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

0.0.1 Semi-Structured Damerau-Levenshtein Distance

Before discussing some of the similarity measures it is important to briefly introduce the Damerau-Levenshstein distance. This distance function is a modified version of the Levenshstein distance[2], which is a widely used to compute the edit-distance of two discrete sequences CITE . The most important applications are within the Natural Language Processing (NLP) discipline and the Biomedical Sciences. Within these areas, we often use the Levenshtein distance to compute the edit-distance between two words, two sentences or two DNA sequences. Note, that the elements of these sequences are often atomic symbols instead of multidimensional vectors. Generally, the distance accounts for inserts, deletions and substitutions of elements between the two serquences. 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**[Check how a paper describes the reason for usage]**. In Figure 1, we schematically show two sequneces and their distance.

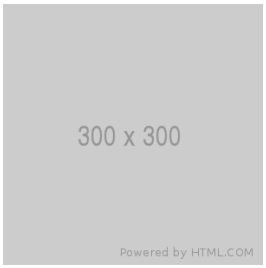

Figure 1: Shows two sequence. The edit distance is the sum of multiple operations. Blue shows an insert, red a deletion, purple a substitution and green a transposition. Therefore the edit distance is 4.

In Equation 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 seperate terms.

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

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 attibute values. Therefore, we characterise the sequences as process events in Equation 2.

$$s^1 = \{a, a, b, a\} \tag{2}$$
$$s^2 = \{a, a^*, b, a\} \tag{3}$$
$$s^3 = \{a, c, b, a\} \tag{4}$$
$$s^4 = \{a, b, a\} \qquad a, b, c \in \mathbb{R}^3 \tag{5}$$
$$a = \begin{bmatrix}2\\1\\4\end{bmatrix} \; a^* = \begin{bmatrix}3\\3\\4\end{bmatrix} \; b = \begin{bmatrix}1\\1\\0\end{bmatrix} \; c = \begin{bmatrix}3\\1\\4\end{bmatrix} \tag{6}$$

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 Figure 2. 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-Levenshstein 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.

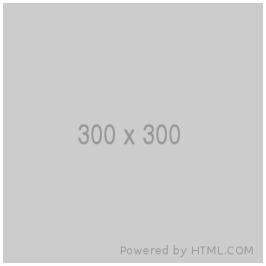

Figure 2: This figure exemplifies the differeneces between the normal DL-distance and this one used.

In order to reflect these differences in attribute values, we introduce a modified version of the Damerau-Levenshstein distance, that not only reflects the difference between two process instances but also the attribute values. We achieve this by introducing a cost function $cost_{a_i,b_j}$, which applies to a normed vector-space[1]. Concretely, we formulate the modified Damerau-Levenshstein distance as shown in Equation 7. For the remainder, we will denote this edit-distance as Semi-strucured Damerau-Levenshtein distance (SSDLD).

[1]A 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*.

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

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 attached to them. As mentioned earlier, two events that refer to the same activity can still be different due to event attributes. The distance between the event attributes determines *how* different these events are. The fourth term handles the substitution of two events. Here, we compute the substitution cost as the sum of an insertion and a deletion. The fifth term computes the cost after transposing both events. This cost is similar to term 3 only that we now consider the differences between both events after they were aligned. The last term relates to the stopping criterion of the recursive formulation of the Damerau-Levenshstein distance.

0.0.2 Discussing the Semi-Structured Damerau-Levenshtein Distance

As with all metrics, it is important to discuss important characteristics. 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 interpretration 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 seperate 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 a and b, but their attribute values are $\binom{2}{3}$ and $\binom{2}{3}$, the events are still relatively similar to each other. 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 that occurs at nighttime is not the same event as if it happens at daytime. The time domain is the main driver of distinction and the content remains a secondary actor. All the terms described in this SSDLD follow the second notion. There are two reasons for this decision. First, treating event activities and event attributes seprately would further complicate the modified Davemerau-Levenshstein-Distance, as we would have to expand the cost structure. Second, the original Davemerau-Levenshstein-Distance applies to discrete sequence, like sequences with atomic/symbolic words. By treating each event as an discrete sequence element, we remain faithful to the original function.

0.0.3 Likelihood-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 out put 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 a odds or the difference between the two likelihoods. We choose to use the odds[Needs some thought or testing. Odds may be too aggressive.]. In Equation 8, we define the function.

$$odds = \frac{p(o^*|e^*)}{p(o^*|e)} \tag{8}$$

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

0.0.4 Feasibility-Measure

[Change names in cf section.] 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 [CITE] 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 determines its feature values. Meaning, we can model density estimators for every event and use them to determine the likelihood of a set of features. In Equation 9, we define both parts of the function.

$$feasibility_e = p(a_n|a_1 \dots e_{n-1}) \approx p(a_n|a_{n-1}) \tag{9}$$
$$feasibility_f = p(f|a_n) \tag{10}$$

Here, a and f are the activity and features of a particular event. The first equation shows the approximation based on the markov assumption.

0.0.5 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. We will use a modified version of the Damerau-Levenshtein distance. This distance computes the costs associated with aligning two sequences by taking changes, inserts, deletes and transpositions of elements into account. Each of these alignment operations is typically associated with a cost of 1. This allows us to directly compute the structural difference between two sequences regardless of their lenghts. Additionally, instead of computing a cost of 1 for every operation, we compute a cost of a distance between the feature vectors of each step. This allows us to take not only the sequential differences into account but the feature differences, too. The similarity distance function is defined in Equation 11.

$$cost(a_i, b_j) = L2(a_i, b_j) \tag{11}$$
$$a_i, b_j \in \mathbb{R}^d \tag{12}$$

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

0.0.6 Sparcity-Measure

Sparsity refers to the number of changes between the factual and counterfactual sequence. We typically want to minimize the number of changes. However, sparcity 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 another modified version of the Damerau-Levenshstein edit distance. The sparcity distance function is defined in Equation 13.

$$cost(a_i, b_j) = \sum_d \mathbb{I}(a_{id} = b_{jd}) \tag{13}$$
$$a_i, b_j \in \mathbb{R}^d \tag{14}$$

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.