

This section dives into the generative models that we will explore in this thesis. They cover fundamentally different approaches to the counterfactual generation of process data. We apply the viability metric established in ?? to evaluate the performance of each model.

Here, we attempt to capture the latent state-space of a model and use this space to sample counterfactual candidates. Last, we explore a technique which does not require to optimise a differentiable objective function. Instead we use the viability measure as a fitness function and maximise the fitness of each counterfactual candidate.

0.0.1 Baseline Model: Random-Search Approach

This model will act as one of the baseline methods. Here, we generate a random sequence of events. Afterwards we generate event attributes, randomly. This approach is reasonably fast, but expected to perform poorly.

As explained earlier, a randomly sampling a possible sequence of events becomes more and more unlikely the longer the sequence is and the more events are possible. One generally has a chance of $\frac{1}{A^T}$ to randomly find an event, that is generally possible given the process model. The chances decrease even more if one also generates event attributes randomly.

Therefore, we expect most models to perform better on average.

0.0.2 Baseline Model: Case-Based Approach

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

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

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

0.0.3 Generative Model: Deep Generative Approach

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 Generative Adversarial Models (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

0.0.4 Generative Model: Evolutionary Approach

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 [\[1\]](#), because it does not require to use differentiable functions. Hence, we can directly optimise the viability metric established in [\[1\]](#).

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.

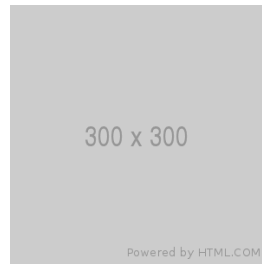


Figure 1: A figure showing the process of uniformly applying characteristics of one sequence to another

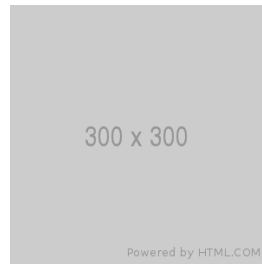


Figure 2: A figure showing the process of applying characteristics of one sequence to another using one split point

Mutation

Mutations introduce random perturbations to the offsprings. Here, only one major approach to apply these mutations was used. However, the extend in

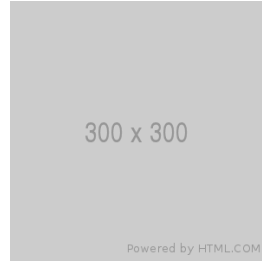


Figure 3: A figure showing the process of applying characteristics of one sequence to another using two split points

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 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.