

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

0.0.1 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[2], which is a widely used to compute the edit-distance of two discrete sequences. The most important domains of applications being the Natural Language Processing (NLP) discipline and Biomedical Sciences. In other words, we often use the Levenshtein-Distance to compute the edit-distance between two words, two sentences or two DNA sequences. Generally, the distance accounts for inserts, deletions and substitutions of elements between the two sequences. Damerau modified the distance function to allow for transpositions. 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. Equation 1 depicts the mathematical formulation of the distance.

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

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 2.

$$s^1 = \{a, a, b, a\} \quad (2)$$

$$s^2 = \{a, a^*, b, a\} \quad (3)$$

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

$$s^4 = \{a, b, a\} \quad a, b, c \in \mathbb{R}^3 \quad (5)$$

$$a = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix} \quad a^* = \begin{bmatrix} 3 \\ 3 \\ 4 \end{bmatrix} \quad b = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad c = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix} \quad (6)$$

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 1. In other words, we cannot simply assume a static cost of 1 for each necessary 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 an appropriate distance function which includes both events and event attributes. Concretely, we can say that s^1 and s^2 only differ in terms of events only by 1 change. However, taking attribute values into account, s^1 and s^2 differ on two 2 values.

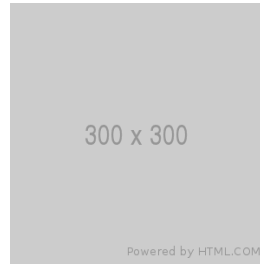


Figure 1: This figure exemplifies the differences between the normal DL-distance and this one used.

Therefore, we will introduce a modified version of the Damerau-Levenshtein-Distance, that not only reflects the difference between two process instances but also the attribute values by introducing an appropriate cost function $cost_{a_i, b_j}$. Concretely, we define a modified Damerau-Levenshtein-Distance as shown in Equation 7.

$$d_{a,b}(i, j) = \min \begin{cases} d_{a,b}(i-1, j) + \text{cost}(0, b_j) & \text{if } i > 0 \\ d_{a,b}(i, j-1) + \text{cost}(a_i, 0) & \text{if } j > 0 \\ d_{a,b}(i-1, j-1) + \text{cost}(a_i, b_j) & \text{if } i, j > 0 \\ & \& a_i = b_j \\ d_{a,b}(i-1, j-1) + \text{cost}(a_1, 0) + \text{cost}(0, b_j) & \text{if } i, j > 0 \\ & \& a_i \neq b_j \\ d_{a,b}(i-2, j-2) + \text{cost}(a_1, b_{j-1}) + \text{cost}(a_{i-1}, b_j) & \text{if } i, j > 1 \\ & \& a_i = b_{j-1} \\ & \& a_{i-1} = b_j \\ 0 & \& i = j = 0 \end{cases} \quad (7)$$

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. $\text{cost}(a, b)$ 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. For simplicity, we use $\text{cost}(x, 0)$ to denote the maximal distance. It can be understood 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 all cost-functions. 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, because it computes the distance between two points.

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. To explain the intuition behind this term, 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. First, we consider the event and its attributes as separate entities. This notion is reasonable some attributes remain static throughout the whole process run. If we take a loan application process as an example, an applicants ethnic background will not change regardless of the event. It can be considered a case attribute which remains static throughout the process run. This understanding would requires us to

modify the costs, which treats 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 $(\frac{2}{3})$ and $(\frac{2}{3})$, 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 night-time 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 modified Davemeru-Levenshtein-Distance follow the second notion. There are two reasons for this decision. First, treating event activities and event attributes separately would further complicate the modified Davemeru-Levenshtein-Distance as we would expand the cost structure. Second, the original Davemeru-Levenshtein-Distance applies to discrete sequences. By treating each event as an atomic sequence element, we remain faithful to the original function.

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 is the stopping criterion of the recursive formulation.

0.0.2 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 output a prediction based 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 odds **[Needs some thought or testing. Odds may be too aggressive.]**. In Equation 8, we define the function.

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

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

0.0.3 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 ^{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.

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

$$feasibility_f = p(f | a_n) \quad (10)$$

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

0.0.4 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.

$$cost(a_i, b_j) = L2(a_i, b_j) \quad (11)$$

$$a_i, b_j \in \mathbb{R}^d \quad (12)$$

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

0.0.5 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 another modified version of the Damerau-Levenshtein edit distance. The sparsity distance function is defined in Equation 13.

$$cost(a_i, b_j) = \sum_d \mathbb{I}(a_{id} = b_{jd}) \quad (13)$$

$$a_i, b_j \in \mathbb{R}^d \quad (14)$$

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.