



**Utrecht
University**

Department of Mathematics and Computer Science
Process Analytics

The Generation of interpretable counterfactual examples by finding minimal edit sequences using event data in complex processes

Master Thesis

Olusanmi A. Hundogan

Supervisors:

Xixi Lu

Yupei Du

July 25, 2022

Abstract

Contents

1	Introduction	4
1.1	Motivation	4
1.2	Problem Space	6
1.3	Related Literature	8
1.3.1	Generating Counterfactuals	8
1.3.2	Generating Counterfactual Sequences	8
1.3.3	Generating Counterfactual Time-Series	9
1.3.4	Generating Counterfactuals for Business Processes	10
1.4	Research Question	11
2	Background	14
2.1	Process Mining	14
2.1.1	A definition for Business Processes	14
2.1.2	What is Process Mining?	16
2.1.3	The Challenges of Process Mining	17
2.2	Multivariate Time-Series Modelling	17
2.2.1	What are Time Series Models?	18
2.2.2	The Challenges of Time Series Modelling	18
2.3	Counterfactuals	19
2.3.1	What are Counterfactuals?	19
2.3.2	The Challenges of Counterfactual Sequence Generation	21
2.4	Formal Definitions	22
2.4.1	Process Logs, Cases and Instance Sequences	22
2.4.2	State-Space Models	23
3	Methods	27
3.1	Methodological Framework	27
3.2	Viability Measure	28
3.2.1	Semi-Structured Damerau-Levenshtein Distance	28
3.2.2	Discussing the Semi-Structured DL Distance	32
3.2.3	Delta-Measure	33

3.2.4	Feasibility-Measure	33
3.2.5	Similarity-Measure	34
3.2.6	Sparcity-Measure	34
3.2.7	Discussion	35
3.3	Models	35
3.3.1	Prediction Model: LSTM	35
3.3.2	Feasibility Model: Markov Model	38
3.3.3	Generative Model: VAE	41
3.3.4	Generative Model: Evolutionary Algorithm	42
4	Evaluation	47
4.1	Datasets	47
4.2	Representation	48
4.3	Preprocessing	49
5	Results	51
5.1	Determine the Prediction Model	51
5.1.1	Practical Matters	51
5.1.2	Results	51
5.1.3	Discussion	51
5.2	Determine the Feasibility Model	52
5.2.1	Results	52
5.2.2	Discussion	53
5.3	Determine the Operators	54
5.3.1	Experimental Setup	54
5.3.2	Results	54
5.3.3	Discussion	57
5.4	Determine the Mutation-Rates	58
5.4.1	Experimental Setup	58
5.4.2	Results	58
5.4.3	Discussion	58
5.5	Determine the Termination Point	60
5.5.1	Experimental Setup	60
5.5.2	Results	60
5.5.3	Discussion	61
5.6	Determine the best Generator Algorithm	62
5.6.1	Experimental Setup	62
5.6.2	Results	62
5.6.3	Discussion	64
5.7	Determine the Robustness given Sequence Length	64
5.7.1	Experimental Setup	64

5.7.2	Results	64
5.7.3	Discussion	65
5.8	Comparison to other Methods in Literature	65
5.8.1	Experimental Setup	65
5.8.2	Results	65
5.8.3	Discussion	65
6	Discussion	66
7	Conclusion	67

Chapter 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[47]. 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[26, 44]. For instance, in the medical domain, models have been shown to predict the outcome or trajectory of a patient's condition[31]. 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[1], 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 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 become 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[36]. Figure 1.1 is a visual representation of the desired goal.

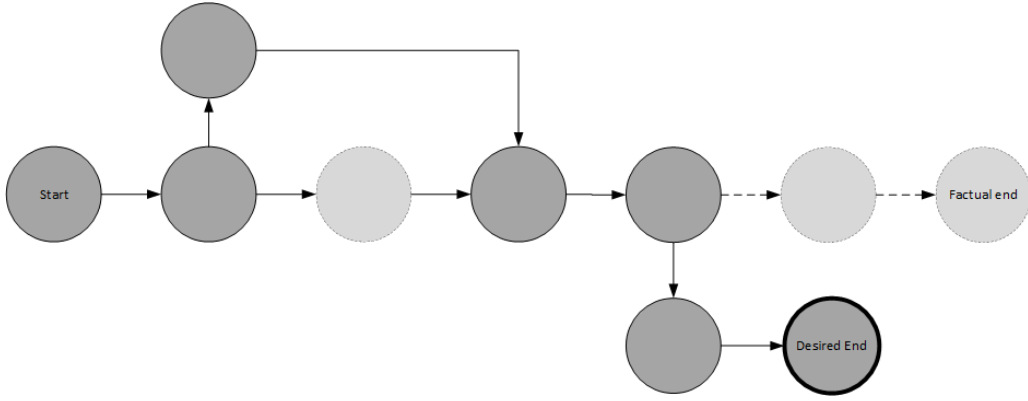


Figure 1.1: This figure illustrates a model, that predicts a certain trajectory of the process. However, we want to change the process steps in such a way, that it changes the outcome.

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[36, p.90] or feature importances[36, p.45]. but some are more comprehensible for humans than others.

A prominent and human-friendly approach are *counterfactuals*[36, 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 will 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 paper, we will 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 data¹. However, little research has focussed on counterfactuals for dynamic data².

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 and each data point is an independent entity. We can take the MNIST dataset[15] or Iris dataset[3, 16] 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[32]. 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. Hence, not all semi-structured data sets 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[18], for instance, are collections of text heavy unstructured information. In Figure 1.2, 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 occurred. A *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 occurred.

¹With static data, we refer to data that does not change over a time dimension.

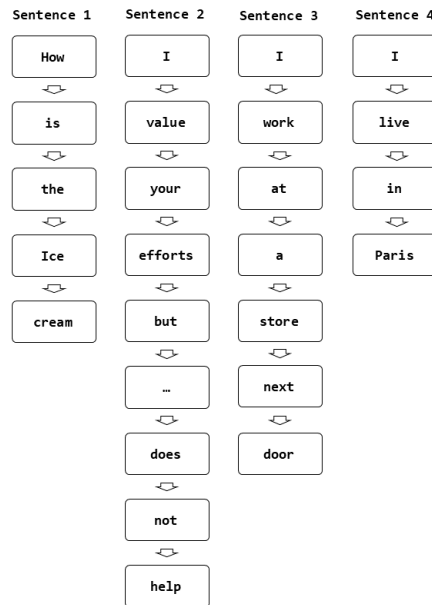
²With dynamic data, we refer to data that has a temporal relationship as a major component, which is also inherently sequential

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) Shows an excerpt of the MNIST dataset. This is a structured dataset.

(b) Shows a number of heterogenous documents. A dataset like this would be called unstructured.



(c) Shows multiple sequences of words. Each word forms a sentence. Therefore, this data is semi-structured.

Figure 1.2: Figure shows schematic examples of static structured, dynamic semistructured data and unstructured data.

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, the time and magnitude.

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[49].

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

1.3 Related Literature

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

1.3.1 Generating Counterfactuals

The topic of counterfactual generation as explanation method was introduced by Wachter et al. in 2017[48]. 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 Wachter et al. did not account for the minimalisation of feature changes and does not penalize unrealistic features. Furthermore, their solution cannot incorporate categorical variables.

A newer approach by Dandl et al. incorporates four main criteria for counterfactuals (see section 2.3) by applying a genetic algorithm with a multi-objective fitness function[12]. 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.

1.3.2 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. Martens and Provost and later Krause et al. are early examples of counterfactual NLP research[27, 33]. Their approach strongly focuses on the manipulation

of sentences to achieve the desired outcome. However, as Robeer et al. puts it, their counterfactuals do not comply with *realisticness*[41].

Instead, Robeer et al. showed that it is possible to generate realistic counterfactuals with a Generative Adversarial Model (GAN)[41]. 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 perturbed and then decoded into a similar instance[34, 50]. By modelling the latent space, we can simply sample from a distribution conditioned on the original instance. Bond-Taylor et al. provides an overview of the strengths and weaknesses of common generative models.

Eventhough, 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*[29]. 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 Krishnan et al.[27, 28]. Their Deep Kalman Filter (DKF) and subsequent Deep Markov Model (DMM) approximate the dynamic latent state-space by modeling the latent space given the data sequence and all previous latent vectors in the sequence. There are many variations[9, 17, 29] of Krishnan et al.’s model, but most use Evidence Lower-Bound (ELBO) of the posterior for the current Z_t given all previous $\{Z_{t-1}, \dots, Z_1\}$ and X_t [20].

1.3.3 Generating Counterfactual Time-Series

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

First, Delaney et al. introduces a case-based reasoning to generate counterfactuals[14]. 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[36, 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 Delaney et al., as it mainly deals with time series classification[14]. 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 Ates et al. also uses a case-based approach[5]. 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 will also introduce a framework that allows for flexibility depending on the dataset.

1.3.4 Generating Counterfactuals for Business Processes

So far, none of the models 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[6, 23, 42, 51]. Among these, Narendra et al. approach is one of the first to include counterfactual reasoning for process optimization[37]. Oberst and Sontag use counterfactuals to generate alternative solutions to treatments, which lead to a desired outcome[38]. Again, the authors do not attempt to provide an explanation of the models outcome and therefore, disregard multiple viability criterions for counterfactuals in XAI. Qafari and W. M. P. van der Aalst published the most recent paper on the counterfactual generation of explanations[40]. 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, Tsirtsis et al. develop the first explanation method for process data[46]. However, their work closely resembles the work of Oberst and Sontag and treat the task as Markov Decision Process

(MDP)[38]. 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 Tsirtsis et al. and Oberst and Sontag contribute an important idea. The idea of incrementally generating the counterfactual instead of the full sequence. Hsieh et al. build on this concept by proposing a system that generates counterfactuals milestone-wise[24]. Their work is the closest to our approach. The authors recognised that some processes have critical events, which govern the overall outcome. Hence, by simply avoiding the undesired outcome from milestone to milestone, it is possible to limit the search space and compute viable counterfactuals with counterfactual generation methods, such as the DiCE algorithm. However, their approach presupposes that the critical event points are known and the outcome is binary. Especially, the first condition makes their approach heavily dependent on the data which is used.

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 will establish the challenges that are open 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 be logically impossible, given the data set. 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:** Which existing counterfactual approaches can be applied to generate sequences?

~~RQ1: Which evaluation metric, reflects the viability of counterfactuals?~~

~~RQ2: To which extend do viable counterfactuals align with domain experts?~~

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

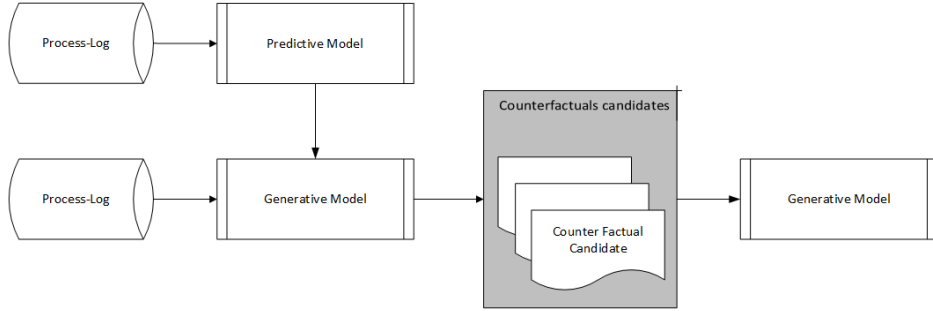


Figure 1.3: This figure shows a simplified schematic representation of the framework which is explored in this thesis.

The framework contains three parts. First, we need a pretrained predictive component which we aspire to explain. The component should be capable of *accurately* predicting 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, 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 test the following hypotheses:

RQ1-H1: If we use a viability function to determine valid counterfactuals, we consistently retrieve more viable counterfactuals, than randomly choosing a counterfactual.

RQ2-H1: The counterfactual generation consistently identifies the most viable counterfactual in the dataset faster than a random search.

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

The first hypothesis RQ1-H1 appears to be trivial, if it does not have a probabilistic component. Therefore, we need to use a model which probabilistically generates the most viable counterfactuals. Hence, we use an evolutionary algorithm, which is intrinsically probabilistic.

Chapter 2

Background

2.1 Process Mining

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

2.1.1 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[13]. Despite, legal or biological processes being valid understandings, 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 will mainly

follow the understanding outlined in W. van der Aalst et al.[47]. Each step, including start and end points, is an process event which was caused by an *activity*. Often, both terms, *event* and *activity*, are used interchangeably. However, there are subtle differences, which will become important later in this thesis. For now, 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 nighttime.

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 2.1's graphical representation is an example of such a *process map*[47].

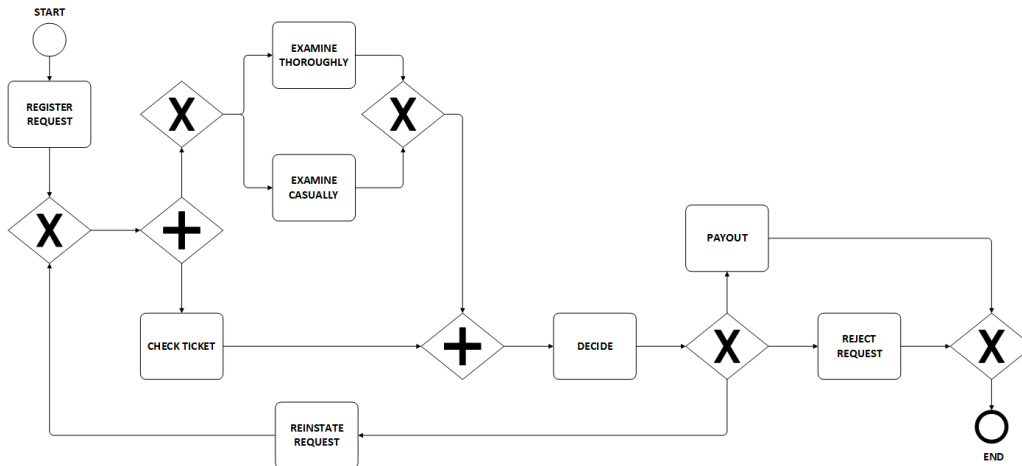


Figure 2.1: This graph shows an example of a Business Process Modell Notation (BPMN) process map.

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

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 from the customers first click to the successful shipment, an Amazon vendor might only be interested in the delivery process of a product.

Third, the example provided in Figure 2.1 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 will 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*.

2.1.2 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 modeling and analysis of processes such as Continuous Process Improvement (CPI) or Business Process Management (BPM)[47]. However, its data-centric approaches originate in Data Mining. The authors W. van der Aalst et al. 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”[47]. The discipline revolves around the analysis of event logs. A 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 Separated Values (CSV) or eXtensible Event Stream (XES)[47].

2.1.3 The Challenges of Process Mining

As mentioned in chapter 1, process data modelling and analysis is a challenging task. W. van der Aalst et al. mentions a number of issues that arise from processes[47].

The first issue arises from the quality of the data set. 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 omission of logged process steps, or wrong levels of granularity[47].

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 data's 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 section 2.2. 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[47].

A third issue which contributes to the datasets' incompleteness and complexity is a phenomenon called *concept drift*[47]. 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[47]. 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[26, 44].

2.2 Multivariate Time-Series Modelling

The temporal and multivariate nature of process instance often turns Process Mining into a Multivariate Time-Series Modelling 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.

2.2.1 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[19]. 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[29]. A time series model attempts to capture the relationship between precedent and antecedent.

2.2.2 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[49]. 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[2]. 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

possible combinations reduce 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 section 2.3.

2.3 Counterfactuals

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

2.3.1 What are Counterfactuals?

Counterfactuals have various definitions. However, their semantic meaning refers to "*a conditional whose antecedent is false*"[10]. A simpler definition from Starr 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[21] literature. A prominent figure within the causal inference discipline is Pearl et al., who postulates that a "*kind of statement – an 'if' statement in which the 'if' portion is untrue or unrealized – is known as a counterfactual*"[39]. What binds all of these definitions is the notion of causality within *what-if* scenarios.

However, for this paper, we will 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 Molnar[36, 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 tool¹ is simple. We understand the outcome of a model, if we know *what* outcome would occur *if* we changed its input. For instance, let's declare a sequence 1 as *ABCDEF \mathbf{G}* . Then a counterfactual *ABCDEX \mathbf{Z}* would tell us that **F** (probably) caused **G** in sequence 1. As counterfactuals only address explanations of one model result and not the model as a whole, they are called *local* explanations[36, p. 212]. According to Molnar *Valid* counterfactuals satisfy **four** criteria[36, p. 212]:

Similarity: A counterfactual should be similar to the original instance. If the counterfactual to sequence 1 was *AACDEX \mathbf{Z}* we would already have difficulties to discern whether B or F or both caused G at the end of sequence 1. Hence, we want to be able to easily compare the counterfactual with the original. We can archive this by either minimizing their mutual distance.

Likelihood: A counterfactual should produce the desired outcome if possible. This characteristic is ingrained in Molnar'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 possible. If the counterfactual *ABCDEX \mathbf{Z}* ends with Z but this sequence is highly unrealistic, we cannot be certain of our conclusion for sequence 1. Therefore, we want the outcome's likelihood to be at least higher under the counterfactual than under the factual instance.

Sparcity: In line with the notion of similarity, we want to change the original instance only minimally. Multiple changes impede the understanding of causal relationships in a sequence.

Feasibility: Each counterfactual should be feasible. In other words, impossible values are not allowed. As an example, a sequence *ABCDE $\mathbf{1G}$* would not be feasible if numericals are not allowed. 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. Especially

¹There are other explanatory techniques in XAI like *feature importances* but counterfactuals are considered the most human-understandable

for long and cyclical sequences, we have to expect previously unseen sequences.

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 F causes G, but what if G is not only caused by F? If we are able to find counterfactuals **VBCDEFH** and **ABCDEXZ** but all other configurations lead to G, then we know positions 1 and 6 cause G.

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 criterions as *individual viability* and *population viability*. However, to remain concise, we will use *viability* to refer to the individual criterions only. We will explicitly mention *population viability* if we refer to criterions that concern the population.

2.3.2 The Challenges of Counterfactual Sequence Generation

The current literature surrounding counterfactuals exposes a number of challenges when dealing with counterfactuals.

The most important disadvantage of counterfactuals is the Rashomon Effect[36, 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 question *how* to evaluate counterfactuals. Every automated measure comes with implicit assumptions and they cannot guarantee a realistic explanation. We still need domain experts to assess their *viability*.

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 addressed 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.

2.4 Formal Definitions

Before diving into the remainder 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.

2.4.1 Process Logs, Cases and Instance Sequences

We start by formalising the log L and its elements. Let \mathcal{E} be the universe of event identifiers and $E \subseteq \mathcal{E}$ a set of events. 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 case. For a set of events $E \subseteq \mathcal{E}$, we use a shorthand s^c being a particular sequence $s^c = \langle e_1, e_2, \dots, e_t \rangle$ with c as case identifier and a length of t . Each s is an element of the process log $s \in L$.

Furthermore, let \mathcal{T} be the time domain and $\pi_t : E \mapsto \mathcal{T}$ a surjective linking function which strictly orders a set of events.

Let \mathcal{A} be a universe of attribute identifiers in which each identifier maps to a set of attribute values $\bar{a}_i \in \mathcal{A}$.

Let \bar{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 $e_t = \{a_1 \in A_1, a_2 \in A_2, \dots, a_I \in A_I\}$ with the size $I = |\mathcal{A}|$, in which each a_i refers to a value within its respective set of possible attribute values. Conversely, we define a mapping from an attribute value to its respective attribute identifier $\pi_{\bar{a}} : A \mapsto \mathcal{A}$.

Furthermore, 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 $d = \pi_d(A_i)$.

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 $\mathbf{e}_t^c = [f_1; f_2; \dots; f_I]$. Therefore, \mathbf{e}_t^c is embedded in a vector space of size D which is the sum of each individual attribute dimension $D = \sum_i \pi_d(A_i)$. Furthermore, if we refer to a specific named attribute set A_i as a name, we will use the shorthand \bar{a}_i .

Figure 2.2 shows a schematic representation of a log L , a case c and an event e .

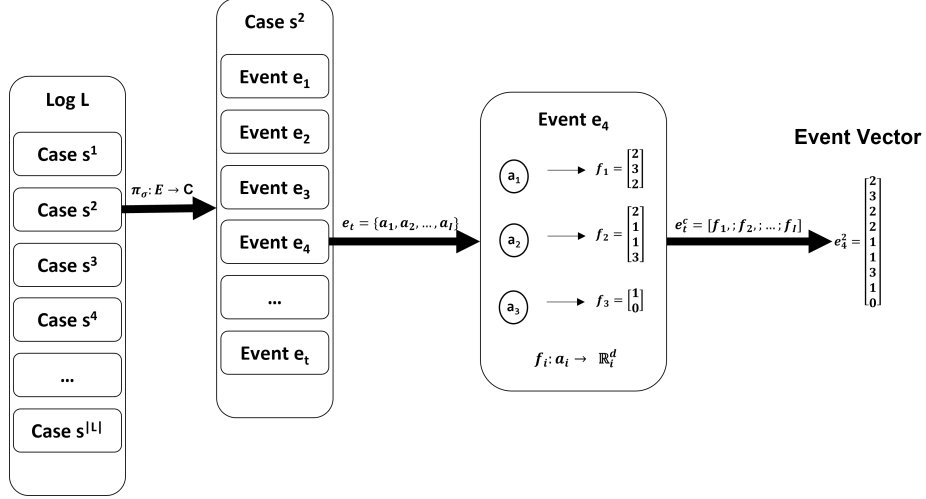


Figure 2.2: This figure shows the representation of a log L which contains a number of cases s . Case s^2 contains a number of events e_t . Each event has attribute values a_i , which are mapped to vector spaces of varying dimensions. At last, all of the vectors are concatenated.

2.4.2 State-Space Models

Generally speaking, every time-series can be represented as a state-space model[25]. 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 2.1.

$$\begin{aligned} \mathbf{z}_{t+1} &= h(t, \mathbf{z}_t, \mathbf{u}_t) \\ \mathbf{e}_t &= g(t, \mathbf{z}_t, \mathbf{u}_t) \\ \mathbf{z}_{t+1} &:= \frac{d}{dt} \mathbf{z}_t \end{aligned} \tag{2.1}$$

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 \mathbf{e}_t decouples *hidden*² states, from *observable* system outputs. Figure 2.3 shows a graphical representation of these equations.

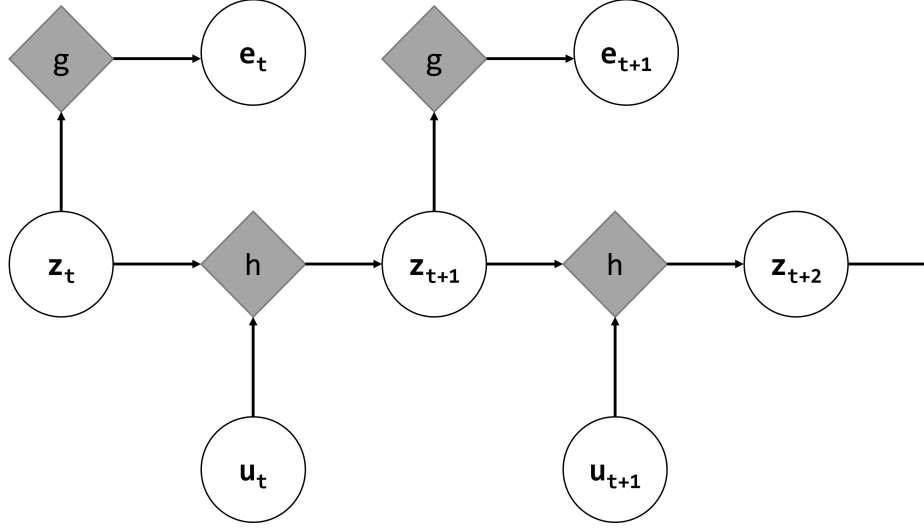


Figure 2.3: This figure shows a simplified graphical representation of a state-space model. Each arrow represents the flow of information.

The body of literature for state-space models is too vast to discuss them in detail³. However, for process mining, we can use this representation to discuss the necessary assumptions for process mining. In line with the process-definition in section 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 will 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 section 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

²A 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.

³For an introduction to state-space models see: XXX

often unknown and therefore, we require probabilistic approaches.

We can formulate Equation 2.1 probabilistically as shown in Equation 2.2.

$$\mathbb{E}[p(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h)] = \int z_{t+1} \cdot p(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h) \quad (2.2)$$

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

Note, that h and g are substituted with probability density functions parametrized 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 Probabilistic Graphical Model (PGM). This framework allows us to leverage simplifying assumptions such as the independence from future values and *d-separation*.

These characteristics change the probabilities in Equation 2.2 to Equation 2.3:

$$p(z_{t+1} \mid z_{1:t}, u_{1:t}, \theta_h) = \prod_{1 \leq \tau \leq t} p(z_{\tau} \mid z_{1:\tau}, u_{\tau}, \theta_h) \quad (2.3)$$

$$p(x_t \mid z_{1:t}, \theta_g) = \prod_{1 \leq \tau \leq t} p(x_{\tau} \mid z_{1:\tau}, \theta_g) \quad (2.4)$$

For $p(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h)$, we ignore future timesteps, as T changes into t . *d-separation* 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-separation*. $p(x_t \mid t, z_{1:T}, u_{1:T}, \theta_g)$ 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 2.3. We then use the

samples to reconstruct the most-likely a counterfactual $e_{1:t}^*$. Hence, our goal is to maximize both likelihoods.

Chapter 3

Methods

3.1 Methodological Framework

To generate counterfactuals, we will need to establish a conceptual framework, which consists of three main components. The three components are shown in Figure 3.1.

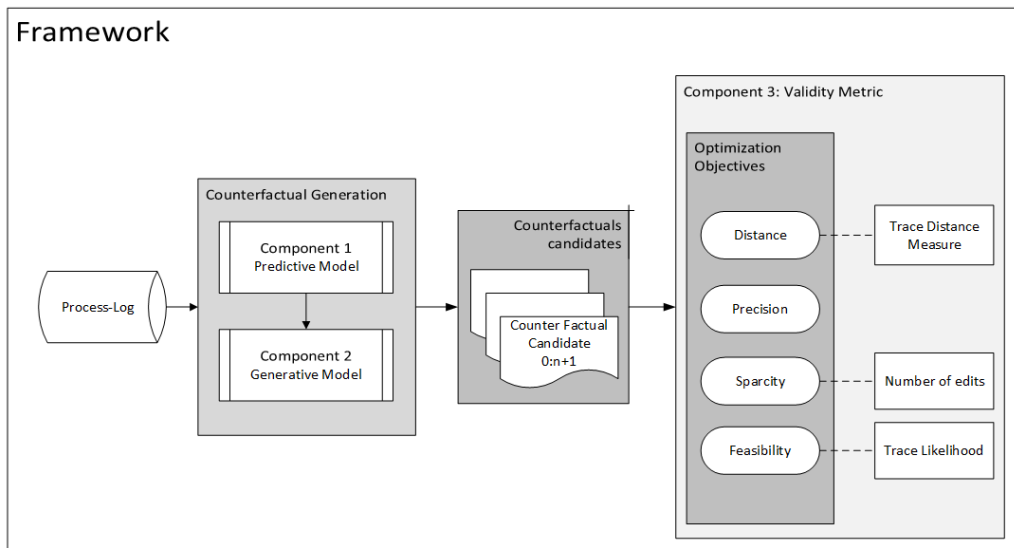


Figure 3.1: Shows the methodological framework of this thesis. The input is the process log. The log will be used to train a predictive model (Component 1) and the generative model (Component 2). This process produces a set of candidates which are subject to evaluation via the validity metric (Component 3).

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. Hence, we cannot use purely deterministic generators. Therefore, we compare two different generative approaches against a baseline. The baseline uses a simple case based heuristic. The generative approaches are a sequential a deep generative model and an evolutionary technique. Both methods allow us to use a factual sequence as starting point for the generative production but 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 section 2.3. We measure the *similarity* between two sequences using a multivariate sequence distance metric. The *precision* of the prediction will compare the likelihood of the counterfactual outcome without changes to the sequence and the counterfactual candidates. For this purpose, we require the predictive model, as it will compute the likelihoods. 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.

3.2 Viability Measure

In this section we define the components of the metric to measure the validity of a sequence.

3.2.1 Semi-Structured Damerau-Levenshtein Distance

Before discussing some of the similarity measures it is important to briefly introduce the Damerau-Levenshtein distance. This distance function is a modified version of the Levenshtein distance[30], which is a widely used to

compute the edit-distance of two discrete sequences[4, 35]. 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 sequences. Damerau 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 3.2, we schematically show two sequences and their distance.

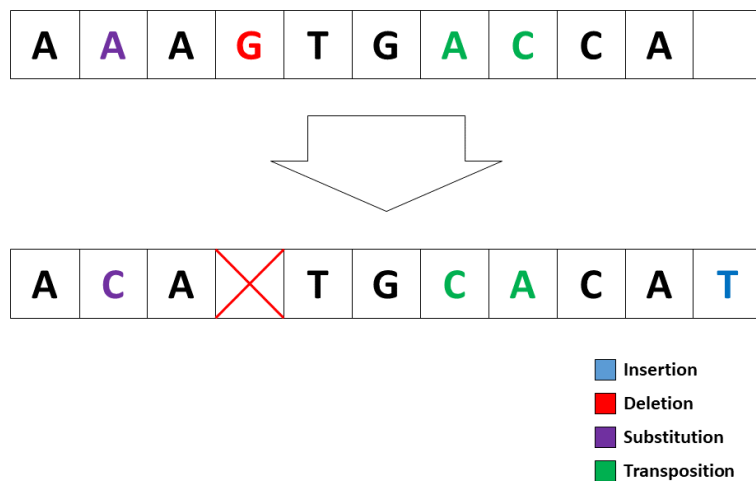


Figure 3.2: Shows two sequences. The edit distance is the sum of multiple operations. Blue shows an insert, red a deletion, purple a substitution and green a transposition. Therefore the edit distance is 4.

In Equation 3.1 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 separate terms.

$$d_{a,b}(i, j) = \min \begin{cases} d_{a,b}(i-1, j) + 1 & \text{if } i > 0 \\ d_{a,b}(i, j-1) + 1 & \text{if } j > 0 \\ d_{a,b}(i-1, j-1) + 1 & \text{if } i, j > 0 \\ d_{a,b}(i-2, j-2) + 1 & \text{if } i, j > 1 \wedge a_i = b_{j-1} \wedge a_{i-1} = b_j \\ 0 & \text{if } i = j = 0 \end{cases} \quad (3.1)$$

The recursive form $d_{a,b}(i, j)$ for sequences a and b with respective elements i and j takes the minimum of each 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. For Process Mining, it becomes necessary to modify the distance further.

To illustrate the issue, we explore a couple of examples. Lets assume, we have two strings $s^1 = aaba$ and $s^2 = acba$. Using the Damerau-Levenshtein-Distance, the edit distance between both sequences is zero, as we can recognise one substitution at the second character in both strings. However, this representation is insufficient for process instances as they may also contain attribute values. Therefore, we characterise the sequences as process events in Equation 3.2.

$$s^1 = \{a, a, b, a\} \quad (3.2)$$

$$s^2 = \{a, a^*, b, a\} \quad (3.3)$$

$$s^3 = \{a, c, b, a\} \quad (3.4)$$

$$s^4 = \{a, b, a\} \quad a, b, c \in \mathbb{R}^3 \quad (3.5)$$

$$a = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix} \quad a^* = \begin{bmatrix} 3 \\ 3 \\ 4 \end{bmatrix} \quad b = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad c = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix} \quad (3.6)$$

If we do not consider attribute values, it becomes clear that s^2 , s^3 and s^4 have an edit distance of 0, 1 and 1 to s^1 . However, with attribute values 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. These examples show that we can neither disregard attribute values nor events, while computing the edit distance of two process instances. We show this in ???. In other words, we cannot simply assume a static cost of 1 for each edit operation. Instead, we have to define a cost function which takes attribute variables

into account. In the following sections, we will establish distances which use a modified Damerau-Levenshtein distance approach. Here, the cost of each edit-operation will be weighted with a distance function that considers the difference between event attributes. In simplified terms, we can say that s^1 and s^2 are identical, if we only consider the activity. However, taking attribute values into account, s^1 and s^2 actually differ on two accounts. In order to reflect these differences in attribute values, we introduce a modified version of the Damerau-Levenshtein 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 normed vector-space¹. Concretely, we formulate the modified Damerau-Levenshtein distance as shown in Equation 3.7. For the remainder, we will denote this edit-distance as Semi-structured Damerau-Levenshtein distance (SSDLD).

$$d_{a,b}(i, j) = \min \begin{cases} d_{a,b}(i-1, j) + cost(\mathbf{0}, b_j) & \text{if } i > 0 \\ d_{a,b}(i, j-1) + cost(a_i, \mathbf{0}) & \text{if } j > 0 \\ d_{a,b}(i-1, j-1) + cost(a_i, b_j) & \text{if } i, j > 0 \\ & \& \bar{a}_i = \bar{b}_j \\ d_{a,b}(i-1, j-1) + cost(a_i, \mathbf{0}) + cost(\mathbf{0}, b_j) & \text{if } i, j > 0 \\ & \& \bar{a}_i \neq \bar{b}_j \\ d_{a,b}(i-2, j-2) + cost(a_i, b_{j-1}) + cost(a_{i-1}, b_j) & \text{if } i, j > 1 \\ & \& \bar{a}_i = \bar{b}_{j-1} \\ & \& \bar{a}_{i-1} = \bar{b}_j \\ 0 & \& i = j = 0 \end{cases} \quad (3.7)$$

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 \bar{a}_i and \bar{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

¹A normed vector-space is a vector space, in which all vectors have the same dimensionality. For instance, if all vectors have three dimensions, we can call the vector-space *normed*.

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-Levenshtein distance.

3.2.2 Discussing the Semi-Structured DL Distance

There are two important characteristics, which require a discussion.

First, 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.

Second, to explain the intuition behind most terms, it is necessary to establish a common understanding between the relationship of an event with its event attributes. Generally, we can have two notions of this relationship.

For the first relationship, we consider the event and its attributes as separate entities. This notion is reasonable, as some attributes remain static throughout the whole process run. If we take a loan application process as an example, an applicant's ethnic background will 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 cost functions, as they treat the activity independently from its attribute values. In other words, if the activities of two events are \bar{a} and \bar{b} , but their attribute values are $(\frac{2}{3})$ and $(\frac{2}{3})$, these events may be seen as more similar than two \bar{a} and \bar{a} with attribute values $(\frac{2}{3})$ and $(\frac{5}{0})$.

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 event as an event at daytime. 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. Second, the unmodified Damerau-Levenshtein

distance applies to discrete sequences, such as textual data with atomic words or characters. By treating each event as an discrete sequence element, we remain faithful to the original function.

3.2.3 Delta-Measure

For this measure, we can evaluate the precision of a counterfactual trace by determining whether a counterfactual would lead to the desired outcome. For this purpose, we use the predictive model, which will output a prediction base on the counterfactual sequence. However, it is often difficult to force a deterministic model to produce a different result. We can relax the condition by maximising the likelihood of the counterfactual outcome. If we compare the likelihood of the desired outcome under the factual sequence with the counterfactual sequence, we can determine an increase or decrease. Ideally, we want to increase the likelihood. We can compute the odds or the difference between the two likelihoods. We choose to use the delta.

$$improvement = p(o^*|a^*) - p(o^*|a) \quad (3.8)$$

Here, $p(o|a)$ describes the probability of an outcome, given a sequence of events.

3.2.4 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 events themselves. This is a difficult task, given the *open world assumption*. In theory, we cannot know whether any event *can* follow after a nother event or not. However, we can assume that the data is representative of the process dynamics. Hence, we can simply 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. We use the Kneser-Ney Smoothing[8] approach to ensure that unseen sequences are accounted for. Second, we have 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. Hence, we compute the joint probability of a case by

using the forward algorithm [CITE forward algorithm](#) . In Equation 3.9 shows the formulation.

$$\begin{aligned}
feasibility_e &= p(a_i | a_{i-1} \dots a_1, \theta) \approx p(a_i | a_{i-1}, \theta) \\
feasibility_f &= p(f_i | a_i, \theta) \\
feasibility &= \prod_{i=1}^n [p(f_i | a_i, \theta) * p(a_i | a_{i-1}, \theta)]
\end{aligned} \tag{3.9}$$

Here, a and f are the activity and features of a particular event. Likewise, θ is the data sample which is used to determine the parameters of the density function. The first equation shows the approximation based on the markov assumption.

3.2.5 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 3.2.1), we want to take the structural distance and the feature distance into account. Henceforth, we will use the previously established SSDLD. The similarity distance uses a cost function as specified in Equation 3.10.

$$\begin{aligned}
cost(a_i, b_j) &= L2(a_i, b_j) \\
a_i, b_j &\in \mathbb{R}^d
\end{aligned} \tag{3.10}$$

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

3.2.6 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 sequences that are compared can have varying lengths. Second, even if they were the same length, the events might not line up in such a way, that we can simply count the changes to a feature. Hence, to solve this issue, we use the previously established SSDLD. The sparsity distance uses a cost function as specified in Equation 3.11.

$$\begin{aligned} cost(a_i, b_j) &= \sum_d \mathbb{I}(a_{id} = b_{jd}) \\ a_i, b_j &\in \mathbb{R}^d \end{aligned} \tag{3.11}$$

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.

3.2.7 Discussion

Given the current viability function we can already determine the optimal counterfactual:

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

The elements that fulfill these criteria make up the pareto surface of this multi-valued viability function. If each of the values are scaled a range between 0 and 1, the theoretical ceiling is 4. This value is only possible if we can flip the outcome of a factual sequence 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 an important threshold. If we score the viability of a factual against itself, a normalised sparsity and similarity value have to at its maximal value of 1. In contrast, the improvement has to be 0. The feasibility is 0 depending on whether the factual was 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 a considerable counterfactual.]**

3.3 Models

3.3.1 Prediction Model: LSTM

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[22]. 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

$$h_{t+1} = \sigma(Vh_t + Ux_t + b) \quad (3.12)$$

$$o_t = \sigma(Wh_t + b) \quad (3.13)$$

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 nonlinearity 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 ?? shows a schematic representation of an RNN.

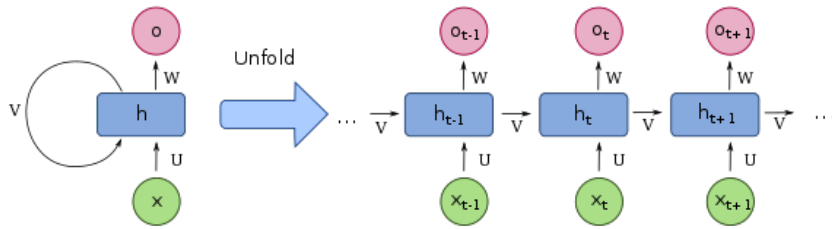


Figure 3.3: A schematic representation of an RNN viewed in compact and unfolded form??.

The architecture of the prediction model is shown in Figure 3.4.

One input consists of an 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 concat 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.

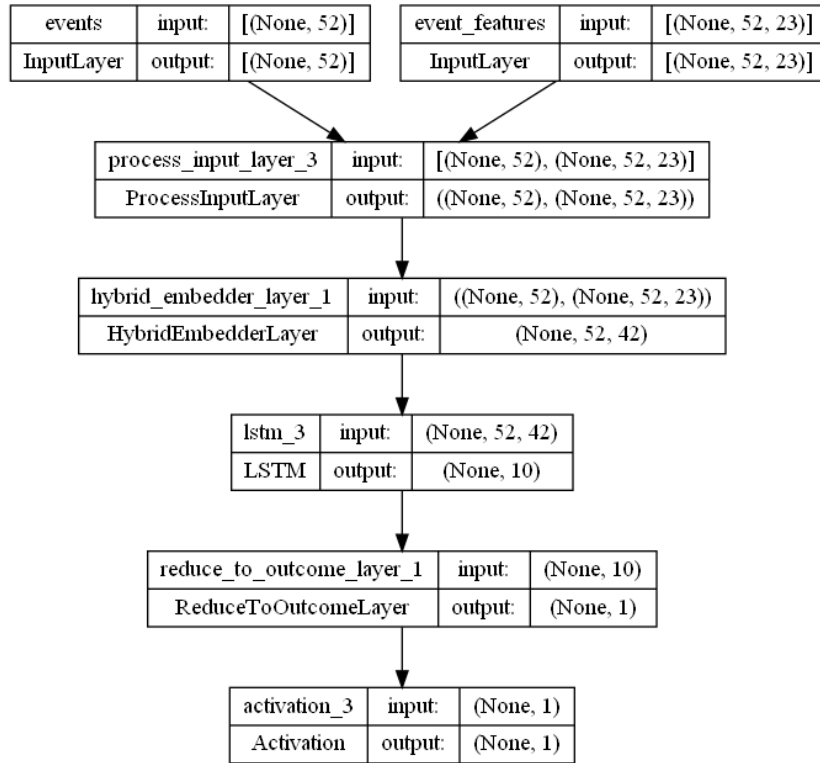


Figure 3.4: Shows the different components of the LSTM architecture. Each elements contains information about the input and output of a layer. None is a placeholder for the batch size.

3.3.2 Feasibility Model: Markov Model

Before testing any model we have to establish two crucial components of the viability measure. First, we require a prediction model which we want to explain using counterfactuals. This is relevant for determining the improvement that a counterfactual yields in contrast to the factual. Second, we need to know to what extent any given counterfactual is feasible given the dataset at hand. Therefore, we will dedicate the first set of experiments to establishing these components.

To compute the viability of a counterfactual we need to determine its feasibility. In other words, we have to determine the possibility or impossibility of the counterfactual. We can use the data log to gauge the feasibility, by estimating the data distribution.

There are many ways to estimate the density of a data set. 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 only dependent on its immediate predecessor and neither on future nor on any any states prior to its immediate predecessor. Third, the collection of attributes within an event depend 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. The probability distributions are shown in ??.

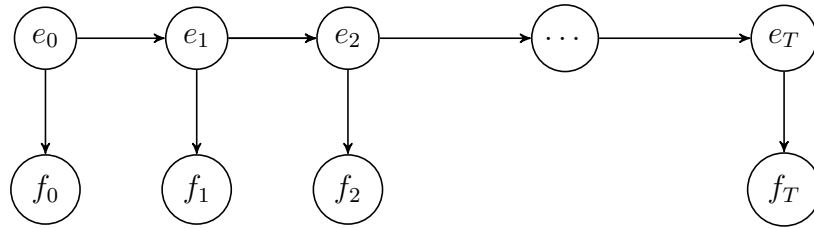


Figure 3.5: The feasibility model in graphical form.

Here, e_t represents the transition from one event state to another. Likewise, f represent the emission of the feature attributes. Hence, the probability of a particular sequence is the product of the transition probability multiplied with state emission probability for each step. Note, that this is the same as the feasibility measure as in Equation 3.9. **[Make formula in viability section consistent with this! Formula needs to use e instead of a and change starting index from 1 to 0.]**

$$p(e_{0:T}, f_{0:T}) = p(e_0) p(f_0 | e_0) \prod_{t=1}^T p(e_t | e_{t-1}) p(f_t | e_t) \quad (3.14)$$

Practical Matters

The general computation of these products is trivial. However, we need to probabilities for $p(e_0)$, $p(f_t | e_t)$ and $p(e_t | e_{t-1})$ as the true distributions are unobservable.

Starting with the transition dynamics part of the equation $p(e_t | e_{t-1})$, we can estimate the model parameters by counting the transitions from one event state (activity) to another (activity). [Define the difference between event und activity in a better way and earlier.] $p(e_0)$ is a special case, as it does not have a preceding event.

The emission probabilities are more complicated for three reasons: First, the event distribution does not necessarily belong to the same family as the feature distribution. Hence, we cannot use any simple method to estimate these conditionals. We need to estimate the probability for each event seperately. The second issue directly follows from the first. If we estimate each event distribution by partitioning the data by events, we naturally have less data to estimate each model's parameters. Although, event partitioning, are not an issue for common events states (activities), they can make emission probabilities of less frequent event states exceptionally hard to estimate. One can turn to Bayesian Methods, which hand these situations better by specifying a prior.

However, the third issue exacerbates the main issue of using bayesian methods. Namely, because features do not necessarily have to be from the same distributional family, we have to model each conditional distribution with a mixture of distributions. Hence, simple bayesian updates are not possible either and require more time expensive methods such as Markov-Chain-Monte-Carlo methods or similar. [Check if this is true. Maybe we can use MCSC].

From these issues, we can conclude there are multiple viable ways to model these conditional distributions and we have to choose an fitting method².

²Note, that we did not mention modelling $p(f_0 | e_0)$ as it is practically the same distribution as $p(f_t | e_t)$.

Model Description

Knowing, there are many ways to model the sequence distribution, we choose to implement a number of different methods. However, we evaluate them based on how well they fit the data distribution and choose the most promising method.

Transition Dynamics: For the transition dynamics we count the individual transitions as mentioned by using a *Transition-Count-Matrix*. Then, we compute the probabilities of each transition by dividing occurrence count of the preceding event-state. We apply the same method on the $p(e_0)$. The only difference is that we count the starts and divide by the number of available cases.

Emission Probability: For $p(f_t | e_t)$, we employ [3] tactics:

Independent: Here, we assume all feature columns are independent variables with no covariations. Hence, for discrete variables, we use Categorical distributions. In other words, if the variable is binary, we count the ones to estimate the parameters of a Bernoulli distribution. Similar holds for categorical distributions. In contrast, we use independent Gaussian distributions for continuous variables.

Grouped: Here, we group variables from the same distributional family and estimate their parameters. Meaning, we take all discrete distributions and compute the parameters of one categorical distribution. Likewise, we group all continuous variables and compute the parameters, mean and covariance, for one multivariate Gaussian distribution.

Grouped with χ^2 : This is similar to the grouped approach. However, a multivariate Gaussian is also a continuous distribution. The likelihood of a continuous density distribution is not limited to a range between 0 and 1. Only the area under the density function has to follow this restriction. Meaning, if we compute the likelihood of a specific data point we might end up values of 30 or even 300000. Therefore, we rather interpret a data point as the mean of another Gaussian. With this assumption, we can use the χ^2 distribution to compute the probability of that distribution belonging to the distribution at hand. Or rather, how likely it is to find another datapoint which is more likely to belong to the distribution. Here, we say $Q = (Y - \mu)^T \Sigma^{-1} (Y - \mu)$ and assume $Q \sim \chi^2(k)$. If Q is bigger than $(x - \mu)^T \Sigma^{-1} (x - \mu)$, then $\mathbb{P}[(y - \mu)^T \Sigma^{-1} (y - \mu) \geq (x - \mu)^T \Sigma^{-1} (x - \mu)] = 1 - \mathbb{P}[Q \leq (x - \mu)^T \Sigma^{-1} (x - \mu)]$

Gaussian Distribution under Event Partitions [FOR XIXI: This section requires an intuitive understanding of linear algebra and the formula of simple gaussians and multivariate gaussians. Shall I put this in the discussion section instead?] Gaussian distributions are particularly vulnerable to data shortages due to event partitionings.

For instance, if we use independent variables, we often face issues of columns without any variation. Those lead to a covariance matrices' determinant being 0. In these cases the covariance is no longer a definite covariance matrix. A determinant with a value of 0 means intuitively, that at least one of the Gaussian distributions has an undefined probability. This undefinable probability occurs, because the variance parameter of a regular Gaussian appears in the denominator of the whole expression. By dividing by 0 and subsequently computing the joint probability we receive an undefined joint probability. Therefore, the covariance often needs to be definite. This issue is the reason, why we often employ numerical solutions like *Singular-Value-Composition* to get an approximation.

However, if the covariance matrix has a higher rank than the number of data points used to estimate it, the issue remains. Meaning, if we have a 20x20 covariance matrix and just 3 datapoints to estimate it, we will likely face numerical issues of computing the determinant again.

To mitigate these issues, we fallback to methods like adding a small constant to the diagonal. If this approach does not work either, we add a small constant to the full matrix. Alternatively, we can compute the Gaussian in a Bayesian way by adding a prior. However, using a fallbacks was deemed as the simpler solution.

3.3.3 Generative Model: VAE

The generative approach assumes, it is possible to capture a latent state z and use this state to generate suitable counterfactual candidates. We condition the generation procedure on the factual instance to generate counterfactuals that show sparse differences to the original sequence. The core idea is to sample randomly $e^* \sim p(z|e)$ to generate counterfactual candidates. We can sort each candidate by their *viability* and choose top-K contenders as viable counterfactuals. There are a multitude of approaches to generate the counterfactuals. However, we will limit our exploration to sequential Variational Autoencoders (VAEs) and sequential GANs. Both techniques allow us to sample from a smooth latent space conditioned on the factual sequence. VAEs approximate $p(z|e)$ by trying to reconstruct the input using Monte-Carlo methods. GANs require a generator model and a discriminator model. The generator model attempts to fool the discriminator model by

generating, results that closely resemble true process instances. In contrast, the discriminator tries to distinguish generated instances from real instances.

Model Architecture

TBD

3.3.4 Generative Model: Evolutionary Algorithm

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 EP as we follow the stereotypical structure of these algorithms and use vector representations directly.

In our algorithm, we randomly generate candidates by modifying the factual sequence and evaluate their fitness based on a fitness function. Those, candidates that are deemed as fit enough are subsequently modified to produce offspring. Afterwards, the procedure will repeat until a termination condition is reached. It differs from , because it does not require to use differentiable functions. Hence, we can directly optimise the viability metric established in section 3.2.

For the algorithm, we follow a rigid structure of operations as outlined in ???. As ??? shows, we define 5 fundamental components. Initiation, Selection, Crossover, Mutation and Recombination.

Algorithm 1 Shows the basic structure of an evolutionary algorithm.

Require: Hyperparameters

Ensure: The result is the final population

survivors \leftarrow initialize population;

while not *termination* **do**

parents \leftarrow select parents;

offspring \leftarrow crossover parents;

mutants \leftarrow mutate offspring;

survivors \leftarrow recombine population and mutants;

termination \leftarrow check if termination criterion is reached

end while

Initiation

The initiation process 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. In this thesis, we call this method the *Default-Initiation*. However, choosing among a subset of the search space can allow for a faster convergence. We chose to implement three different subspaces as a starting point. First, by sampling from the data distribution of the Log (*Data-Distribution-Initiation*). Second, by picking individuals from a subset of the Log (*Casebased-Initiation*). And lastly, we can use the factual case itself as a reasonable starting point (*Factual-Initiation*).

Selection

The selection process chooses a set of individuals among the population according to a selection procedure. These individuals will go on to act as material to generate new individuals. Again, there are multiple ways to accomplish this. In this thesis, we explore three methods. First, the *Roulette-Wheel-Selection*. Here, we compute the fitness of each individual in the population and choose a random sample proportionate to their fitness values. Next, the *Tournament-Selection*, which randomly selects pairs of population individuals and uses the individual with the higher fitness value to succeed. Last, we select individuals based on the elitism criterion. In other words, only a top-k amount of individuals are selected for the next operation.

Crossover

Within the crossover procedure, we select random pairing of individuals to pass on their characteristics. Again allowing a multitude of possible procedures. We can uniformly distribute characteristics by copying one individual of the pair and pass on a fraction of the complementary individual (*Uniform-Crossover*). By repeating this process towards the opposite direction, we can create two new offsprings, which share characteristics of both individuals. The second approach is suitable for sequential data of same lengths. We can choose a point in the sequence and pass on characteristics of the complementary individual onto the first individual from that point onwards and backwards (*One-Point-Crossover*). Thus creating two new offsprings. The last option is called *Two-Point-Crossover* and resembles its single-point counterpart. However, this time, we choose two points in the sequence and pass on the overlap and the disjoint to generate two new offsprings.

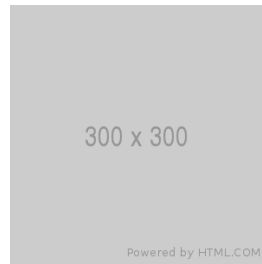


Figure 3.6: A figure showing the process of uniformly applying characteristics of one sequence to another

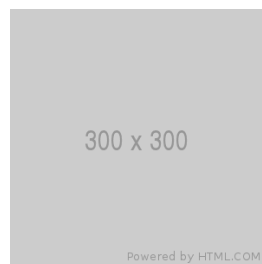


Figure 3.7: A figure showing the process of applying characteristics of one sequence to another using one split point

Mutation

Mutations introduce random perturbations to the offsprings. Here, only one major approach to apply these mutations was used. However, the extend in which these mutations are applicable can still vary.

Before elaborating on the details, we have to briefly discuss four modification types that we can apply to sequences of data. Reminiscent of edit distances, which were introduced earlier in this thesis, we can either insert, delete or change a step. Furthermore, we can transpose two adjacent steps in

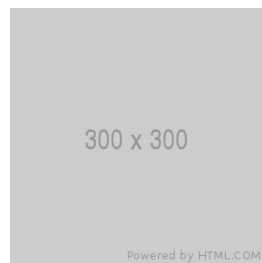


Figure 3.8: A figure showing the process of applying characteristics of one sequence to another using two split points

the sequence. These are the fundamental ways we can use to edit sequences.

However, we can change the rate 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 will experience a modification which results in the deletion of a step. Same applies to the other modifications. Further, we modify the amount to which each modification applies to the sequence. We call this rate *edit-rate* and keep it constant accross every edit-type. Meaning, if the edit-rate is 0.5 and the delete-rate is 1, then each individual will have 50% of their sequence deleted.

There are still three 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 would naturally occur over repeated iterations in an evolutionary algorithm.

However, these operations follow the structure of established edit-distances like the Damerau-Levenshtein distance. Furthermore, they allow for efficient restrictions with respect to the chosen data encoding. For instance, we can restrict delete operations to steps that are not padding steps. In constrast insert operations can be restricted to padding steps only.

Second, we could introduce different edit-rates for each edit-type. However, this adds additional complexity and needlessly increases the search space for hyperparameters.

Third, as we chose the hybrid encoding scheme, we have to define what an insert or a change means for the data. Aside from changing the activity, we also have to choose reasonable data attributes. This necessity requires to define two ways to produce them. We can either choose the features randomly, or choose to sample from a distribution which depends on the previous activities. We name the former approach *Default-Mutation*. We can simplify the latter approach by invoking the markov assumption and sample the feature attributes given the activity in question (*Data-Distribution-Mutation*).

Recombination

This process refers to the creation of a new population for the next iteration. We have to point out that in the literature, recombination is often synonymous with crossover. However, in this thesis recombination refers to the update process which generates the next population. Here, we use introduce two variants.

We name the strinct selection of the best individuals among the offsprings and the previous population *Fittest-Individual-Recombination*. 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. The latter keeps suboptimal individuals, while adding a constant pool of better individuals to select.

Chapter 4

Evaluation

In this chapter, we are going to establish most of the methods, the results section will cover. In detail, we will discuss the datasets, we will use, the pre-processing pipeline and the final representation for each of the algorithms.

4.1 Datasets

In this thesis, we use a multitude of datasets for generating the counterfactuals. All of the data sets were taken from Teinemaa et al. Each dataset consists of log data and contains labels which signify the outcome of a process. They were introduced by [some author]. We focus on binary outcome predictions. Hence, each dataset will provide information about one of two possible outcomes associated with the case. For instance, a medical process might be deemed a success if the patient is cured or a failure if the patient remains ill. A loan application process might deem granting the loan a success or the rejection as failure. The determination of the outcome depends on the use-case and the stakeholders involved. An insurance provider might deem a successful claim as a failure, while the client deems it as a success.

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 individual case relates to one loan application process and can be accepted (success) or cancelled (deviant).

The next dataset is the Sepsis-Dataset.

Lastly, we apply our approach to a third dataset of a different domain [To show validity across datasets]. [Add a dataset description here.] Below we list all the important descriptive statistics in ?? [num deviant and num regular should be based on the counts within the cases.] [Time should just get seconds not this format.]

Dataset	Num Cases	Min Seq Len	Max Seq Len	Ratio Distinct Traces	Num Dist
Dice4EL	3728	15	50	0.000268	
BPIC12-25	866	15	25	0.001155	
BPIC12-50	3728	15	50	0.000268	
BPIC12-75	4461	15	75	0.000224	
BPIC12-100	4628	15	100	0.000216	
Sepsis25	707	5	25	0.001414	
Sepsis50	770	5	47	0.001299	
Sepsis75	777	5	66	0.001287	
Sepsis100	779	5	88	0.001284	
TrafficFines	129615	2	20	0.000008	

4.2 Representation

To process the data in subsequent processing steps, we have to discuss the way we will encode the data. There are a multitude of ways to represent a log. We will discuss 4 of them in this thesis.

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, eventhough, 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.

Therefore, it is simpler 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 has the dimensions (case-index, max-sequence-length) and the latter (case-index, max-sequence-length, feature-attributes). **[This requires a change into formal symbols**

that were defined prior.]

4.3 Preprocessing

To prepare the data for our experiments, we employed basic tactics for pre-processing. First, we split the log into a training and a test set. The test set will act as our primary source for evaluating factuals, that are completely unknown to the model. We further split the training set into a training set and 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 val and 75 train.

First, we filter out every case, whose' sequence length exceeds 25. We keep this maximum threshold for most of the experiments that forucs on the evolutionary algorithm. The reason is . Furthermore, two components of the proposed viability measure have a run time complexity of at least 2. Hence, limiting the sequence length saves a substantial amount of ressources.

Next, we extract time variables if they are not provided in the log in the first place. Then, we convert all binary columns to the values 1 and 0. **[In cases know time relevant information is available, we will XXX...]**

Each categorical variable is converted using binary encoding. Binary encoding is very similar to onehot encoding. However, it is still distinct. 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. **[from different from onehot encoding as we, encode each categorical as binary value rather than .]**

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 are standardized to have a zero mean and a standard deviation of 1.

We omit the case id, the case activity and label column from this preprocessing procedure, for reasons explained in section 4.2. The case activity are label-encoded. In other words, every category is assigned to a unique integer. The label column is binary as we focus on outcome prediction.

The entire pipeline is visualized in Figure 4.1.

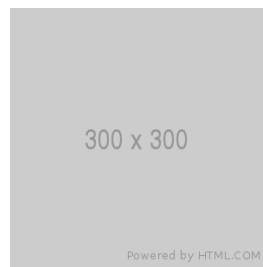


Figure 4.1: shows the entire preprocessing pipeline based on an example case.

Chapter 5

Results

Before testing any model we have to establish two crucial components of the viability measure. First, we require a prediction model which we want to explain using counterfactuals. This is relevant for determining the improvement that a counterfactual yields in contrast to the factual. Second, we need to know to what extent any given counterfactual is feasible given the dataset at hand. Therefore, we will dedicate the first set of experiments to establishing these components.

5.1 Determine the Prediction Model

We use counterfactuals primarily to explain predictive models. This explanation requires a to define the prediction model we use in this thesis.

5.1.1 Practical Matters

[Mention everything necessary to repeat this experiment: For instance, unbalanced data]

5.1.2 Results

[Show how this model is fine to use by reporting training and validation scores.]

5.1.3 Discussion

Mention whatever is noticeable: 1. The models are extremely good for longer maximal sequence lengths of the BPIC dataset. The

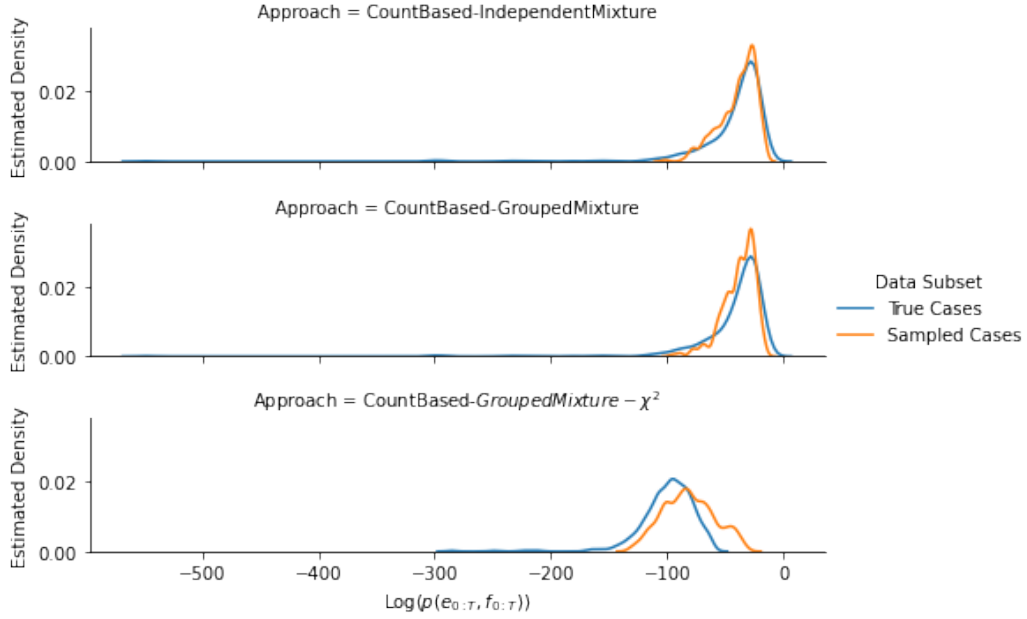


Figure 5.1: The figure shows the distribution of feasibility values given the approach that was used. It is not entirely clear which of the approaches fit better to the data.

question is whether the model genuinely learns intrinsic features or just waits for trivial patterns like: How many padding events does the sequence have? What cyclical patterns are present? Does a particular event occur or not. Could be solved using attention and a visualisation method.

5.2 Determine the Feasibility Model

5.2.1 Results

With all the configurations in mind we compare the distribution of the dataset with the distribution drawn from samples from the distribution. The configuration which best represent the data distribution is the best candidate for the next experimental steps. For this purpose, compute we the feasibility values for the Log's cases for the whole training dataset. Additionally, we sample the same number of values using our distributional approach. We show their distributions in Figure 5.1.

As the figure is difficult to interpret, we also compute various distances using the same subsets of data. The Kolgomorov-Smirnoff Test (KST) is

Table 5.1: Table show the computation of various distances. Showing that the combination of Countbased Transition Estimation and the Goupded- χ^2 approach consistently yields lower distances.

Eval-Method	Transition Approach	Emmision Approach	value
KS-Test	CountBased	GroupedMixture	0.411523
KS-Test	CountBased	GroupedMixture- χ^2	0.353909
KS-Test	CountBased	IndependentMixture	0.419753
L2	CountBased	GroupedMixture	0.000004
L2	CountBased	GroupedMixture- χ^2	0.000000
L2	CountBased	IndependentMixture	0.000004
L1	CountBased	GroupedMixture	0.000033
L1	CountBased	GroupedMixture- χ^2	0.000000
L1	CountBased	IndependentMixture	0.000031
Correlation	CountBased	GroupedMixture	1.004166
Correlation	CountBased	GroupedMixture- χ^2	1.004104
Correlation	CountBased	IndependentMixture	1.010678

particularly interesting as it is a common method to compute the difference between two distributions.

Table 5.1 shows that the third approach yields lower distances accross all distance methods employed.

5.2.2 Discussion

As the best configuration seems to be the **[CountBased-Grouped- χ^2]** approach, we continue with this configuration for the subsequent experiments.

However, it is important to stress that the proposed way of estimating the data distribution is one of many. The markovian approach explicitly removes the effect of past and future states. It is needless to say, a process step does not have to depend on its immediate previous state. A process outcome may be influenced by all past or future events. For instance, if one has to approve a loan in a second stage, one might be more inclined to approve something that a trusted employee already approved. Likewise, one might apply more scrutiny, knowing that a certain supervisor is going to approve the third stage.

Furthermore, this approach assumes strictly sequential processes. If the sequence has events running in parallel, we also have to record in greater detail which event has triggered a subsequent event in a given sequence. Often this knowledge is not even available.

[Mention to Discuss issue with underflow]

5.3 Determine the Operators

As explained in ??, there are many possible configurations for an evolutionary algorithm. Therefore, we run simulations for all possible combinations of operators. We use the results, to filter out operators for subsequent evaluation steps that may not be promising.

5.3.1 Experimental Setup

To avoid confusion, we refer to each unique phase combination as a model-configuration. For instance, one model-configuration would consist of [a DatabBasedInitiator, an ElitismSelector, a OnePointCrosser, SamplinBasedMutator and a FittestSurvivorRecombiner]. We refer to a specific model-configuration in terms of its abbreviated operators. For instance, the earlier example is denoted as [DBI-ES-OPC-SBM-FSR].

The model-configuration set contains [144] elements. We choose to run each model-configuration for [50] evolution cycles. For all model-configurations, we use the same set of [4] factual process instances, which are randomly sampled from the test set. We ensure, that the outcomes of these factuals are evenly divided. We decide to return a maximum of [1000] counterfactuals for each factual case. Within each evolutionary cycle, we generate [100] new offsprings. We keep the mutation rate at [0.1] for each mutation type. Hence, across all cases that are mutated, the algorithm deletes, inserts, and changes [1%] of events. We collect the means viability and its components across the iterative cycles of the model.

5.3.2 Results

Figure 5.2 shows the bottom and top-[k] model-configurations based on the viability after the final iterative cycle. We also show how the viability evolves for each iteration.[change evolutionary cycle to iterative cycle] The results reveal a couple of patterns. First, all of the top-[k] algorithms use [either CaseBasedInitiator or SampleBasedInitiator] as initiation operation. In contrast, the bottom-[k] all use [RandomInitiator] as initialisation. Second, we see that most of the top-[k] algorithms use the [Elitism-Selector].

The complete table of results is in ??.

ATTACHMENT: tbl:average-viability

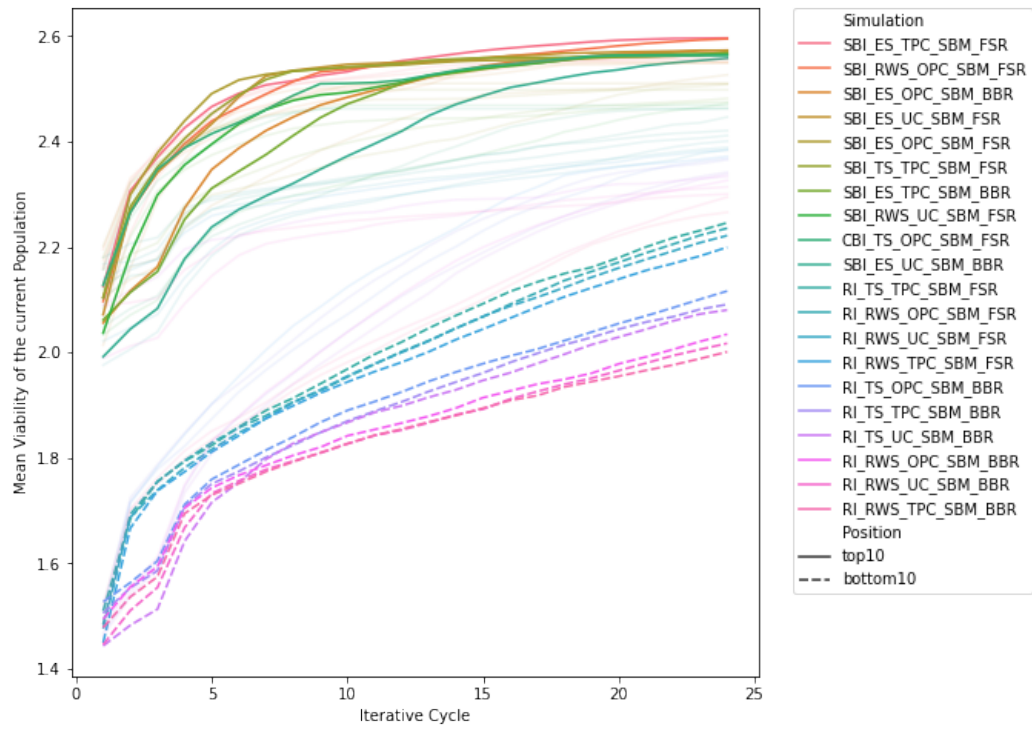


Figure 5.2: This figure shows the average viability of the [10] best and worst model-configurations. The x-axis shows how the viability evolves for each evolutionary cycle.

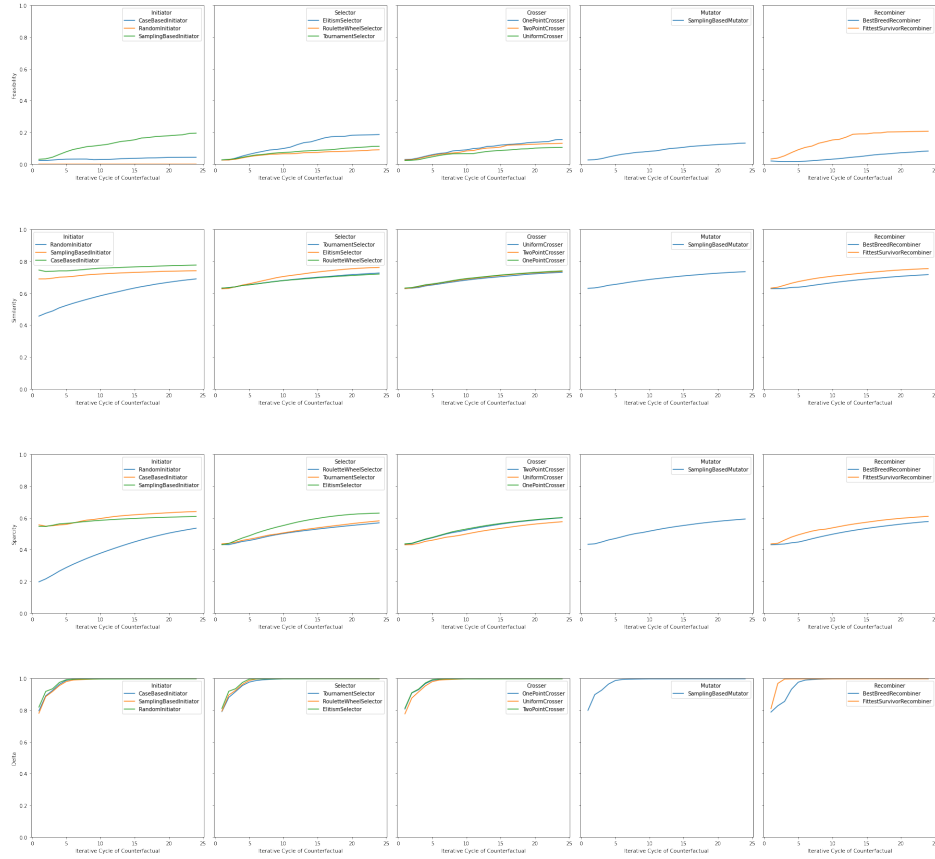


Figure 5.3: Shows the evolution of each viability measure over the entire span of iterative cycles. Each figure adjust the respective operator type by taking the average over all other operator types. **[Make sure, that the legends are aligned in color.]**

In Figure 5.3, we see the effects of each operator type, except the mutation operation.

Starting with some commonalities accross operator-type and measure, the figure shows that the initiator heavily determines the starting point for each measure. For instance, the **[RandomInitiator]** starts well below the other initiator operations when it comes to sparcity and similarity. Another noteworthy general observation is the delta measure **[Change some of the names to align with Dice4EL]**. Here, for each operator type we see a movement towards the highest possible delta value. Hence, most configurations are capable of changing the source class to the desired class.

In terms of viability Figure ??, shows an increase only for cases, in which the **[SampleBasedInitiator]** is used. Similar holds for recombination with **[FittestSurvivorRecombiner]**.

The results for the selection operator type are undeniably in favor of **[ElitismSelector]** for all viability measures. The same holds for the recombination operation. Here, the **[FittestSurvivorRecombiner]** yields better results.

When it comes to the crossing operation, the results indicate, the differences between **[OnePointCrosser]** and **[TwoPointCrosser]** are inconclusive for all viability measures except feasibility. One can explain that by noting, that both operations are very similar in nature. However, cutting the sequence only once retains produces less impossible sequences for the child sequences.

5.3.3 Discussion

Moving forward, we have to choose a set of operators and also determine suitable hyperparameters for each. In the next experiment we consider the following operators: We choose the **[SampleBasedInitiator]** as it might increase our chances to generate feasible variables.

For selection, we will use the **[ElitismSelector]**, as it generally appears to return better results.

Furthermore, we choose to move forward with the **[OnePointCrosser]**. This crossing operation is slightly better in yielding feasible results.

For selection and recombination, we use the **[ElitismSelector]** and the **[FittestIndividualRecombiner]**, respectively. They all outcompete their alternatives for all similarity, sparcity and feasibility.

In the next experiment we vary mutation rates, using **[SBI-ES-OPC-SBM-FIR]**.

5.4 Determine the Mutation-Rates

5.4.1 Experimental Setup

Given the chosen configuration in subsection 5.3.1, we will now explore optimal parameter settings. Therefore, we run the model [50] times with different mutation-rates. For the chosen, we uniformly sample the mutation rates for *delete*, *insert* and *change* between 0 and 1 for each run. The remaining procedure follows the process described in subsection 5.3.1.

5.4.2 Results

To avoid confusion, we refer to each triplet consisting of *delete*, *insert* and *change* as one *rate-configuration*. Hence, if we discuss a number of rate-configurations, we discuss a set of triplets.

As we can see in Figure 5.4, every rate-configuration will eventually lead to convergence. However, its striking, that most rate configurations converge at around the same iterative cycle. This holds for a majority of rate-configurations, except for two cases within the top5 and a small number of other rate-configuration. A comparison is available in ??

ATTACHMENT: att:all-mutation-rate-results

5.4.3 Discussion

While it is expected, that every rate-configuration eventually converges towards an optimal value, it remains surprising that most rate-configurations suddenly converge around the 10 iteration. There are a multitude of possible reasons for this phenomenon. As the viability measure incorporates structural information and event-related information, we assume that the algorithm focuses on finding a structural optimum first. Hence, the algorithm first optimizes similarity, sparsity and delta, before focusing on feasibility. This behaviour is reinforced by the viability measures' tendency to favor shorter sequences first. This phenomenon is discussed further, when we look at the event and feature structure of the results in subsection 5.6.1.

However, as we have to choose a rate-configuration, we take the configuration which not only yields the highest viability but also does not converge. Therefore, we maintain the models ability to still improve beyond [50] iteration cycles. We move forward with a delete-rate of [0.14], an insert rate of [0.21] and a change rate of [0.23].

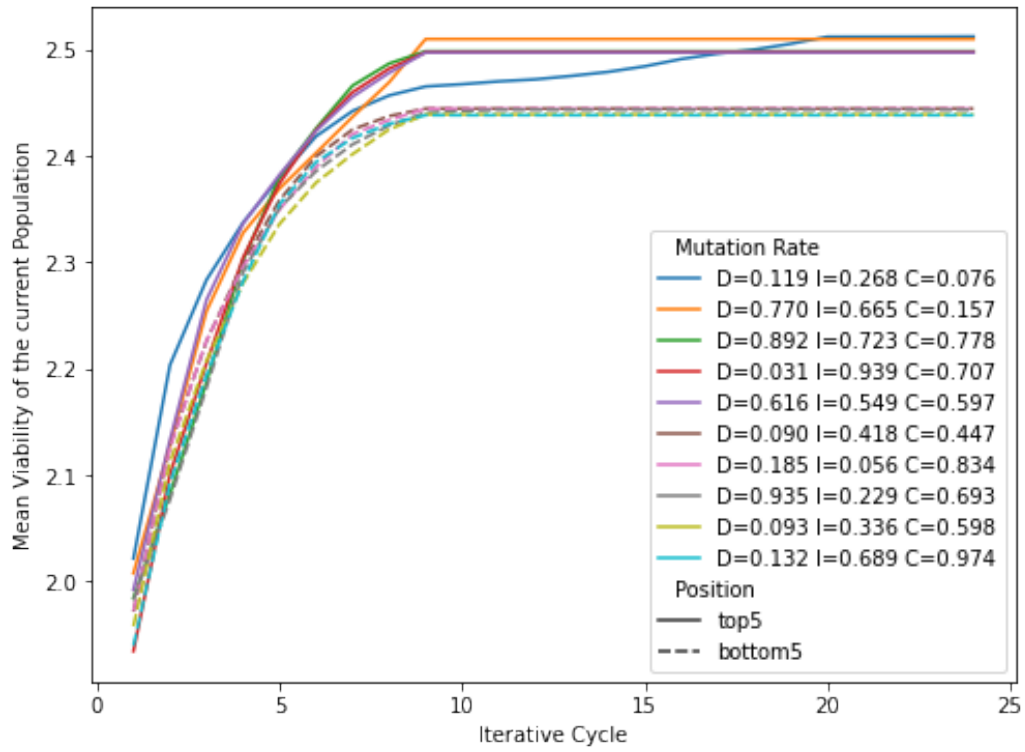


Figure 5.4: This figure shows the average viability of the [5] best and worst rate-configurations. The x-axis shows how the viability evolves for each evolutionary cycle. The legend shows rate-configuration was used with D, I and C standing for delete, insert and change, respectively. The line-type marks, whether the configuration is among the top or bottom rate-configurations in the list.

5.5 Determine the Termination Point

5.5.1 Experimental Setup

Given the chosen configuration from subsection 5.3.1 and the chosen hyperparameters of subsection 5.4.1, we explore the effects of different stopping criteria for the evolutionary algorithm. 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 termination point. Hence, a termination point at 5 means the algorithm, will not proceed to optimize its results, further after reaching the fifth iteration.

We choose to run the configuration for evolution cycles from 5 to 100 in steps of 5. We keep the mutation rate at **[0.1]** for each mutation type. The remaining procedure follows the process described in subsection 5.3.1.

5.5.2 Results

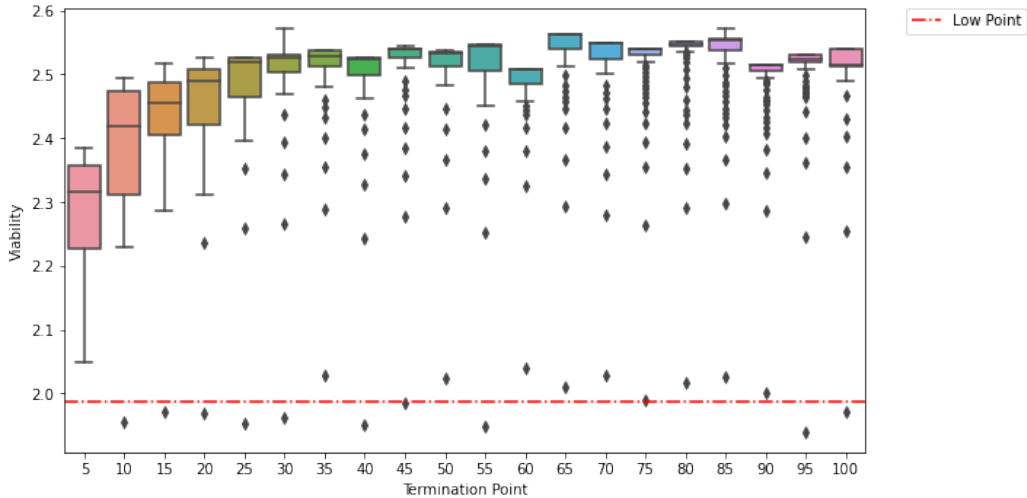


Figure 5.5: This figure shows boxplots for the viability of each maximum number of iteration cycles tested. The x-axis shows how the viability evolves for each termination point.

In Figure 5.5, 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 episode**[introduce term 'episode']**. However, the results do converge towards a viability of **[2.5]**. Hence, increasing the termination point further is unlikely to generate better results.

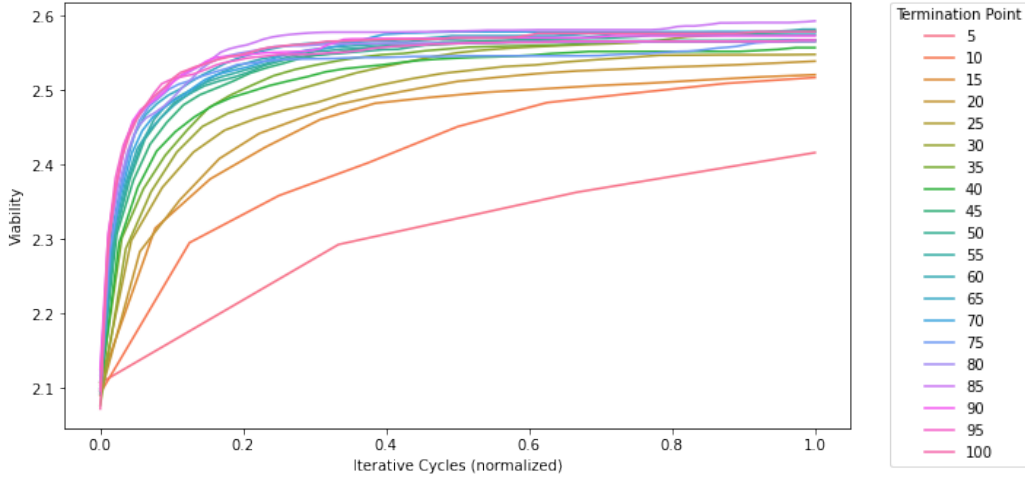


Figure 5.6: shows the viability reached given the termination point. All termination points are normalized for better comparability.

Furthermore, in Figure 5.6 reflects the spread of the individual results. Unsurprisingly, low termination points yields a larger spread dispersion of viability. The values become less dispersed, the higher the termination point.

However, there are some surprising outliers. First, the termination points 55, 60 and 90 seem to have an extremely low dispersion. Meaning most of their results have the same viability. It is not clear whether this is a rule or mere coincidence. Also, the number of outliers increase with the termination point as well. A last noteworthy observation is the amount of solutions near the mark of **[1.8]**, as depicted with the dotted line.

5.5.3 Discussion

Most of the result were expected. The longer the algorithm runs, the more it narrows its solution space. Typically, this is a favorable characteristic. However, for counterfactual explanations, this advantage becomes a nuisance. We generally seek more diverse solutions. As discussed in section 3.2, we favor diverse solutions, but this aspect is not reflected in the viability measure.

The low dispersion cases are probably solutions that are stuck in local optima. As mentioned earlier, this is a common flaw for evolutionary algorithms.

With regards to the low viability outliers, we can safely ignore those as we are mostly interested in the higher ranking counterfactuals.

Table 5.2: Table shows the result of Experiment 4. The colors indicate the model configurations that were examined. The results are based on the average viability each counterfactual a model produces across all factials that were tested.

Short Name	Unnamed: 0	experiment	rank	likelihoo
CBGW-CBG-CBGW-IM	249.500000	1.000000	25.500000	0.39380
EGW-ES-EGW-SBI-ES-OPC-SBM-FSR-IM	1249.500000	1.000000	25.500000	0.49671
EGW-ES-EGW-SBI-ES-OPC-SBM-HR-IM	1749.500000	1.000000	25.500000	0.49799
EGW-ES-EGW-SBI-ES-OPC-SBM-RR-IM	2249.500000	1.000000	25.500000	0.49933
RGW-RG-RGW-IM	749.500000	1.000000	25.500000	0.31499

For the next experiments we are going to use [35] as a termination point. This number is high enough to yield reasonable counterfactuals with high viability, while not being too narrow in its solution space.

5.6 Determine the best Generator Algorithm

5.6.1 Experimental Setup

Knowing the , we compare the evolutionary algorithm with other algorithms.

In this comparison, we employ the other models mentioned in section 3.3. Namely, the *Case-Based Generator* and the *Random Generator*.

For the evolutionary algorithm, we choose the model-configuration from subsection 5.3.1, the rate-configuration determined in subsection 5.4.1 and the termination point from subsection 5.5.1. Furthermore, we randomly sample [20] factials from the test set and use the same factials for every generator. We ensure, that the outcomes are evenly divided. The remaining procedure follows the established procedure of previous experiments.

5.6.2 Results

the results shown in Figure 5.7 show that the evolutionary algorithm [model-specifier] returns better results on average. The worst model is the random generated model.

Table 5.2 shows the detailed results.

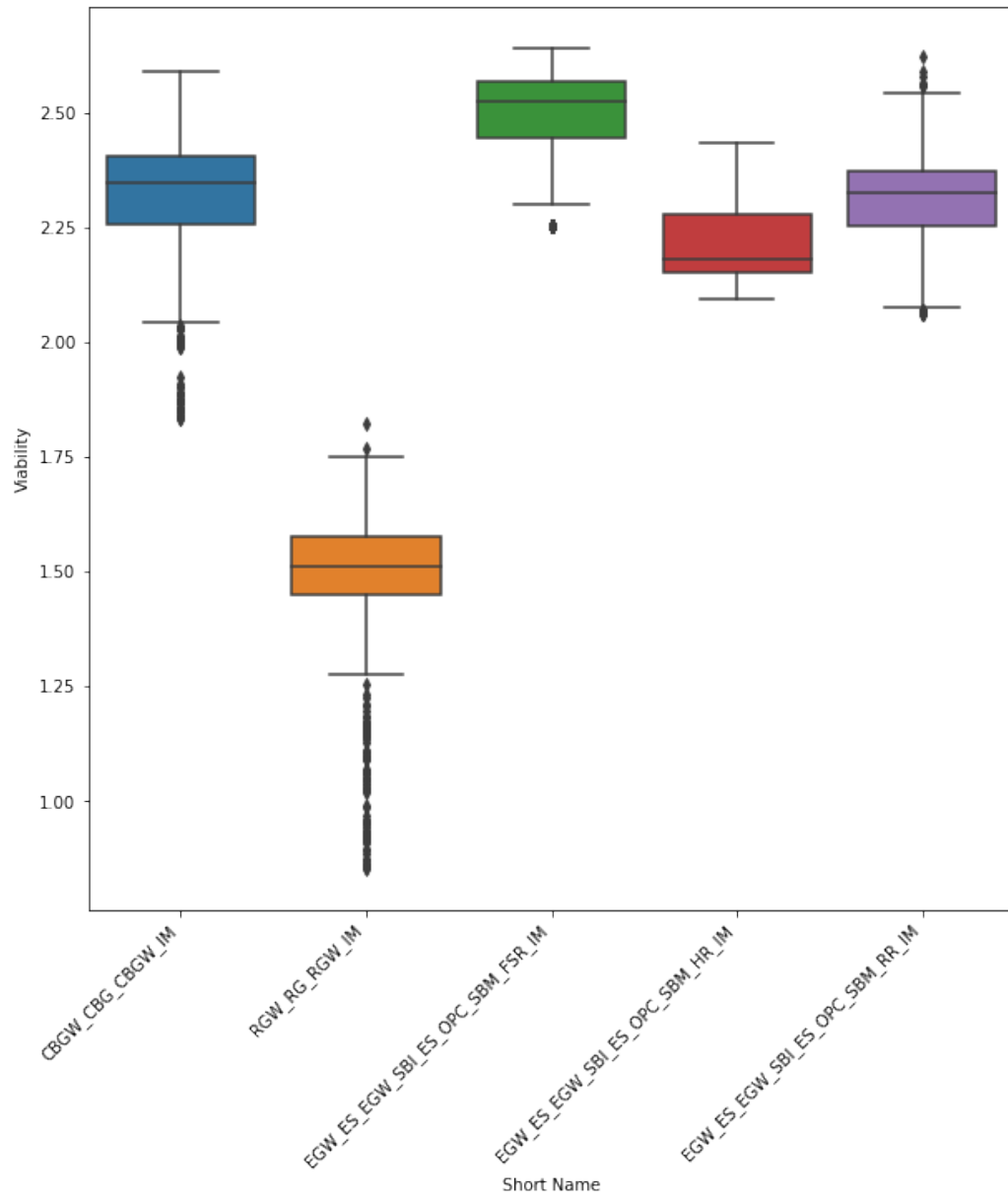


Figure 5.7: This figure shows boxplots of the viability of each models' generated counterfactual.

5.6.3 Discussion

These results show that the model [SBI-ES-OPC-FSR] is clearly superior to the other models. This result is unsurprising, as the baselines do not actively search for an optimal solution. However, knowing these results, a couple of questions remain. Namely, whether the results remain consistent for longer sequences and for other datasets? Furthermore, how does this procedure compare to other methods in the literature? The remaining experiments will address these questions.

5.7 Determine the Robustness given Sequence Length

So far, the experiments were conducted on a maximal sequence length of [25]. Also, we conducted each experiment[FOR XIXI: In terms of wording, should I change 'experiment' to 'simulation'?] on the same dataset. [MENTION the RQ to have an approach which is process model agnosite] With these result, we are not able to to claim, that thew model consistently outperforms the other approaches. Therefore, this section will explore the results on different data-set sizes and types.

5.7.1 Experimental Setup

For this experiment, we run the same experiment of subsection 5.6.1 on all the datasets, that where introduced in section 4.1.

5.7.2 Results

Again, Figure 5.8 shows a clear dominance of the evolutionary model ([Model Name]) across all datasets.

The results show that the performance remains consistent for [for a specific model]. Here, [a specific model] returns an avarage viability of [some value]. Hence, we can savely assume that the model outperforms the baseline approaches in general.

However, a notable observation occurs in the processing time. Although all models require more time to produce their results, if we increase the maximum length of sequences, the evolutionary algorithms require substantially more time. ?? shows, the processing time is much higher than the baseline models. On average, [evo model] require [duration in seconds], while [other models] require [XX], [XX] and [XX], respectively.

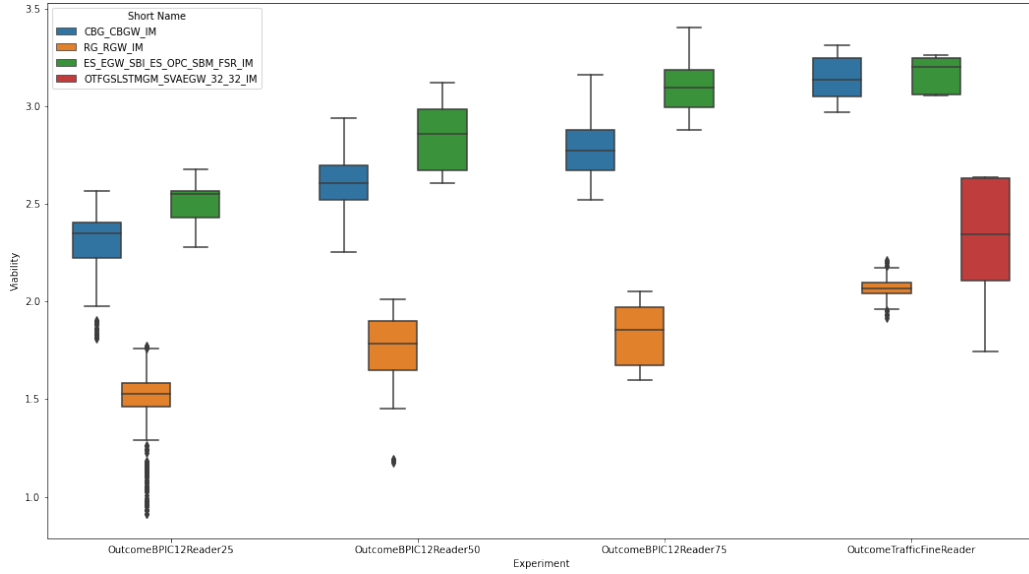


Figure 5.8: This figure shows boxplots of the viability of each models' generated counterfactual.

5.7.3 Discussion

The results show that [... TBD]. The increase of generation time can be explained by the viability measure. More specifically, the current implementation of the SSDLD within the sparsity and similarity measure has a quadratic time complexity. The time complexity primarily depends on the maximal sequence length. The number of cases that are compared is less of a factor as we use a highly vectorized implementation of the distance using numpy.

5.8 Comparison to other Methods in Literature

5.8.1 Experimental Setup

5.8.2 Results

5.8.3 Discussion

Chapter 6

Discussion

Chapter 7

Conclusion

Bibliography

- Agrawal, K. (2010). To study the phenomenon of the Moravec’s Paradox. *ArXiv*.
- Anastasiou, A., Hatzopoulos, P., Karagrigoriou, A., & Mavridoglou, G. (2021). Causality Distance Measures for Multivariate Time Series with Applications. *Mathematics*, 9(21), 2708. doi:10.3390/math9212708
- Anderson, E. (1936). The Species Problem in Iris. *Annals of the Missouri Botanical Garden*, 23(3), 457–509. doi:10.2307/2394164. JSTOR: 2394164
- Apostolico, A., & Giancarlo, R. (1998). Sequence Alignment in Molecular Biology. *Journal of Computational Biology*, 5(2), 173–196. doi:10.1089/cmb.1998.5.173
- Ates, E., Aksar, B., Leung, V. J., & Coskun, A. K. (2021, May 19). Counterfactual Explanations for Multivariate Time Series. In *2021 International Conference on Applied Artificial Intelligence (ICAPAI)* (pp. 1–8). 2021 International Conference on Applied Artificial Intelligence (ICAPAI). doi:10.1109/ICAPAI49758.2021.9462056
- Baker, J., Song, J., & Jones, D. R. (2017). Closing the Loop: An Empirical Investigation of Causality in IT Business Value. *undefined*. Retrieved March 1, 2022, from <https://www.semanticscholar.org/paper/Closing-the-Loop-%3A-An-Empirical-Investigation-of-in-Baker-Song/df210060211bdc598f2d3382c68c615319287f71>
- Bond-Taylor, S., Leach, A., Long, Y., & Willcocks, C. G. (2021, April 14). *Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models*. arXiv: 2103.04922 [cs, stat]. Retrieved October 1, 2021, from <http://arxiv.org/abs/2103.04922>
- Chen, S. F., & Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4), 359–394. doi:10.1006/csla.1999.0128
- Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A., & Bengio, Y. (2016, April 6). A Recurrent Latent Variable Model for Sequential Data. In *Proceedings of the 28th International Conference on Neural Informa-*

- tion Processing Systems - Volume 2* (pp. 2980–2988). Cambridge, MA, USA: MIT Press. Retrieved February 3, 2022, from <http://arxiv.org/abs/1506.02216>
- Counterfactual. (n.d.). doi:10.1093/oi/authority.20110803095642948
- Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3), 171–176. doi:10.1145/363958.363994
- Dandl, S., Molnar, C., Binder, M., & Bischl, B. (2020). Multi-Objective Counterfactual Explanations. In T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. Emmerich, & H. Trautmann (Eds.), *Parallel Problem Solving from Nature – PPSN XVI* (pp. 448–469). doi:10.1007/978-3-030-58112-1_31
- Definition of PROCESS. (n.d.). Retrieved February 17, 2022, from <https://www.merriam-webster.com/dictionary/process>
- Delaney, E., Greene, D., & Keane, M. T. (2021). Instance-Based Counterfactual Explanations for Time Series Classification. In A. A. Sánchez-Ruiz & M. W. Floyd (Eds.), *Case-Based Reasoning Research and Development* (pp. 32–47). doi:10.1007/978-3-030-86957-1_3
- Deng, L. (2012). The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine*, 29(6), 141–142. doi:10.1109/MSP.2012.2211477
- Fisher, R. A. (1936). The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7(2), 179–188. doi:10.1111/j.1469-1809.1936.tb02137.x
- Fraccaro, M., Sønderby, S. K., Paquet, U., & Winther, O. (2016, December 5). Sequential neural models with stochastic layers. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (pp. 2207–2215). Red Hook, NY, USA: Curran Associates Inc.
- Francis, W. N., & Kucera, H. (1979). *Brown corpus manual*. Department of Linguistics, Brown University, Providence, Rhode Island, US. Retrieved from <http://icame.uib.no/brown/bcm.html>
- Frank, S. L., & Christiansen, M. H. (2018). Hierarchical and sequential processing of language. *Language, Cognition and Neuroscience*, 33(9), 1213–1218. doi:10.1080/23273798.2018.1424347
- Girin, L., Leglaive, S., Bie, X., Diard, J., Hueber, T., & Alameda-Pineda, X. (2021). Dynamical Variational Autoencoders: A Comprehensive Review. *Foundations and Trends® in Machine Learning*, 15(1-2), 1–175. doi:10.1561/22000000089. arXiv: 2008.12595
- Hitchcock, C. (2020). Causal Models. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Summer 2020). Metaphysics Research Lab,

- Stanford University. Retrieved February 10, 2022, from <https://plato.stanford.edu/archives/sum2020/entries/causal-models/>
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780. doi:10.1162/neco.1997.9.8.1735
- Hompes, B. F. A., Maaradji, A., La Rosa, M., Dumas, M., Buijs, J. C. A. M., & van der Aalst, W. M. P. (2017). Discovering Causal Factors Explaining Business Process Performance Variation. In E. Dubois & K. Pohl (Eds.), *Advanced Information Systems Engineering* (pp. 177–192). doi:10.1007/978-3-319-59536-8_12
- Hsieh, C., Moreira, C., & Ouyang, C. (2021, October 31). DiCE4EL: Interpreting Process Predictions using a Milestone-Aware Counterfactual Approach. In *2021 3rd International Conference on Process Mining (ICPM)* (pp. 88–95). 2021 3rd International Conference on Process Mining (ICPM). doi:10.1109/ICPM53251.2021.9576881
- Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME—Journal of Basic Engineering*, 82, 35–45.
- Klimek, J., Klimek, J., Kraskiewicz, W., & Topolewski, M. (2021, August 17). *Long-term series forecasting with Query Selector – efficient model of sparse attention*. arXiv: 2107.08687 [cs]. Retrieved November 9, 2021, from <http://arxiv.org/abs/2107.08687>
- Krause, J., Perer, A., & Ng, K. (2016, May 7). Interacting with Predictions: Visual Inspection of Black-box Machine Learning Models. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (pp. 5686–5697). doi:10.1145/2858036.2858529
- Krishnan, R., Shalit, U., & Sontag, D. (2017). Structured Inference Networks for Nonlinear State Space Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1). Retrieved February 22, 2022, from <https://ojs.aaai.org/index.php/AAAI/article/view/10779>
- Leglaive, S., Alameda-Pineda, X., Girin, L., & Horaud, R. (2020, February 10). *A Recurrent Variational Autoencoder for Speech Enhancement*. arXiv: 1910.10942 [cs, eess]. Retrieved February 7, 2022, from <http://arxiv.org/abs/1910.10942>
- Levenshtein, V. (1965). Binary codes capable of correcting deletions, insertions, and reversals. *undefined*. Retrieved April 15, 2022, from <https://www.semanticscholar.org/paper/Binary-codes-capable-of-correcting-deletions%2C-and-Levenshtein/b2f8876482c97e804bb50a5e2433881ae31d0cdd>
- Mannhardt, F., & Blinde, D. (2017). Analyzing the trajectories of patients with sepsis using process mining: RADAR + EMISA 2017. *RADAR+EMISA 2017, Essen, Germany, June 12-13, 2017*, 72–80. Retrieved April 22,

- 2022, from <http://www.scopus.com/inward/record.url?scp=85022001209&partnerID=8YFLogxK>
- Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a large annotated corpus of English: The penn treebank. *Computational Linguistics*, 19(2), 313–330.
- Martens, D., & Provost, F. (2014). Explaining data-driven document classifications. *MIS Quarterly*, 38(1), 73–100. doi:10.25300/MISQ/2014/38.1.04
- Melnyk, I., Santos, C. N. dos, Wadhawan, K., Padhi, I., & Kumar, A. (2017, December 4). *Improved Neural Text Attribute Transfer with Non-parallel Data*. arXiv: 1711.09395 [cs]. Retrieved February 28, 2022, from <http://arxiv.org/abs/1711.09395>
- Mitton, R. (2010). Fifty years of spellchecking. *Writing Systems Research*, 2(1), 1–7. doi:10.1093/wsr/wsq004
- Molnar, C. (2019). *Interpretable machine learning. A Guide for Making Black Box Models Explainable*. Retrieved from <https://christophm.github.io/interpretable-ml-book/>
- Narendra, T., Agarwal, P., Gupta, M., & Dechu, S. (2019). Counterfactual Reasoning for Process Optimization Using Structural Causal Models. In T. Hildebrandt, B. F. van Dongen, M. Röglinger, & J. Mendling (Eds.), *Business Process Management Forum* (pp. 91–106). doi:10.1007/978-3-030-26643-1_6
- Oberst, M., & Sontag, D. (2019, June 6). *Counterfactual Off-Policy Evaluation with Gumbel-Max Structural Causal Models*. arXiv: 1905.05824 [cs, stat]. Retrieved September 22, 2021, from <http://arxiv.org/abs/1905.05824>
- Pearl, J., Glymour, M., & Jewell, N. P. (2016). *Causal inference in statistics: A primer*. Chichester, West Sussex: Wiley.
- Qafari, M. S., & van der Aalst, W. M. P. (2021). Case Level Counterfactual Reasoning in Process Mining. In S. Nurcan & A. Korthaus (Eds.), *Intelligent Information Systems* (pp. 55–63). doi:10.1007/978-3-030-79108-7_7
- Robeer, M., Bex, F., & Feelders, A. (2021, November). Generating Realistic Natural Language Counterfactuals. In *Findings of the Association for Computational Linguistics: EMNLP 2021* (pp. 3611–3625). EMNLP-Findings 2021. doi:10.18653/v1/2021.findings-emnlp.306
- Shook, C. L., Ketchen Jr., D. J., Hult, G. T. M., & Kacmar, K. M. (2004). An assessment of the use of structural equation modeling in strategic management research. *Strategic Management Journal*, 25(4), 397–404. doi:10.1002/smj.385

- Starr, W. (2021). Counterfactuals. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Summer 2021). Metaphysics Research Lab, Stanford University. Retrieved February 9, 2022, from <https://plato.stanford.edu/archives/sum2021/entries/counterfactuals/>
- Tax, N., Verenich, I., La Rosa, M., & Dumas, M. (2017). Predictive Business Process Monitoring with LSTM Neural Networks. In E. Dubois & K. Pohl (Eds.), *Advanced Information Systems Engineering* (pp. 477–492). doi:10.1007/978-3-319-59536-8_30
- Teinemaa, I., Dumas, M., La Rosa, M., & Maggi, F. M. (2018, October 23). *Outcome-Oriented Predictive Process Monitoring: Review and Benchmark*. arXiv: 1707.06766 [cs]. Retrieved May 7, 2022, from <http://arxiv.org/abs/1707.06766>
- Tsirtsis, S., De, A., & Gomez-Rodriguez, M. (2021, July 6). *Counterfactual Explanations in Sequential Decision Making Under Uncertainty*. arXiv: 2107.02776 [cs, stat]. Retrieved September 9, 2021, from <http://arxiv.org/abs/2107.02776>
- van der Aalst, W., Adriansyah, A., de Medeiros, A. K. A., Arcieri, F., Baier, T., Blickle, T., . . . Wynn, M. (2012). Process Mining Manifesto. In F. Daniel, K. Barkaoui, & S. Dustdar (Eds.), *Business Process Management Workshops* (pp. 169–194). doi:10.1007/978-3-642-28108-2_19
- Wachter, S., Mittelstadt, B., & Russell, C. (2017). Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR. *ArXiv*. doi:10.2139/ssrn.3063289
- Wang, J., Song, S., Zhu, X., & Lin, X. (2013). Efficient recovery of missing events. *Proceedings of the VLDB Endowment*, 6(10), 841–852. doi:10.14778/2536206.2536212
- Wang, K., Hua, H., & Wan, X. (2019, December 12). *Controllable Unsupervised Text Attribute Transfer via Editing Entangled Latent Representation*. arXiv: 1905.12926 [cs]. Retrieved February 28, 2022, from <http://arxiv.org/abs/1905.12926>
- Wang, Z., Zhang, J., Xu, H., Chen, X., Zhang, Y., Zhao, W. X., & Wen, J.-R. (2021, July 11). Counterfactual Data-Augmented Sequential Recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 347–356). doi:10.1145/3404835.3462855