We introduced most operator types in **??**. In this section, we describe the concrete set of operators and select a subset that we want to explore further.

For our purposes, the *gene* of a sequence consists of the sequence of events within a process instance. Hence, if an offspring inherits one parent gene, it inherits the activity associated with the event and its event attributes.

$$\overset{\text{Parent 1}}{\left(\begin{array}{cc} \underbrace{\boxed{\begin{array}{cc} a & b \\ 0.6 & 0.25 \\ 0 & 0 \\ 1.2 & 4.5 \end{array}}}_{\text{Genes passed on}} & \begin{array}{c} a \\ 0.70 \\ 1 \\ 2.3 \end{array} \end{array}\right)} + \overset{\text{Parent 2}}{\left(\begin{array}{cc} \begin{array}{cc} a & b \\ 0.6 & 0.75 \\ 0 & 0 \\ 1.2 & 4.5 \end{array} & \underbrace{\boxed{\begin{array}{cc} a & c \\ 0.64 & 0.57 \\ 1 & 0 \\ 3.3 & 3.0 \end{array}}}_{\text{Genes passed on}} \end{array}\right)} = \overset{\text{Offspring}}{\left(\begin{array}{cc} \underbrace{\boxed{\begin{array}{cc} a & b \\ 0.6 & 0.25 \\ 0 & 0 \\ 1.2 & 4.5 \end{array}}}_{\text{Inherited}} & \underbrace{\boxed{\begin{array}{cc} a & c \\ 0.64 & 0.57 \\ 1 & 0 \\ 3.3 & 3.0 \end{array}}}_{\text{Inherited}} \end{array}\right)}$$

Figure 1: A newly generated offspring inheriting genes in the form of activities and event attributes from both parents.

Our goal is to generate candidates by evaluating the sequence based on our viability measure. Our measure acts as the fitness function. The candidates that are deemed fit enough are subsequently selected to reproduce offspring. This process is explained in Figure 1. The offspring is subject to mutations. Then, we evaluate the new population and repeat the procedure until a termination condition is reached. We can optimise the viability measure established in **??**.

Operators

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

Inititation

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

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

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

Algorithm 1 The basic structure of an evolutionary algorithm.

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

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

Selection

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

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

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

Crossing

UCx: We can uniformly choose a fraction of genes of one individual (*Parent 1*) and overwrite the respective genes of another individual (*Parent 2*). The result is a new individual. We call that (*Uniform-Crossover*). Figure 2 shows a simple schematic example. By repeating this process in the opposite direction, we create two new offsprings that share both individuals' characteristics. The number of inherited genes can be adjusted using a rate factor. The number of selected positions is determined by a crossing rate between 0 and 1. The higher the crossover rate, the higher the risk of disrupting possible sequences. If we turn to Figure 2 again, we see how the second child has 2 repeating genes at the end. If a process does not allow the transition from *activity 8* to another *activity 8*, then the entire process instance becomes infeasible.

OPC: *One-Point-Crossing* is an approach suitable for sequential data of the same lengths. We can choose a point in the sequence and pass on genes of *Parent 1* onto the *Parent 2* from that point onwards and backwards (*One-Point-Crossover*). Thus, creating two new offsprings as depicted in Figure 3.

TPC: *Two-Point-Crossing* resembles its single-point counterpart. However, this time, we choose two points in the sequence and pass on the overlap, and the disjoints to generate two new offsprings. Again, Figure 4 describes the procedure visually. We can increase the number of crossover points even further. However, this increase comes at the risk of disrupting sequential dependencies.

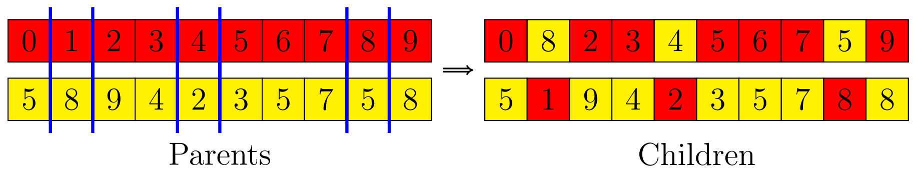

Figure 2: A crossing process of uniformly applying characteristics of one sequence to another.

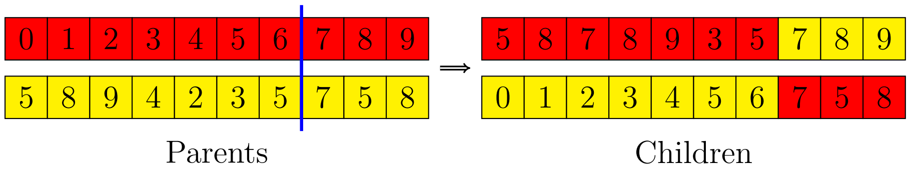

Figure 3: A One-Point example of applying characteristics of one sequence to another using one split point

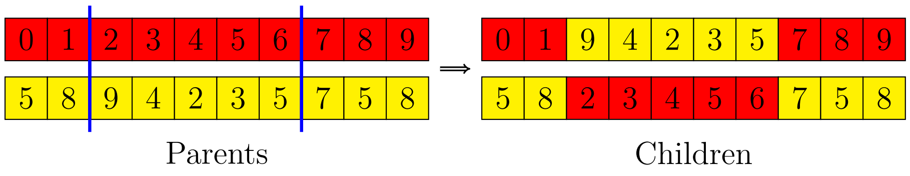

Figure 4: The process of applying characteristics of one sequence to another using two split points.

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

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

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

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

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

There are still two noteworthy topics to discuss.

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

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

Third, using the random sampler automatically disrupts the feasibility for most offspring if either of the two conditions is met. First, if the log contains categorical/binary event attributes, Gaussian samples cannot reflect these types of random variables. Second, if the vector space with which event attributes are represented is too large, it becomes less and less likely to sample something within the correct bounds. For instance, let us again consider the example on 1. However, instead of having 3 event attributes, each event had 100. Then, it becomes extremely difficult to randomly sample a set that fits the event attribute vectors.

Recombination

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

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

RR: *Ranked-Recombination* selects the new population differently than the former recombination operators. Instead of using the viability directly, we sort each individuum by every viability component separately. This approach allows us to select individuals regardless of the scales of every individual viability measure. We refer to this method as *Ranked-*

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

Naming-Conventions

We use abbreviations to refer to them in figures, tables, appendices, etc. For instance, *CBI-RWS-OPC-RM-RR* refers to an evolutionary operator configuration that samples its initial population from the data, probabilistically samples parents based on their fitness, crosses them on one point and so on. For the *Uniform-Crossing* operator, we additionally indicate its crossing rate using a number. For instance, *CBI-RWS-UC3-RM-RR* is a model using the *Uniform-Crossing* operator. The child receives roughly 30% of the genome of one parent and 70% of another parent.

Hyperparameters

The evolutionary approach comes with a number of hyperparameters.

We first discuss the *model configuration.* As shown in this section, there are a 54 to combine all operators. Depending on each operator combination, we might see very different behaviours. For instance, it is obvious that initiating the population with a random set of values can hardly converge at the same speed as a model which leverages case examples. Similarly, selecting only the fittest individuals is heavily prone to local optima issues. The decision of the appropriate set of operators is by far the most important in terms of convergence speed and result quality.

The next hyperparameter is the *termination point.* Eventually, most correctly implemented evolutionary algorithms will converge to a local optimum. Especially if only the best individuals are allowed to cross over. If we choose the termination point too early, the generated individual most likely underperforms. In contrast, selecting a termination point too far in the future might yield optimal results at the cost of time performance. Furthermore, the existence of local optima may result in very similar solutions in the end. Optimally, we find a termination point, which acts as a reasonable middle point.

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