Representing Temporal Attributes for Schema Matching

Yinan Mei
BNRist, Tsinghua University
Beijing, China
myn18@mails.tsinghua.edu.cn

Shaoxu Song
BNRist, Tsinghua University
Beijing, China
sxsong@tsinghua.edu.cn

Yunsu Lee
Samsung Research
Seoul, South Korea
yunsu16.lee@samsung.com

Jungho Park
Samsung Research
Seoul, South Korea
j0106.park@samsung.com

Soo-Hyung Kim
Samsung Research
Seoul, South Korea
sooh0721.kim@samsung.com

Sungmin Yi
Samsung Research
Seoul, South Korea
sungmin.yi@samsung.com

ABSTRACT

Temporal data are prevalent, where one or several time attributes present. It is challenging to identify the temporal attributes from heterogeneous sources. The reason is that the same attribute could contain distinct values in different time spans, whereas different attributes may have highly similar timestamps and alike values. Existing studies on schema matching seldom explore the temporal information for matching attributes. In this paper, we argue to order the values in an attribute A by some time attribute T as a time series. To learn deep temporal features in the attribute pair (T, A), we devise an auto-encoder to embed the transitions of values in the time series into a vector. The temporal attribute matching (TAM) is thus to evaluate matching distance of two temporal attribute pairs by comparing their transition vectors. We show that computing the optimal matching distance is NP-hard, and present an approximation algorithm. Experiments on real datasets demonstrate the superiority of our proposal in matching temporal attributes compared to the generic schema matching approaches.

CCS CONCEPTS

• **Information systems → Mediators and data integration**.

KEYWORDS

Data Integration; Schema Matching; Temporal Data

ACM Reference Format:
Yinan Mei, Shaoxu Song, Yunsu Lee, Jungho Park, Soo-Hyung Kim, and Sungmin Yi. 2020. Representing Temporal Attributes for Schema Matching. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403115

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403115

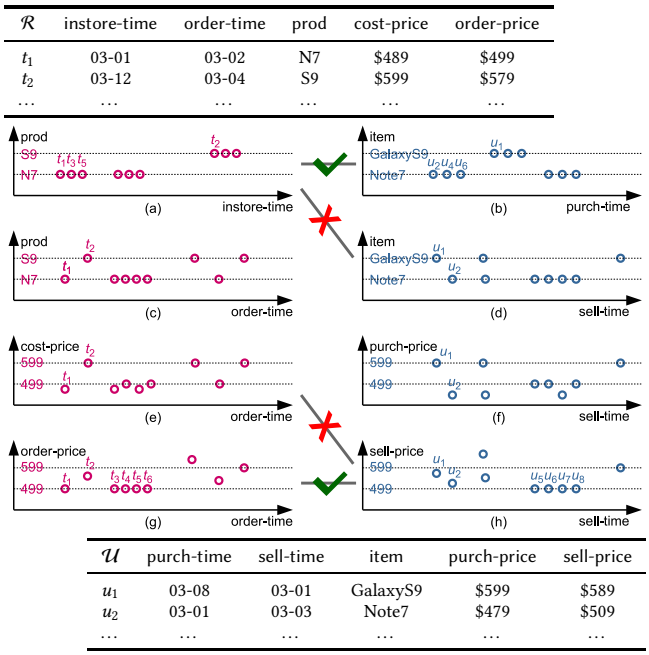

Figure 1: Order tuples (hole circles) by different time attributes to capture various temporal features for schema matching between $\mathcal{R}$ (pink) and $\mathcal{U}$ (blue)

1 INTRODUCTION

Temporal data are widely collected from various sources and stored in heterogeneous databases, ranging from traditional sale or production records to the emerging sensor readings of IoT devices. It is highly demanded to integrate the temporal data for various data mining tasks.

However, schema matching over temporal data could be very challenging. (1) Different attributes may have highly similar timestamps and alike values, e.g., cost price vs. sell price of the same products in a similar period. (2) On the other hand, the same attribute for matching may contain distinct values in different time spans. For instance, temperatures of the same wind turbine would be very different in summer and winter. While generic schema matching has been extensively studied [3], exploring the temporal information for matching attributes is surprisingly untouched.

1.1 Motivation

Consider two relations[1] $\mathcal{R}$ and $\mathcal{U}$ of sales records collected from different retailers in Figure 1. There are multiple attributes with confusing names and values, which prevent the existing schema matching approaches [3] from performing.[2] For instance, both prod in $\mathcal{R}$ and item in $\mathcal{U}$ denote the name of products. However, they are challenging to identify since product names from heterogeneous sources could be diverse. On the other hand, time attributes are hard to distinguish given the very close timestamp values, e.g., in purch-time or sell-time. Various price attributes are similar as well.

Intuitively, we explore the temporal information over various time attributes for schema matching. Interesting patterns could be observed after ordering the tuples by the corresponding timestamps. For instance, after ordering the products by instore-time in $\mathcal{R}$ in Figure 1(a), we notice that the same products are always purchased in a batch such as t_1, t_3, t_5. Similar patterns are also observed when ordered by purch-time in $\mathcal{U}$ in Figure 1(b). It is different from the ordering by order-time or sell-time in Figure 1(c) or (d), since products could be sold in any order. That is, by considering together the time attributes with others, we can successfully match (instore-time, prod) and (purch-time, item).

To give another example, we consider the price attributes with similar values.[3] After ordering the tuples by order-time in $\mathcal{R}$, there is no particular pattern on the cost-price attribute, as shown in Figure 1(e). However, an interesting pattern of consecutive tuples, t_3, t_4, t_5, t_6 with the same order-price value, is observed in Figure 1(g), since product N7 is occasionally on sale promotion in that period. In Figure 1(h), we observe the similar patterns in the sell-price attribute ordered by sell-time in $\mathcal{U}$. The attribute pairs (order-time, order-price) and (sell-time, sell-price) can thus be matched.

Motivated by both examples,[4] we propose to consider an attribute together with some time attribute, namely a temporal attribute pair. In addition to comparing single attributes, schema matching is enhanced by comparing the temporal features of temporal attribute pairs, e.g., (instore-time, prod) vs. (purch-time, item).

1.2 Contribution

Our contributions in this paper are summarized as follows. Figure 2 in Section 2 presents an overview of our proposal.

We propose a novel value transition graph to capture the temporal information in a temporal attribute pair. Values in an attribute are ordered as a time series by the other time attribute in the pair. We extract the fine-grained irregular time intervals of value transitions in the time series as discriminative features for representing the temporal attribute pair. For instance, in the time series ordered by instore-time in Figure 1(a), the transitions between N7 and S9 are infrequent with long time intervals. In contrast, there are more transitions between N7 and S9 with short time intervals in the temporal attribute pair (order-time, prod) in Figure 1(c).

[1] There are often a huge number of relations for integration, e.g., hundreds of sensors (attributes) from thousands of wind turbines (relations) with evolving schemas during maintenance. Manually matching or labeling the attributes is not practical.

[2] It is notable that considering the correlations between attributes [18] may not help. The instore-time is not necessary to be earlier than order-time, since some products are pre-ordered. And cost-price is not always lower than order-price, in case of sales.

[3] For numerical attributes, we often discretize them into a finite number of value intervals. See Section 5.2.1 for details.

[4] See the experiments in Section 5 for more examples, e.g., in mobile data or IoT data.

We devise an auto-encoder to learn the deep temporal features from the time intervals of value transitions in the time series. Intuitively, the number of transitions between the two values could be various. For example, N7 and S9 have only one transition in Figure 1(a), while there are (at least) two transitions between Note7 and GalaxyS9 in Figure 1(b). To enable the comparison over various numbers of transitions and capture deep temporal features, an auto-encoder naturally fits by embedding the time intervals of transitions (varied in number) into a unified vector.

We evaluate the matching distance of two temporal attribute pairs by matching their graphs of value transitions. It is to find the minimum matching distances on the aforesaid embedded vectors between two transition graphs. We show NP-hardness of the optimal matching problem (Theorem 1 in Section 4.2) and develop an approximate algorithm (Algorithm 1 in Section 8.2).

Experiments on real datasets demonstrate that our proposal can successfully distinguish the attributes with similar values, as well as time attributes with similar timestamps. On the other hand, we can also match the attributes with distinct values and distant timestamps. Remarkably, our temporal attribute matching (TAM) complements and cooperates with the existing schema matching approaches such as TAM(FOD) and TAM(LSTM) to achieve the best performance.

2 TAM OVERVIEW

In this section, we first present an overview of temporal attribute matching (TAM) in Figure 2. It includes two steps, (1) matching temporal attribute pairs from two relations in Section 2.1, and (2) matching transition graphs of two temporal attribute pairs in Section 2.2. Details of each step are presented in Sections 3 and 4, respectively.

2.1 Matching Temporal Attribute Pairs

Consider two relations with schemas

$$\mathcal{R}(T_1, T_2, \ldots, A_1, A_2, \ldots),$$

$$\mathcal{U}(S_1, S_2, \ldots, B_1, B_2, \ldots)$$

for schema matching, where $T_1, T_2, \ldots$ and $S_1, S_2, \ldots$ are time attributes. When two relations $\mathcal{R}$ and $\mathcal{U}$ have a different number of attributes, virtual attributes are often introduced to make them the same size [7].

The mapping between $\mathcal{R}$ and $\mathcal{U}$ is thus a bijective function $M : \mathcal{R} \to \mathcal{U}$, i.e., one-to-one match.[5] For instance, $M(A) = B$ denotes that an attribute $A \in \mathcal{R}$ is mapped to attribute $B \in \mathcal{U}$; and similarly for time attributes $M(T) = S$.

For any $T, A \in \mathcal{R}$, we call (T, A) a *temporal attribute pair*. It captures the temporal relationships between time attribute T and some other A. Similarly, we have pairs (S, B) in $\mathcal{U}$. When multiple time attributes exist, one may also consider two time attributes as a temporal attribute pair, such as (T_1, T_2). Such a relationship between time attributes is also meaningful, e.g., production-time is usually earlier than maintenance-time.

[5] We leave the more complicated many-to-one/many-to-many match as future study.

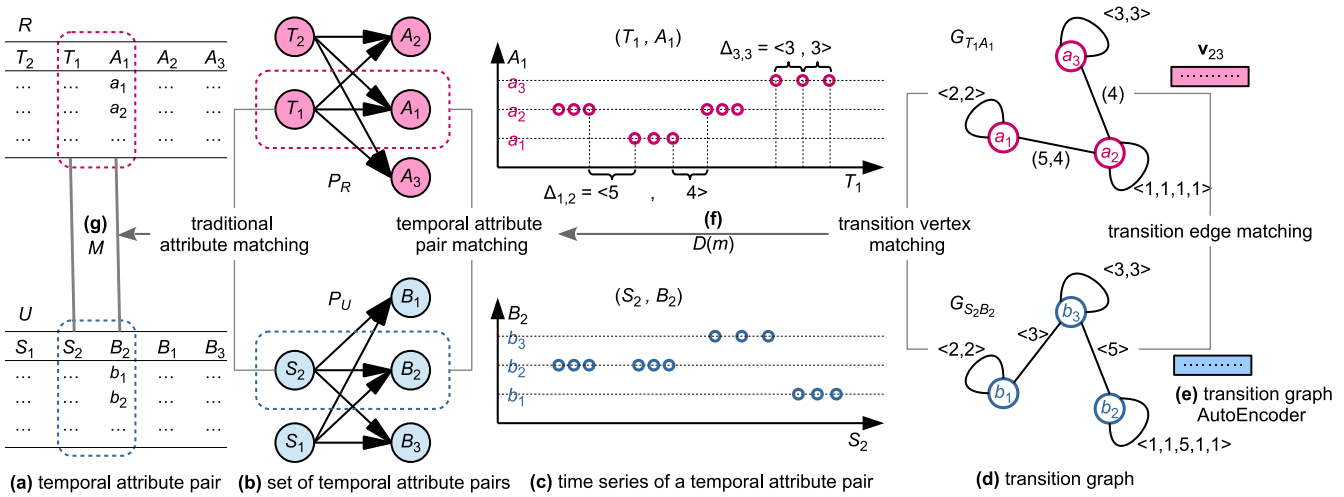

Figure 2: Overview of temporal attribute matching TAM. (a) A temporal attribute pair (T, A) in relation $\mathcal{R}$ consists of an attribute A together with a time attribute T. (b) Consider all the temporal attribute pairs in $\mathcal{R}$, denoted by $\mathcal{P}_{\mathcal{R}}$. (c) For each temporal attribute pair (T, A), we order its values a in A by the corresponding timestamps in T as a time series. Transitions between two values a_i and a_j in the time series are investigated. (d) The time series of temporal attribute pair (T, A) is thus represented by a transition graph G_{TA}. Each edge (a_i, a_j) in G_{TA} is associated with a sequence Δ_{ij} of time intervals on the transitions between two values a_i and a_j. (e) The sequences Δ_{ij} in various lengths are embedded into a unified vector $\mathbf{v}_{ij}$ by a carefully designed transition graph AutoEncoder. (f) By comparing the distances on vectors, we calculate the matching distance $D(m)$ of two transition graphs G_{TA} and G_{SB}, representing the temporal attribute pairs (T, A) and (S, B) of $\mathcal{R}$ and $\mathcal{U}$, respectively. (g) Considering the matching distances on all temporal attribute pairs, in conjunction with the traditional attribute matching distances, we find a mapping $M : \mathcal{R} \to \mathcal{U}$ between $\mathcal{R}$ and $\mathcal{U}$ with the minimum matching distance $\mathcal{D}(M)$.

Let $\mathcal{P}_{\mathcal{R}}$ and $\mathcal{P}_{\mathcal{U}}$ be the corresponding sets of temporal attribute pairs in $\mathcal{R}$ and $\mathcal{U}$, respectively. To evaluate a mapping M, we consider the matching distance between $\mathcal{R}$ and $\mathcal{U}$,

$$\mathcal{D}(M) = \sum_{T \in \mathcal{R}} \mathcal{D}_s(T, M(T)) + \sum_{A \in \mathcal{R}} \mathcal{D}_s(A, M(A)) + \sum_{(T,A) \in \mathcal{P}_{\mathcal{R}}} \mathcal{D}_p(T, A, M(T), M(A)). \quad (1)$$

$\mathcal{D}_s(T, M(T))$ is the distance between two time attributes $T \in \mathcal{R}$ and $M(T) \in \mathcal{U}$, and similarly for $\mathcal{D}_s(A, M(A))$. Existing schema matching approaches (see [26] for a survey) on evaluating the distances between two attributes can be applied, e.g., considering correlation/dependency among attributes [18]. In this sense, our proposal is complementary to the existing techniques.[6]

$\mathcal{D}_p(T, A, M(T), M(A))$ is the distance between two temporal attribute pairs (T, A) and $(M(T), M(A))$. It is the major focus of this study. Please see a brief introduction below in Section 2.2 and refer to Sections 3 and 4 for details.

The optimal schema matching problem is thus to find a mapping M with the minimum matching distance $\mathcal{D}(M)$.

2.2 Matching Transition Graphs

To evaluate the distance $\mathcal{D}_p(T, A, M(T), M(A))$ between two temporal attribute pairs (T, A) and $(M(T), M(A))$, we propose to represent the temporal features in a temporal attribute pair. Intuitively, we study the transition relationships between values in an attribute, say A, referring to time in T. It forms a transition graph[7] of the temporal attribute pair (T, A), denoted by $G_{TA} = (A, E)$. Each vertex $a \in A$ denotes a value in the attribute. Each edge, say $(a_i, a_j) \in E$, represents the transitions between values a_i and a_j in A on T. For simplicity, we denote $S = M(T), B = M(A)$. Likewise, let $G_{SB} = (B, F)$ be the transition graph of temporal attribute pair (S, B).

To evaluate the distance between temporal attribute pairs, we study the mapping between the corresponding transition graphs $m : A \to B$. It is again a bijective function. If attributes A and B have a different number of values, virtual values are introduced to make them the same size [16]. The matching distance w.r.t. m is

$$D(m) = \sum_{a_i \in A} D_v(a_i, m(a_i)) + \sum_{(a_i, a_j) \in E} D_e(a_i, a_j, m(a_i), m(a_j)), \quad (2)$$

where $D_v(a_i, m(a_i))$ is the distance between two values a_i in attribute A and $m(a_i)$ in attribute B, as defined in Formula 5 in Section 4. $D_e(a_i, a_j, m(a_i), m(a_j))$ is the distance on transitions of (a_i, a_j) and $(m(a_i), m(a_j))$, as defined in Formula 6 in Section 4.

We are interested in mapping m with the minimum $D(m)$. While the graph matching problem is generally hard as analyzed in Theorem 1, efficient approximation is possible as presented in Section 4. Finally, the distance between two temporal attribute pairs is given by $\mathcal{D}_p(T, A, M(T), M(A)) = \mathcal{D}_p(T, A, S, B) = D(m)$.

2.3 Application in Practice

Formula 1 shows that TAM is complementary to the traditional schema matching approaches, such as TAM(FOD) and TAM(LSTM)

[6] See Section 5 for an evaluation on how this proposal improves the existing methods.

[7] See a formal definition of transition graph in Definition 2.

in the experiments in Section 5. In practice, when the generic features inside/among attributes fail to work, the temporal features may perform. We can distinguish different attributes sharing similar values, given the distinct temporal features in Figures 1(e) and (h). On the other hand, by capturing the deep temporal features on value transitions, TAM successfully identifies same attributes with distinct values, such as Figures 1(a) and (b).

3 REPRESENTING TEMPORAL ATTRIBUTE

As briefly mentioned in Section 2.2, we propose a transition graph G_{TA} to denote the transitions between values in A referring to the time in T. In this way, we can represent the temporal features in a temporal attribute pair (T, A). Note that the number of transitions between two values could be various in different relations, as illustrated in Section 1.2. To enable the matching over various transitions and capture the deep temporal features, we devise a transition graph auto-encoder to embed the transitions (and values) into a unified space in Section 3.2. Graph matching can thus be performed over the transition embeddings to evaluate the distance of two temporal attribute pairs in Section 4.

3.1 Transition Graph for Temporal Attribute

The tuples in a relation r with schema $\mathcal{R}$ can be considered as a time series ordered by the timestamps in T. To capture the temporal information in (T, A), motivated by the state transition in a state machine [5], we study the transitions between values (states) of attribute A in the time series.

Definition 1 (Transition). *For two values a_i and a_j in an attribute A, we call (t_k, t_l) a transition between a_i and a_j, where $t_k, t_l \in r$, $t_k[T] < t_l[T]$, and $\{t_k[A], t_l[A]\} = \{a_i, a_j\}$.*

That is, a switch of values a_i and a_j occur on $t_k[A]$ and $t_l[A]$, referring to $\{t_k[A], t_l[A]\} = \{a_i, a_j\}$. To study the temporal information, we investigate $t_l[T] - t_k[T]$ the time interval of a transition (t_k, t_l). It denotes *how long the transition takes to switch values a_i and a_j*. We often consider only the transitions that occur in a small time window $t_l[T] - t_k[T] \leq \delta$, where δ is an optional transition time threshold. The reason is that the switch of two values too distant in time is often meaningless. For instance, studying the transitions between Samsung SGH-A188 (around 2000) and S10 (around 2019) does not help much in representing and matching the attributes, since the sales/production of products are seldom affected by those in years ago.[8]

Instead of studying only consecutive events [31, 32], we build a transition graph to represent the transitions on all the value pairs as well as self-transitions on a single value.

Definition 2 (Transition Graph). *For a temporal attribute pair (T, A), we build an undirected* transition graph $G_{TA} = (A, E)$, *where (1) each vertex $a_i \in A$ denotes a value in attribute A, and (2) each edge $e_{ij} = (a_i, a_j) \in E$ is associated with a sequence Δ_{ij} of time intervals on T, for representing transitions between values a_i and a_j.*

In order to capture both the lengths and frequencies of transitions, we study the sequence Δ_{ij} of time intervals. Note that self-edge $(a_i, a_i) \in E$ is also associated with a sequence Δ_{ii} of time intervals on T, for representing self-transitions on a_i.

[8] Please see Section 5 for an evaluation of various transition time threshold δ.

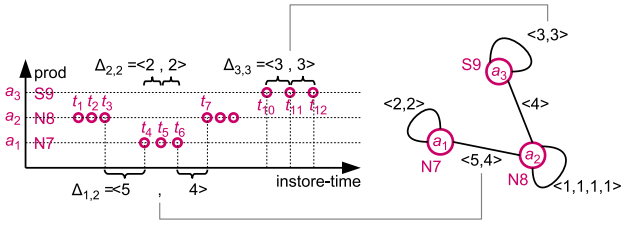

Figure 3: Transition graph for representing the temporal attribute pair (instore-time, prod), built upon the time series of prod values ordered by instore-time

Example 1. *For the temporal attribute pair (instore-time, prod), Figure 3 presents the time series ordered by instore-time, and the corresponding transition graph $G_{instore\text{-}time,prod}$. For simplicity, we denote a_1 =N7, a_2 =N8, a_3 =S9. Suppose that the transition time threshold is $\delta = 5$. For the edge $e_{12} = (a_1, a_2)$ =(N7,N8), the first transition between these two values occur in t_3 and t_4, with time interval $\Delta_{1,2}[1] = 5$. It is worth noting that t_2 and t_4 is not a valid transition, whose time interval 6 is greater than δ=5. Similarly, another transition occurs between t_6 and t_7, but not t_5 and t_7 with time interval 6. The sequence of time intervals on transitions between a_1 and a_2 is thus $\Delta_{1,2} = \langle 5, 4\rangle$. Self-transitions on a_3 = S9 occur in (t_{10}, t_{11}) and (t_{11}, t_{12}) with the same interval 3. The corresponding sequence of time intervals on self-transitions is $\Delta_{3,3} = \langle 3, 3\rangle$.*

3.2 Transition Graph AutoEncoder

After representing temporal attribute pairs as transition graphs, as introduced in Section 2.2, we need to match the transition graphs in order to evaluate the distances of two temporal attribute pairs. Unfortunately, directly evaluating the syntactic distance [9] on vertices of values a_i and edges of transition time intervals Δ_{ij} may not perform well. The reason is that values as well as the number of transitions on values could be distinct in different relations (see the intuitions in detail below). To enable matching and capture the deep temporal features, we embed vertices and edges into a low dimensional space. The distances of two transition graphs are then evaluated on the embeddings $\mathbf{v}_i$ and $\mathbf{v}_{ij}$ of vertices and edges, respectively, in Section 4.

3.2.1 Intuition. (1) Directly evaluating the syntactic distance of values in vertices may not work well, for two reasons. (1a) The same values (or timestamps) in different attributes could have different semantics, e.g., either instore-time or order-time in Figure 1. (1b) On the other hand, the values of the same attributes could be distant and not overlapping in relations $\mathcal{R}$ and $\mathcal{U}$ for matching. For example, the temperature values of the same wind turbine would be in different ranges in summer and winter. (2) It is difficult to evaluate the distance of the sequences on edges, which have different lengths of transitions. For instance, in Figure 2, there are two transitions with time intervals $\Delta_{1,2} = \langle 5, 4\rangle$ in the edge (a_1, a_2), while the edge (b_1, b_2) has only one value in the sequence $\Delta'_{1,2} = \langle 12\rangle$.

In order to match various values a_i and sequences Δ_{ij} of time intervals on transitions in different lengths, we propose to embed vertices and edges into a unified space.

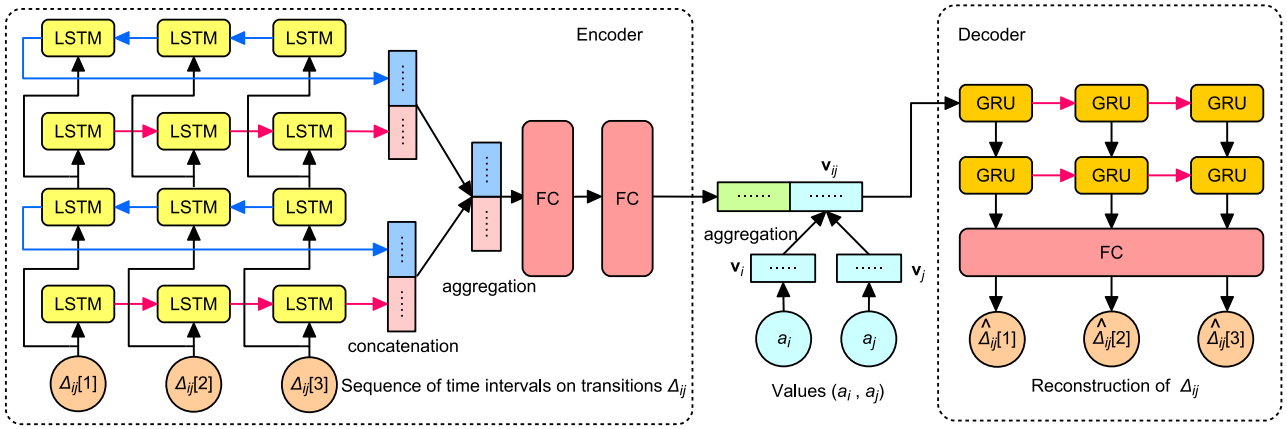

Figure 4: Structure of transition graph AutoEncoder, for learning vertex embeddings $\mathbf{v}_i$ and edge embeddings $\mathbf{v}_{ij}$. The encoder takes the sequence of time intervals on transitions Δ_{ij} as the input and puts it into two layers of Bi-LSTM. We concatenate and aggregate the outputs of two Bi-LSTM layers. The output is passed to two fully connected layers (FC). The embedding $\mathbf{v}_{ij}$ for edge e_{ij} is obtained by aggregating the corresponding vertex embeddings $\mathbf{v}_i$ and $\mathbf{v}_j$ of values a_i and a_j. The edge embedding $\mathbf{v}_{ij}$ is concatenated with the output of FC layers as the context vector of the encoder. The decoder passes the context vector to two layers of GRU and one fully connected layer to reconstruct $\widehat{\Delta}_{ij}$.

Existing graph representation methods [4, 12, 30] or graph convolution network (GCN) [20] cannot be used to represent the transition graph. Rather than single values on edges that can be embedded into an adjacent matrix, our edges e_{ij} are attached with a sequence Δ_{ij} of transition time intervals.

3.2.2 Embedding. Without providing labels on matched vertices or edges, the embedding relies on unsupervised models. Given Δ_{ij} as a sequence, we employ sequence modeling with neural networks, such as LSTM [14] and GRU [6].

Figure 4 presents our transition graph auto-encoder. It takes the sequence Δ_{ij} of each edge e_{ij} as the input, i.e., $\langle \Delta_{ij}[1], \Delta_{ij}[2], \ldots, \Delta_{ij}[n] \rangle$, and reconstruct $\langle \widehat{\Delta}_{ij}[1], \widehat{\Delta}_{ij}[2], \ldots, \widehat{\Delta}_{ij}[n] \rangle$, where n is the length of Δ_{ij}, and $\widehat{\Delta}_{ij}$ is the reconstructed sequence. The learning process is thus to minimize the reconstruction error,

$$\sum_{\kappa=1}^{n} \| \Delta_{ij}[\kappa] - \widehat{\Delta}_{ij}[\kappa] \|_2^2. \tag{3}$$

The output vector $\mathbf{z}$ of the encoder is the input of the decoder. It is concatenated by two parts, (1) the output of the sequence embedding layer w.r.t. the sequence Δ_{ij}, and (2) the output of the value embedding layer w.r.t. the values a_i and a_j in the edge e_{ij}. The input $\mathbf{v}_i$ of the value a_i can be initialized by Xavier initialization [11]. By aggregating $\mathbf{v}_i$ and $\mathbf{v}_j$, it forms the encoding $\mathbf{v}_{ij}$ of the edge e_{ij}, e.g., $\mathbf{v}_{ij} = \mathbf{v}_i + \mathbf{v}_j$. Both the vertex and edge embeddings $\mathbf{v}_i$ and $\mathbf{v}_{ij}$ are gradually trained by back propagation.

4 DETERMINING MATCHING DISTANCE

Once the vertices and edges are represented as vectors by the transition graph AutoEncoder, as illustrated in Figure 2, we are now ready to match two transition graphs. It is to find a mapping m with the minimum distance $D(m)$ between the transition graphs of two temporal attribute pairs (T, A) and (S, B). While the optimal matching problem is generally hard (Theorem 1), we develop an approximate matching algorithm (Algorithm 1).

4.1 Matching Distance

Given two transition graphs for matching, $G_{TA} = (A, E)$, $G_{SB} = (B, F)$, let $m : A \to B$ be a mapping between them. We define the distances D_v and D_e on vertices of values and edges on transitions, respectively, for evaluating $D(m)$ of transition graphs in Formula 2.

4.1.1 Vertex Distance. Given the embeddings $\mathbf{v}_i$ of the value $a_i \in A$ and $\mathbf{v}_i^m$ of the mapped value $m(a_i) \in B$, we calculate the distances of two vectors, e.g., using the squared Euclidean distance [16, 17],

$$d(\mathbf{v}_i, \mathbf{v}_i^m) = \sum_{k=1}^{n} (\mathbf{v}_i[k] - \mathbf{v}_i^m[k])^2, \tag{4}$$

where $\mathbf{v}_i[k]$ is the k-th dimension of vector $\mathbf{v}_i$, and similarly for $\mathbf{v}_i^m[k]$.

While the vectors are embedded with temporal features of transitions, the importance of each value is not considered. Recall that the length of sequence $|\Delta_{ij}|$ denotes how often the transitions occur. Intuitively, vertices and edges, which are involved in more transitions, play a more critical role in matching. By considering both the self-transitions and inter-value-transitions of a value a_i, we define its weight in matching $\frac{\sum_{a_j \in A} |\Delta_{ij}|}{\sum_{e_{kl} \in E} |\Delta_{kl}|}$, where $\sum_{e_{kl} \in E} |\Delta_{kl}|$ is the total number of transitions in G_{TA}. It denotes the normalized occurrence frequency of the value $a_i \in A$. Similarly, let $|\Delta_{ij}^m|$ be the length of sequence for transitions between the mapped values $(m(a_i), m(a_j))$. The weight of the mapped value $m(a_i)$ is $\frac{\sum_{a_j \in A} |\Delta_{ij}^m|}{\sum_{e_{kl} \in F} |\Delta_{kl}|}$.

The vertex distance between a value $a_i \in A$ and its mapped value $m(a_i) \in B$ is finally given by

$$D_v(a_i, m(a_i)) = d(\mathbf{v}_i, \mathbf{v}_i^m) * \left(\frac{\sum_{a_j \in A} |\Delta_{ij}|}{\sum_{e_{kl} \in E} |\Delta_{kl}|} + \frac{\sum_{a_j \in A} |\Delta_{ij}^m|}{\sum_{e_{kl} \in F} |\Delta_{kl}|}\right), \tag{5}$$

i.e., the embedding distance weighted by both values.

4.1.2 Edge Distance. Similar to the vertex distance, we consider the embeddings $\mathbf{v}_{ij}$ of an edge $(a_i, a_j) \in E$ and $\mathbf{v}_{ij}^m$ of the corresponding mapped edge $(m(a_i), m(a_j)) \in F$. The vector distance $d(\mathbf{v}_{ij}, \mathbf{v}_{ij}^m)$ is first calculated.

To evaluate the weight of the edge $(a_i, a_j) \in E$, we again consider $|\Delta_{ij}|$ with normalization $\frac{|\Delta_{ij}|}{\sum_{e_{kl} \in E} |\Delta_{kl}|}$. And similarly, the weight of the corresponding mapped edge $(m(a), m(a_j)) \in F$ is $\frac{|\Delta_{ij}^m|}{\sum_{e_{kl} \in F} |\Delta_{kl}|}$.

The weighted edge distance between $(a_i, a_j) \in E$ and its mapped $(m(a_i), m(a_j)) \in F$ is given by

$$D_e(a_i, a_j, m(a_i), m(a_j)) = d(\mathbf{v}_{ij}, \mathbf{v}_{ij}^m) * \left(\frac{|\Delta_{ij}|}{\sum_{e_{kl} \in E} |\Delta_{kl}|} + \frac{|\Delta_{ij}^m|}{\sum_{e_{kl} \in F} |\Delta_{kl}|}\right). \quad (6)$$

Example 2. *Consider two transition graphs $G_{T_1A_1}$ and $G_{S_2B_2}$ in Figure 2. Let m be the mapping in consideration, having $m(a_1) = b_1$, $m(a_2) = b_2$, $m(a_3) = b_3$. We have $\sum_{e_{kl} \in E} |\Delta_{kl}| = 11$ and $\sum_{e_{kl} \in F} |\Delta_{kl}| = 11$ for weight normalization. Suppose that the distance between the embeddings of a_1 and b_1 is $d(\mathbf{v}_1, \mathbf{v}_1^m) = 2$. The matching distance on vertex a_1 is $D_v(a_1, m(a_1)) = (\frac{|\Delta_{1,1}|+|\Delta_{1,2}|+|\Delta_{1,3}|}{11} + \frac{|\Delta_{1,1}^m|+|\Delta_{1,2}^m|+|\Delta_{1,3}^m|}{11}) * d(\mathbf{v}_1, \mathbf{v}_1^m) = (\frac{4}{12} + \frac{3}{11}) * 2$. Similarly, given $d(\mathbf{v}_{2,3}, \mathbf{v}_{2,3}^m) = 2.5$, the edge distance between (a_2, a_3) and the mapped (b_2, b_3) is $D_e(a_2, a_3, m(a_2), m(a_3)) = (\frac{|\Delta_{2,3}|}{11} + \frac{|\Delta_{2,3}^m|}{11}) * d(\mathbf{v}_{2,3}, \mathbf{v}_{2,3}^m) = (\frac{1}{11} + \frac{1}{11}) * 2.5$.*

4.2 Hardness Analysis

As stated in Sections 2.1 and 2.2, the optimal graph matching is to find a mapping with the minimum distance $D(m)$ or $\mathcal{D}(M)$. While the bipartite graph matching that can be efficiently solved in polynomial time [21], it is not the case for transition graph matching. We need to further evaluate the matching on edges referring to the mappings on vertices. Given the hardness of the subgraph isomorphism problem [19], it is not surprising to show the hardness of matching the transition graphs.

Theorem 1. *Given two transition graphs, G_{TA}=(A, E), G_{SB}=(B, F), with vertex and edge embeddings, the problem to determine whether there is a mapping $m : A \to B$ such that $D(m) = 0$ is NP-complete.*

4.3 Approximate Matching

Referring to the hardness in Theorem 1, we focus on approximation algorithms for graph matching. While the matching with both vertex and edge distances is generally hard, as discussed before Theorem 1, the matching considering only vertex distances could be efficient. Intuitively, we may start from a relatively good mapping using only the vertex matching, and then gradually improve the matching considering edge distances. Algorithm 1 in Section 8.2 presents the pseudo-code of approximation.

To obtain an initial mapping m, we consider only the vertex distances $D_v(a_i, m(a_i))$ in Formula 5. That is, the matching distance is

$$D'(m) = \sum_{a_i \in A} D_v(a_i, m(a_i)) \quad (7)$$

in contrast to the one with edge distances in Formula 2. For each pair of values $a_i \in A$ and $b_j \in B$, from graphs G_{TA} and G_{SB}, respectively, we calculate $D_v(a_i, b_j)$. By calling the Kuhn-Munkres algorithm [21], it returns an initial mapping m with the minimum $D'(m)$.

Next, we gradually improve the initial mapping m. Consider each value pair $a_i, a_j \in A$ of the graph $G_{TA}(A, E)$. By swapping the mapping of these two values, we obtain a new candidate mapping m. As illustrated in Line 5 in Algorithm 1 in Section 8.2, if the new mapping has a better distance $D(m) \leq D(m')$, it will replace m' as the new improved mapping. Note that $D(m)$ is the matching distance with both vertex and edge distances in Formula 2.

For each pair of values $a_i \in A$ and $b_j \in B$, the distance $D_v(a_i, b_j)$ can be calculated in $O(|A|^2)$ time, referring to Formula 5, where $|A|$ is the total number of values in attribute A.[9] Considering all the value pairs in A needs $O(|A|^4)$ time. The Kuhn-Munkres algorithm [21] can find the initial mapping m with the minimum $D'(m)$ in $O(|A|^3)$ time. The approximation complexity is thus $O(|A|^4)$. To further improve the mapping, we may call Algorithm 1 several times.

Example 3. *Consider initial mapping m on two transition graphs $G_{T_1A_1}$ and $G_{S_2B_2}$ in Figure 2, having $m(a_1) = b_1$, $m(a_2) = b_2$, $m(a_3) = b_3$. The corresponding matching distance is $D(m) = 3$. Let m' be another mapping by swapping the matching on a_1 and a_2, i.e., $m'(a_1) = b_2$, $m'(a_2) = b_1$, $m'(a_3) = a_3$. If we have $D(m') = 2.9 < D(m) = 3$, then m' is an improvement of m with smaller matching distance.*

5 EXPERIMENT

In the experiments, we compare our proposal to the existing schema matching approaches over general data without considering the temporal information, as well as deep learning methods for capturing temporal features in schema matching. The settings of the experiments are described in Section 8.3.

5.1 Comparison with Existing Approaches

In this experiment, we compare our approach to the uninterpreted schema matching methods, OPAQUE [18] and FOD [17], for general data without considering the temporal information. The reason is that sensor readings have measurement errors, e.g., distinct temperatures may be returned by different sensors in the same place. Uninterpreted methods without inferring the meaning of each value are thus preferred.[10] To capture the temporal features, we consider STS3 [24] in time series matching, which outperforms the dynamic time warping (DTW) algorithm as reported in [24]. LSTM [14] is also compared as the embedding approach.

Tables 1, 2 and 3 present the matching results over the WindTurbine, MotionSense and Excavator datasets, respectively. By varying the settings of each method,[11] we report the best results.

FOD and OPAQUE perform well over the WindTurbine dataset in Table 1. The reason is that the igbt attribute has a distribution different from those of topbox and pbox1, as shown in Figure 7(a). FOD and OPAQUE, investigating the distributions of value occurrences, show high accuracy in matching the igbt attribute. Our

[9] Including the virtual values in Section 3.1 to make attributes A and B the same size.
[10] See Section 6 for an introduction on uninterpreted schema matching.
[11] See Section 5.2 for an evaluation of our proposal under various settings.

Table 1: Average matching accuracy in WindTurbine (since there is only one time attribute, its matching accuracy is always 1), with 5 vertices, threshold $\delta = 200$, 1 iteration for TAM

Method	time	topbox	pbox	igbt	Average
OPAQUE	1.00	0.47	0.47	**1.00**	0.73
FOD	1.00	0.53	0.49	0.89	0.73
FOD(concat)	1.00	0.35	0.28	0.28	0.48
STS3	1.00	0.40	0.30	0.36	0.51
LSTM	1.00	0.40	0.46	0.27	0.53
TAM(–)	1.00	0.61	0.78	0.57	0.74
TAM(FOD)	1.00	**0.75**	**0.85**	0.79	**0.85**
TAM(LSTM)	1.00	0.61	0.78	0.57	0.74

Table 2: Average matching accuracy in MotionSense (since there is only one time attribute, its matching accuracy is always 1), with 7 vertices, threshold $\delta = 1.0$ and 1 iteration for TAM

Method	time	acc.x	acc.y	acc.z	Average
OPAQUE	1.00	0.35	0.47	0.34	0.54
FOD	1.00	0.38	0.36	0.28	0.51
FOD(concat)	1.00	0.35	0.28	0.28	0.48
STS3	1.00	0.86	**0.97**	0.88	0.93
LSTM	1.00	0.80	0.89	**0.91**	0.90
TAM(–)	1.00	0.80	0.92	0.77	0.87
TAM(FOD)	1.00	0.70	0.82	0.65	0.79
TAM(LSTM)	1.00	**0.89**	0.96	0.89	**0.94**

TAM(FOD) also benefits from the attribute distances of FOD. Moreover, by capturing the distinct transition features of values in topbox and pbox1 attributes, we can further distinguish these two attributes. As presented in Figure 7(a) in Section 8.3, the time intervals of transitions between two values in pbox1 are longer than those in topbox.

For the same reason, FOD and OPAQUE do not work over MotionSense in Table 2, given the very similar distributions of values in Figure 7(b) in Section 8.3. Simply concatenating the time attributes does not help, since data are collected with regular time intervals. STS3 with time series matching distance performs well by studying the trend of values. LSTM also shows a good performance by embedding the deep temporal features. By investigating the transition features of values, our TAM(LSTM) further improves the matching accuracy.

LSTM, considering sequences with regular time intervals, does not work over Excavator in Table 3 with irregular time intervals as shown in Figure 7(c) in Section 8.3. Our TAM successfully handles the irregular time intervals. By embedding the distinct features on time intervals of two time attributes, time1 and time2, we can distinguish them.

To sum up, our proposed TAM show the best performances (together with either FOD or LSTM) in all datasets. Indeed, by using only the transition features between values without considering the attribute matching distance, TAM(–) already shows relatively high accuracy. The results verify the rationale of studying the transitions of values over time in matching temporal attributes. For the data with different value distributions in the relations to match, such as WindTurbine data in Figure 7(a) or Excavator data in Figure 7(c), TAM(FOD) considering value distributions in attribute matching is preferred. For those data with very similar values and regular time intervals, e.g., the MotionSense data in Figure 7(b), TAM(LSTM) may perform.

Table 3: Average matching accuracy in Excavator (since there are two alternatives in matching time attributes, either both correct or both wrong, time1 and time2 share the same matching accuracies; and similarly for la and lo), with 10 vertices, threshold $\delta = 300$ and 1 iteration for TAM

Method	time1	time2	la	lo	Average
OPAQUE	0.49	0.49	0.52	0.52	0.50
FOD	0.50	0.50	0.45	0.45	0.48
FOD(concat)	0.53	0.53	0.47	0.47	0.50
STS3	0.51	0.51	0.50	0.50	0.51
LSTM	0.49	0.49	0.54	0.54	0.52
TAM(–)	0.60	0.60	0.71	0.71	0.66
TAM(FOD)	0.60	0.60	0.71	0.71	0.66
TAM(LSTM)	**0.61**	**0.61**	**0.73**	**0.73**	**0.67**

5.2 Evaluation of Proposed Methods

Next, we evaluate how the proposed method TAM performs under various settings. In addition to the average matching accuracy, we report the major time costs of different parts in our proposal, including (1) training LSTM to obtain the attribute distances $\mathcal{D}_s(T, M(T))$ and $\mathcal{D}_s(A, M(A))$ as mentioned in Section 2.1, (2) training AutoEncoder to embed the transitions of value pairs (a_i, a_j) into $\mathbf{v}_{ij}$ in Section 3, and (3) matching transition graphs G_{TA} and G_{SB} to evaluate the matching distances of two temporal attribute pairs (T, A) and (S, B) in Section 4. We omit the time costs of using FOD to compute the attribute matching distances which are very efficient compared to the LSTM training.

5.2.1 Varying Transition Graph Size. Recall that each vertex in the transition graph G_{TA} denotes a value a in the attribute A, as introduced in Definition 2. For numerical attributes, we discretize values into a finite number of intervals. That is, each value interval denotes a vertex in the transition graph. By setting different lengths of intervals, we vary the size of transition graphs.

Figures 5(a), 5(c) and 5(e), present the results by varying the number of vertices over three datasets. When the graph size is too small (the extreme case would have only two vertices), the fine-grained transitions of values cannot be adequately represented. The corresponding matching accuracy is low. On the other hand, if the graph size is too large, the number of transitions observed between two extremely fine-grained value intervals would be limited. Without sufficient observations, the training of AutoEncoder in Section 3.2 is unreliable. The matching accuracy drops.

The graph size affects the time costs of both AutoEncoder training and transition graph matching. With the increase of graph size,

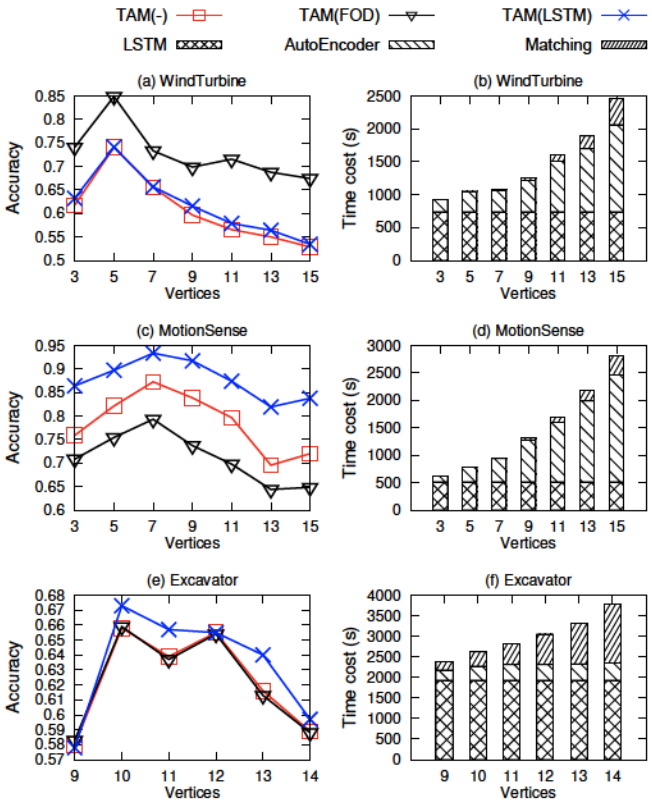

Figure 5: Varying vertices in transition graphs

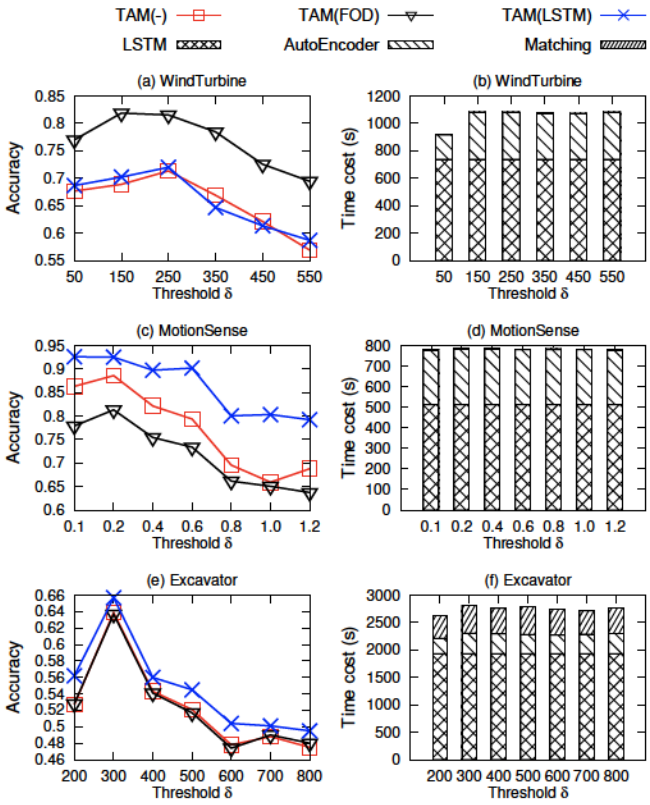

Figure 6: Varying threshold δ for transition time intervals

although the number of transitions between two vertices reduces, there are more vertices and edges in training. As illustrated in Figures 5(b), 5(d) and 5(f), the AutoEncoder training time costs generally increase with graph size. It is not surprising that the time costs of transition graph matching increase heavily given more vertices.

Thereby, a moderately large graph size is preferred, e.g., 5 to 10, which has a better matching accuracy and keeps time costs low. When values fluctuate significantly in a short period, such as WindTurbine and MotionSense in Figure 7, a small number of vertices are sufficient. On the other hand, if the value changes are small in transitions, e.g., in Excavator, fine-grained vertices are expected.

5.2.2 Varying Transition Time Threshold δ. When studying transitions between two values, we often ignore those with time intervals greater than a threshold δ, as discussed after Definition 1. The relationship between two values distant in time is often weak and thus does not help much in representing the temporal attribute pair. Varying the transition time threshold δ affects the number of transitions considered in representing a temporal attribute pair.

Figures 6(a), 6(c) and 6(e) report the results under various thresholds of transition time intervals. If the threshold δ is too small, only the transitions with very short time intervals will be considered. Without sufficient observations, AutoEncoder training over the transitions in Section 3.2 would be unreliable and leads to lower matching accuracy. When setting threshold δ too large, however, those meaningless transitions with long time intervals would disturb AutoEncoder training. The matching accuracy drops again.

The time cost of AutoEncoder training increases slightly with the threshold δ, since more transitions are considered in training. It is not significant, given the fixed graph size.

6 RELATED WORK

Schema Matching. Generic schema matching across relational databases has been widely studied [2, 3, 26]. A natural idea is to represent each attribute, e.g., either by the attribute name or the values appearing in the attribute. Once the attributes are represented, attributes with smaller matching distances on representations will be matched [22]. Since data values from heterogeneous sources may have distinct encoding, uninterpreted matching techniques are proposed. FOD [17] employs the probability mass function and the probability density function to represent attributes. OPAQUE [18] utilizes inter-attribute dependency relations in computing the matching distances of two relations. However, all the methods are dedicated to the general data, while the temporal attribute matching considered in this paper is not studied. As illustrated in the Introduction, capturing the temporal information is essential in representing and matching temporal attributes. Our proposed TAM complements and can also utilize the existing techniques on matching attributes, e.g., TAM(FOD) in the experiments.

Learning over Data Sequences. To represent temporal features, machine learning techniques over data sequences can be considered. With the feedback loop network structure, Recurrent Neural Network (RNN) performs well on various types of sequential data such as text [23], video [8], time series [13, 25], etc. Long-Short Term Memory (LSTM) [14] is one such popular variant that can handle long term event dependencies by utilizing a gated architecture. LSTM-based AutoEncoder has been proposed to learn representations of video sequences [29]. However, these RNN (LSTM) models assume regular time intervals between consecutive data in a sequence. Time-Aware LSTM [1], a variant of LSTM, takes irregular time intervals of data as inputs. Unfortunately, such a model can only represent the irregular temporal information between consecutive tuples.

Graph Matching. As shown in Figure 2, our TAM framework needs to match two transition graphs, in order to determine the matching distance of two temporal attribute pairs. It is worth noting that both vertices and edges in the transition graph are labeled with embeddings, conventional algorithms such as graph edit distance (GED) [9, 27] do not apply. Graph matching is also used in matching event data [28, 33]. However, the event data dependency graph [10, 32] does not capture the extensive transition information as introduced in Section 3.1 in this paper. Moreover, different from the bipartite graph matching [15], edge matching distances need to be studied. Nevertheless, by considering only the vertex matching distances as bipartite graph matching, Kuhn-Munkres algorithm [21] provides an initial mapping for our approximation in Section 4.3.

7 CONCLUSIONS

In this paper, we study the matching of temporal attributes from heterogeneous sources. The problem is particularly challenging since values as well as timestamps from different attributes could be similar. On the other hand, values of the same attribute may unfortunately be distinct in different time spans and thus difficult to identify. Intuitively, we can rely on the temporal information embedded in the attribute values. Existing generic schema matching techniques, however, seldom take account of such temporal information. In this sense, we propose to represent an attribute A together with a time attribute T to capture the temporal information, namely a temporal attribute pair (T, A). Values in A are ordered as a time series referring to the corresponding timestamps in T. To capture the temporal features, we embed the fine-grained transitions between two values in the time series into a unified vector. The matching distance of two temporal attribute pairs (T, A) and (S, B) is thus evaluated by comparing their transition embeddings. While computing the optimal matching distance is NP-hard (Theorem 1), we devise an approximate matching algorithm (Algorithm 8.2).. Experiments on several real-world datasets demonstrate that our proposal can successfully distinguish different attributes with similar values and timestamps, and match the same attributes with distinct values and timestamps.

Acknowledgement. This work is supported in part by the National Key Research and Development Plan (2019YFB1705301), the National Natural Science Foundation of China (61572272, 71690231), and Samsung Research, Samsung Electronics Co., Ltd.

REFERENCES

[1] I. M. Baytas, C. Xiao, X. Zhang, F. Wang, A. K. Jain, and J. Zhou. Patient subtyping via time-aware LSTM networks. In *SIGKDD*, pages 65–74. ACM, 2017.
[2] Z. Bellahsene, A. Bonifati, and E. Rahm, editors. *Schema Matching and Mapping*. Data-Centric Systems and Applications. Springer, 2011.
[3] P. A. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. *PVLDB*, 4(11):695–701, 2011.
[4] S. Bonner, I. Kureshi, J. Brennan, G. Theodoropoulos, A. S. McGough, and B. Obara. Exploring the semantic content of unsupervised graph embeddings: An empirical study. *Data Science and Engineering*, 4(3):269–289, 2019.
[5] K. Cheng and A. S. Krishnakumar. Automatic generation of functional vectors using the extended finite state machine model. *ACM TODAES*, 1(1):57–79, 1996.
[6] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.
[7] H. Elmeleegy, M. Ouzzani, and A. K. Elmagarmid. Usage-based schema matching. In *ICDE*, pages 20–29, 2008.
[8] Y. Fan, X. Lu, D. Li, and Y. Liu. Video-based emotion recognition using CNN-RNN and C3D hybrid networks. In *ICMI*, pages 445–450, 2016.
[9] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *PAA*, 13(1):113–129, 2010.
[10] Y. Gao, S. Song, X. Zhu, J. Wang, X. Lian, and L. Zou. Matching heterogeneous event data. *IEEE Trans. Knowl. Data Eng.*, 30(11):2157–2170, 2018.
[11] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010.
[12] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864. ACM, 2016.
[13] N. Gugulothu, V. TV, P. Malhotra, L. Vig, P. Agarwal, and G. Shroff. Predicting remaining useful life using time series embeddings based on recurrent neural networks. *CoRR*, abs/1709.01073, 2017.
[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
[15] A. Itai, M. Rodeh, and S. L. Tanimoto. Some matching problems for bipartite graphs. *J. ACM*, 25(4):517–525, 1978.
[16] A. R. Jaiswal, D. J. Miller, and P. Mitra. Uninterpreted schema matching with embedded value mapping under opaque column names and data values. *IEEE TKDE*, 22(2):291–304, 2010.
[17] A. R. Jaiswal, D. J. Miller, and P. Mitra. Schema matching and embedded value mapping for databases with opaque column names and mixed continuous and discrete-valued data fields. *ACM TODS*, 38(1):2:1–2:34, 2013.
[18] J. Kang and J. F. Naughton. On schema matching with opaque column names and data values. In *SIGMOD*, pages 205–216. ACM, 2003.
[19] R. M. Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2010.
[20] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*. OpenReview.net, 2017.
[21] H. W. Kuhn. The hungarian method for the assignment problem. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 29–47. Springer, 2010.
[22] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128. IEEE Computer Society, 2002.
[23] R. O. Messina and J. Louradour. Segmentation-free handwritten chinese text recognition with LSTM-RNN. In *ICDAR*, pages 171–175. IEEE Computer Society, 2015.
[24] J. Peng, H. Wang, J. Li, and H. Gao. Set-based similarity search for time series. In *SIGMOD*, pages 2039–2052. ACM, 2016.
[25] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. W. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. In *IJCAI*, pages 2627–2633. ijcai.org, 2017.
[26] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.
[27] A. Sanfeliu and K. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE SMC*, 13(3):353–362, 1983.
[28] S. Song, Y. Gao, C. Wang, X. Zhu, J. Wang, and P. S. Yu. Matching heterogeneous events with patterns. *IEEE TKDE*, 29(8):1695–1708, 2017.
[29] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using lstms. In *ICML*, pages 843–852, 2015.
[30] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *SIGKDD*, pages 1225–1234. ACM, 2016.
[31] J. Wang, S. Song, X. Zhu, X. Lin, and J. Sun. Efficient recovery of missing events. *IEEE TKDE*, 28(11):2943–2957, 2016.
[32] X. Zhu, S. Song, X. Lian, J. Wang, and L. Zou. Matching heterogeneous event data. In *SIGMOD*, pages 1211–1222. ACM, 2014.
[33] X. Zhu, S. Song, J. Wang, P. S. Yu, and J. Sun. Matching heterogeneous events with patterns. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 376–387, 2014.

8 SUPPLEMENT

8.1 Proof of Theorem 1

Proof. The problem is clearly in NP, since $D(m)$ can be computed by Formulas 2, 5 and 6 in $O(|A|^2)$ time, where $|A|$ is the number of total values.

To prove the NP-hardness, we build a reduction from the subgraph isomorphism problem, which is one of Karp's 21 NP-complete problems [19]. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, it is to determine whether there is a subgraph $G_0 = (V_0, E_0) : V_0 \subseteq V_2, E_0 \subseteq E_2 \cap (V_0 \times V_0)$ such that there is a mapping $\pi : V_1 \rightarrow V_0$ having $(v_i, v_j) \in E_1 \Leftrightarrow (\pi(v_i), \pi(v_j)) \in E_0$.

We build two transition graphs $G_{TA} = (A, E)$, $G_{SB} = (B, F)$, having $V_1 \subseteq A$, $E_1 \subseteq E$, $V_2 = B$, $E_2 \subseteq F$. Each $a_i \in A$ has $|\Delta_{ii}| = 1$ and $\mathbf{v}_i = (1)$, and similarly for $b_i \in B$. For each $(v_i, v_j) \in E_1$, the corresponding $(a_i, a_j) \in E$ has $|\Delta_{ij}| = 1$ and $\mathbf{v}_{ij} = (2)$. For each $(v_i, v_j) \in E_2$, the corresponding $(b_i, b_j) \in F$ has $|\Delta_{ij}| = 0$ and $\mathbf{v}_{ij} = (2)$. For the other edges in both graphs, we set $|\Delta_{ij}| = 0$ and having $\mathbf{v}_{ij} = (0)$ or $\mathbf{v}_{ij} = (1)$.

We show that there is a mapping π having $(v_i, v_j) \in E_1 \Leftrightarrow (\pi(v_i), \pi(v_j)) \in E_0$ if and only if there is a mapping m for $G_{TA} = (A, E)$ and $G_{SB} = (B, F)$ having $D(m) = 0$.

First, if there exists a mapping π, for each $v_i \in V_1$, we can find a $\pi(v_i) \in V_2$ that corresponds to $a_i \in A$ and $a_i^m \in B$, respectively, having $\mathbf{v}_i = (1)$ and $\mathbf{v}_i^m = (1)$ referring to the transformation. It follows $D_v(a_i, m(a_i)) = 0$ according to Formula 5. For each $(v_i, v_j) \in E_1$ with $\mathbf{v}_{ij} = (2)$, the corresponding $(\pi(v_i), \pi(v_j)) \in E_2$ has $\mathbf{v}_{ij}^m = (2)$ referring to the transformation. That is, $D_e(a_i, a_j, m(a_i), m(a_j)) = 0$ according to Formula 6. For the other $(v_i, v_j) \notin E_1$, we have $|\Delta_{ij}| = 0$, $|\Delta_{ij}^m| = 0$ in transformation. It leads to $D_e(a_i, a_j, m(a_i), m(a_j)) = 0$ as well. To sum up, we have $D(m) = 0$.

Next, if there exists a mapping m with $D(m) = 0$, we could find $(\pi(v_i), \pi(v_j)) \in E_0$ for each $(v_i, v_j) \in E_1$. According to the non-negative distances, for any $(v_i, v_j) \in E_1$, it always has $D_e(a_i, a_j, m(a_i), m(a_j)) = 0$. Referring to the transformation, the corresponding $(m(a_i), m(a_j)) \in F$ should have $|\Delta_{ij}^m| = 0$ and $\mathbf{v}_{ij}^m = (2)$. That is, we have $(\pi(v_i), \pi(v_j)) \in E_0$. □

8.2 Approximate Matching Algorithm

We show below the simple approximate matching algorithm as introduced in Section 4.3. The code is available online.[12]

Algorithm 1 ImproveMatching(G_{TA}, G_{SB}, m)

Input: An initial mapping m between graphs G_{TA} and G_{SB}
Output: An improved mapping m' with $D(m') \leq D(m)$

```
m' = m
for each a_i, a_j ∈ A of G_TA(A, E) do
    m(a_i) = m'(a_j)
    m(a_j) = m'(a_i)
    if D(m) ≤ D(m') then
        m' = m
    end if
end for
return m'
```

[12] https://github.com/EliasMei/TAM/

8.3 Experimental Settings

The experiments run on a machine with 2.1GHz CPU and 128GB memory.

8.3.1 Datasets. We employ three real datasets in experiments.

WindTurbine Data. In this dataset, we collect a number of 36,285 tuples over four attributes (time, topbox, pbox1, igbt) from the sensors of a wind turbine. The tuples are recorded with regular time intervals (every 7 seconds). Figure 7(a) presents the sample data, where each time series corresponds to a temporal attribute pair, such as (time, topbox). To evaluate schema matching, we split the dataset into 10 parts in time order as 10 relations $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{10}$. Schema matching is performed over different pairs of relations, e.g., between $\mathcal{R}_2$ and $\mathcal{R}_3$. As discussed at the beginning of this paper, the schema matching over temporal data could be very challenging. Different attributes topbox and pbox1 have similar values, while the values of the same attribute igbt in various relations for matching are very different.

MotionSense Data. [13] The dataset collects the readings of motion sensors in smartphones when the users are walking. Each user has about 6,500 records over four attributes (time, acceleration.x, acceleration.y, acceleration.z). It is also collected with regular time intervals of 0.02 second. Figure 7(b) presents the sample data in time series. As shown, the accelerations on three dimensions are very similar and hard to distinguish using only the value distance. Again, the data of a user are split into 10 parts in time order as 10 relations $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{10}$ for schema matching. We repeat the experiments on 2 users and report the average results.

Excavator Data. We collect the locations of excavators in a mine. Unlike the aforesaid datasets, it has two time attributes, stamped by CAN bus and device, respectively. The time intervals are irregular. There are 20 relations (about 12,000 tuples) collected from two excavators in 10 days over four attributes (time1, time2, lo, la). Figures 7(c) and (d) present the time series of different temporal attribute pairs, such as (time1, lo) or (time2, lo). As shown, the timestamps of two time attributes are slightly different, and thus difficult to distinguish. Instead of matching $\mathcal{R}_i$ from the same context as the previous datasets, the schema matching is performed between relations $\mathcal{R}_i$ and $\mathcal{R}_j$ from two excavators, respectively.

8.3.2 Criteria. The accuracy of schema matching over an attribute A is evaluated by the proportion of correct mappings

$$\text{accuracy}(A) = \frac{\sum_{i=1}^{n} \sum_{j=i+1}^{n} 1_{\mathcal{R}_i.A=M_{ij}(\mathcal{R}_j.A)}}{n * (n-1)}$$

where $n = 10$ is the total number of relations for matching, and $1_{\mathcal{R}_i.A=M_{ij}(\mathcal{R}_j.A)}$ is an indicator function having $1_{\mathcal{R}_i.A=M_{ij}(\mathcal{R}_j.A)} = 1$ if $\mathcal{R}_i.A = M_{ij}(\mathcal{R}_j.A)$, otherwise 0. If A is correctly matched in all relations, we have accuracy$(A) = 1$. Instead, accuracy$(A) = 0$ denotes that none of the matches for the attribute is correct.

8.3.3 Implementation Details. FOD [17] represents each attribute by a probability mass function (PMF) over the values in the attribute. The distance between any two attributes is thus calculated on their PMFs, i.e., both $\mathcal{D}_s(T, M(T))$ for time attribute $T \in \mathcal{R}$ and

[13] https://www.kaggle.com/malekzadeh/motionsense-dataset

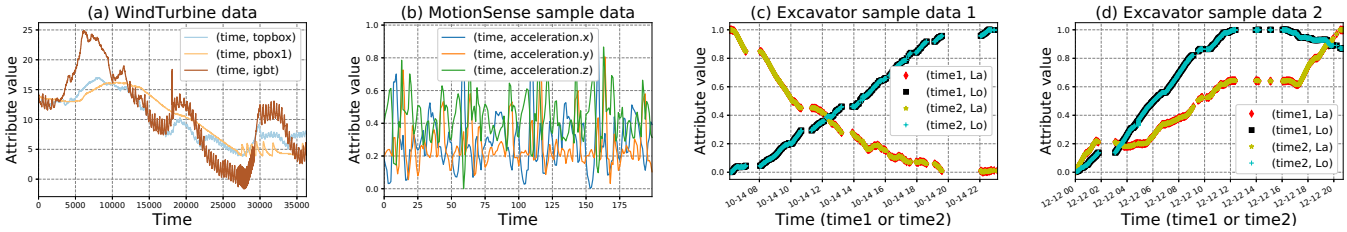

Figure 7: Time series of various temporal attribute pairs in WindTurbine, MotionSense and Excavator data

$\mathcal{D}_s(A, M(A))$ 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.

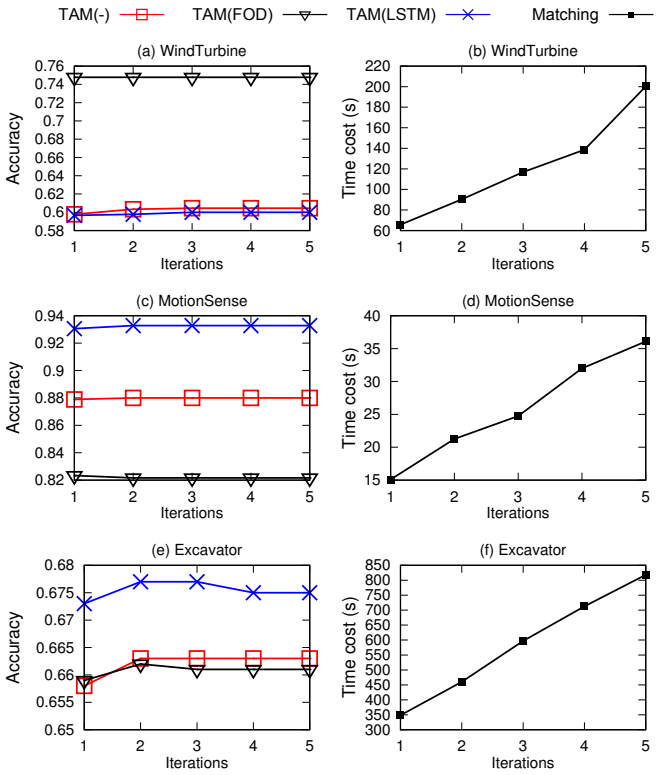

Figure 8: Varying iterations of improving matching

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