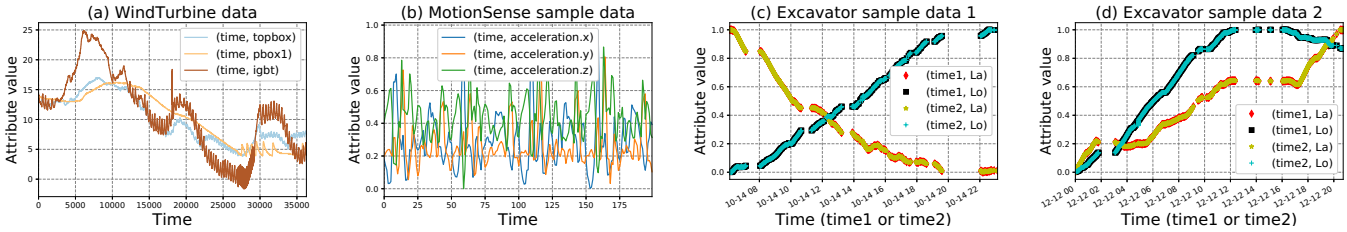

Figure 7: Time series of various temporal attribute pairs in WindTurbine, MotionSense and Excavator data

$\mathcal{D}_s(A, M(A))$ for other attribute $A \in \mathcal{R}$ in Formula 1. The relationships between attributes, however, are ignored. To utilize the temporal information, a straightforward adaption is to concatenate the values in the temporal attribute pair (T, A), namely FOD(concat).

OPAQUE [18] considers not only the distances on two attributes $\mathcal{D}_s(T, M(T))$, $\mathcal{D}_s(A, M(A))$, but also the distances of two attribute pairs, including the temporal attribute pairs $\mathcal{D}_p(T, A, M(T), M(A))$. However, it treats (T, A) as normal attributes, and considers only the co-occurrence relationships of values in attributes (T, A), without exploring the embedded temporal information.

STS3 [24] is a time series matching method. We treat each temporal attribute pair (T, A) as a time series. The time series matching distance thus measures the distance of $\mathcal{D}_p(T, A, M(T), M(A))$ for temporal attribute pairs.

LSTM [14] embeds a sequence of values into a vector. Instead of studying as time series with irregular time intervals in Figure 3, we can only treat each attribute A as a sequence ordered by the time attribute. An embedding can thus be learned by minimizing the mean square error (MSE) between the original sequence of the attribute and the predicted sequence. Indeed, the time attribute T can also be interpreted as a sequence of timestamp values, and thus apply LSTM to generate a vector. Finally, the schema matching is performed by comparing the vector distances of attributes. It is worth noting that the irregular time intervals between values are ignored as a sequence.

Our TAM can cooperate with the existing methods for evaluating attribute distances, as discussed in Section 2.1. Therefore, we have three versions, TAM(–) without using attribute distances $\mathcal{D}_s(T, M(T))$ and $\mathcal{D}_s(A, M(A))$ in Formula 1, TAM(FOD) using the attribute distances computed by FOD, and TAM(LSTM) with LSTM-based attribute distances.[14]

When there are multiple time attributes, e.g., in Excavator, we also consider two time attributes as a temporal attribute pair, i.e., (time1,time2), as introduced in Section 2.1. The corresponding distance $\mathcal{D}_p(\text{time1}, \text{time2}, M(\text{time1}), M(\text{time2}))$ is utilized in determining the mapping.

All approaches are implemented in python and report the best results by testing various parameter settings.

8.4 Reproducibility for Determining Iterations of Improving Matching

In Section 4.3, we mention to call Algorithm 1 several times to iteratively improve the transition graph matching, i.e., m'=Improve Matching(G_{TA}, G_{SB}, m'). Figures 8(a), 8(c) and 8(e), present the results by varying the number of iterations in improving the transition graph matching. LSTM and AutoEncoder time costs are not affected in this experiment on varying the iterations of improving matching. As shown, the matching accuracy is already high by calling Algorithm 1 once. The results demonstrate the effectiveness of the approximation proposed in Section 4.3. Indeed, with the increase of iterations, the matching accuracy may decrease a bit in some cases. The reason is that no transitions are observed between some values a_i and a_j, i.e., Δ_{ij} is null. In this case, the vector $\mathbf{v}_{ij}$ is simply assigned to 0, which distracts the matching. As aforesaid, it is also the same reason why the matching accuracy drops with the increase of graph size in Section 5.2.1. The corresponding matching time cost always increases with more iterations. In this sense, it is usually sufficient to call Algorithm 1 once in practice.

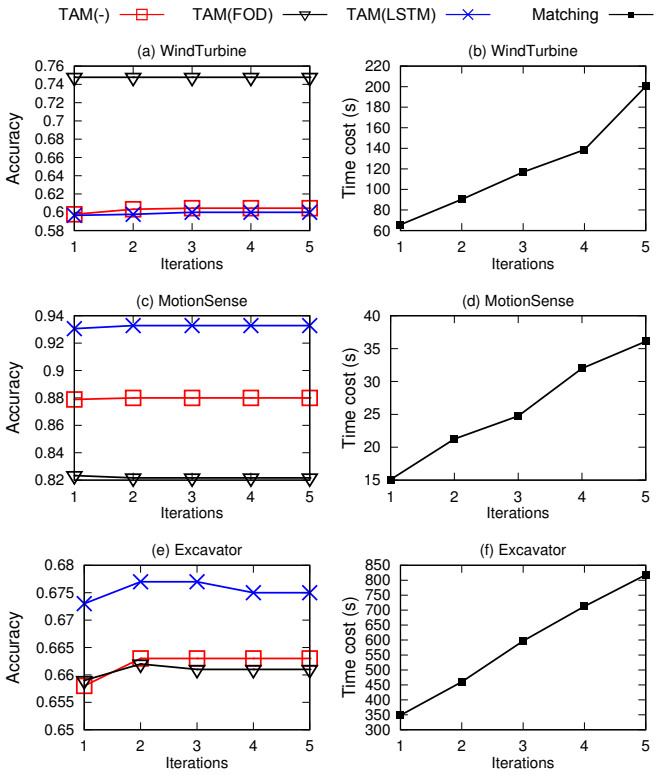

Figure 8: Varying iterations of improving matching

[14] See a discussion of choosing TAM(FOD) or TAM(LSTM) in practice in Section 5.1.