1 Data Utility Metrics

Data-utility-based metrics primarily aim at assessing the similarity between original and anonymized data without computing intermediate data abstractions such as statistical models. In the context of process mining a prominent method is to compare the respective variant distributions of two event logs in a meaningful manner. We demonstrate that these distribution-oriented algorithms can be complemented with methods analyzing event logs based on absolute statistics. The underlying challenge rather lies in the interpretation of similarity and data quality which in turn depends on the data sources, -structure and experimental setup. To illustrate this ambiguity we additionally discuss brief examples.

1.1 Relative Log Similarity

As a baseline for distribution-based utility metrics we refer to the *data utility* definition in [1]. An algorithmic example of the workflow is depicted in Figure 1.1. The algorithm first reads in the original- and the anonymized event log (*Log A*, *Log B*). To obtain both variant distributions, all absolute frequencies are then converted into relative numbers per log (*Distribution A*, *Distribution B*). As an example *Log A* contains 100 traces distributed over 3 variants which leads to a fraction of 0.5 for $\sigma_3 = \langle a, b, c \rangle$.

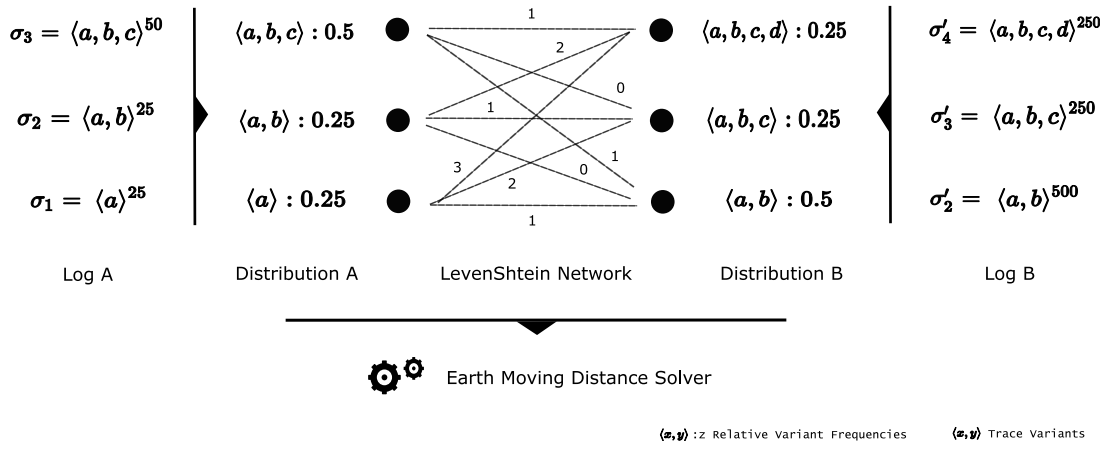

Figure 1.1: Algorithmic example of the *relative log similarity* workflow according to [1]. *Log A* and *Log B* are simple event logs, serving as inputs. Next, both variant distributions are computed by converting all absolute variant counts into relative frequencies. The similarity between single variants is expressed with a per-activity *LevenShtein* cost network. Eventually, we employ an *Earth Mover Distance* optimization to find the minimum cost alignment of the distributions. The final utility metric is returned as this cost subtracted from 1.

Next, we define the similarity between single variants as their per-activity *LevenShtein* string edit distance. The underlying idea lies in the fact, that two activity sequences equal each other more with increasing sequence overlap. In our example $\langle a, b, c \rangle$ thus shares a cost of 1 edit operations with $\langle a, b \rangle$ due to the missing activity c. Recall that in [1] all distance values are additionally normalized by the maximum sequence length of the two involved traces to constrain the final *relative log similarity* within a range of 0 to 1. Having prepared the full *LevenShtein* cost network, the essential goal is to

align both variant distributions while minimizing the transfer cost. In a more technical sense this means all variant names with their relative frequencies in *Distribution A* have to be optimally converted into *Distribution B*. Whenever sequences cannot be perfectly matched during this procedure, the cost of the respective distance metric is accumulated (e.g. half of the share of $\langle a, b, c\rangle$ (A) can be assigned to the 0.25 $\langle a, b, c\rangle$ (B) without penalty whereas the remaining half leads to imperfect alignments at either $\langle a, b, c, d\rangle$ or $\langle a, b\rangle$ (B)). The corresponding optimization problem is solved by standard *Earth Moving Distance* (EMD) routines [1]. As a result, an optimum indicates how much penalty we need at minimum to transform *Distribution A* into *Distribution B*, i.e. how similar both variant distributions are with respect to activity changes. The *relative log similarity* finally denotes this accumulated cost subtracted from 1.

Although the metric allows to measure the distribution similarity in terms of sequence assignments, there are two caveats for event data applications. The first problem is related to situations where the minimum cost assignment does not coincide with logical variant matching. A simple example is depicted in Figure 1.2 (*left*). Due to the large cost differences, σ_3 will be converted into σ_2', avoiding the penalty 100 between σ_3 and σ_1'. Consequently, both variants σ_2, σ_2' never produce a full link (0.5 to 0.5) which is undesired behaviour since they originate from the same sequence. Under the goal of evaluating event log similarities, it is therefore more logical to ensure matching all equal traces before computing the remaining mismatch (even if the overall solution is mathematically suboptimal).

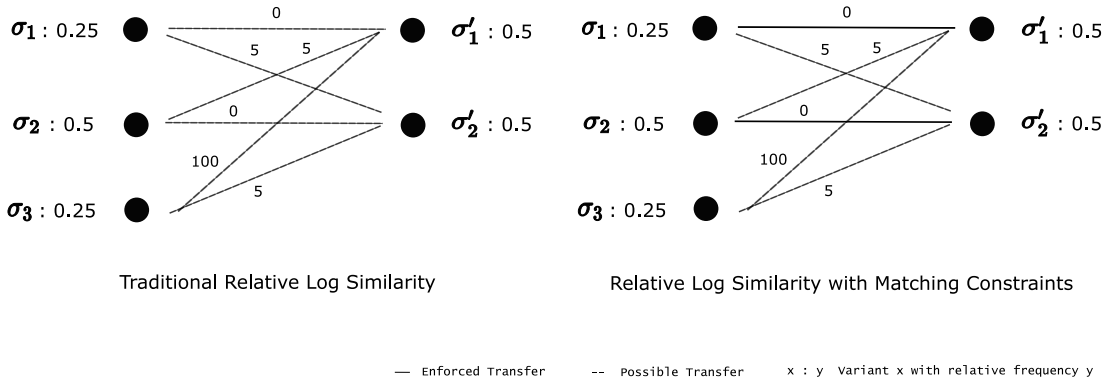

Figure 1.2: Simplified situation for illogical variant matching during *relative log similarity* computation (*left*) and the impact of our direct matching constraints (*right*). Again all vertices $\sigma : y$ denote variants σ with their relative frequencies y and the edges represent transfer costs. The specific cost structure (*left*) prevents a complete assignment of $\sigma_2 - \sigma_2'$ which would be required for estimating the similarity between both event logs. As an improvement, we enforce pairs of equal variants before running the *EMD* optimizer (*right*).

The second issue lies in the long runtime of standard *EMD* solvers on big inputs and thus rather represents an implementational challenge. During our experiments the *relative log similarity* computations demonstrated clear bottlenecks compared to both DP privatization engines and all other evaluation metrics. As a remedy for both caveats we extend the traditional algorithm by prior direct matching constraints. In accordance

with our aforementioned example this means that all equal variants are paired before optimization. Figure 1.2 (*right*) shows the corresponding enforced assignments on $\sigma_1 - \sigma_1'$ and $\sigma_2 - \sigma_2'$. Due to the fact that most event logs share a large fraction of their traces with their (ϵ, δ)-DP counterparts, this improvement therefore not only prevents illogical variant conversions, but also significantly speeds up the optimization runtime. From a technical perspective, there are multiple ways to integrate such constraints into the *relative log similarity* algorithm. Subsequently we highlight the most common options and briefly justify our choice.

- Removal of all mutual variants within both event logs before computing distributions as relative frequencies. In Figure 1.1 this refers to $\langle a, b, c \rangle^{50}$ and $\langle a, b \rangle^{25}$, yielding only $\langle a \rangle^{25}$ in *Log A* and $\{\langle a, b, c, d \rangle^{250}, \langle a, b, c \rangle^{200}, \langle a, b, d, c \rangle^{475}\}$ in *Log B*. As a result all equal pairs are separated from the *EMD* optimizer which solves the matching problem and improves performance. However, the interaction with absolute counts distorts the distribution orientation of the final metric and thus makes it difficult to interpret.

- Direct integration of suitable edge constraints for all matching pairs of relative variant frequencies into the optimization problem. Recalling the bipartite-graph structure depicted in Figure 1.1, we would therefore enforce a minimum flow of 0.25 between the $\langle a, b, c \rangle$ and $\langle a, b \rangle$ vertices respectively. Consequently, the solution domain significantly shrinks and the *EMD* algorithm cannot select illogical matches. Although this approach operates on the distribution level, the optimizer still needs to deal with a potentially large number of constraints which reflects a new source of inefficiencies.

- Removal of all mutual relative variant frequencies (distribution overlap) with renormalization before *EMD* optimization. In Figure 1.1 this denotes $\langle a, b, c \rangle : 0.25$ and $\langle a, b \rangle : 0.25$, leading to $\{\langle a, b, c \rangle : 0.25, \langle a \rangle : 0.25\}$ ($\{\langle a, b, c \rangle : 0.5, \langle a \rangle : 0.5\}$ after renormalization) in *Log A* and $\{\langle a, b, c, d \rangle : 0.25, \langle a, b \rangle : 0.25\}$ ($\{\langle a, b, c, d \rangle : 0.5, \langle a, b \rangle : 0.5\}$ after renormalization) in *Log B*. Compared with the last alternative, we obtain the same effects on the *relative log similarity* computation, but avoid any inefficient constraints. Moreover, the resulting metric maintains its statistical interpretation and is comparable with the traditional version from [1]. Due to these advantages, we choose this option for our enhancement.

The final extended *relative log similarity* routine is presented as Algorithm 1. In accordance with our schematic explanation from Figure 1.1, we first transform both event logs into their simple form (if not done during preprocessing), followed by the relative frequency derivation (S, D). Next, the cost matrix is constructed as a network of normalized *LevenShtein* distances for all variant pairs (*cost*). At the same time, any overlapping variant frequencies (direct matches) between the two logs are removed from S, D to implicitly link their frequency contribution. Eventually, the cleaned distributions S, D together with the cost matrix *cost* are provided as inputs to the *EMD* solver function. Note, that the aforementioned renormalization steps on S, D are commonly part of the solver's data preprocessing pipelines and therefore omitted in Algorithm 1. Our implementation is integrated into the *dpv* Python package and can be found on Github (*data_utility* in the *TraVaS* implementation folder). For our experiments, we employ this metric whenever

two event logs need to be statistically compared with respect to their variant distribution.

Algorithm 1: Direct Matching Relative Log Similarity

Input: Original event log L, DP event log L'
Output: Data utility u
function utility (L, L')
  forall $\sigma \in L$ **do**
    add tuple $(\sigma, L(\sigma)/|L|)$ to set S `// compute relative frequencies`
  forall $\sigma' \in L'$ **do**
    add tuple $(\sigma', L'(\sigma')/|L'|)$ to set D `// compute relative frequencies`
  forall $(\sigma, f) \in S$ **do**
    forall $(\sigma', f') \in D$ **do**
      $norm = \max(\text{len}(\sigma), \text{len}(\sigma'))$ `// max variant length`
      add triple $(\sigma, \sigma',$ LevenShtein$(\sigma, \sigma')/norm)$ to set $cost$
      if $\sigma = \sigma'$ **then**
        $corr = \min(f, f')$ `// direct matching frequencies`
        update (σ, f) in S to $(\sigma, f - corr)$ `// remove intersection`
        update (σ', f') in D to $(\sigma, f' - corr)$ `// remove intersection`
  $u = 1 - \text{EMD\_solve}(S, D, cost)$ `// run Earth Moving Solver`
  return u

1.2 Absolute Log Difference

In contrast to the *relative log similarity*, we introduce our *absolute log difference* as a metric which measures the difference between two event logs based on absolute statistics instead of distributions. The main motivation for this additional evaluation tool originates from situations where the *EMD*-conditions of the *relative log similarity* are not well suited for measuring differences between event logs. In particular, when comparing original data to (ϵ, δ) differentially private outputs, there are two important cases to differentiate. First, we consider the scenario of two DP event logs, maintaining similar variant distributions, but significant size differences. The *relative log similarity* would thus produce similar scores for both logs with respect to the original data although the DP method that produced the data most similar in size to the original log is clearly more accurate. An extreme example is shown in Figure 1.3.

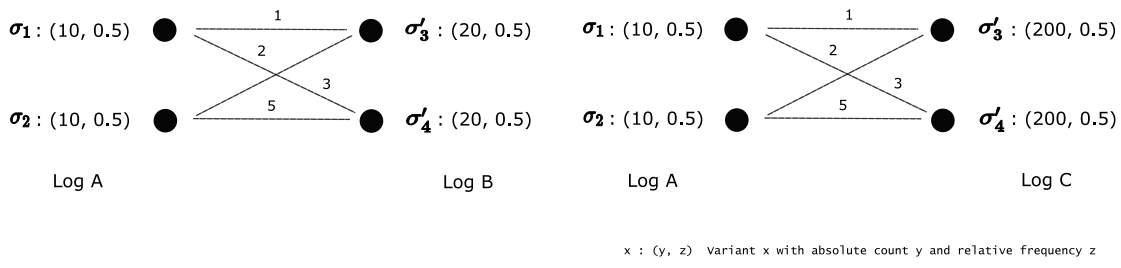

Figure 1.3: Simplified example for two event log pairs (*left* and *right*) producing the same *relative log similarity* despite significant size differences. The vertices $\sigma : (y, z)$ describe variants σ with absolute counts y and relative frequencies z. Again all edges additionally refer to the *LevenSthein* distance matrix. Since *Log B* and *Log C* maintain the same variant distribution, our *relative log similarity* fails to detect a potential data quality difference.

Whereas the left part depicts the variants $(\sigma_1, \sigma_2; \sigma'_3, \sigma'_4)$ and the *LevenShtein* cost network between an original log *Log A* and its privatized complement *Log B*, the right part

describes the same situation for another privatized event log *Log C* that only differs from *Log B* by its absolute variant numbers (200 instead of 20). As a result, both pairs would possess the exact same frequency distribution (0.5 on $\sigma_1, \sigma_2; \sigma'_3, \sigma'_4$) and thus the same *relative log similarity*, although *Log B* and *Log C* are distinct anonymized outcomes.

The second issue with Algorithm 1 is linked to the relative *LevenShtein* cost construction instead of the distributions. As an example we consider the corresponding distances of three variant pairs. Let $\sigma_0 = \langle a, a \rangle$, $\sigma_1 = \langle a, a, b, b, \ldots, b \rangle$ (10 times activity b), $\sigma_2 = \langle a, a, b, b, \ldots, b \rangle$ (20 times activity b) and $\sigma_3 = \langle a, a, b, b, \ldots, b \rangle$ (30 times activity b) be sample variants. The relative per-activity *LevenShtein* distance between $\sigma_0 - \sigma_1$, $\sigma_0 - \sigma_2$ and $\sigma_0 - \sigma_3$ then yield 10/12, 20/22 and 30/32 respectively. Even though the pairs $\sigma_0 - \sigma_1$ to $\sigma_0 - \sigma_2$ to $\sigma_0 - \sigma_3$ constantly increase by an activity difference of 10, the penalty of the relative *LevenShtein* grows nonlinear-degressively and thus represents an illogical transfer cost function for highly divergent variant lenghts. Furthermore, absolute item edit distances without normalization are easier to interpret than relative measures, in particular if they are utilized within an integer optimization routine.

Since different DP methods typically have considerably different impacts on both absolute variant frequencies and trace length, these two cases can be problematic when aiming at measuring the method performance with our *relative log similarity*. To provide a remedy for both issues, we design an *absolute log difference* metric that compares variants based on their absolute sequence changes and then finds an optimal variant assignment between two event logs. A simplified example operation is presented in Figure. 1.4.

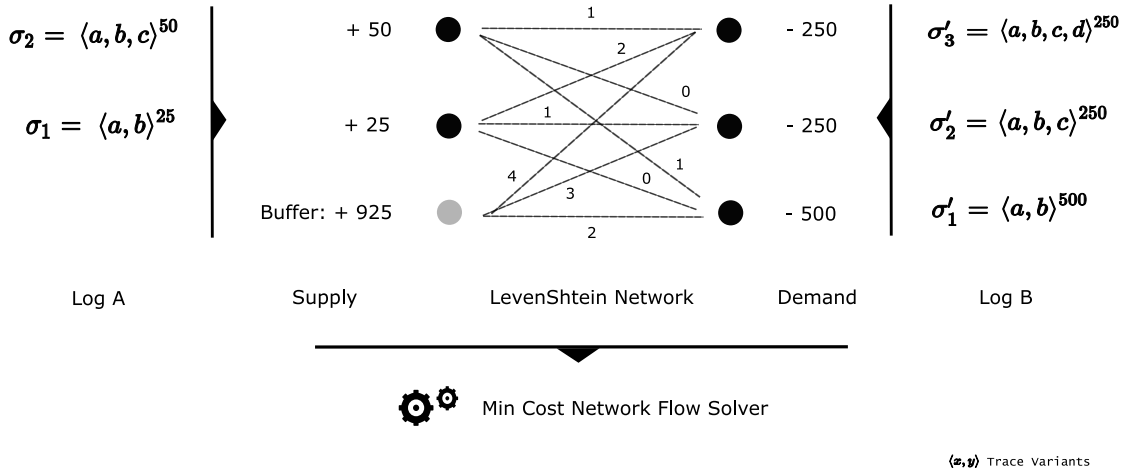

Figure 1.4: Algorithmic example of the *absolute log difference* workflow, similar to Figure 1.1. In a first step, the two input logs (*Log A*, *Log B*) are used to transfer all absolute variant frequencies into supply and demand vertices (*Supply*, *Demand*). Subsequently, a bipartite graph is constructed by adding a traditional *LevenShtein* cost network and a *Buffer* node for unmatched frequencies. At last, we evaluate the *absolute log difference* as the cost of an optimal variant assignment between *Log A* and *Log B*.

After the absolute variant frequencies are extracted from both logs (*Log A* and *Log B*), they are identified as supply and demand nodes of a bipartite graph (*Supply, Demand*). Subsequently, the overall frequency mismatch between *Log A* and *Log B* is assigned to a

buffer node (*Buffer*) and the graph edges are established as a transfer cost network based on per-activity absolute *LevenShtein* distances. We consider the examples $\sigma_2 = \langle a, b, c \rangle^{50}$ and $\sigma_3' = \langle a, b, c, d \rangle^{250}$, leading to a supply of 50, a demand of 250 and an edit distance 1 (activity d). Note that the *Buffer* represents a vertex to be assigned to all variants that are impossible to match similar neighbor variants. Hence, the cost of the incoming edges (4, 3 and 2) should always denote a separate maximum penalty with respect to the different transfer costs of the respective connected vertices (variants of *Log* B). Finally, the resulting assignment problem is optimally solved with a minimum cost network flow algorithm. As a result, the procedure measures event log differences by attempting to transform one event log into another event log while minimizing the required *LevenShtein* operations. The optimal solution, i.e. the minimal accumulated penalty thus declares the number of activity changes by which the two logs differ.

Our complete *absolute log difference* algorithm is described as Algorithm 2. In analogy to Algorithm 1, the supply and demand nodes (S, D) are constructed first, to allow for direct matching constraints when computing the cost matrix.[1] Since we do not need to deal with an intermediate conversion to frequency distributions, these constraints are then implemented by direct removal of all overlapping variants (*corr*). After the main bipartite graph generation is finished, we insert the artificial buffer node including its cost edges into the graph ($\cdot$ with *dem* and *sup*). The resulting assignment setup can finally be solved with any efficient conventional min-cost-flow optimizer. Our implementation is part of the *dpv* Python package and available on Github (*log_diff* in the *TraVaS* implementation folder)).

Algorithm 2: Absolute Log Difference

Input: Original event log L, DP event log L'
Output: Log difference cost c

```
function log_diff (L, L')
    forall σ ∈ L do
        add tuple (σ, L(σ)) to set S                          // create supply nodes
    forall σ' ∈ L' do
        add tuple (σ', −L'(σ')) to set D                      // create demand nodes
    forall (σ, f) ∈ S do
        forall (σ', f') ∈ D do
            add triple (σ, σ', LevenShtein(σ, σ')) to set cost
            if σ = σ' then
                corr = min(f, f')                  // direct matching frequencies
                update (σ, f) in S to (σ, f − corr)          // remove intersection
                update (σ', f') in D to (σ, f' + corr)       // remove intersection
    sup = sum({f | (σ, f) ∈ S})                              // obtain overall supply
    dem = sum({|f'| | (σ, f') ∈ D})                          // obtain overall demand
    if sup < dem then
        add ( · , dem − sup) to set S                        // add supply buffer
        add ( · , σ', len(σ')) for all σ' ∈ D to set cost    // add buffer cost
    else
        add ( · , −sup + dem) to set D                       // add demand buffer
        add (σ, · , len(σ)) for all σ ∈ S to set cost        // add buffer cost
    c = Min_Cost_Flow_solve(S, D, cost)                      // run Min Cost Flow Solver
    return c
```

In conclusion, we highlight that both data utility metrics have their individual use cases and limitations. The decision which metric to select should be entirely based on the

[1] An explanation of direct matching challenges can be found at Figure 1.2.

measurement and evaluation objectives. For our experiments in the paper, we typically deploy both metrics to not only demonstrate their applicability, but also to compare DP methods based on data distribution similarity as well as absolute statistics.

1.3 Note of Authorship

This document is part of the supplementary material created for the paper "TraVaG: Differentially Private Trace Variant Generation Using GANs", written by Majid Rafiei, Frederik Wangelik, Mahsa Pourbafrani and Wil M.P. Van der Aalst. Please contact *frederik.wangelik@rwth-aachen.de* for further information.

References

[1] M. Rafiei and W. M. P. van der Aalst, "Towards quantifying privacy in process mining," in *Process Mining Workshops - ICPM 2020 International Workshops*, ser. Lecture Notes in Business Information Processing. Springer, 2020.