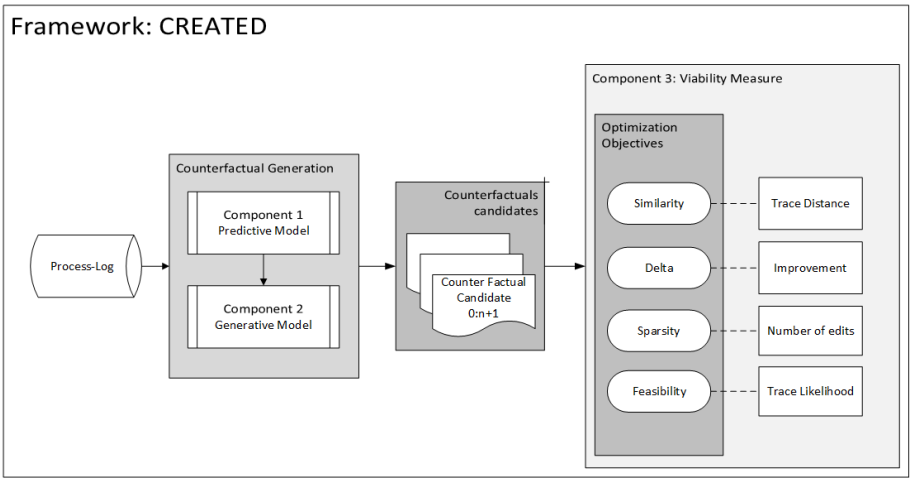


Figure 11: The methodological framework of this thesis. The input is the process log. The log will be used to train a predictive model (Component 1) and the generative model (Component 2). This process produces a set of candidates which are subject to evaluation via the validity metric (Component 3).

two sequences using a multivariate sequence distance metric. The *delta* 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. This requires splitting the feasibility into two parts. First, the likelihood of the sequence of each event and second, the likelihood of the features given the event that occurred.

As we heavily rely on evolutionary algorithms to generate our counterfactuals, we refer to this framework as CREATED. The name describes 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 counterfactuals.

Differences to DiCE4EL ? has recently published a paper on the counterfactual generation of process data. They call their framework DiCE4EL and share many ideas with our framework. Therefore, we want to highlight the key differences and similarities.

In their approach, they attempt to solve various issues that we have also highlighted in subsection 1.2. Furthermore, they do so by incrementally generating the model in sequential order. However, unlike **?**, whose solution creates counterfactuals for every step in the sequence, **?** focus on critical decision points they call milestones.

To gain a better understanding, it is important to outline the event log the authors use briefly. It was taken from a Dutch bank which processes loan applications in which customers request a certain amount of money. The activities

are related to either application states or manual work activities. The application states consist of tasks generated by a machine and manual work activities produced by humans. Hence, the manual work items occur in reference to the application state. For instance, if the loan application is in a *pre accepted* state, then the next events are often produced by humans reviewing the state. Those events are essentially sub-events of the application state. Human activities do not have to happen sequentially. They can occur in parallel. The moment all manual work items conclude marks the decision for the next application state. For instance, from *pre accepted* to *accepted*. Now, to understand why the milestone approach works requires knowing that an application loan process will change to a rejection state, for instance, if all manual work items are completed. There will not be applications that suddenly switch to another state, although the work items of a previous state have not concluded yet. Thus, one can split the entire sequence into subsequences or ignore the sub-events, reducing the search space significantly.

One issue with this approach is the fact that one would first have to identify these milestones. Hence, a crucial distinction to our proposed framework is their dependence on knowledge about the true process, as displayed in this section. Our framework does not leverage structural information about the true process model in question. We believe this is the core contribution in contrast to their approach.

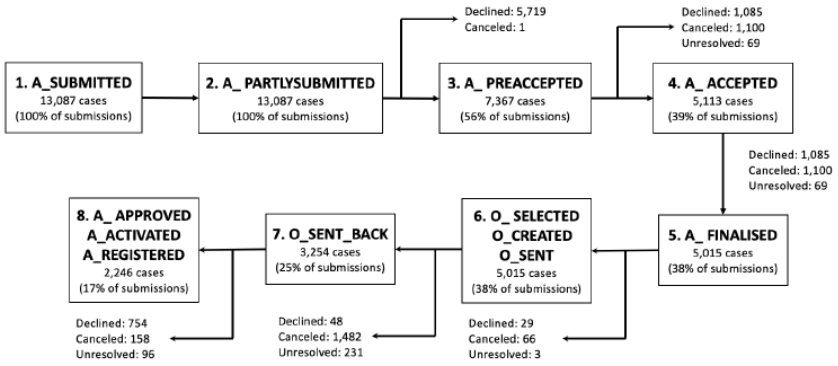


Figure 12: Milestones of loan application process captured in BPIC2012 as identified in **?**

However, similarities between both frameworks do exist. Mainly, our approach also relies on the prediction scores of the model we attempt to explain. Similar to **?**, we incorporate these scores into our quality measure.

3.2 Semi-Structured Damerau-Levenshtein Distance

Before discussing the viability function, we have to introduce an edit-distance for sequences. An edit-distance is used to compute distance between two sequences. Therefore, they take their *structural* patterns like the length or deletions or inserts into account. However, most approaches tend to focus on the sequence of items (letters or words) without taking into account that each item may have additional attributes. Therefore, we propose a custom edit-distance measure.

Semi-Structured Damerau-Levenshtein In order to reflect these differences in attribute values, we introduce a modified version of the Damerau-Levenshstein distance, that not only reflects the difference between two process instances, but also the attribute values. We achieve this by introducing a cost function $cost_{a_i,b_j}$, which applies to a vector-space. Concretely, we formulate the modified Damerau-Levenshstein distance as shown in Equation 13. For the remainder, we refer to this edit-distance as Semi-strucured Damerau-Levenshtein distance (SSDLD).

$$d_{a,b}(i,j) = \min \begin{cases} d_{a,b}(i-1,j) + cost(\mathbf{0}, b_j) & \text{if } i > 0 \\ d_{a,b}(i,j-1) + cost(a_i, \mathbf{0}) & \text{if } j > 0 \\ d_{a,b}(i-1,j-1) + cost(a_i, b_j) & \text{if } i,j > 0 \\ & \quad \& \ \overline{a}_i = \overline{b}_j \\ d_{a,b}(i-1,j-1) + cost(a_i, \mathbf{0}) + cost(\mathbf{0}, b_j) & \text{if } i,j > 0 \\ & \quad \& \ \overline{a}_i \neq \overline{b}_j \\ d_{a,b}(i-2,j-2) + cost(a_i, b_{j-1}) + cost(a_{i-1}, b_j) & \text{if } i,j > 1 \\ & \quad \& \ \overline{a}_i = \overline{b}_{j-1} \\ & \quad \& \ \overline{a}_{i-1} = \overline{b}_j \\ 0 & \quad \& \ i = j = 0 \end{cases} \tag{13}$$

Here, $d_{a,b}(i,j)$ is the recursive form of the Damerau-Levenshtein-Distance. a and b are sequences and i and j specific elements of the sequence. $cost(a,b)$ is a cost function which takes the attribute values of a and b into account. The first two terms correspond to a deletion and an insertion from a to b. The idea is to compute the maximal cost for that the wrongfully deleted or inserted event. The third term adds the difference between two events with identical activities $\overline{a}_i$ and $\overline{b}_j$. As mentioned earlier, two events that refer to the same activity can still be different due to event attributes. The distance between the event attributes determines *how* different these events are. The fourth term handles the substitution of two events. Here, we compute the substitution cost as the sum of an insertion and a deletion. The fifth term computes the cost after transposing both events. This cost is similar to term 3 only that we now consider the differences between both events after they were aligned. The last term relates to the stopping criterion of the recursive formulation of the Damerau-Levenshstein distance.

Discussion There are two noteworthy discussion points, as they might incite disagreements with the validity of our viability measure.

If we assess the first two terms, we use $cost(x, 0)$ to denote the maximal distance of inserting and deleting x. $cost(x, 0)$ can be read as cost between x and a null-vector of the same size. However, it is noteworthy to state that this interpretation does not hold for any arbitrary cost function. For instance, the cosine distance does not work with a null vector, as it is impossible to compute the angle between x and a null vector. Here, the maximum distance would just amount to 1. In contrast, the family of Minkowsky distance works well with this notion because they compute a distance between two points and not two directions.

Furthermore, the intuition behind most terms requires an established notion between events and their attribute. Generally, we can have two notions of this relationship.

We consider the event and its attributes separate entities for the first relationship. This notion is reasonable, as some attributes remain static throughout the process. If we take a loan application process as an example, an applicant's ethnic background does not change regardless of the event. This characteristic can be considered a case attribute, which remains static throughout the process run. This understanding would require us to modify the viability measures, as they treat the activity independently from its attribute values. In other words, if the activities of two events are $\overline{a}$ and $\overline{b}$, but their attribute values are $\binom{2}{3}$ and $\binom{2}{3}$, these events may be seen as more similar than two $\overline{a}$ and $\overline{a}$ with attribute values $\binom{2}{3}$ and $\binom{5}{0}$.

In contrast, a second notion would treat each event as an independent and atomic point in time. Hence, a and b would be considered completely different even if their event attributes are the same. This understanding is also a valid proposition, as you could argue that an event which occurs at nighttime is not the same as a daytime event. Here, the time domain is the main driver of distinction, and the content remains a secondary actor.

All the terms described in the SSDLD follow the second notion. There are two reasons for this decision. First, treating event activities and event attributes separately would further complicate the SSDLD, as we would have to expand the cost structure to account for unchangeable event attributes. Second, the unmodified Damerau-Levenshstein distance applies to discrete sequences, such as textual data with atomic words or characters. By treating each event as a discrete sequence element, we remain faithful to the original function.

3.3 Viability Measure

Earlier, in subsection 2.8, we have discussed what determines *good* counterfactuals. However, we have not introduced our approach to operationalize the notion of *viability*. To recall, a counterfactual is hardly useful, if it is vastly different from the factual example or, if it requires changes that are logically implausible. For instance, if patients are required to vastly change their behavior in many aspects of their life or change their race these counterfactuals are hardly useful

for the patient or a medical professional. We are more interested in what we have to change *at least*. Also, if the counterfactual is, per se, unrealistic or bears no change in outcome, we lack any interest in those counterfactuals, as well. For processes, these issues become even more complicated as they are semi-structured and often multivariate. How we operationalize these criteria is explained in the following.

Similarity-Measure We use a function to compute the distance between the factual sequence and the counterfactual candidates. Here, a low distance corresponds to a small change. For reasons explained earlier (subsection 3.2), we want to consider the structural and feature distances. Henceforth, we use the previously established SSDLD. The similarity distance uses a cost function as specified in Equation 14.

$$\begin{aligned} cost(a_i, b_j) &= L2(a_i, b_j) \\ a_i, b_j &\in \mathbb{R}^d \end{aligned} \tag{14}$$

Here, $dist(x, y)$ is an arbitrary distance metric. i and j are the indices of the sequence elements a and b, respectively. $L2$ denotes the euclidean distance.

Sparsity-Measure Sparsity refers to the number of changes between the factual and counterfactual sequence. We typically want to minimize the number of changes. However, sparsity is hard to measure, as we cannot easily count the changes. There are two reasons why this is the case: First, the compared sequences can have varying lengths. Second, even if they were the same length, the events might not line up, so we can simply count the changes to a feature. Hence, we use the previously established SSDLD to solve this issue. The sparsity distance uses a cost function as specified in Equation 15.

$$\begin{aligned} cost(a_i, b_j) &= \sum_d \mathbb{I}(a_{id} = b_{jd}) \\ a_i, b_j &\in \mathbb{R}^d \end{aligned} \tag{15}$$

Here, $\sum_d \mathbb{I}(a_{id} = b_{jd})$ is an indicator function, that is used to count the number of changes in a vector.

Feasibility-Measure To determine the feasibility of a counterfactual trace, it is important to recognise two components.

First, we have to compute the probability of the sequence of event transitions. This is a difficult task, given the *Open World assumption*. In theory, we cannot know whether or not any event *can* follow after another event. However, if the data is representative of the process dynamics, we can make simplifying assumptions. For instance, we can compute the first-order transition probability

by counting each transition. However, the issue remains that longer sequences tend to have a zero probability if they have never been seen in the data.

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 none of the previous events and features. Meaning, we can model density estimators for every event and use them to determine the likelihood of a set of features.

There are many ways to estimate the density of a dataset. For our purposes, we incorporate the sequential structure of the log data and make simplifying assumptions. First, we consider every activity as a state in the case. Second, each state is dependent on its immediate predecessor and neither on future nor on any states prior to its predecessor. Third, the collection of attributes within an event depends on the activity which emits it. The second assumption is commonly known as *Markov Assumption*. With these assumptions in place, we can model the distribution by knowing the state transition probability and the density to emit a collection of event attributes given the activity.

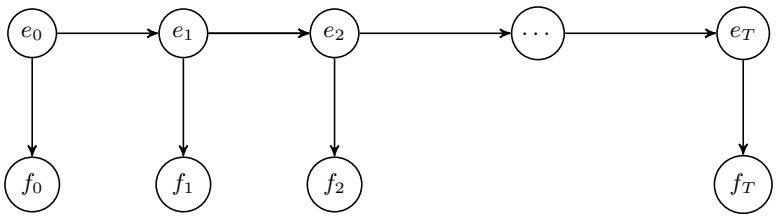


Figure 13: The feasibility model in graphical form. e_t represents an event and f_t the features it emits.

Here, e_t represents the transition from one event state to another. 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. Note that this is the same as the feasibility measure in Equation 16.

$$p\left(e_{0:T}, f_{0:T}\right) = p\left(e_0\right) p\left(f_0 \mid e_0\right) \prod_{1}^{T} p\left(e_t \mid e_{t-1}\right) p\left(f_t \mid e_t\right) \tag{16}$$

To conclude this section, we must stress again that many ways to define feasibility exists. We chose a probabilistic approach. There is an issue with this approach. Shorter sequences naturally have higher probabilities. Hence, we introduce a bias into our viability measure towards short sequences. This bias can be beneficial or detrimental depending on the use case. For instance, a medical process model might favour shorter counterfactual explanations mainly because we want to understand how we can effectively reduce the time of illness. However, if we want to explain a highly standardised manufacturing process that

went wrong in one instance, we would rather keep the counterfactual as close as possible to the factual.

Delta-Measure For this 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**?**. 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*. However, the binary case introduces some noteworthy considerations.

Within this task setting, we have to consider multiple cases. First, the prediction score is typically limited to a domain within 0 and 1, which we can interpret as a probability distribution. Hence, if the model score is 0.6, then the model has the confidence of 60% that the input can be categorised as belonging to class 1. For instance, within a medical process, we could say the model is 75% confident that the patient can be cured. Conversely, there's a 25% confidence that the process instance belongs to class 0. We can make decisions by using a threshold. Typically, this threshold lies at 50%. Hence, we determine that a patient can make decisions by using a threshold. Typically this threshold lies at 50%. Figure 14 illustrates how this threshold behaves given the factual prediction score.

We identify 2 cases:

Case 1: A counterfactual generator *flips* the prediction score to the opposite side of the decision threshold. Then, we achieve our general aim, and the difference between the scores directly indicates the counterfactual's success. If we look at Figure 14, the two quadrants with only positive delta values cover this case.

Case 2: A counterfactual has the same decision as the factual. For instance, when the counterfactual and factual prediction scores for a patient's recovery chance are below 0.5 or above 0.5. Then, we must consider whether the counterfactual predictions score is moving towards the desired outcome or away from it.

– [2.1] If the prediction for the factual decides an outcome of 0, but the predictions score for the counterfactual is even lower, then we did not change the prediction at all. In fact, we increase the chance of the actual factual outcome. That situation is worse than what we desire. For instance, a patient would not want to pursue a counterfactual situation in which his odds of recovery are worse than his current.

– [2.2] In contrast, if a prediction model's score leads to an outcome of 0 but the counterfactual returns a higher prediction score than the factual

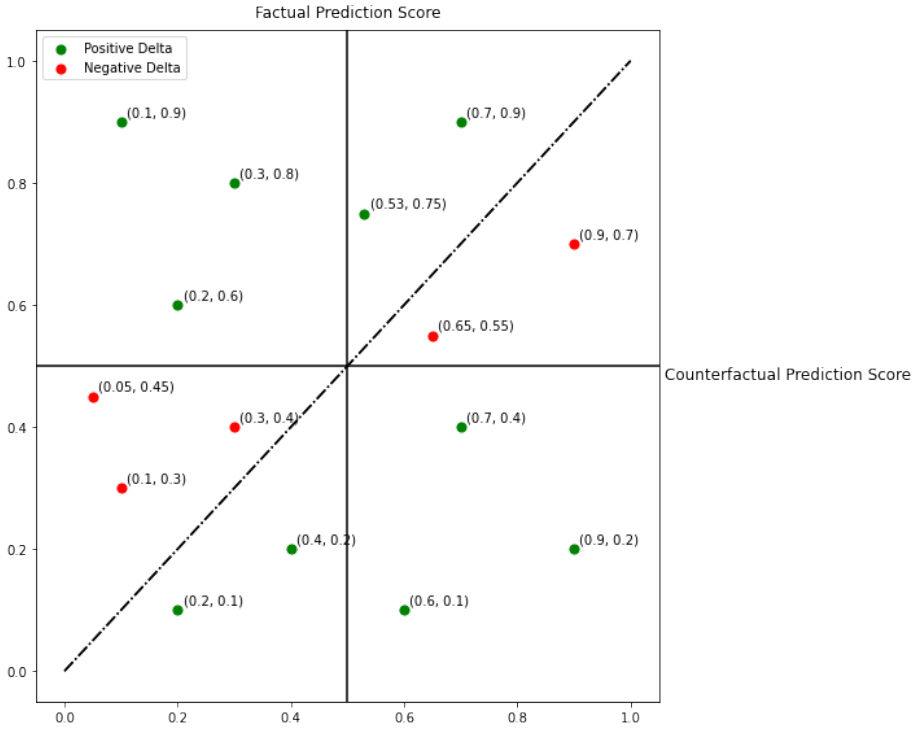


Figure 14: An example of which points yield a positive delta given the factual predictions score. Green means the delta is an improvement. Red points signify a negative improvement.

predictions score, a patient might still be interested in the counterfactual. In some situations, even a small improvement is desirable.

The sub-cases of case 2 go in both ways. Hence, we have to incorporate each case differently in the delta score. The two quadrants in Figure 14 that have positive and negative deltas reflect how we interpret these cases.

$$delta = \begin{cases} |p(o|s^*) - p(o|s)| & \text{if } p(o|s) > 0.5 \ \& \ p(o|s) > p(o|s^*) \\ -|p(o|s^*) - p(o|s)| & \text{if } p(o|s) > 0.5 \ \& \ p(o|s) < p(o|s^*) \\ |p(o|s^*) - p(o|s)| & \text{if } p(o|s) < 0.5 \ \& \ p(o|s) > p(o|s^*) \\ -|p(o|s^*) - p(o|s)| & \text{if } p(o|s) < 0.5 \ \& \ p(o|s) < p(o|s^*) \end{cases} \tag{17}$$

Discussion Given the current viability measure, we can already determine the optimal counterfactual:

> The optimal counterfactual flips a model's strongly expected factual outcome to the desired outcome, maintaining the same trajectory as the factual in terms of events, with minimal changes in its event attributes, while remaining feasible according to the data.

The elements that fulfil these criteria make up the Pareto surface of this multi-valued viability function. If each value is scaled with a range between 0 and 1, the theoretical ceiling is 4. This value is only possible if we flip a factual sequence's outcome without changing it. As this is naturally impossible for deterministic model predictions, the viability has to be lower than 4.

Furthermore, we can already postulate that a viability of 2 is a critical threshold. If we score the viability of a factual against itself, a normalised sparsity and similarity value have to be at its maximal value of 1. In contrast, the improvement has to be 0. The feasibility is 0 depending on whether the factual were used to estimate the data distribution or not. With these observations in mind, we determine that any counterfactual with a viability of at least 2 is already better than the factual.

Differences to DiCE4EL ? follow a similar pattern of assessing the quality of their counterfactuals. The authors also focus on similarity, sparsity, feasibility and likelihood-improvement. However, they incorporate and operationalise them differently. Their approach is most apparent in their loss function.

Similarity: Similar to our approach, the authors use a distance function and optimise it using gradient descent. They evaluate the quality of their counterfactuals using the same function[8]. However, we use a modified Damerau-Levenshstein distance algorithm to incorporate structural differences such as the sequence lengths or transposed events.

[7] Obviously, the domain of the application decides where this threshold lies. One can always argue that confidence of 51% is close to randomly guessing.

[8] They call it proximity during evaluation

Sparsity: The DiCE4EL approach does not consider this.

Feasibility: This quality criterion is embodied by two loss functions: Category loss and scenario loss. The category loss ensures that categorical variables remain categorical after generation. The scenario loss adds emphasis on only generating counterfactuals that are in the event log. Unlike our probabilistic interpretation, they treat the existence of feasible counterfactuals as a binary criterion[9].

Likelihood: Similar to the authors' scenario loss, they treat the improvement of a class as a binary state. Either the counterfactual changes the model's prediction of the desired outcome, or it does not.

The details of each criterion's operationalisation are explained in **?**. By assessing their interpretation of quality criteria, we see the clear distinction between our approach and the approach of **?**.

First, their viability measure decisively discourages the generation of counterfactuals that are not present in the dataset. In contrast, our approach treats this aspect as a soft constraint.

Second, while our approach acknowledges general improvements in likelihoods, DiCE4EL treats all counterfactuals that do not lead to better desires as detrimental solutions. However, one can argue that improving the likelihood of the desired outcome just slightly is already beneficial.

Third, **?** do not optimise sparsity, while we include it within our framework. One can argue that similarity automatically incorporates aspects of sparsity, but we disagree with this notion. We can see this by employing a simple example: Let factual A have features signifying the biological sex (binary), the income (normalised) and the age (normalised) $\begin{pmatrix}1\\1\\1\end{pmatrix}$ as event attributes. Let counterfactual B have the same event attributes with $\begin{pmatrix}0\\1\\1\end{pmatrix}$. Let's assume the distance measure uses the L1-norm. Then, a counterfactual C with event attributes $\begin{pmatrix}1\\0.5\\0.5\end{pmatrix}$ would have the same distance to factual A as B has. However, C requires the change of two event attributes, while B only requires 1 change. In a scenario in which we seek to reduce the number of edits, B is preferable to C, regardless of the distance to A.

The last difference stems from the fact that **?** do not include structural sequence characteristics in their similarity measure.

[9] They call it plausibility during evaluation

3.4 Prediction Model: LSTM

The architecture of the prediction model is shown in Figure 15. The model architecture is inspired by **?**. However, we do not separate the input into dynamic and static features.

One input consists of a 2-dimensional event tensor containing integers. The second input is a 3-dimensional tensor containing the remaining feature attributes. The first dimension in each layer represents the variable batch size, and *None* acts as a placeholder.

The next layer is primarily concerned with preparing the full vector representation. We encode each activity in the sequence into a vector space. We chose a dense-vector representation instead of a one-hot representation. We also create positional embeddings. Then we concatenate the activity embedding, positional embedding and the event attribute representation to a final vector representation for the event that occurred.

Afterwards, we pass the tensor through a LSTM module. We use the output of the last step to predict the outcome of a sequence using a fully connected neural network layer with a sigmoid activation as this is a binary classification task.

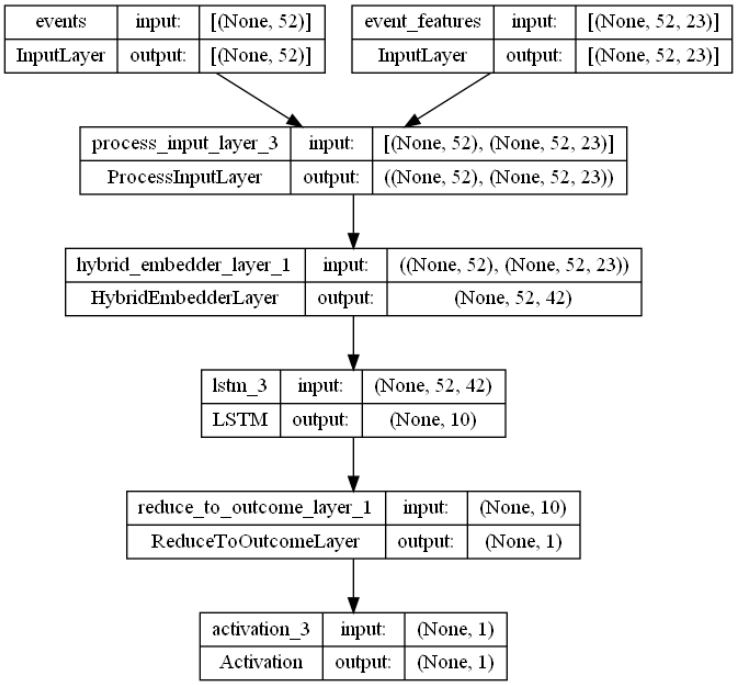


Figure 15: The different components of the LSTM architecture. Each element contains information about the input and output of a layer. None is a placeholder for the batch size.

3.5 Counterfactual Generators

Generative Model: Evolutionary Algorithm We introduced most operator types in subsection 2.15. 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.

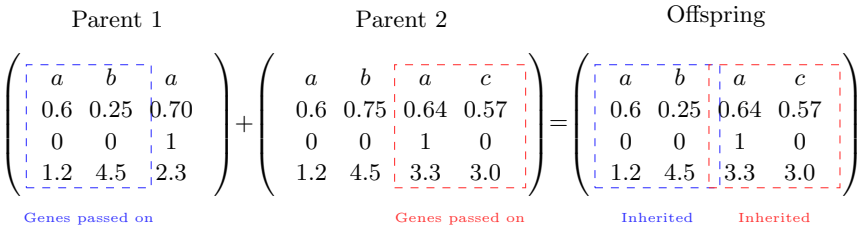


Figure 16: A newly generated offspring inheriting genes in the form of activities and event attributes from both parents.

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 Figure 16. The offspring is subject to mutations. Then, we evaluate the new population and repeat the procedure until a termination condition is reached. We can optimise the viability measure established in subsection 3.3.

Operators We implemented several different evolutionary operators. Each one belongs to one of five categories. The categories are initiation, selection, crossing, mutation and recombination.

Inititation

RI: The *Random-Initiation* generates an initial population entirely randomly. The activity is just a randomly chosen integer, and each event attribute is drawn from a normal distribution.

SBI: The *Sampling-Based-Initiation* generates an initial population by sampling from a data distribution estimated from the data directly.

CBI: *Case-Based-Initiation* samples individuals from a subset of the Log (*Case-Based-Initiation*). Those individuals are used to initiate the population.

The initiation procedure might be the most important operation in terms of computation time. The reason is that we expect more sophisticated initiation procedures like Sampling-Based-Initiation and Case-Based initiation to start with much higher viability and reach their convergence much sooner.

Algorithm 2 The basic structure of an evolutionary algorithm.

Require: factual
Require: configuration
Require: sample-size
Require: population-size
Require: mutation-rate
Require: termination-point
Ensure: The result is the final counterfactual sequences
  $counterfactuals \leftarrow initialize$(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

Selection

RWI: *Roulette-Wheel-Selection* Selects individuals randomly. However, we compute each individual's fitness in the population and choose a random sample proportionate to their fitness values. Hence, sequences with high fitness values have a higher chance to crossover their genes, while fewer fit individuals also occasionally get their chance.

TS: *Tournament-Selection* compares two or more individuals and selects a winner among them. We choose two competing individuals we randomly sample with replacement. Hence, some individuals have multiple chances to compete. The competing individuals are randomly chosen as winners in proportion to their viability. Hence, if an individual with a viability of 3 is pitted against an individual with a viability of 1, then there's a 3:1 chance that the first individual will move on to crossover its genes.

ES: *Elitism-Selection* selects each individual solely on their fitness. In other words, only a top-k amount of individuals are selected for the next operation. There is no chance for weaker individuals to succeed. This approach is deterministic and, therefore, subject to getting stuck in local minima.

Crossing

UCx: We can uniformly choose a fraction of genes of one individual (*Parent 1*) and overwrite the respective genes of another individual (*Parent 2*). The result is a new individual. We call that (*Uniform-Crossover*). Figure 17 shows a simple schematic example. By repeating this process in the opposite direction, we create two new offsprings that share both individuals' characteristics. The number of inherited genes can be adjusted using a rate factor. The number of selected positions is determined by a crossing rate between 0 and 1. The higher the crossover rate, the higher the risk of disrupting possible sequences.

If we turn to Figure 17 again, we see how the second child has 2 repeating genes at the end. If a process does not allow the transition from *activity 8* to another *activity 8*, then the entire process instance becomes infeasible.

OPC: *One-Point-Crossing* is an approach suitable for sequential data of the same lengths. We can choose a point in the sequence and pass on genes of *Parent 1* onto the *Parent 2* from that point onwards and backwards (*One-Point-Crossover*). Thus, creating two new offsprings as depicted in Figure 18.

TPC: *Two-Point-Crossing* resembles its single-point counterpart. However, this time, we choose two points in the sequence and pass on the overlap, and the disjoints to generate two new offsprings. Again, Figure 19 describes the procedure visually. We can increase the number of crossover points even further. However, this increase comes at the risk of disrupting sequential dependencies.

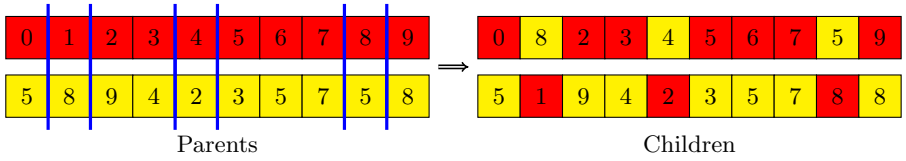


Figure 17: A crossing process of uniformly applying characteristics of one sequence to another.

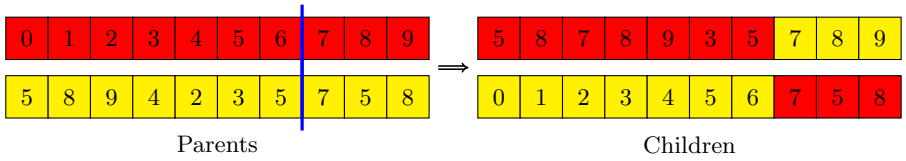


Figure 18: A One-Point example of applying characteristics of one sequence to another using one split point

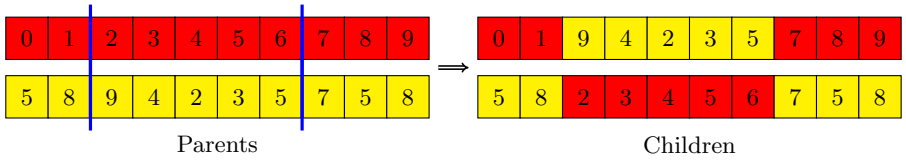


Figure 19: The process of applying characteristics of one sequence to another using two split points.

Mutation Before elaborating on the details, we have to briefly discuss four modification types we can apply to data sequences. Reminiscent of edit distances, which were introduced earlier in this thesis, we can either insert, delete, change and transposition a gene. These edit types are the fundamental edits we use to modify sequences. For a visual explanation of each edit-type we refer again to Figure 9 in subsection 2.14.

However, we can change the extent to which each operation is applied over the sequence. We call these parameters *mutation-rates*. In other words, if the delete rate equals 1, every individual experiences a modification which results in the deletion of a step. The same applies to other edit types.

As we chose the hybrid encoding scheme, we must define what an insert or a change means for the data. Aside from changing the activity, we also have to choose a new set of data attributes. This necessity requires defining two ways to produce them. We can either choose the features randomly or choose to take a more sophisticated approach.

RM: *Random-Mutation* creates entirely random features for inserts and substitution. The activity is just a randomly chosen integer, and each event attribute is drawn from a normal distribution.

SBM: *Sampling-Based-Mutation* creates sampled features based on data distribution for inserts and substitution. We can simplify the approach by invoking the *Markov Assumption* and sample the feature attributes given the activity in question (*Sample-Based-Mutation*).

There are still two noteworthy topics to discuss.

First, these edit types are disputable. One can argue that change and transpose are just restricted versions of delete-insert compositions. For instance, if we want to change the activity *Buy-Order* with *Postpone-Order* at timestep 4, we can first delete *Buy-Order* and insert *Postpone-Order* at the same place. Similar holds for transpositions, albeit more complex. Hence, these operations naturally occur over repeated iterations in an evolutionary algorithm. However, these operations follow the structure of established edit distances like the Damerau-Levenshstein distance. Furthermore, they allow us to restrict their effects efficiently. For instance, we can restrict delete operations to steps that are not padding steps. In contrast, insert operations can be limited to padding steps only.

Second, we can apply different edit rates for each edit type. However, this adds additional complexity and increases the search space for hyperparameters.

Third, using the random sampler automatically disrupts the feasibility for most offspring if either of the two conditions is met. First, if the log contains categorical/binary event attributes, Gaussian samples cannot reflect these types of random variables. Second, if the vector space with which event attributes are represented is too large, it becomes less and less likely to sample something within the correct bounds. For instance, let us again consider the example on 2. However, instead of having 3 event attributes, each event had 100. Then, it becomes extremely difficult to randomly sample a set that fits the event attribute vectors.

Recombination

FSR: *Fittest-Survivor-Recombination* strictly determines the survivors among the mutated offsprings and the current population by sorting them in terms of viability. The operator guarantees that the population size remains the same across all iterations. Nonetheless, this approach is subject to getting stuck in local maxima. This is mainly because this recombination scheme does not allow for the exploration of unfavourable solutions that may evolve into better ones in the long run.

BBR: *Best-of-Breed-Recombination* Determines mutants that are better than the average within their generation and adds them to the population. The operator only removes individuals after the maximum population size is reached. Afterwards, the worst individuals are removed to make way for new individuals.

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. We refer to this method as *Ranked-Recombination*. In our order, we choose to favour feasibility first. Feasibility values are by far the lowest as they are joint probability values that become smaller with every multiplication. Second, we favour delta, then sparsity and at last similarity. Mainly because it is more important to flip the outcome than to change as little as possible, and it is more important to change as little as event attributes as possible than to become more similar to the factual.

Naming-Conventions We use abbreviations to refer to them in figures, tables, appendices, etc. For instance, *CBI-RWS-OPC-RM-RR* refers to an evolutionary operator configuration that samples its initial population from the data, probabilistically samples parents based on their fitness, crosses them on one point and so on. For the *Uniform-Crossing* operator, we additionally indicate its crossing rate using a number. For instance, *CBI-RWS-UC3-RM-RR* is a model using the *Uniform-Crossing* 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 54 to combine all operators. Depending on each operator combination, we might see very different behaviours. For instance, it is obvious that initiating the population with a random set of values can hardly converge at the same speed as a model which leverages case examples. Similarly, selecting only the fittest individuals is heavily prone to local optima issues. 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*. Eventually, most correctly implemented evolutionary algorithms will converge to a local optimum. Especially if only the best individuals are allowed to cross over. If we choose the termination point too early, the generated individual most likely underperforms. In contrast, selecting a termination point too far in the future might yield optimal results at the cost of time performance. Furthermore, the existence of local optima may result in very similar solutions in the end. Optimally, we find a termination point, which acts as a reasonable middle point.

The *mutation rate* is another hyperparameter. It signifies how much a child can differ from its parent. Again, choosing a rate that is too low does not explore the space as much as it could. In turn, a mutation rate that is too high significantly reduces the chance to converge. The optimal mutation rate allows for exploring novel solutions without immediately pursuing suboptimal solution spaces. Our case is special, as we have four different mutation rates to consider. The change rate, the insertion rate, the deletion rate and the transposition rate. Naturally, these strongly interact. For instance, if the deletion rate is higher than the insertion rate, there's a high chance that the sequence will be shorter, if not 0, at the end of its iterative cycles. Mainly because we remove more events than we introduce. However, we cannot assume this behaviour across the board as other hyperparameters interplay. Most prominently, the fitness function. Let us assume we have a high insertion rate, but the fitness function rewards shorter sequences. Subsequently, both factors cancel each other out. Hence, the only way to determine the best set of mutation rates requires an extensive search.

Baseline Model: Random Generator This model acts as one of the baseline methods. Here, we generate a random sequence of events. Afterwards we generate event attributes, randomly. This approach is reasonably fast, but expected to perform poorly.

As explained earlier, any possible sequence of events becomes more and more unlikely the longer the sequence is. One generally has a chance of $\frac{\#UniqueTraces}{A^T}$ to randomly find an event sequence, that is the process log. The chances decrease even more if one also generates event attributes randomly. Therefore, we expect most models to perform better that this model. If a model happens to be worse, it would indicate that it is more likely to just randomly pick numbers and get a better counterfactual.

Baseline Model: Sample-Based Generator This baseline resembles the random baseline. However, we use the feasibility model to guide the random search for the generation of counterfactuals. We refer to the model specified in Equation 16. The sampling procedure utilises the model structure for the sampling process. We first generate a random seed of possible starting events ($p\left(e_0\right)$). Afterwards, we randomly sample subsequent events by iteratively sampling new activities according to the transition probabilities we gathered from the data ($\prod_1^T p\left(e_t \mid e_{t-1}\right)$). Given the sequence, we simply sample the features per event from $p\left(f_t \mid e_t\right)$.

Baseline Model: Case-Based Generator Case-based techniques leverage the data by using example instances. The idea is to find suitable candidates that best fulfil the counterfactual criteria. We treat this model as a baseline. Therefore, we keep this approach simple. We find candidates by searching by randomly sampling cases from the log and then evaluating them using the viability measure.

Inherently, this approach is restricted by the *representativeness* of the data. It is not possible to generate counterfactuals that have not been seen before. This method works for cases where the data hold enough information about the process. If this condition is not met, it is impossible to produce suitable candidates.

Note that this approach will automatically fulfil the criterion of being feasible, as the counterfactuals are drawn from the log directly. Hence, we expect their feasibility to be often higher than other methods.

4 Evaluation

In this chapter, we discuss the datasets, the preprocessing pipeline, and the final representation for each of the algorithms. All the experiments were run on a Windows machine with 12 processor cores (Intel Core i7-9750H CPU 2.60GHz) and 32 GB Ram. The main programming language was python. The models were mostly developed with Tensorflow**?** and NumPy**?**. We provide the full code on Github**?**. There, you will find instructions on how to install and run the experiments yourself.

4.1 Datasets

In this thesis, we use ten publicly available datasets. All of the datasets were taken from **?**. Each dataset consists of log data and contains labels that signify a process's outcome. We focus on binary outcome predictions. Hence, each dataset provides a binary label for each process instance that indicates the outcome of that process instance.

BPIC12: The first dataset is the popular BPIC12 dataset. This dataset was originally published for the Business Process Intelligence Conference and contains events for a loan application process. Each case relates to one loan application process and can be accepted (regular) or cancelled (deviant).

Sepsis: The next dataset is the Sepsis-Dataset. It is a medical dataset that records patients with life-threatening sepsis conditions. The outcome describes whether the patient returns to the emergency room within 28 days from initial discharge.

TrafficFines: Third, we apply our approach to the Traffic-Fines-Dataset. This dataset contains events related to notifications sent related to a fine. The dataset originates in a log from an Italian local police force.

DiCE4EL: Lastly, we include a variation of the BPIC dataset. It is the dataset which was used by **?**. The difference between this dataset and the original dataset is two-fold. First, **?** omit most variables except two. Second, it is primarily designed for next-activity prediction and not outcome prediction. We modified the dataset to fit the outcome prediction model.

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

Table 1: All datasets used within the evaluation. DiCE4EL is used for the qualitative evaluation, and the remaining are used for quantitative evaluation purposes.

For more information about these datasets we refer to **?**'s comparative study**?**. We list all the important descriptive statistics in Table 1.

	precision			recall			f1-score			support		
Subset	test	training	validation	test	training	validation	test	training	validation	test	training	validation
Dataset												
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

Table 2: The evaluation metrics for the prediction component on all datasets. Includes precision, recall and f1 score for test, training and validation data.

We list the predictions of our prediction component in Table 2. The F1-Scores on the test sets are generally higher for the BPIC dataset. Furthermore, in the case of the BPIC datasets, the length of the dataset determines whether the prediction model always predicts correctly or not. It is fair to assume that the length of a loan application process determines the chance of getting rejected or not.

4.2 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. The test set will act as our primary source for evaluating factuals entirely unknown to the model.

We split the training set into a training set and a validation set. This procedure is a common tactic to employ model selection techniques. In other words, Each dataset is split into 25% Test and 75 remaining, and from the remaining, we take 25% validation and 75% training data.

First, 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 run time 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. For a time format, we encode all information from seconds to a year. If the complete log occurs within one time unit only, e.g. every event that happened within a year, drop the extracted column—afterwards, we standard scale all remaining time features.

Each categorical variable is converted using binary encoding. Binary encoding is very similar to one-hot encoding. However, it is still distinct. The binary encoding uses a binary representation for each class encoded. This representation saves a lot of space as binary encoded variables are less sparse than one-hot encoded variables.

We also add an offset of 1 to binary and categorical columns to introduce a symbol which represents padding in the sequence. All numerical columns have a zero mean and a standard deviation of 1.

We omit the case id, the activity and the label column from this preprocessing procedure for reasons explained in subsection 2.12. The activity is label-encoded. Hence, every category is assigned to a unique integer. The label column is binary encoded, as we focus on outcome prediction.

4.3 Experimental Setup

As mentioned in subsection 2.8, counterfactual generation is notorious for lacking a standardised evaluation procedure. Nonetheless, we attempt to address our research questions with the following experiments.

Experiment 1: Model Selection Before comparing models, we reduce the number of possible models that *can* be compared. In terms of operators, we introduced three initiators, three selectors, three crossers, two mutators and three recombiners. Hence, comparing all possible evolutionary operator combinations requires examining a total of 54 different models. Furthermore, each model has hyperparameters we have to define, too. Therefore, the first set of experiments is dedicated to choosing among a subset of operator combinations and selecting appropriate hyperparameters.

First, we compute all possible configurations without changing any hyperparameter. We refer to each unique operator combination as a model configuration to avoid confusion. For instance, one model configuration would consist of *a*

Sampling-Based-Initiator, an Elitism-Selector, a One-Point-Crosser, Sampling-Based-Mutator and a Fittest-Survivor-Recombiner. For the sake of brevity, we refer to a specific model configuration in terms of its abbreviated operators. For instance, the earlier example is denoted as *SBI-ES-OPC-SBM-FSR*.

Afterwards, we explore the hyperparameters of the model. We start with the termination point. Hence, we want to examine the effects of the iterative cycles that each evolutionary algorithm will run. The goal is to find a stopping criterion which yields reasonably good counterfactuals while reducing the computation time. We will only consider the number of iterative cycles as a stopping criterion. We refer to each different criterion as a termination point. Hence, a termination point at 5 means the algorithm will not proceed to optimise its results further after reaching the fifth iteration. We can choose the termination point by inspecting how the average population viability evolves across each cycle. We keep every other experimental setting as established beforehand.

For determining the mutation rate for every mutation type, we choose the best evolutionary algorithm and run the configuration with six rates from 0 to 0.5 in steps of 0.1. We omit everything beyond 0.5 to preserve information about the parent. For instance, if we use a change rate of 0.9, we mutate 90% of the genes the child inherited. This would defeat the purpose of evolving better counterfactuals through breeding. We use the termination point established in the prior experiment. We keep every other experimental setting as set beforehand.

After executing all preliminary experiments, we choose the evolutionary generators and compare them with all baseline models in all subsequent experiments.

Experiment 2: Comparing with Baseline Generators In this experiment, we assess the viability of all the chosen evolutionary and baseline generators. For this purpose, we sample 10 factuals and use the models to generate 50 counterfactuals. We determine the median viability across the counterfactuals. With this experiment, we show that a model that optimises counterfactual quality criteria produces better results than models that do not. Hence, we expect the evolutionary algorithm to perform best, as it can directly optimise multiple viability criteria. We move on with the best-performing models.
Under *RQ1-H1* and *RQ1-H2* we expect the evolutionary algorithms to outperform the baselines when it comes to viability.

Experiment 3: Comparing with alternative Literature The model comparison is not enough to establish the validity of our solution, as we defined the viability measure ourselves. Therefore, we also assess each model based on the evaluation criteria of an alternative work. More precisely, we quantify the viability of our models using the metrics employed by **?**. Hence, we measure the sparsity by computing the average Levenshstein difference and proximity using the L2-Norm. Furthermore, we compute the average intra-list-diversity and plausibility

Similar to **?**, we focus on the *activities* that are generated by each model and its accompanying *resource* event-attribute. For diversity and plausibility,

we remain close to the original evaluation protocol by **?** as we also treat each counterfactual trace sequence as a symbol. Hence, a sequence ABC is treated as a completely different symbol than $ABCD$.

The goal is to show that models, which optimise viability criteria, perform better, even if viability is assessed differently, as stated in *RQ2-H1* of our research question (subsection 1.4).

Experiment 4: Qualitative Assessment For the last assessment, we follow **?**'s procedure of assessing the models qualitatively. We use the dataset as the authors do. However, as we focus on outcome prediction, we attempt to answer one of two questions:

1. *what would I have had to change to prevent the cancellation/rejection of the loan application process*
2. *what would I have had to change to cause a cancelled/rejected loan application process*

The goal is to show that the results are viable despite not having a standardised protocol to measure their viability.

5 Results

This chapter presents the results of each evaluation step. Furthermore, we analyse the results.

5.1 Experiment 1: Model Selection

Model Configuration

Results As there are many ways to combine each configuration, we select a few configurations by examining them through simulations.

The set of model-configuration contains 54 elements. We choose to run each model configuration for 100 evolution cycles. For all model configurations, we use the same four factual process instances randomly sampled from the test set. We ensure that the outcomes of these factuals are evenly divided. We decide to limit the population size to a maximum of 1000 counterfactuals. Within each evolutionary cycle, we generate 100 new offspring. We keep the mutation rate at 0.01 for each mutation type. Hence, across all cases that are mutated, the algorithm deletes, inserts, and changes 1% of events per cycle. We collect the mean viability and its components across the iterative cycles of the model.

Figure 20 shows the bottom and top-5 model configurations based on the viability after the final iterative cycle. We also show how the viability evolves for each iteration. The results reveal a couple of patterns. First, all top-5 algorithms

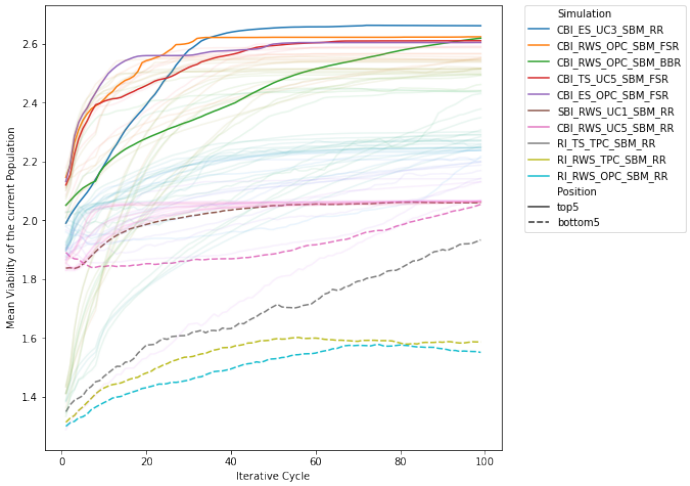


Figure 20: This figure shows the average viability of the five best and worst model configurations. The x-axis shows how the viability evolves for each evolutionary cycle.

use either *Case-Based-Initiator* as initiation operation. In contrast, the bottom-5 use *Random-Initiator* as initialisation. Hence, the initialisation appears to be majorly important for the algorithm. The complete table of results is in **??**.

In Figure 21, we see the effects of each operator type.

Starting with some commonalities, across operator-type and measure, the figure shows that the initiator heavily determines the starting point for each measure. For instance, the *Random-Initiator* starts well below the other initiator operations regarding sparsity and similarity. Similarly, most of the *RI-x* model configurations begin at much lower viability than the other model configurations. This pattern is evident in Figure 20.

Another noteworthy general observation is the delta measure. Here, we see a movement towards the highest possible delta value for each operator type. Hence, most configurations are capable of changing the source outcome to the desired outcome.

Regarding feasibility, Figure 21 shows an increase for most operators. This is especially true if the operator has a random component or if it optimises for feasibility. Similar holds for recombination with *Fittest-Survivor-Recombiner*. The feasibility has not reached convergence yet. In many cases, the feasibility monotonously increases.

Among the top-5 *CBI-ES-OPC-SBM-FSR* grows the fastest in terms of viability and reaches convergence the earliest. On the opposite side, we find *CBI-RWS-OPC-SBM-BBR* to have the slowest growth. However, it is also the only one not reaching convergence at that point.

When it comes to the crossing operation, the results indicate that the differences between *One-Point-Crosser* and *Two-Point-Crosser* are inconclusive for all viability measures except feasibility. One can explain that by noting that both

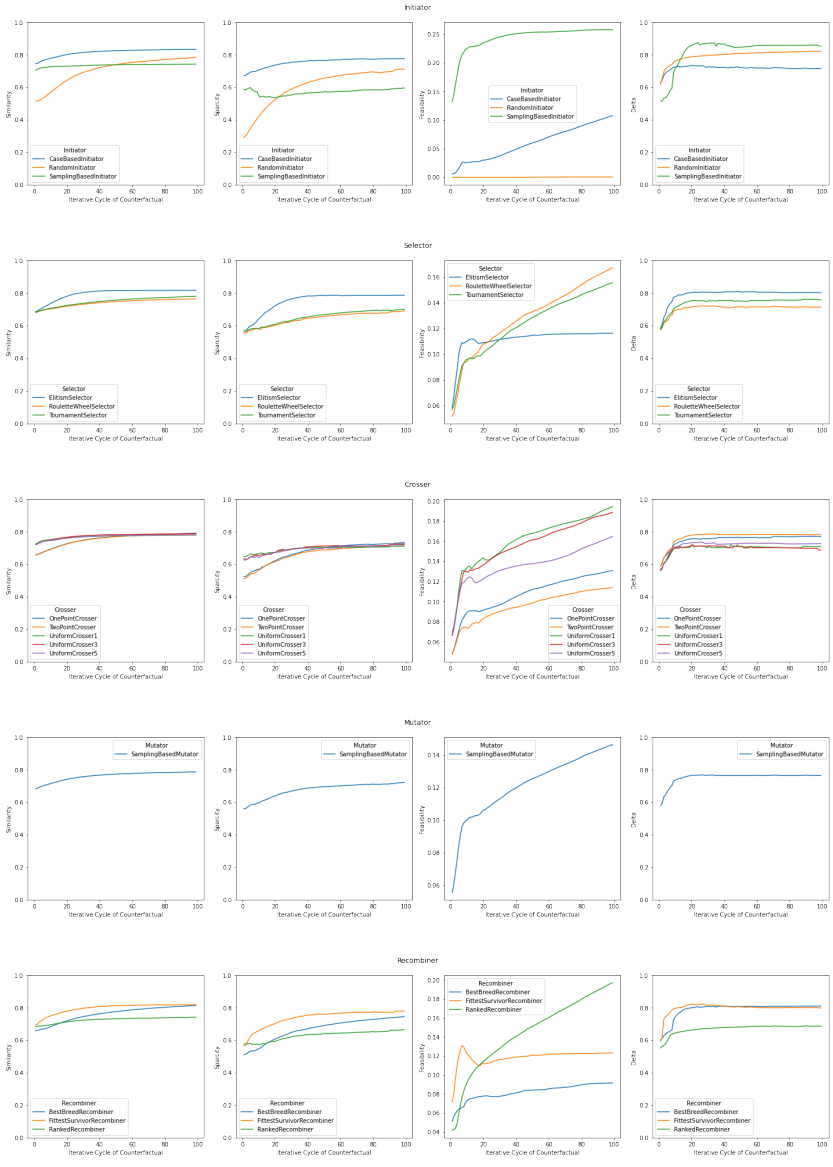


Figure 21: The evolution of each viability measure over the entire span of iterative cycles. Each figure adjust the respective operator type by taking the average over all other operator types.

operations are very similar. However, cutting the sequence only once produces fewer impossible sequences for the child sequences.

Discussion The results show us that the initiation procedure heavily determines the starting point of the algorithm. Hence, this result is hardly surprising. We have discussed the reasons in section 3.5. Namely, more sophisticated methods than random initiation can heavily influence the starting point of the evolutionary algorithm and determine how fast the algorithm reaches convergence.

Interestingly, among the top-5 configurations, only the 5th operation has the *Elitism-Selector* and the *Fittest-Survivor-Recombiner*. Both operators heavily focus on deterministic selections of the very best individuals. The fact that only one of these approaches reached the top tells us that this combination is naturally prone to local maxima. We can also see how much faster it ran and converged at its highest viability. Therefore, *CBI-RWS-OPC-SBM-BBR* is much more interesting as it did not converge after 100 iterative cycles. Hence, it is likely it may reach higher viability scores if we choose to let it run longer.

It is equally interesting that the best model turned out to be the one that sorts the individuals based on a given sorting order. As we chose the order in favour of the least impactful viability component (feasibility), this may suggest that ranked sorting may act as a suitable balancing mechanism.

The monotonous increase of the feasibility may have two possible reasons. Either, this behaviour displays the bias within the feasibility component. We mentioned that the feasibility is biased towards shorter sequences. Hence, the feasibility might increase until only one event is left. Therefore, there might not even be a convergence. Another reason could be that the more dominant viability components are optimised first and, afterwards, the feasibility. Hence, after 100 iterations, there is still much room to improve. In other words, we would have to increase the termination point before encountering convergence. The results with regards to the recombiners provide a clue. Here, we see that the *Fittest-Survivor-Recombiner* and *Best-Breed-Recombiner* do converge on feasibility, while the Ranked-Recombiner does not. In other words, we lose a lot of potential because the algorithm prioritises other components.

Model Termination Point

Results For the experiment, we chose a termination point of *200* which is twice the length of the previous simulation. We keep the mutation rate at *0.01* for each mutation type. The remaining procedure follows the process described in section 5.1.

In Figure 22, we see a general increase in viability for each termination point. It shows that increasing the termination point also yields better results at the end of the generation process. We see that *CBI-ES-UC3-SBM-RR* returns the best results in the shortest time span. The model converges after roughly 50 iterative cycles. *CBI-RWS-OPC-SBM-BBR* appears to have not reached convergence. The

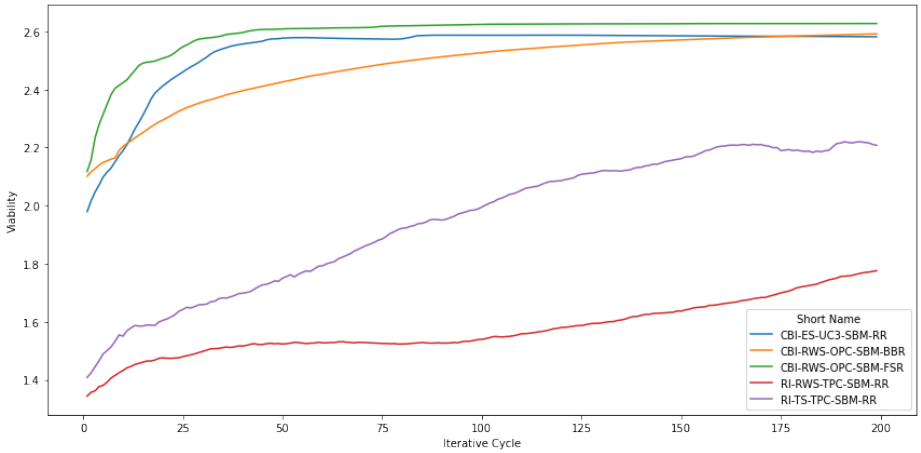


Figure 22: This figure shows the viability across the iteration cycles.

randomly initiated models have not reached convergence as well. However, they remain far below models that use a more sophisticated method to initialise their population.

Figure 23 shows a decomposed view on how the viability measure evolves. Furthermore, we show the average amount of events within a generated counterfactual. In terms of similarity and sparsity, all models behave similarly. This is no surprise as both measures are inherently interlinked. We see that the randomly initiated models (RI-x) decrease the number of events they generate. Case-based initiated models appear to gain more viability slightly. Although, *CBI-RWS-OPC-SBN-BBR* appears that reaches its saturation point significantly later (*100*th cycle). Interestingly, the *CBI-RWS-OPC-SBM-BBR* model struggles to maintain feasibility and collapses to near 0 after the *100th* iterative cycle. Another surprise is the steep ascension of the only model that uses tournament selection (*RI-TS-TPC-SBM-RR*) towards the end of the generation process. The model even overtakes the model that leads the model configurations in terms of *viability*. Furthermore, we see that *CBI-ES-UC-SBM-RR* has the highest feasibility among all models. However, it also quickly converges after 50 iterative cycles.

Discussion The results are not surprising. The longer the algorithm runs, the closer it gets to a local minimum. We expect every evolutionary algorithm to converge at some point, as only the best within the population are chosen for the next iteration. If the model does not include enough non-deterministic components, the results collapse to one optimal case in terms of structure. Hence, the counterfactual activities remain unchanged for the rest of the generation process. The events ratio should optimally approach a number around 0.5 if the factuals are evenly distributed in length. All model configurations seemingly follow this trajectory. However, models (*RI-TS-TPC-SBM-RR*) falls below this level. This

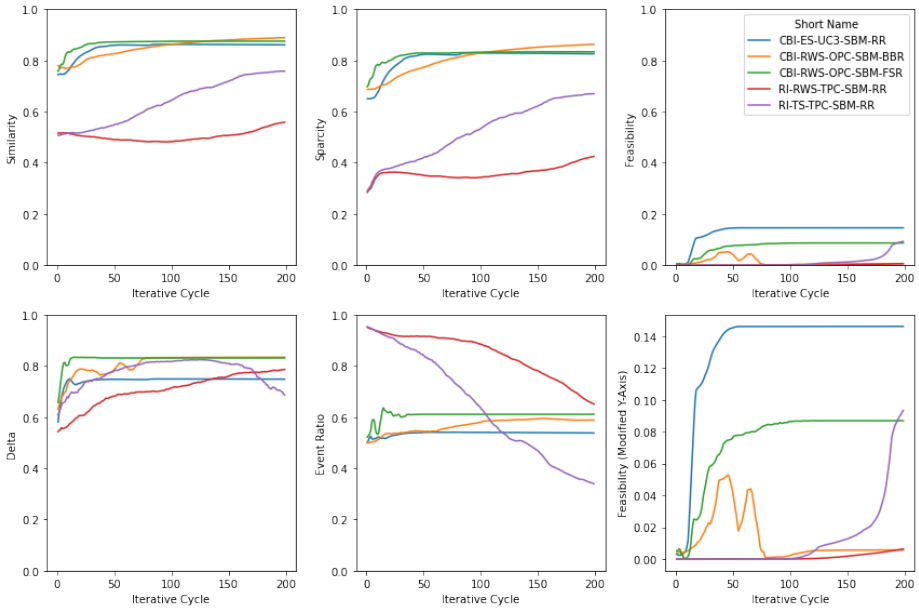


Figure 23: This figure shows the remaining measure components. Additionally, we show the ratio of events within the population. We also show a magnified version of the feasibility measure.

coincides with its sharp rise in feasibility. We assume this behaviour relates to a bias of the feasibility measure towards shorter sequences. The rise and decline of *CBI-RWS-OPC-SBM-BBR* shortly before overtaking all other models in terms of similarity and sparsity indicate a trade-off between how close the counterfactual is to the factual and how feasible it is.

For the following experiments, we use **50** as a termination point. It appears to be a reasonable point in which most models reach their highest viability yield and have not converged yet. We do not seek convergence, as we want to maintain the diversity of our counterfactuals.

Model Parameters

Results As explained earlier, we run the same configuration as previously established for this simulation. However, we vary the rate with which we apply a mutation type this time.
As we can see in Figure 24, a mutation rate of 0 yields better results on average. We are suggesting that mutating the children might impede the model. For model configurations that use the Fittest-Survivor-Recombination, we observe a sharp pattern of convergence before the 10th iterative cycle.

Figure 25 reveals the reason for this behaviour. In all plots of a sharp change right before the 10th iterative cycle. However, the feasibility measure also displays a sudden stop of improvement for all mutation rates except 0.0 and 0.4.

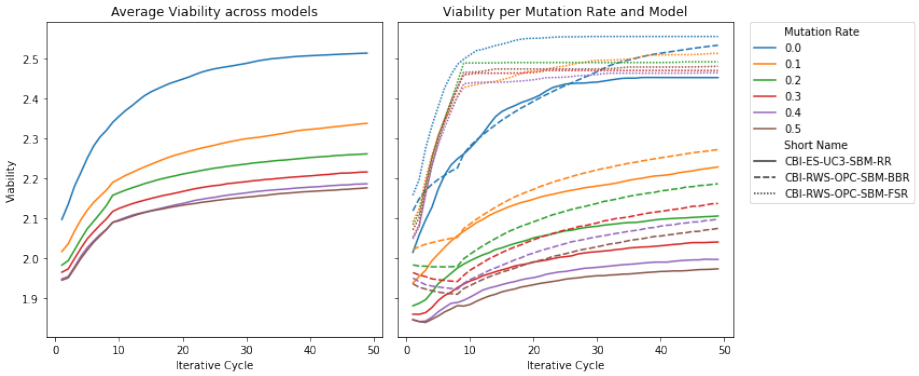


Figure 24: This figure shows the viability for each model and mutation rate per iterative cycle. The first plot shows the average across models. The second figure shows the same information per model. The x-axis shows how the viability evolves for each evolutionary cycle. The colour indicates the mutation rate. The line-type marks each model tested.

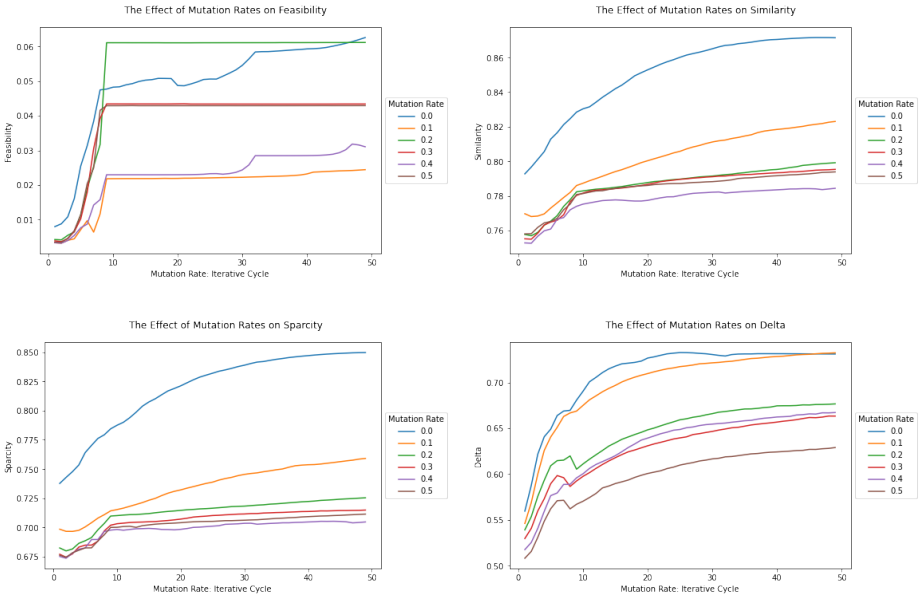


Figure 25: Shows all components of the viability measure.

These exceptions also change their growth rate but improve shortly after the 30th iterative cycle. The figure also shows that a mutation rate of 0.2 reaches the highest feasibility among the other edit rates. However, after 48 cycles, the mutation rate of 0.0 overtakes 0.2.

Discussion While it is expected that every rate configuration eventually converges towards an optimal value, it remains surprising that most configurations suddenly converge around the 10th iteration. There are several possible reasons for this phenomenon. As the viability measure incorporates structural and event-related information, we assume that the algorithm first focuses on finding a structural optimum.

Hence, the algorithm first prioritizes finding the best sequence in terms of activities. After finding an activity sequence, the model primarily focuses on improving the event attributes. Another explanation could be the ratio between the number of generated children and the population threshold. In this experiment, we generated 200 new children while limiting the population size to 1000.

With these observations in mind, we choose to set the mutation rate to 0.01. This decision implicates that mutations occur very rarely. Therefore, the main driving force for finding the best counterfactual is now the crossing operation. With this setting, we maintain the model's ability to improve beyond **50**th iterative cycle.

Model Candidates To conclude this section, we summarize the model selection by choosing the models and their respective hyperparameters. Furthermore, we provide a quick overview of their characteristics. All models use the same mutator, the *Sample-Based-Mutator*.

1. **CBI-ES-UC3-SBM-RR:** This model initializes the first population using process instances from the data. For each iterative cycle, the individuals with the highest viability will go on to cross over their genome. Every child will receive 30% and 70% of its parents, respectively. After the mutation phase, the generator re-ranks the entire population and discards all individuals who have not met the threshold. We choose this model as it promises to return the most feasible counterfactuals. However, the model most likely does not return the highest viability compared to other generators.
2. **CBI-RWS-OPC-SBM-FSR:** This model initializes the first population using actual process instances. For each iterative cycle the individuals that pass on their genes are probabilistically selected based in proportion to their viability. For every child, a crossover point decides how much of a parent's genome is inherited. After the mutation phase, the generator selects the most viable individuals as the next population. We choose this model as it promises to return the highest value in terms of viability. However, the model is prone to reaching convergence very quickly.

5.2 Experiment 2: Comparing with Baseline Generators

In this section we examine the results of each model's average viability across all datasets.

Results In this comparison, we employ the baseline models mentioned in subsection 3.5. Namely, the *Case-Based Generator*, the *Sample-Based Generator* and the *Random Generator*.

We randomly sample *20* factuals from the test set and use the same factuals for every generator. We ensure that the outcomes are evenly divided. The remaining procedure follows the established practice of previous experiments.

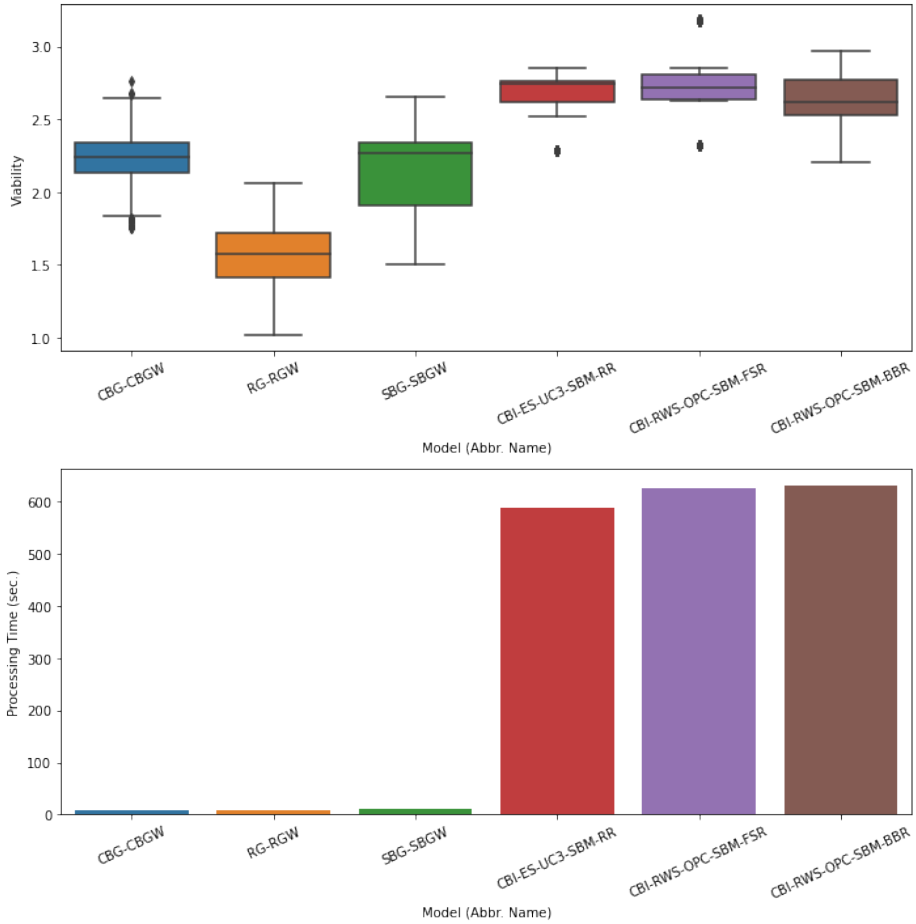


Figure 26: This figure shows boxplots of the viability of each model's generated counterfactual.

The results shown in Figure 26 show that the evolutionary algorithm *CBI-ES-UC3-SBM-RR* slightly returns better results when it comes to the median 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.

Figure 26 also displays the vast difference in computation time for the evolutionary algorithms. Only the model using the *Ranking-Recombination* seems slightly faster than the ones using Best-Breed and Fittest-Survivor as recombination methods.

Table 3 shows the detailed results.

Table 3: The result of Experiment 4. The colours indicate the model configurations that were examined. The results are based on the average viability over each counterfactual a model produces across all factuals that were tested.

Model (Abbr. Name)	Prediction Score	Viability	Sparcity	Similarity	Feasibility	Delta	Num. Paddings	Processing Time (sec.)	Max. Seq. Length
CBG-CBGW	0.514867	2.230507	0.764022	0.818115	0.014585	0.633786	14.584000	9.414627	27.000000
CBI-ES-UC3-SBM-RR	0.497746	2.678977	0.870874	0.896964	0.087737	0.823403	15.448000	588.550365	27.000000
CBI-RWS-OPC-SBM-BBR	0.445966	2.612767	0.851280	0.882271	0.095409	0.783807	15.560000	631.307437	27.000000
CBI-RWS-OPC-SBM-FSR	0.463966	2.728961	0.870071	0.899039	0.160373	0.799478	15.432000	625.714404	27.000000
RG-RGW	0.569685	1.554904	0.338077	0.578003	0.000000	0.638824	1.034000	8.175288	27.000000
SBG-SBGW	0.487669	2.151321	0.717582	0.755577	0.171964	0.506198	25.016000	9.927904	27.000000

Figure 27 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 6 out of 9 datasets, we see an improvement of at least 0.15. From **??** we see that the gap often occurs because of much higher similarity and sparsity scores. 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.

In tables 4, 5, 6 and 7 we show generations of all models and compare them to DiCE4EL. We see that all models return reasonable counterfactuals, except the Random-Search Generator . This model does not seem to follow any pattern.

Analysis These results show that the model *CBI-RWS-OPC-SBM-FSR* outperforms the other models. This result is unsurprising, as the baselines do not actively search for an optimal solution. Furthermore, we see that most evolutionary models surpass their baselines by a wide margin.

The difference in computation time is most likely due to the similarity and sparsity measures. The computation of the Damerau-Levenshstein distance is quadratic. As we also apply an additional custom cost function, the longer

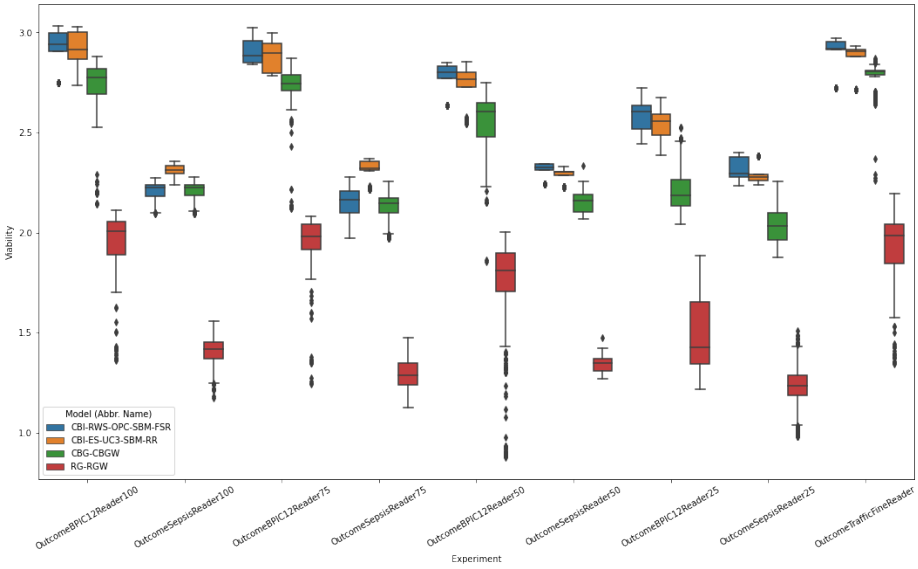


Figure 27: Boxplots of the viability of each model's generated counterfactuals across a heterogeneous collection of datasets.

Factual Seq.				Our CF Seq.				DiCE4EL CF Seq.		
Amount	Activity	Outcome	Resource	Amount	Activity	Outcome	Resource	Activity	Resource	Amount
15 500	A-SUBMITTED	0	112							
15 500	A-PARTLYSUBMITTED	0	112							
15 500	A-PREACCEPTED	0	112							
15 500	W-Completeren aanvraag	0	112							
15 500	W-Completeren aanvraag	0	111							
15 500	W-Completeren aanvraag	0	889							
15 500	W-Completeren aanvraag	0	889							
15 500	W-Completeren aanvraag	0	9							
15 500	A-ACCEPTED	0	9							
15 500	A-FINALIZED	0	9	15 500	A-SUBMITTED	1	112			
15 500	O-SELECTED	0	9	15 500	A-PARTLYSUBMITTED	1	112			
15 500	O-CREATED	0	9	15 500	A-PREACCEPTED	1	112	A-SUBMITTED	112	15 500
15 500	O-SENT	0	9	15 500	W-Completeren aanvraag	1	111	A-PARTLYSUBMITTED	112	15 500
15 500	W-Completeren aanvraag	0	9	15 500	W-Completeren aanvraag	1	111	A-PREACCEPTED	112	15 500
15 500	W-Nabellen offertes	0	9	15 500	A-ACCEPTED	1	111	A-ACCEPTED	1	15 500
15 500	W-Nabellen offertes	0	9	15 500	A-FINALIZED	1	111	O-SELECTED	1	15 500
15 500	O-SENT-BACK	0	129	15 500	O-SELECTED	1	111	A-FINALIZED	1	15 500
15 500	W-Nabellen offertes	0	129	15 500	O-CREATED	1	111	O-CREATED	1	15 500
15 500	W-Valideren aanvraag	0	138	15 500	O-SENT	1	111	O-SENT	1	15 500
15 500	O-DECLINED	0	138	15 500	W-Completeren aanvraag	1	111	W-Completeren aanvraag	1	15 500
15 500	A-DECLINED	0	138	15 500	O-SENT-BACK	1	149	O-SENT-BACK	11259	15 500
15 500	W-Valideren aanvraag	0	138	15 500	W-Nabellen offertes	1	149	W-Nabellen offertes	11259	15 500
				15 500	O-ACCEPTED	1	629	O-ACCEPTED	9	15 500

Table 4: A comparison between the Casebased-Search Generator and D4EL

Factual Seq.				Our CF Seq.				DiCE4EL CF Seq.		
Amount	Activity	Outcome	Resource	Amount	Activity	Outcome	Resource	Activity	Resource	Amount
15 500	A-SUBMITTED	0	112							
15 500	A-PARTLYSUBMITTED	0	112							
15 500	A-PREACCEPTED	0	112							
15 500	W-Completeren aanvraag	0	112							
15 500	W-Completeren aanvraag	0	111							
15 500	W-Completeren aanvraag	0	889							
15 500	W-Completeren aanvraag	0	889							
15 500	W-Completeren aanvraag	0	9							
15 500	A-ACCEPTED	0	9	15 000	A-SUBMITTED	1	112			
15 500	A-FINALIZED	0	9	15 000	A-PARTLYSUBMITTED	1	112			
15 500	O-SELECTED	0	9	15 000	A-PREACCEPTED	1	112			
15 500	O-CREATED	0	9	15 000	A-ACCEPTED	1	861	A-SUBMITTED	112	15 500
15 500	O-SENT	0	9	15 000	A-FINALIZED	1	861	A-PARTLYSUBMITTED	112	15 500
15 500	W-Completeren aanvraag	0	9	15 000	O-SELECTED	1	861	A-PREACCEPTED	112	15 500
15 500	W-Nabellen offertes	0	9	15 000	O-CREATED	1	861	A-ACCEPTED	1	15 500
15 500	W-Nabellen offertes	0	9	15 000	O-SENT	1	861	O-SELECTED	1	15 500
15 500	O-SENT-BACK	0	129	15 000	W-Completeren aanvraag	1	861	A-FINALIZED	1	15 500
15 500	W-Nabellen offertes	0	129	5 000	W-Nabellen offertes	1	11189	O-CREATED	1	15 500
15 500	W-Valideren aanvraag	0	138	15 210	W-Nabellen offertes	1	861	O-SENT	1	15 500
15 500	O-DECLINED	0	138	15 000	O-SENT-BACK	1	129	W-Completeren aanvraag	1	15 500
15 500	A-DECLINED	0	138	15 363	W-Nabellen offertes	1	912	O-SENT-BACK	11259	15 500
15 500	W-Valideren aanvraag	0	138	14 537	W-Valideren aanvraag	1	129	W-Nabellen offertes	11259	15 500
				15 000	O-ACCEPTED	1	138	O-ACCEPTED	9	15 500

Table 5: A comparison between the CBI-ES-UC3-SBM-RR and D4EL

Factual Seq.				Our CF Seq.				DiCE4EL CF Seq.		
Amount	Activity	Outcome	Resource	Amount	Activity	Outcome	Resource	Activity	Resource	Amount
15 500	A-SUBMITTED	0	112							
15 500	A-PARTLYSUBMITTED	0	112							
15 500	A-PREACCEPTED	0	112							
15 500	W-Completeren aanvraag	0	112	5 000	A-SUBMITTED	1	112			
15 500	W-Completeren aanvraag	0	111	5 000	A-PARTLYSUBMITTED	1	112			
15 500	W-Completeren aanvraag	0	889	5 000	A-PREACCEPTED	1	112			
15 500	W-Completeren aanvraag	0	889	5 000	W-Completeren aanvraag	1	861			
15 500	W-Completeren aanvraag	0	9	7 500	W-Completeren aanvraag	1	861			
15 500	A-ACCEPTED	0	9	7 500	W-Completeren aanvraag	1	861			
15 500	A-FINALIZED	0	9	7 500	A-ACCEPTED	1	861			
15 500	O-SELECTED	0	9	7 500	A-FINALIZED	1	861			
15 500	O-CREATED	0	9	7 500	O-SELECTED	1	861	A-SUBMITTED	112	15 500
15 500	O-SENT	0	9	7 500	O-CREATED	1	861	A-PARTLYSUBMITTED	112	15 500
15 500	W-Completeren aanvraag	0	9	7 500	O-SENT	1	861	A-PREACCEPTED	112	15 500
15 500	W-Nabellen offertes	0	9	7 500	W-Completeren aanvraag	1	861	A-ACCEPTED	1	15 500
15 500	W-Nabellen offertes	0	9	7 500	W-Nabellen offertes	1	109	O-SELECTED	1	15 500
15 500	O-SENT-BACK	0	129	7 500	W-Nabellen offertes	1	861	A-FINALIZED	1	15 500
15 500	W-Nabellen offertes	0	129					O-CREATED	1	15 500
15 500	W-Valideren aanvraag	0	138	7 500	O-SENT-BACK	1	789	O-SENT	1	15 500
15 500	O-DECLINED	0	138	7 500	W-Nabellen offertes	1	789	W-Completeren aanvraag	1	15 500
15 500	A-DECLINED	0	138					O-SENT-BACK	11259	15 500
15 500	W-Valideren aanvraag	0	138	5 000	W-Valideren aanvraag	1	138	W-Nabellen offertes	11259	15 500
				7 500	O-ACCEPTED	1	11289	O-ACCEPTED	9	15 500

Table 6: 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
				11 247	A-PARTLYSUBMITTED	1				
				10 159	A-FINALIZED	1				
				13 939	O-SENT	1				
15 500	A-SUBMITTED	0	112	10 550	A-APPROVED	1				
15 500	A-PARTLYSUBMITTED	0	112	25 608	A-APPROVED	1				
15 500	A-PREACCEPTED	0	112	29 560	W-Nabellen offertes	1				
15 500	W-Completeren aanvraag	0	112	4 665	O-CANCELLED	1				
15 500	W-Completeren aanvraag	0	111	19 918	O-SELECTED	1				
15 500	W-Completeren aanvraag	0	889	16 922	W-Completeren aanvraag	1				
15 500	W-Completeren aanvraag	0	889							
15 500	W-Completeren aanvraag	0	9	19 969	A-SUBMITTED	1				
15 500	A-ACCEPTED	0	9	13 066	A-PREACCEPTED	1				
15 500	A-FINALIZED	0	9							
15 500	O-SELECTED	0	9	25 535	A-DECLINED	1				
15 500	O-CREATED	0	9					A-SUBMITTED	112	15 500
15 500	O-SENT	0	9	21 437	O-CREATED	1		A-PARTLYSUBMITTED	112	15 500
15 500	W-Completeren aanvraag	0	9	17 469	W-Beoordelen fraude	1		A-PREACCEPTED	112	15 500
15 500	W-Nabellen offertes	0	9					A-ACCEPTED	1	15 500
15 500	W-Nabellen offertes	0	9	29 395	W-Completeren aanvraag	1		O-SELECTED	1	15 500
15 500	O-SENT-BACK	0	129	9 053	O-SENT-BACK	1		A-FINALIZED	1	15 500
15 500	W-Nabellen offertes	0	129					O-CREATED	1	15 500
15 500	W-Valideren aanvraag	0	138	11 627	A-FINALIZED	1		O-SENT	1	15 500
15 500	O-DECLINED	0	138	41 470	W-Valideren aanvraag	1		W-Completeren aanvraag	1	15 500
15 500	A-DECLINED	0	138	16 381	O-SENT-BACK	1		O-SENT-BACK	11259	15 500
15 500	W-Valideren aanvraag	0	138	2 828	A-ACCEPTED	1		W-Nabellen offertes	11259	15 500
				8 834	O-ACCEPTED	1		O-ACCEPTED	9	15 500

Table 7: A comparison between the Random-Search Generator and D4EL

the sequence, these computation times increase. The evolutionary algorithm, as described in subsection 2.15 is a sequential operation that also increases with the sequence length. However, we can deduce that the time difference between the *CBI-ES-UC3-SBM-RR* stems from either the *Ranking-Recombination* or the *Uniform-Crossing* operation. As those two are the only discernible operators.

In contrast, the baselines have been implemented in ways that vectorise most operations using NumPy. Meaning they can vastly decrease their computation time. On the other hand, the evolutionary algorithms are subject to python's notorious slow-looping procedures. However, this is not a vital issue for two reasons. First, it is possible to run evolutionary algorithms in a parallel manner. Second, we have not explored more optimised implementations of either the SSDLD or the evolutionary algorithm. However, we are confident that there are better and fast implementations available.

Knowing these results, a couple of questions remain, namely, whether the results remain consistent for longer sequences and other datasets. Furthermore, how does this procedure compare to other methods in the literature? The remaining experiments will address these questions.

The results for Figure 27 show that both evolutionary algorithms outperform the competition across all datasets and against all baselines. The fact that sparsity and similarity are the main drivers for this consistent improvement indicates a higher structural alignment between counterfactual and factual.

This remarkable result shows that the algorithm can outperform baselines regardless of the process log and its length.

The underperformance of the random model was expected. In subsection 3.3, we indicated that viable algorithms must at least reach a viability of 2.

Furthermore, we expected the search space for the Random-Search Generator is too vast to find viable results.

The fact that 8 of 9 datasets showed that the random model cannot exceed the threshold of two supports this claim. Additional support is the observation that every Casebased-Search Generator reaches at least 2.

7 All suggestions

5.3 Experiment 3: Evaluation under a different Viability Measure

Results Table 8 shows how each model performs under the evaluation metrics chosen by **?**. All of them apply separately to the sequence of resources and the sequence of activities. Each evaluation metric is the mean across all counterfactual results per model.

First, plausibility measures whether the sequence of activities or resources was found in the data—next, proximity is the normalised euclidian similarity between two sequences. The third is sparsity, computed using the normalised Levenshtein similarity.

We see that the evolutionary models are often comparable and sometimes even better than the DiCE4EL solution by **?**. We see that, for instance, for proximity. If the proximity of our model is lower than the proximity of the DiCE4EL solution, we can say that our models are, on average, closer to the

Generator	Dimension	Model Property Factual	CREATED Diversity	CREATED Plausibility	CREATED Proximity	CREATED Sparsity	D4EL Diversity	D4EL Plausibility	D4EL Proximity	D4EL Sparsity
CBG-CBGW-IM	Activity	0	0.972000	1.000000	0.063218	0.046497	0.000000	1.000000	0.000000	0.000000
		1	0.976000	1.000000	0.059401	0.129841	0.000000	1.000000	0.000000	0.214286
		2	0.976800	1.000000	0.086785	0.077784	0.000000	1.000000	0.000000	0.000000
		3	0.972800	1.000000	0.058631	0.053378	0.000000	1.000000	0.000000	0.055556
		4	0.976800	1.000000	0.059257	0.050848	0.000000	1.000000	0.000000	0.000000
		5	0.978400	1.000000	0.047923	0.132316	0.000000	1.000000	0.000000	0.052632
	Resource	0	0.980000	1.000000	0.506296	0.242407	0.000000	0.000000	0.277778	0.111111
		1	0.980000	1.000000	0.511616	0.249226	0.000000	1.000000	0.642857	0.214286
		2	0.979200	1.000000	0.527694	0.180480	0.000000	1.000000	0.642857	0.142857
		3	0.978400	1.000000	0.479959	0.222916	0.000000	1.000000	0.500000	0.222222
		4	0.980000	1.000000	0.446653	0.193787	0.000000	1.000000	0.409091	0.181818
		5	0.980000	1.000000	0.438579	0.201254	0.000000	0.000000	0.473684	0.157895
ES-EGW-CBI-ES-UC3-SBM-RR-IM	Activity	0	0.000000	1.000000	0.055556	0.000000	0.000000	1.000000	0.000000	0.000000
		1	0.320000	0.000000	0.062500	0.362500	0.000000	1.000000	0.000000	0.214286
		2	0.112800	0.000000	0.000000	0.347143	0.000000	1.000000	0.000000	0.000000
		3	0.039200	0.000000	0.165556	0.001111	0.000000	1.000000	0.000000	0.055556
		4	0.000000	1.000000	0.045455	0.045455	0.000000	1.000000	0.000000	0.000000
		5	0.000000	0.000000	0.052632	0.157895	0.000000	1.000000	0.000000	0.052632
	Resource	0	0.000000	0.000000	0.666667	0.277778	0.000000	0.000000	0.277778	0.111111
		1	0.640800	0.000000	0.875000	0.550000	0.000000	1.000000	0.642857	0.214286
		2	0.792000	0.000000	0.714286	0.490000	0.000000	1.000000	0.642857	0.142857
		3	0.936000	0.000000	0.832222	0.294444	0.000000	1.000000	0.500000	0.222222
		4	0.420000	0.000000	0.636364	0.227273	0.000000	1.000000	0.409091	0.181818
		5	0.554400	0.000000	0.631579	0.315789	0.000000	0.000000	0.473684	0.157895
ES-EGW-CBI-RWS-OPC-SBM-FSR-IM	Activity	0	0.000000	0.000000	0.055556	0.055556	0.000000	1.000000	0.000000	0.000000
		1	0.000000	0.000000	0.000000	0.357143	0.000000	1.000000	0.000000	0.214286
		2	0.000000	0.000000	0.000000	0.571429	0.000000	1.000000	0.000000	0.000000
		3	0.000000	1.000000	0.111111	0.166667	0.000000	1.000000	0.000000	0.055556
		4	0.000000	1.000000	0.090909	0.045455	0.000000	1.000000	0.000000	0.000000
		5	0.000000	0.000000	0.052632	0.210526	0.000000	1.000000	0.000000	0.052632
	Resource	0	0.000000	0.000000	0.666667	0.444444	0.000000	0.000000	0.277778	0.111111
		1	0.000000	0.000000	0.785714	0.428571	0.000000	1.000000	0.642857	0.214286
		2	0.000000	0.000000	0.714286	0.714286	0.000000	1.000000	0.642857	0.142857
		3	0.000000	0.000000	0.777778	0.388889	0.000000	1.000000	0.500000	0.222222
		4	0.584800	0.000000	0.772727	0.181818	0.000000	1.000000	0.409091	0.181818
		5	0.000000	0.000000	0.684211	0.210526	0.000000	0.000000	0.473684	0.157895
RG-RGW-IM	Activity	0	0.980000	0.000000	0.021696	0.000000	0.000000	1.000000	0.000000	0.000000
		1	0.980000	0.000000	0.031518	0.000000	0.000000	1.000000	0.000000	0.214286
		2	0.980000	0.000000	0.021023	0.000000	0.000000	1.000000	0.000000	0.000000
		3	0.980000	0.000000	0.034096	0.000000	0.000000	1.000000	0.000000	0.055556
		4	0.980000	0.000000	0.039796	0.000000	0.000000	1.000000	0.000000	0.000000
		5	0.980000	0.000000	0.042676	0.000000	0.000000	1.000000	0.000000	0.052632
	Resource	0	0.000000	0.000000	0.122198	0.000000	0.000000	0.000000	0.277778	0.111111
		1	0.000000	0.000000	0.151669	0.000000	0.000000	1.000000	0.642857	0.214286
		2	0.000000	0.000000	0.151084	0.000000	0.000000	1.000000	0.642857	0.142857
		3	0.000000	0.000000	0.130005	0.000000	0.000000	1.000000	0.500000	0.222222
		4	0.000000	0.000000	0.092543	0.000000	0.000000	1.000000	0.409091	0.181818
		5	0.000000	0.000000	0.121023	0.000000	0.000000	0.000000	0.473684	0.157895

Table 8: A comparison between our model and D4EL

factual. Similar holds for sparsity. We see this behaviour for both evolutionary generators. However, the Casebased-Search Generator also displays better proximity and sparsity scores than DiCE4EL. Only the Random-Search Generator appears to show worse results.

Analysis Based on these results, we can see that our model does seem to optimize properly under our viability function. If we compare our results with a different set of results, we see they often compare or even outperform the alternative solution. Unsurprisingly, the Casebased-Search Generator achieves the highest plausibility as all the counterfactuals were drawn from the data.

5.4 Experiment 4: Qualitative Assessment

Results In the result tables, you can see some of the factuals generated by our model and the model of **?**.

In this section, we show how both models (*CBI-ES-UC3-SBM-RR* and *CBI-RWS-OPC-SBM-FSR*) are capable of changing the outcome of the factual. Both models also return reasonable counterfactuals. However, *CBI-ES-UC3-SBM-RR* appears to be more consistent with the counterpart of **?**. Especially in terms of the activity sequence. For instance, both 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

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	15 105	A-SUBMITTED	1	112			
5 000	A-PARTLYSUBMITTED	0	112	14 214	A-PARTLYSUBMITTED	1	112			
5 000	A-PREACCEPTED	0	101	14 716	A-PREACCEPTED	1	112			
5 000	W-Afhandelen leads	0	101	15 373	A-ACCEPTED	1	9	A-SUBMITTED	112	5 000
5 000	A-ACCEPTED	0	111	15 038	O-SELECTED	1	912	A-PARTLYSUBMITTED	112	5 000
5 000	O-SELECTED	0	111	14 962	A-FINALIZED	1	912	A-PREACCEPTED	112	5 000
5 000	A-FINALIZED	0	111	14 887	O-CREATED	1	111	A-ACCEPTED	1	5 000
5 000	O-CREATED	0	111	14 597	O-SENT	1	103	O-SELECTED	1	5 000
5 000	O-SENT	0	111	15 236	W-Completeren aanvraag	1	111	A-FINALIZED	1	5 000
5 000	W-Completeren aanvraag	0	111	15 474	W-Nabellen offertes	1	111	O-CREATED	1	5 000
5 000	W-Nabellen offertes	0	111					O-SENT	1	5 000
5 000	O-CANCELLED	0	111					W-Completeren aanvraag	1	5 000
5 000	A-CANCELLED	0	111					O-SENT-BACK	11259	5 000
5 000	W-Nabellen offertes	0	111	14 474	W-Nabellen offertes	1	111	W-Nabellen offertes	11259	5 000
				14 716	A-REGISTERED	1	111	O-ACCEPTED	9	5 000

Table 9: A comparison between the CBI-ES-UC3-SBM-RR 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

Table 10: A comparison between the CBI-RWS-OPC-SBM-FSR and D4EL

an acceptance of the counterfactual. Both generate the latter activity correctly aligned with the factual. For instance, both evolutionary algorithms start the process with the correct sequence of A-SUBMITTED, A-PARTLYSUBMITTED and A-PREACCEPTED.

Furthermore, our model appears to be much closer in terms of sequences than the model by **?**. *CBI-RWS-OPC-SBM-FSR* (the model that only chooses the fittest survivors) has gaps. These gaps indicate that the model also attempts to align toward the correct structure of the factual model. We do not see that in *CBI-ES-UC3-SBM-RR*, as it ranks feasibility above similarity and sparsity. The introduction of gaps in the sequence automatically reduces the feasibility of the model.

We also see that the value for *Amount* fluctuates for the evolutionary generators. Similar holds for the resource field. The model focuses on event structure first and event attributes second. This might be seen as a limiting factor when it comes to event attributes. However, one could argue that the most revealing information the counterfactuals provide for sequences is within the sequence structure and less the event attributes.

Analysis Most of the results are reasonable. Surprisingly, the models did not necessarily create counterfactuals much shorter than their factual counterparts. In fact, most of **?**'s counterfactuals are shorter in length. This characteristic can be an advantage for use cases, such as medicine. The fluctuations in the loan amount were expected, as well. We did not implement any safeguard option to keep certain attributes fixed. The values our generative models produce are more

or less an indication of what the prediction model deems as a useful change to turn over the outcome at a specific step in the process.

We are also not surprised that all models manage to capture the first few activities. These are mostly the same across all cases. If our generative models had not recognised these, one could question their utility.

All models successfully flip the outcome of the prediction model and are surprisingly close to the factual compared to the model by **?**. However, we must remember that these observations tell us more about the model than the true process. More specifically, our model can show which events and attributes have to be present at a specific point within the process.

All in all, we claim that the generator model can teach us more about the prediction model primarily. Further improvement might show even more nuance in the model's behaviour. We discuss some of them in the discussion chapter.

6 Discussion

In this chapter, we are going to reexamine many of the past decisions we made. We critically assess the results of experiments and how we interpret them. We also propose possible improvements and opportunities for future reasearch.

6.1 Interpretation of Results

In the following, we discuss the results in three aspects:

1. The quality in terms of the viability of the counterfactual sequences generated by our models.
2. Their quality compared to two baseline approaches and the state-of-the-art DICE4EL approach.
3. Their implications in terms of the general utility of our solution.

Our first two experiments show that we can optimise towards viability successfully. We defined four criteria for the viability of counterfactuals (similarity, sparsity, feasibility, and delta in likelihood) and showed that a model optimising towards those criteria can return superior results. Furthermore, we created models capable of optimising complicated operationalisations of these criteria without the limitation of a function with a clearly defined gradient.

Based on the results, we have seen in the latter experiments that we can confidently say the models can generate viable counterfactuals. Compared to other methods in the literature, we show that our counterfactuals attempt to be closer to the factual we desire to understand. We have to note that these counterfactuals are primarily a reflection of the underlying prediction model. One might argue that this does not translate to a real-world scenario. However, a model never truly does. If our framework attempts to explain how a prediction model behaves, then its applicability to real-world scenarios depends on that model's viability. But regardless of the prediction model's performance, we can clearly gain an understanding of its internal reasoning pattern.

The viability measure we proposed shows that structural difference can help us better understand when and where we must apply counterfactual changes. Other approaches often seem to overlook the importance of the sequence structure. However, the *CBI-RWS-OPC-SBM-FSR* model shows that it may be reasonable to incorporate structural differences in our viability measures. Especially, if we talk about sequences and processes. The gaps within the counterfactuals our models produced clearly indicate that. If a model attempts to align sequences, it becomes much easier to compare them side-by-side.

In contrast to the closest alternative approach by **?**, we show that we can create these counterfactuals without incorporating domain-specific knowledge such as an understanding of milestone patterns. Domain knowledge can always help us create better solutions. However, we do not always have access to them. We believe that showing it is possible to create viable counterfactuals without domain-specific knowledge is our most significant contribution. Furthermore, our models can generate solutions not currently present within the data. Case-based solutions often overlook this aspect, as they are heavily biased towards the data input. Second, they can fail to deliver the necessary structural nuance when understanding sequences.

6.2 Limitations

There were also several limitations to our approach. We begin with the most obvious flaw. The generation of counterfactuals is always hard to gauge when it comes to their usefulness. There is no standardised way to evaluate the viability of a counterfactual. In fact, this is still an open research question**??**. Therefore, we often have to evaluate the counterfactuals in some subjective and qualitative way. In this thesis, we decided to compare the counterfactuals with another approach in the literature and the factual themselves. Because our counterfactuals did not produce nonsensical results, we deemed them viable. A domain expert might strongly disagree. Therefore, we advise also to incorporate experts to evaluate such an approach. The lack of domain expertise is a clear limitation of our approach, and we must acknowledge it.

Next, we introduced a novel way to measure the viability of a multivariate sequence. However, we did not compare its result to other approaches in the literature. Mainly because very few researchers have touched upon this topic. This lack of good multivariate sequence distances needs to be explored further. However, our viability measure does introduce new ideas to this sphere of research. Mainly the idea of incorporating structure. We believe that this might benefit disciplines such as *Process Mining* the most.

The viability components we chose showed they can lead to an optimised solution, but there are most likely better ways to operationalize viability criteria. However, what makes an excellent counterfactual and how we can quantify that is still a subject of debate. Many researchers fall back on defining their custom evaluation methods. However, we believe a good approach is a direct and qualitative comparison between two different approaches.

Furthermore, we did not take diversity into account. Our models strictly optimize towards the optimization goal. However, as we discussed, diversity can help us better understand factuals.

When it comes to the evolutionary algorithm, we have to admit that there are most likely more advanced and more efficient algorithms that utilise the notion of evolution. Our approach mainly followed the basic structure of an evolutionary algorithm. However, there are methods such as CMA-ES capable of improving the efficiency of the evolutionary generation.

6.3 Improvements

There are several improvements we propose. First, the feasibility metric often resulted in far lower values than other metrics. The small probabilities we saw are emblematic of the probabilistic sphere. However, it would undoubtedly help to find ways to operationalise feasibility and make it comparable to other viability components. Our ranking-based method showed that it is possible to overcome this issue. However, a less opinionated solution would be more beneficial.

Furthermore, we would like to stress that our approach is only as good as the prediction model it attempts to explain. To gain further insights into *true* process models, one must make sure that the prediction model accurately reflects the real world. Again, a domain expert might help to deduce which model is the best reflection of natural phenomena.

6.4 Future Work

Regarding future directions, it is worth pointing out whether employing other components of the viability structure is beneficial. The measure described here clearly operationalised a set of criteria. However, there may be more aspects to consider and generate even better counterfactuals. A good example would be diversity. In terms of other evolutionary approaches, applying modern state-of-the-art methods with the same viability measure would be interesting.

7 Conclusion

In conclusion, we researched **how we generate counterfactual sequences while incorporating structural differences between the factual sequence and the counterfactual sequence**. We showed it is possible to use a viability measure and incorporate structural differences. We can also use an evolutionary algorithm to optimise this viability measure.

Concerning RQ1[1], we can design a SSDLD that can compute distances even if the semi-structured data is multivariate. We employ an evolutionary model to achieve this goal in Experiment 1 (subsection 5.1).

For RQ2[2], we see the extent to which our counterfactuals fulfill viability. We show that by conducting Experiment 2 (subsection 5.2), which confirms

the hypothesis that our models outperform both random-based (RQ1-H1) and case-based approaches (RQ1-H2).

For RQ3[3], we showed that our counterfactuals are viable. We confirm that in Experiment 3 (subsection 5.3) and Experiment 4 (subsection 5.4). These experiments show that our counterfactuals are not only comparable to existing work in the literature (RQ3-H1) but can even align with factuals to make both more comparable.

To summarize, we answered all research questions and confirmed all hypotheses. Domain experts can still contest the viability of the counterfactuals. However, we believe that counterfactuals primarily explain the model we attempt to understand. Therefore, they are a valid and transparent reflection of a particular model. Furthermore, we show it is worth pursuing more research and insights into the counterfactual generation of processes. Examples within this thesis showed that processes are a ubiquitous part of our life. Many things can be understood as a process. Hence, shying away from complicated problems like multivariate sequence problems heavily limits our progress and understanding of the cause and effect relations within our daily lives.

8 First Section

8.1 A Subsection Sample

Please note that the first paragraph of a section or subsection is not indented. The first paragraph that follows a table, figure, equation etc. does not need an indent, either.

Subsequent paragraphs, however, are indented.

Sample Heading (Third Level) Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 11 gives a summary of all heading levels.
Displayed equations are centered and set on a separate line.

$$x + y = z \tag{18}$$

[1] How can we employ existing methods to compute viability so that its optimization incorporates information about the structure of the sequence?
[2] To what extent can we generate counterfactuals that fulfill the criteria to be viable?
[3] How does an algorithm which optimizes multiple viability quality metrics perform against other approaches?

Table 11: Table captions should be placed above the tables.

Heading level	Example	Font size and style
Title (centered)	**Lecture Notes**	14 point, bold
1st-level heading	**1 Introduction**	12 point, bold
2nd-level heading	**2.1 Printing Area**	10 point, bold
3rd-level heading	**Run-in Heading in Bold.** Text follows	10 point, bold
4th-level heading	*Lowest Level Heading.* Text follows	10 point, italic

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, we prefer the use of square brackets and consecutive numbers. Citations using labels or the author/year convention are also acceptable. The following bibliography provides a sample reference list with entries for journal

Bibliography