Matching Heterogeneous Events with Patterns

Xiaochen Zhu[#], Shaoxu Song[#], Jianmin Wang[#], Philip S. Yu[†], Jiaguang Sun[#]

[#]*Key Laboratory for Information System Security, MOE; TNList; School of Software, Tsinghua University, Beijing, China*
zhu-xc10@mails.tsinghua.edu.cn,{sxsong,jimwang,sunjg}@tsinghua.edu.cn
[†]*Department of Computer Science, University of Illinois at Chicago, USA psyu@cs.uic.edu*

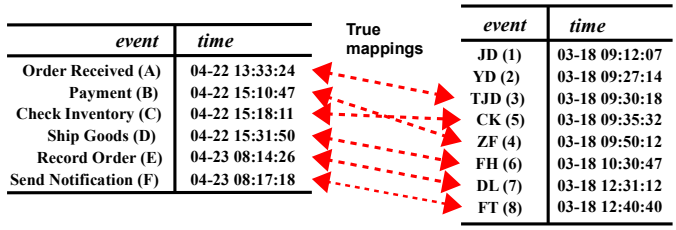
***Abstract*— A large amount of heterogeneous event data are increasingly generated, e.g., in online systems for Web services or operational systems in enterprises. Owing to the difference between event data and traditional relational data, the matching of heterogeneous events is highly non-trivial. While event names are often opaque (e.g., merely with obscure IDs), the existing structure-based matching techniques for relational data also fail to perform owing to the poor discriminative power of dependency relationships between events. We note that interesting patterns exist in the occurrence of events, which may serve as discriminative features in event matching. In this paper, we formalize the problem of matching events with patterns. A generic pattern based matching framework is proposed, which is compatible with the existing structure based techniques. To improve the matching efficiency, we devise several bounds of matching scores for pruning. Since the exploration of patterns is costly and incrementally, our proposed techniques support matching in a pay-as-you-go style, i.e., incrementally update the matching results with the increase of available patterns. Finally, extensive experiments on both real and synthetic data demonstrate the effectiveness of our pattern based matching compared with approaches adapted from existing techniques, and the efficiency improved by the bounding/pruning methods.**

I. Introduction

Information systems (e.g. OA and ERP systems) of different divisions or branches in large corporations keep on generating heterogeneous event logs. It is strongly desired to integrate the event data, e.g., for finding steps leading to a same data (provenance analysis [21]) in multiple sectors, identifying similar complex procedures (complex event processing [6]) in different branches, or obtaining a global picture of business processes (workflow views [4]) in various divisions. Without exploring the correspondence among heterogeneous events, query and analysis on the event data (simply merged together) may not yield any meaningful result.

Unfortunately, directly applying existing schema matching techniques [19] may fail to obtain the right mapping of heterogeneous events. Owing to the independent encoding systems in different sources, the widely used methods based on typographic similarity (e.g. string cosine similarity [10]) or linguistic similarity (using dictionary of ontology like WordNet [18]) of event names are often unlikely to perform (see examples below).

To solve the matching problem with "opaque" names, graph based matching approaches [13] exploit the structural information among attributes (events in our case). It relies on the statistics of dependency relationships, e.g., how often two events appear consecutively. The more similar the dependency relationship is, the more likely the corresponding events can



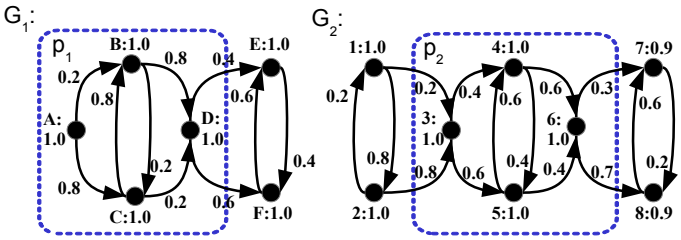
(a) A trace σ_1^1 of events in $\mathcal{L}_1$ (b) A trace σ_1^2 of events in $\mathcal{L}_2$

ID	Trace
σ_1^1	<ABCDEF>
σ_2^1	<ACBDEF>
σ_3^1	<ACBDFE>
σ_4^1	<ABCDFE>
σ_5^1	<ACBDEF>
σ_6^1	<ACBDEF>
σ_7^1	<ACBDFE>
σ_8^1	<ACBDFE>
σ_9^1	<ACBDFE>
σ_{10}^1	<ACBDFE>

(c) Event log $\mathcal{L}_1$

ID	Trace
σ_1^2	<12354678>
σ_2^2	<12345678>
σ_3^2	<2135468>
σ_4^2	<1235467>
σ_5^2	<12345687>
σ_6^2	<12345687>
σ_7^2	<21354687>
σ_8^2	<12345687>
σ_9^2	<12354687>
σ_{10}^2	<12354687>

(d) Event log $\mathcal{L}_2$



(e) Dependency graph G_1 for $\mathcal{L}_1$ (f) Dependency graph G_2 for $\mathcal{L}_2$

Fig. 1: An example of heterogeneous event logs

be mapped with each other. The matching problem is to find a "best" mapping that can maximize the similarity of dependency relationships between two datasets. Unfortunately, as illustrated in the following example, the dependency relationships (w.r.t. two consecutive events) are not discriminative enough to find the right matching.

Example 1. *Figure 1 illustrates two event logs, $\mathcal{L}_1$ and $\mathcal{L}_2$, from order processing systems of a bus manufacturer, which belong to two departments located at distinct industrial parks, respectively. Each trace, e.g., σ_1^1 in Figure 1(a), denotes a sequence of events (steps) for processing one order. An event log consists of many traces, among which the sequences of events may be different, since some of the events can be executed concurrently (e.g.* Payment(B) *and* Check Inventory(C) *in $\mathcal{L}_1$), or alternatively (e.g.,* FT(8) *in σ_3^2 or* DL(7) *in σ_4^2 in $\mathcal{L}_2$).*

As shown between Figures 1(a) and (b), events in $\mathcal{L}_1$ and $\mathcal{L}_2$

have opaque names. According to our manual investigation, Ship Goods in $\mathcal{L}_1$ is found corresponding to an event namely FH (which is an abbreviation of Chinese phonetic representation) in $\mathcal{L}_2$. Such a mapping cannot be automatically identified through a string similarity comparison (even with the help of dictionaries). For simplicity, we use ABCDEF to denote opaque event names in $\mathcal{L}_1$, while 12345678 are events in $\mathcal{L}_2$.

Figures 1(e) and (f) capture the statistical and structural information of $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. Each vertex in the directed graph denotes an event, while an edge between two events (say AB in Figure 1(e) for instance) indicates that they appear consecutively in at least one trace (e.g., σ_1^1 in Figure 1(c)). The numbers attached to vertices and edges represent the normalized frequencies of corresponding events and consecutive event pairs. For instance, 0.2 of AB means that A,B appear consecutively in 20% traces of the event log.

Frequencies of individual events are obviously not discriminative for matching, e.g., events A,B,...,1,2,... share the same frequency 1.0. According to the dependency graphs, DE shares the same frequency with 34, as well as DF with 35, BD with 23, and CD with 13. Following the intuition of high dependency relationship similarity, $D \rightarrow 3$ may be mapped referring to the aforesaid similar dependency edges. However, D and 3 denote two different events in real world.

We note that besides the simple dependency relationships, more complex event patterns (a.k.a. composite events [6]) often exist in event logs and may serve as more discriminative features. Informally, an event pattern is a group events with several dependency relationships declared inside. It is not surprising that such a complex event pattern (with multiple dependency relationships) is more discriminative than single dependency relationships.

Example 2 (Example 1 continued)**.** *Consider the pattern p_1 in blue dashed line in G_1 with four events $\{A, B, C, D\}$ and six edges $\{AB, AC, BC, CB, BD, CD\}$. It states that events B and C must occur after A before D, in either the order of BC or CB. A trace (say σ_1^1) matches with the pattern if a substring of the trace <ABCD> is a topological sort of all events in the pattern.*

Note that in G_2, there is a subgraph p_2 with events $\{3, 4, 5, 6\}$ isomorphic to p_1. It means that there may exist traces in $\mathcal{L}_2$ following the pattern p_1 as well. As event vertices and edges, we can also study the frequency of event patterns, i.e., the number of traces matching the pattern. By evaluating in $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively, p_1 and p_2 are found to share the same normalized frequency 1.0. It suggests that these two patterns may represent the same tasks (composite events). A mapping, say $A \rightarrow 3, B \rightarrow 4, C \rightarrow 5, D \rightarrow 6$, among pattern p_1 and p_2 is probably reliable, rather than the aforesaid $D \rightarrow 3$ in Example 1 by a single dependency relationship.

It is notable that vertices and edges (dependency relationships) can be interpreted as special patterns. More complex event patterns can be declared by users for certain interests, or discovered from data [2], [16], [3] (also see a discussion of choosing discriminative patterns in Section II). Therefore, in this paper, given certain patterns over event logs, we study *the problem of finding an optimal mapping that can maximize the frequency similarity (matching score) w.r.t. the patterns*.

Challenges: The main challenge of event matching originates from the large space of all possible mappings. To support efficient search of optimal mapping, it is essential to devise bounds of matching scores w.r.t. the patterns and prune those mappings with low matching scores. Due to the existence of various mappings, e.g., a pattern (say $\{D, E, F\}$) can either be mapped to $\{3, 4, 5\}$ or $\{6, 7, 8\}$, computing tight bound for each possibly mapped pattern is unpractical. Indeed, it is already hard to find all possibly mapped patterns, which is known as the subgraph isomorphism problem [9].

Moreover, we cannot expect that all the event patterns are given ahead. Obtaining event patterns would be costly, especially when involved with business semantics or manpower, and is often conducted gradually. Thereby, the event matching should also be performed in a pay-as-you-go style [20]. Efficiently updating the optimal matching after the arrival of a new pattern, however, is highly nontrivial. The matching results could be completely diverse, e.g., with or without the pattern p_1 in Figure 1. A naive idea is to recompute the optimal matching from scratch, which is obviously inefficient.

Contributions: Our major contributions in this paper are summarized as follows.

- We propose a pattern based generic framework for event matching, which is compatible with existing structure based matching methods. Efficient bounding and pruning w.r.t matching scores of possible mappings are developed.
- We devise an advanced bounding function together with two indices to accelerate the computation of the optimal event matching. In particular, a tighter bound is calculated without the costly subgraph isomorphism step.
- We extend the proposed techniques to support matching in a pay-as-you-go style and returning a list of top-k optimal mappings.
- We report extensive experimental evaluations on both real and synthetic datasets. It demonstrates that our proposed pattern based matching methods can achieve higher accuracy compared with the state-of-the-art approaches, and the advanced bounding function significantly reduces the time cost (up to 2 orders of magnitudes improvement).

The rest of this paper is organized as follows: Section II introduces preliminaries and formalizes the event matching problem. Section III describes the generic pattern based event matching framework with a simple bounding function. Section IV illustrates an advanced bounding function. SectionV and Section VI present the extensions of pay-as-you-go matching and top-k mappings, respectively. We report the experimental evaluation in Section VII. Finally, Section VIII discusses related work and Section IX concludes this paper.

II. Event Matching Problem

In this section, we formalize syntax and definitions for the event matching problem. Graph based uninterpreted matching techniques are introduced for event matching, which motivate us to study more complex event patterns. Table I lists the frequently used notations in this paper.

A. Uninterpreted Event Matching

Let V be a set of events. A trace σ is a finite sequence of events $v \in V$ ordered by their occurrence timestamps. An event log $\mathcal{L}$ is a collection of traces.

To capture the structural and statistical information among events, we introduce the dependency graph (originally for schema matching [13]) to event logs.

Definition 1 (Event Dependency Graph)**.** *An* event dependency graph *G is a labeled directed graph denoted by (V, E, f), where each event in V corresponds to a vertex, E is the edge set, and f is a labeling function of normalized frequencies that*

- *for each $v \in V$, $f(v, v)$ is the normalized frequency of event v, i.e., the number of traces in $\mathcal{L}$ that contain v (divided by $|\mathcal{L}|$), and*
- *for each edge $(v_1, v_2) \in E$, $f(v_1, v_2)$ is the normalized frequency of two consecutive events v_1v_2, i.e., the number of traces in $\mathcal{L}$ where v_1v_2 occur consecutively at least once (divided by $|\mathcal{L}|$).*

We ignore those edges with frequency 0, i.e., no dependency relationship between two events that do not appear consecutively in any trace of the event log.

Consider two event logs $\mathcal{L}_1$ and $\mathcal{L}_2$, with event sets V_1 and V_2 (without loss of generality supposing that $|V_1| \leq |V_2|$). A mapping M of events between V_1 and V_2 is an injective mapping $M : V_1 \to V_2$. For an event $v_1 \in V_1$, $v_2 = M(v_1)$ is called the corresponded event of v_1, and $v_1 \to v_2$ is called a matching/corresponding event pair.

Owing to the absence of typographic or linguistic similarity, the uninterpreted schema matching method [13] relies on the similarity of dependency relationships. A score function is employed w.r.t. any mapping M, namely *normal distance*, to evaluate the similarity of two event logs.

Definition 2 (Normal Distance)**.** *Let M be a mapping of vertices (events) over dependency graphs $G_1(V_1, E_1, f_1)$ and $G_2(V_2, E_2, f_2)$. The* normal distance *of M is defined as:*

$$D^N(M) = \sum_{v_1, v_2 \in V_1} (1 - \frac{|f_1(v_1, v_2) - f_2(M(v_1), M(v_2))|}{f_1(v_1, v_2) + f_2(M(v_1), M(v_2))})$$

Two forms of normal distances are studied. If $v_1 = v_2$ is required in the formula, the normal distance considers only the frequencies of individual events, i.e., *vertex form*. Otherwise, the normal distance is in *vertex+edge form* which considers both vertex frequencies and edge frequencies.

Normal distance is indeed the summation of frequency similarities (or differences) of corresponding vertices or edges w.r.t. mapping M. The higher the normal distance is, the more similar the vertices and edges captured by M are. As discussed in [13], the summation of similarities is favored for matching rather than the normalization of similarities. The rationale is that the normal distance function attempts to find corresponding events (vertices) as many as possible, whereas the similarity normalization function may only return one matching event pair with the highest frequency similarity.

Consequently, the matching problem is to find a mapping M that has the highest normal distance.

TABLE I: Frequently used notations

Symbol	Description
$v \in V$	an event v in event set V
$p \in \mathcal{P}$	an event pattern p in pattern set $\mathcal{P}$
$G(V, E, f)$	event dependency graph
$D^N(M)$	normal distance of a mapping M
$\delta(p)$	contribution of a pattern p to normal distance
$\Delta(p, U)$	upper bound of $\delta(p)$

Problem 1 (Event Matching Problem)**.** *Given two event logs $\mathcal{L}_1$ and $\mathcal{L}_2$, the event matching problem is to find an event mapping M that maximizes $D^N(M)$.*

Example 3 (Example 1 continued)**.** *Consider the dependency graphs G_1 and G_2 shown in Figures 1 (e) and (f). We can use normal distance to evaluate event mappings. For the true mapping $M = \{A \to 3, B \to 4, C \to 5, D \to 6, E \to 7, F \to 8\}$ illustrated in Figure 1, we have normal distance $D^N_v(M) = 5.89$ ($E \to 7$ and $F \to 8$ have similarities $1 - \frac{1-0.9}{1+0.9} = 0.947$, where other four event pairs have similarities 1.0) in vertex form and $D^N_{v+e}(M) = 13.91$ in vertex+edge form based on the Definition 2. However, neither of them is the highest normal distance. In fact, the event mapping with the highest normal distance is $M' = \{A \to 6, B \to 2, C \to 1, D \to 3, E \to 4, F \to 5\}$, where $D^N_v(M') = 6.00$ and $D^N_{v+e}(M) = 14.00$. Hence, the vertex and edge frequencies are not discriminative to find the right mapping.*

B. Event Matching with Patterns

As mentioned in Section I, complex patterns can be discriminative features in event matching. Following the convention of expressing complex event processing queries [6], we define event patterns with SEQ and AND operators as follows.

Definition 3 (Event Pattern)**.** *An event pattern specifies particular orders of event occurrence, which are defined recursively:*

- *A single event e is an event pattern;*
- *SEQ$(p_1, p_2, \ldots, p_k)$ is an event pattern in which the patterns $p_i, i \in 1, \ldots, k$, occur sequentially;*
- *AND$(p_1, p_2, \ldots, p_k)$ is an event pattern that requires the concurrent occurrence of the patterns $p_i, i \in 1, \ldots, k$, i.e., the order of p_i does not matter.*

To keep the pattern discriminative, we do not allow any other events to appear between the two patterns addressed by two consecutive parameters in the operators.

An event pattern can naturally be represented as a directed graph, where each vertex corresponds to an event [24]. Intuitively, SEQ operator specifies edges between consecutive p_i and $p_{i+1}, i \in 1, \ldots, k-1$, while AND operator indicates edges between any two p_i and $p_j, i \neq j, i, j \in 1, \ldots, k$. It is worth noting that for all the events $e_1, e_2, ..., e_k$ included in a pattern and $i \neq j, i, j \in 1, \ldots, k$, we assume that there should be $e_i \neq e_j$, since some translated graphs of distinct patterns may be the same if the duplication of events are permitted (e.g. SEQ(A, B, A, B) and AND(A, B)).

Definition 4 (Trace Matching Pattern)**.** *Let σ be a trace and p be an event pattern in graph form. We say that σ* matches *with p, if there is a substring of σ which is a topological sort of all events in p.*

We define the normalized frequency $f(p)$ of a pattern p as the number of traces matching pattern p divided by the total number of traces in event log $\mathcal{L}$. Without loss of generality, in the following, we denote p as the patterns in $\mathcal{L}_1$ by default. For the patterns in $\mathcal{L}_2$, the computation is symmetric.

Definition 5 (Pattern Normal Distance)**.** *Let M be a mapping of events over dependency graphs $G_1(V_1, E_1, f_1)$ and $G_2(V_2, E_2, f_2)$. For a set of patterns $\mathcal{P}$, the* pattern normal distance *of M is defined as:*

$$D^N(M) = \sum_{p \in \mathcal{P}} 1 - \frac{|f_1(p) - f_2(M(p))|}{f_1(p) + f_2(M(p))}, \qquad (1)$$

where $M(p)$ is the pattern in G_2 corresponding to p in G_1 via the mapping M such that each event v in p maps to an event $M(v)$ in $M(p)$.

We denote $\delta(p) = 1 - \frac{|f_1(p) - f_2(M(p))|}{f_1(p) + f_2(M(p))}$ for convenience in the following, i.e., $D^N(M) = \sum_{p \in \mathcal{P}} \delta(p)$.

Following the intuition of dependency similarity in vertex/edge based matching, we expect that the normalized frequencies of p in $\mathcal{L}_1$ and its corresponding $M(p)$ in $\mathcal{L}_2$ are as similar as possible, i.e., maximize the pattern normal distance.

Note that vertices and edges are special patterns. Therefore, pattern based matching can be interpreted as a generalization of the existing vertex/edge based matching.

Example 4 (Example 3 continued)**.** *Consider a pattern p_1 =SEQ(A,AND(B, C),D) in Figure 1 (e). We describe p_1 as a graph. The vertices of events are {A,B,C,D}. We add two edges BC, CB due to pattern AND(B, C). According to SEQ(A,AND(B, C),D), both B and C can be performed after A and should be done before D. Thus, we add another 4 edges AB, AC, BD and CD. The graph translated from p_1 is a subgraph of G_1 surrounded by blue dashed line in Figure 1(e).*

For the true mapping $M = \{A \to 3, B \to 4, C \to 5, D \to 6, E \to 7, F \to 8\}$, pattern p_1 (in G_1) corresponds to a subgraph p_2 in G_2. Since all traces in $\mathcal{L}_1$ and $\mathcal{L}_2$ match with p_1 and p_2, respectively, we have $f_1(p_1) = f_2(p_2) = 1.0$. By considering all vertices and edges as patterns in formula (1), the pattern normal distance of M is $D^N(M) = 14.91$.

However, the pattern p_1 has no mapped pattern w.r.t. $M' = \{A \to 6, B \to 2, C \to 1, D \to 3, E \to 4, F \to 5\}$. The pattern normal distance of M' is still 14. By introducing p_1, the true mapping M with the highest pattern normal distance beats M'.

Although it is not the focus of this study, we note that interesting event patterns are often obtained in two ways. 1) Event patterns may be available in business process analyzing systems of enterprises [6]. 2) There are many existing methods for discovering event patterns in event log [2], [16], [3].

We provide some guidelines instead, for choosing possibly "good" event patterns for matching. Intuitively, an event pattern is probably discriminative if no other patterns can be found with the same structure, or its frequency is different from other patterns with the same structure. On the other hand, a pattern with common structure (e.g., a 3-vertex-path pattern {A,B,D} in Figure 1) may be less discriminative, since it has a high chance of mapping to many irrelevant patterns.

III. A Generic Event Matching Framework

The total number of distinct mapping M is $n(n-1)(n-2)...(n-m+1)$, where $n = \max(|V_1|, |V_2|)$, $m = \min(|V_1|, |V_2|)$. Obviously, it is highly time-consuming to enumerate all the possible corresponding relations and choose the one that maximizes the normal distance. Instead, we employ the A* search strategy to gradually construct the optimal mapping, and prune other mappings according to their upper bounds of normal distances. There are two key issues to address in the search algorithm: 1) the efficient computation of a pattern's contribution $\delta(p)$, in particular its frequency $f(p)$ in event logs; 2) the effective estimation of upper bounds of contributions $\delta(p)$ in possible mappings.

A. Overview of A* Search

The process of A* search algorithm follows the growth of A* search tree, e.g., in Figure 2. Each node in the tree represents an intermediate result (M, U_1, U_2), where M is the current partial matching on a subset of events $V_1 \setminus U_1$ and $V_2 \setminus U_2$, U_1 is the set of unmapped events (vertices) in V_1, and U_2 is the set of unmapped events in V_2. Two important values *g* and *h* are defined on each tree node. The value *g* is the normal distance of the current partial matching, i.e., $g = D^N(M)$. The value *h* is an upper bound of normal distances which can be further contributed by matching the remaining events among U_1 and U_2. Consequently, $g(M, U_1, U_2) + h(M, U_1, U_2)$ serves as an upper bound of all mappings expanded from M. (The computation of g and h will be presented soon.)

Algorithm 1 presents the pseudo code of the A* search algorithm. Initially, we add $(\emptyset, V_1, V_2)$ as the root of search tree, in Line 1. In each iteration, we select an un-visited tree node (M, U_1, U_2) with the maximum $g + h$ value (i.e., with the maximum upper bound, in Line 3). If either of U_1 and U_2 is empty, the optimal mapping M is obtained; otherwise, we further expand the mapping. In the latter case, we pick up one event a from U_1 (see the selection of a below). For each $b \in U_2$, we create a child node for (M, U_1, U_2) by appending

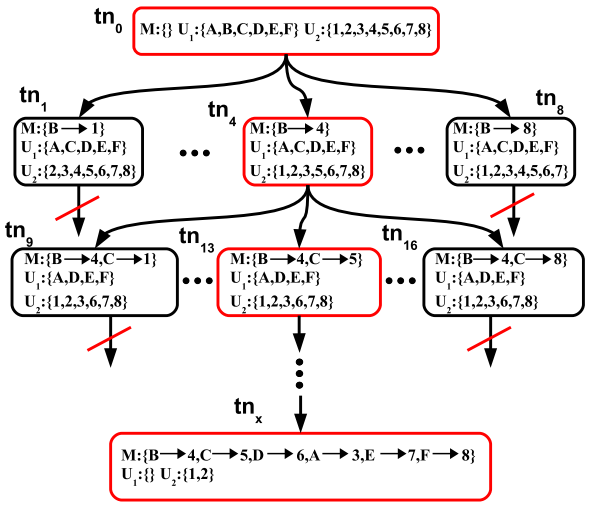


Fig. 2: An example of A* search tree

$a \to b$ to M and removing a, b from U_1 and U_2, respectively. We denote (M', U_1', U_2') a child node of (M, U_1, U_2) where $U_1' = U_1 \backslash \{a\}, U_2' = U_2 \backslash \{b\}$ and $M' : V_1 \setminus U_1' \to V_2 \setminus U_2'$. That is, we expand M to M' with the new mapping $a \to b$, in Line 7.

To select $a \in U_1$ for expanding in Line 5, we consider an ordering of events by the number of patterns that the event is involved in. Intuitively, the early the patterns are included in M, the higher the change is of pruning other mappings. Thereby, we select a vertex which is included by most of the patterns in each step.

Algorithm 1 EVENT MATCHING ALGORITHM

Input: Two event logs $\mathcal{L}_1$ and $\mathcal{L}_2$ with event sets V_1 and V_2
Output: The optimal event mapping M with the maximum pattern normal distance

1: $Q := \{(\emptyset, V_1, V_2)\}$
2: **repeat**
3: $(M, U_1, U_2) := \arg\max_{(M^i, U_1^i, U_2^i) \in Q} g(M^i, U_1^i, U_2^i) + h(M^i, U_1^i, U_2^i)$
4: $Q := Q \backslash \{(M, U_1, U_2)\}$
5: **if** $a :=$ the next event in U_1 exists **then**
6: **for** each $b \in U_2$ **do**
7: compute g and h for child (M', U_1', U_2') of (M, U_1, U_2) by expanding $a \to b$
8: $Q := Q \cup \{(M', U_1', U_2')\}$
9: **until** $U_1 = \emptyset$ or $U_2 = \emptyset$
10: **return** M

Example 5. *Figure 2 illustrates an example of conducting the A* search algorithm for matching $\mathcal{L}_1$ and $\mathcal{L}_2$ in Figure 1. Suppose that all the vertices, edges and p_1 are patterns defined on $\mathcal{L}_1$. According to Algorithm 1, we create a root node tn_0 at the first iteration. To expand the mapping, B is selected from U_1, since B is included by 6 patterns. For each vertex $v \in U_2$, we add a child node of tn_0 which appends $B \to v$ to M. Suppose that tn_4 is currently the tree node with the maximum $g+h$ score. In the second iteration, we visit tn_4 and generate its child nodes tn_9 to tn_{16}. After several times of iteration, it reaches the node tn_x with empty U_1. The mapping M of tn_x is returned as the optimal matching result. There is no need to visit any other un-visited nodes remained in the A* search tree, since their upper bounds of normal distances are no greater than the normal distance of tn_x.*

B. Efficiently Computing the Normal Distance G

Let $\mathcal{P}_{M'}$ be the set of patterns whose events are all defined in M', and $\mathcal{P}_M$ be all the patterns corresponding to M. It is evident that $\mathcal{P}_M \subseteq \mathcal{P}_{M'}$. We denote $\mathcal{P}_{\mathsf{new}} = \mathcal{P}_{M'} \setminus \mathcal{P}_M$ as the set of new introduced matching pattern pairs by appending $a \to b$ to M. According to the definition of $g(M, U_1, U_2)$, i.e., $D^N(M)$, we have

$$g(M', U_1', U_2') = g(M, U_1, U_2) + \sum_{p \in \mathcal{P}_{\mathsf{new}}} \delta(p)$$

according to formula (1).

Therefore, instead of recalculating g for each node, we use a more efficient three-step method to conduct the incremental computation of g of a child node from the g of its parent node. Firstly, we capture the patterns which are newly introduced. Then, for each newly introduced pattern p, we find out whether the corresponding structure w.r.t. M' exists in the other G_2. Finally, we calculate the frequency of all patterns for computing $\delta(p)$.

Capturing newly introduced patterns: As the first step of calculating g, we need to know which patterns are new introduced, i.e., $\mathcal{P}_{\mathsf{new}}$. An inverted index I_p is employed, where each $v \in V$ maps to a list of patterns involving v, denoted as $I_p(v)$. Consequently, the newly introduced patterns can be computed as $\mathcal{P}_{\mathsf{new}} = (\bigcup_{v \in V_1 \setminus U_1'} I_p(v) \setminus \bigcup_{v \in U_1'} I_p(v)) \setminus (\bigcup_{v \in V_1 \setminus U_1} I_p(v) \setminus \bigcup_{v \in U_1} I_p(v))$. That is, all the patterns with vertices from $V_1 \setminus U_1'$, but not those ones that have already been included by the previous M (i.e., not $\mathcal{P}_M$).

Pruning no contribution patterns: For a pattern $p \in \mathcal{P}_{\mathsf{new}}$ from $\mathcal{L}_1$, it is possible that no trace in the other side $\mathcal{L}_2$ matches the corresponding $M(p)$, i.e., $f_2(M(p)) = 0$. This pattern p will have no contribution to the normal distance having $\delta(p) = 0$. Recall that each pattern can be represented as a directed graph. We can identify and prune such patterns, without evaluating frequencies in the event log, by the following pattern existence property in dependency graph.

Proposition 1 (Pattern Existence)**.** *For a pattern p over an event log $\mathcal{L}$ and the dependency graph G of $\mathcal{L}$, if p is not a subgraph of G, we have $f(p) = 0$ w.r.t. $\mathcal{L}$.*

Consequently, for each newly introduced pattern $p \in \mathcal{P}_{\mathsf{new}}$, if $M(p)$ is not a subgraph of G_2, we directly conclude $\delta(p) = 0$ without computing over $\mathcal{L}_2$.

Calculating the frequency of pattern: Finally, to calculate $\delta(p)$, it is indeed to compute $f_1(p)$ and $f_2(M(p))$. Instead of scanning the whole event log to count frequencies, we employ another inverted index I_t, where each event, say $v \in V_1$,

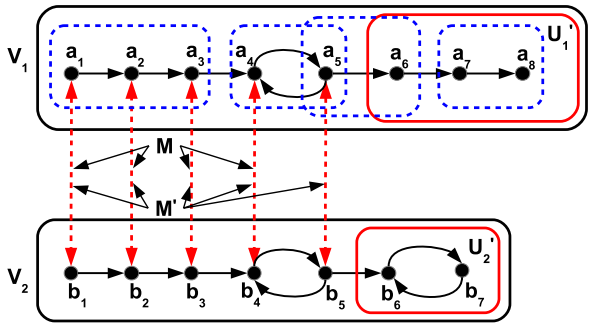


Fig. 3: Expanding a new mapping $a_5 \to b_5$ to M

corresponds to a list of traces $\sigma \in \mathcal{L}_1$ containing v, denoted by $I_t(v)$. Let $V(p)$ denote all the events involved in p. Instead of the whole event log $\mathcal{L}_1$, we only need to scan a part of traces $\bigcap_{v \in V(p)} I_t(v)$ for counting $f_1(p)$. It is similar for computing $f_2(M(p))$.

C. A Simple Upper Bound of H

To compute the upper bound $h(M', U_1', U_2')$ of normal distances that can be further contributed, we consider all the remaining patterns, $\mathcal{P} \setminus \mathcal{P}_{M'}$. In the following, we show that an upper bound $\Delta(p, U_2)$ of $\delta(p)$ can be computed such that the events in p can only be mapped to events in U_2 (see the following Problem 2 for formal definition). Consequently, we have

$$h(M', U_1', U_2') = \sum_{p \in \mathcal{P} \setminus \mathcal{P}_{M'}} \Delta(p, M'(V(p) \setminus U_1') \cup U_2').$$

Note that a simple bound of $\delta(p)$ is 1.0, as each pattern contributes at most 1.0 to the pattern normal distance according to formula (1). It follows $h(M', U_1', U_2') = |\mathcal{P} \setminus \mathcal{P}_{M'}|$.

Example 6. *Figure 3 illustrates an example of expanding a new mapping $a_5 \to b_5$ to a partial matching $M = \{a_1 \to b_1, a_2 \to b_2, a_3 \to b_3, a_4 \to b_4\}$. Then we have $\mathcal{P}_M = \{SEQ(a_1, a_2, a_3)\}$, $\mathcal{P}_{M'} = \{SEQ(a_1, a_2, a_3), AND(a_4, a_5)\}$, thus $\mathcal{P}_{new} = \{AND(a_4, a_5)\}$. By checking the existence of edge b_4b_5 ($M(a_4)M(a_5)$) and b_5b_4 ($M(a_5)M(a_4)$) in G_2, we confirm that $M(AND(a_4, a_5))$ may exist in G_2 and cannot be pruned. At last, we have the remaining patterns $SEQ(a_7, a_8)$ and $SEQ(a_5, a_6)$.*

Algorithm analysis: First, to compute the frequency of a pattern in the event log $\mathcal{L}$, we need to traverse every events of each trace in $\mathcal{L}$. The maximum length of trace is $|V_1|$, the possible total number of events in $\mathcal{L}$ is $|V_1| * |\mathcal{L}|$. That is, the complexity of evaluating a pattern p in the event log is $O(|V_1| * |\mathcal{L}|)$. In the worst case, the optimal mapping will be obtained after the A* search tree is fully expanded to all its leaf nodes (either $U_1 = \emptyset$ or $U_2 = \emptyset$). Let $n = \max(|V_1|, |V_2|)$. As mentioned, there are at most $n!$ possible mappings, i.e., possible leaf nodes. Note that during the expansion of a mapping, any pattern will be evaluated once for computing $\delta(p)$ in g function, while the estimation of h function does not need to evaluating the event log. Therefore, the complexity of the event matching algorithm is $O(|V_1| * |\mathcal{L}| * |\mathcal{P}| * n!)$.

IV. A Tighter Bound on Normal Distance

Rather than simply assigning the largest $\Delta = 1.0$, in this section, we study a tighter (smaller) upper bound of $\delta(p)$ for estimating h. The tighter the upper bound is, the more tree nodes can be pruned during the A* algorithm. We first formalize the problem of finding bound $\Delta(p, U_2)$.

Problem 2 (Upper Bound Problem)**.** *Given a pattern p from $\mathcal{L}_1$ and a set of events $U_2 \subseteq V_2$ of $\mathcal{L}_2$, it is to find an upper bound $\Delta(p, U_2)$ of $\delta(p)$ w.r.t. any mapping $M : V(p) \to U_2$.*

Unfortunately, obtaining a tight bound is highly nontrivial. The most tight bound is $\Delta(p, U_2) = 0$. We identify a trivial case such that a tightest bound $\Delta(p, U_2) = 0$ can be concluded efficiently. That is, if a remaining pattern p is larger than the size of U_2, the upper bound $\Delta(p, U_2)$ is 0. For instance, in Figure 3, we have $\Delta(\text{SEQ}(a_6, a_7, a_8), \{b_6, b_7\}) = 0$.

To obtain tight bounds for general patterns, it is unlikely to capture all the possible subgraphs isomorphic to the pattern. Instead, we compute the bounds by utilizing vertex and edge frequencies as follows to avoid subgraph isomorphism.

A. Upper Bound via Vertex Frequency

We first investigate the relationships between the bound of normal distances and the bound of pattern frequencies. Let f_2^U denote an upper bound of $f_2(M(p))$, for all the possible mapping $M : V(p) \to U_2$.

Lemma 2. *If $f_2^U \le f_1(p)$, we have $\Delta(p, U_2) = 1 - \frac{f_1(p) - f_2^U}{f_1(p) + f_2^U}$.*

Proof. According to $f_1(p) \ge f_2^U \ge f(M(p))$ for all possible M, we have $|f_1(p) - f_2^U| = f_1(p) - f_2^U$ and $|f_1(p) - f_2(M(p))| = f_1(p) - f_2(M(p))$. It follows $\Delta(p, U_2) = 1 - \frac{f_1(p) - f_2^U}{f_1(p) + f_2^U} \ge 1 - \frac{f_1(p) - f_2(M(p))}{f_1(p) + f_2(M(p))} = \delta(p)$. That is, $\Delta(p, U_2) = 1 - \frac{f_1(p) - f_2^U}{f_1(p) + f_2^U}$ is an upper bound of $\delta(p)$. □

Obviously, the smaller the bound f_2^U is, the tighter $\Delta(p, U_2)$ would be. A straightforward method is to capture every $M(p)$ w.r.t. all possible mappings M, calculate $f_2(M(p))$ for each $M(p)$ and use the highest $f_2(M(p))$ as f_2^U. It would be prohibitively expensive, as enumerating all subgraphs isomorphic to p is already unlikely to afford. We attempt to find a more efficient way to estimate f_2^U.

A natural intuition is to consider the frequencies of a pattern's substructures. Since vertices are the most primitive substructures in a pattern, we start from the frequency relationship between a pattern and its vertices.

Proposition 3. *For a pattern p over an event log $\mathcal{L}$ with event set V, it always has $f(p) \le f(v), \forall v \in V(p)$.*

Proof. Consider any event $v \in V(p)$ in the pattern p. For each trace matching the pattern, event v must appear at least once in this trace. It follows that $f(v)$ is no less than $f(p)$. □

That is, $f_2(M(p))$ should be no greater than the frequency of any event involved in $M(p)$.

Let $f_n = \max_{v \in U_2} f_2(v)$ be the highest vertex frequency among U_2. We have $f_2(M(p)) \le f_n$, i.e., an upper bound f_2^U

of $f_2(M(p))$ for possible $M : V(p) \to U_2$. [1]

According to Lemma 2, if $f_n \leq f_1(p)$, we can obtain a tighter upper bound

$$\Delta(p, U_2) = 1 - \frac{f_1(p) - f_n}{f_1(p) + f_n}.$$

Example 7. *Suppose that we need to compute $\Delta(AND(B,C), \{7,8\})$ in Figure 1. According to $\mathcal{L}_1$, we have $f(AND(B,C)) = 1.0$, and the highest vertex frequency f_n in $\{7,8\}$ is 0.9. Thus we have $\Delta(AND(B,C), \{7,8\}) = 1 - \frac{1.0-0.9}{1.0+0.9} = 0.95$. That is, the pattern $AND(B,C)$ can contribute at most 0.95 of normal distance by possible mappings from $\{B,C\}$ to $\{7,8\}$.*

B. Upper Bound via Edge Frequency

Next, besides vertices, more complex substructures, e.g., edges or paths, can be exploited for further tightening the upper bound $\Delta(p, U_2)$. We first consider two simple types of SEQ and AND patterns without other patterns nested. Then, the techniques are extended to general patterns.

Upper bound for simple SEQ patterns: We start with a special case $\text{SEQ}(v_1, \ldots, v_k)$, where all sub-patterns $v_1, \ldots, v_k$ are individual events, named simple SEQ pattern. Dealing with this special case is meaningful, since we will show below that AND patterns and general SEQ patterns with AND patterns nested as sub-patterns can be split into a set of simple SEQ patterns w.r.t. frequencies.

We consider the frequency of the smallest subsequence which only contains two events, i.e., the frequency of edges in dependency graph.

Proposition 4. *For a simple pattern $p = SEQ(v_1, v_2, ..., v_k)$ over an event log $\mathcal{L}$, it always has $f(p) \leq f((v_i, v_{i+1})), i \in 1, \ldots, k-1$.*

Proof. Consider any consecutive events (v_i, v_{i+1}) in the pattern $p = \text{SEQ}(v_1, v_2, ..., v_k)$. For each trace matching the pattern, (v_i, v_{i+1}) must appear at least once in this trace, and not vice versa. Therefore, $f((v_i, v_{i+1}))$ is no less than $f(p)$. □

That is, the frequency of a pattern p is always no greater than that of any edge w.r.t. two consecutive events (v_i, v_{i+1}) in the pattern.

Let $f_e = \max_{v,u \in U_2} f_2((v,u))$ be the highest edge frequency in a subgraph of G_2 with vertices from U_2 only. According to Proposition 4, we have $f_2(M(p)) \leq f_e$, i.e., an upper bound f_2^U of $f_2(M(p))$. [2]

Consequently, for a simple SEQ pattern $p = \text{SEQ}(v_1, v_2, ..., v_n)$, if $f_n \leq f_1(p)$, according to Lemma 2, we have a tighter upper bound

$$\Delta(p, U_2) = 1 - \frac{f_1(p) - f_e}{f_1(p) + f_e}.$$

[1] Indeed, it is also sufficient to use $|p|$-th highest vertex frequency in U_2 as f_n, since $f_2(M(p))$ should be no greater than the lowest frequency among $|p|$ events involved in $M(p)$.

[2] Again, it is indeed sufficient to use $(|p|-1)$-th highest edge frequency as f_e, since $f_2(M(p))$ should be no greater than the lowest frequency of any edge of $|p|-1$ edges in $M(p)$.

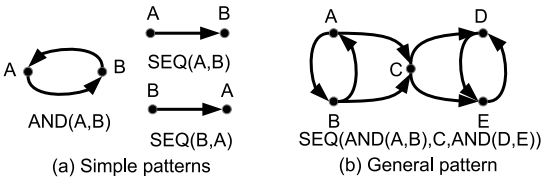


Fig. 4: Simple and general patterns

Example 8. *Suppose that we need to compute $\Delta(SEQ(B,D), \{7,8\})$ in Figure 1. According to $\mathcal{L}_1$, we have $f(SEQ(B,D)) = 0.8$, and the highest edge frequency f_e in the subgraph of G_2 w.r.t. $\{7,8\}$ is 0.6. Thus we have $\Delta(SEQ(B,D), \{7,8\}) = 1 - \frac{0.8-0.6}{0.8+0.6} = 0.86$.*

Upper bound for simple AND patterns: To study the bound, we first give a split of AND patterns into a set of SEQ patterns w.r.t. frequencies.

Lemma 5. *Given any pattern $p = AND(p_1, p_2, ..., p_k)$, we have $f(p) = \sum_{(p'_1, p'_2, ..., p'_k) \in \mathcal{Q}} f(SEQ(p'_1, p'_2, ..., p'_k))$, where $\mathcal{Q}$ is a set containing all distinct permutations of $p_1, p_2, ..., p_k$.*

Proof. Since the order of sub-patterns does not matter for the AND pattern, there will be $k!$ possible orders of the sub-patterns which correspond to all the possible permutations of $p_1, p_2, ..., p_k$. As illustrated in Figure 4(a), for instance, $\text{AND}(A,B)$ can be split into $\text{SEQ}(A,B)$ and $\text{SEQ}(B,A)$.

Referring to the definition of AND pattern, any trace of events matching with p should follow exactly one of the orders in $\mathcal{Q}$ (since an event appear at most once in a pattern or trace). It follows the conclusion of frequency summation. □

Now, we consider a simple AND pattern $\text{AND}(v_1, ..., v_k)$ where $v_1, \ldots, v_k$ are individual events.

Proposition 6. *For a simple pattern $p = AND(v_1, v_2, ..., v_k)$ over $\mathcal{L}_1$, we always have $f_2(M(p)) \leq k! f_e$, for any mapping $M : V(p) \to U_2$.*

Proof. According to Lemma 5, a simple AND pattern with k events can be split into at most $k!$ simple SEQ patterns. As mentioned in Lemma 4, the frequency of a SEQ pattern is bounded by the highest edge frequency f_e. So the frequency of a simple AND pattern with k events is bounded by $k! f_e$. □

In other words, for a simple AND pattern p containing k events, $k! f_e$ serves as an upper bound f_2^U of $f_2(M(p))$. Referring to Lemma 2, if $k! f_e \leq f_1(p)$, the upper bound is

$$\Delta(p, U_2) = 1 - \frac{f_1(p) - k! f_e}{f_1(p) + k! f_e}.$$

Example 9. *Suppose that we need to compute $\Delta(AND(B,C), \{1,3\})$ in Figure 1. According to $\mathcal{L}_1$, we have $f(AND(B,C)) = 1.0$, and the highest edge frequency f_e in the subgraph of G_2 w.r.t. $\{1,3\}$ is 0.2. Thus we have $\Delta(AND(B,C), \{1,3\}) = 1 - \frac{1.0-2!\times 0.2}{1.0+2!\times 0.2} = 0.57$.*

Upper bound for general patterns: Finally, we consider a general pattern p. Following the same intuition of dealing with simple AND patterns, we first study the number of simple SEQ patterns can be split from a pattern p, denoted by $\omega(p)$. The number $\omega(p)$ can be calculated by a recursive way.

TABLE II: Summary of Tighter Upper Bounds

Case of patterns p	Upper Bound
a general pattern	$1 - \frac{f_1(p)-f_n}{f_1(p)+f_n}$
a simple pattern SEQ$(v_1, \ldots, v_k)$	$1 - \frac{f_1(p)-f_e}{f_1(p)+f_e}$
a simple pattern AND$(v_1, ..., v_k)$	$1 - \frac{f_1(p)-k!f_e}{f_1(p)+k!f_e}$
a general pattern	$1 - \frac{f_1(p)-\omega(p)f_e}{f_1(p)+\omega(p)f_e}$

Lemma 7. *Given a general pattern p, we have:*

$$\omega(p) = \begin{cases} 1 & \text{if } p \text{ is a single event} \\ \prod_{i\in1,\ldots,k} \omega(p_i) & \text{if } p \text{ is SEQ}(p_1, ..., p_k) \\ k! \prod_{i\in1,\ldots,k} \omega(p_i) & \text{if } p \text{ is AND}(p_1, ..., p_k) \end{cases}$$

Proof. $\omega(p) = 1$ is evident for a single event. If p is a SEQ$(p_1, ..., p_k)$ pattern, $\omega(p)$ is the combination of $\omega(p_1), ..., \omega(p_k)$, which is $\prod_{i\in1,\ldots,k} \omega(p_i)$. If p is an AND$(p_1, ..., p_k)$ pattern, we have $\omega(p) = k! \prod_{i\in1,\ldots,k} \omega(p_i)$ referring to Lemma 5. □

For instance, we consider the pattern SEQ(AND(A, B), C, AND(D, E)) illustrated in Figure 4(b). Recall that AND patterns, e.g., AND(A, B), can be split into SEQ(A, B) and SEQ(B, A). Therefore, the pattern SEQ(AND(A, B), C, AND(D, E)) corresponds to four possible orders SEQ(A, B, C, D, E), SEQ(B, A, C, D, E), SEQ(A, B, C, E, D) and SEQ(B, A, C, E, D). That is, we have ω(SEQ(AND(A, B),C,AND(D, E))) $= \omega(\text{AND}(A, B)) \times \omega(C) \times \omega(\text{AND}(D, E)) = 2! \times 1 \times 2! = 4$.

Proposition 8. *Given any pattern p over $\mathcal{L}_1$, we always have $f_2(M(p)) \leq \omega(p)f_e$, for any mapping $M : V(p) \to U_2$.*

Proof. According to Lemma 7, a general pattern p can be split into at most $\omega(p)$ simple SEQ patterns. As mentioned in Lemma 4, the frequency of a simple SEQ pattern is bounded by the highest edge frequency f_e. Therefore, the frequency of p is bounded by $\omega(p)f_e$. □

Again, according to Lemma 2, if $\omega(p)f_e$ as the upper bound f_2^U of $f_2(M(p))$ has $\omega(p)f_e \leq f_1(p)$, we have an upper bound

$$\Delta(p, U_2) = 1 - \frac{f_1(p) - \omega(p)f_e}{f_1(p) + \omega(p)f_e}.$$

Example 10. *Suppose that we need to compute $\Delta(p_1, \{1, 3, 6, 7\})$ in Figure 1. By evaluating p_1 in $\mathcal{L}_1$, we have $f(p_1) = 1.0$. The highest edge frequency f_e in the subgraph of G_2 w.r.t. $\{1, 3, 6, 7\}$ is 0.3, and $\omega(p_1) = 2$ owing to Lemma 7. Thus we have $\Delta(p_1, \{1, 3, 6, 7\}) = 1 - \frac{1.0-2\times0.3}{1.0+2\times0.3} = 0.75$.*

C. Put the Upper Bounds Together

Thus far, we have presented the rationale of computing the upper bound $\Delta(p, U_2)$ for various patterns p by either using vertex frequencies or edge frequencies (as summarized in Table II). Obviously, the smaller one will be returned as a tighter upper bound.

Algorithm 2 illustrates an overview for computing the upper bound $\Delta(p, U_2)$. First, Line 1 to Line 2 examines if the upper bound is 0 as stated in the beginning of Section IV. If not, the algorithm tries to estimate the upper bound via vertex frequency as stated in Section IV-A, or via edge frequency as stated in Section IV-B, from Line 3 to Line 10. When none of these estimation methods is applicable, we just return 1.0, i.e., the simple upper bound in Line 12.

Algorithm 2 UPPERBOUND(p, U_2)

Input: p is a pattern in $\mathcal{L}_1$ and U_2 is the set of events in $\mathcal{L}_2$.
Output: The upper bound $\Delta(p, U_2)$ of $\delta(p)$ w.r.t. possible mappings $M : V(p) \to U_2$.

```
1:  if |V(p)| > |U2| then
2:      return 0
3:  fn := the highest frequency of events (vertices) in U2
4:  fe := the highest frequency of edges in the subgraph of G2 with vertices of U2
5:  if fn ≥ ω(p)fe then
6:      fmin := ω(p)fe
7:  else
8:      fmin := fn
9:  if fmin ≤ f(p) then
10:     return 1 − (f(p)−fmin)/(f(p)+fmin)
11: else
12:     return 1.0
```

V. PAY-AS-YOU-GO MATCHING

In real event data management, interesting event patterns are often gradually identified by managers or business analysis in a pay-as-you-go style. When new pattern comes, the previously found optimal matching of events may change. Instead of re-computing the optimal matching from scratch, event matching approaches are expected to support incremental computation, as other pay-as-you-go data integration systems [20].

Suppose that a new pattern p is added to $\mathcal{L}_1$. We first discuss special cases that the previously found matching is still the optimal after p comes. Then, two strategies are presented to improve the efficiency of recomputing the optimal matching.

A. The Previous Event Matching is Still The Optimal

In some case, introducing a new pattern p may not affect the optimal matching. According to Definition 5, a new pattern can contribute at most 1.0 to the pattern normal distance.

Proposition 9. *Let M_{old} be the previous optimal event mapping. Given a new pattern p', if $\delta(p') = 1.0$ w.r.t. M_{old}, the previous optimal mapping is still the optimal mapping under the new pattern set $\mathcal{P} \cup \{p'\}$.*

Proof. Let $D'^N(M_{\mathsf{old}})$ be the normal distance of M_{old} over the new pattern set $\mathcal{P} \cup \{p'\}$, having $D'^N(M_{\mathsf{old}}) = D^N(M_{\mathsf{old}}) + \delta(p') = D^N(M_{\mathsf{old}}) + 1.0$. For any other mapping M, according to the previous optimal mapping, we have $D^N(M) \leq D^N(M_{\mathsf{old}})$. Moreover, we have $\delta(p') \leq 1.0$ for any mapping M. It concludes that $D'^N(M_{\mathsf{old}})$ is still the largest among all possible mappings. □

Otherwise, for the case of $\delta(p') < 1.0$ w.r.t. M_{old}, we update the normal distance of the previous optimal matching M_{old}

by adding $\delta(p')$, i.e., $D'^N(M_{\mathsf{old}}) = D^N(M_{\mathsf{old}}) + \delta(p')$. It is compared with the highest upper bound $(g+h)$ among all the intermediate results remaining in queue Q in Algorithm 1, i.e., the upper bound of the second largest normal distance.

Proposition 10. *If $D^N(M_{old}) + \delta(p') \geq g + h + 1.0$, M_{old} is still the optimal mapping after introducing p.*

Proof. As mentioned before, the new pattern p' can contribute at most 1.0 to the pattern normal distance. If $D^N(M_{\mathsf{old}}) + \delta(p') \geq g + h + 1.0$, there could not be any matching result derived from intermediate results in queue Q with normal distance higher than $D^N(M_{\mathsf{old}}) + \delta(p')$. □

Indeed, since $D^N(M_{\mathsf{old}})$ is the previous optimal mapping, we have $D^N(M_{\mathsf{old}}) \geq g + h$. Therefore, Proposition 9 can be derived by Proposition 10 as a special case, where $D'^N(M_{\mathsf{old}}) = D^N(M_{\mathsf{old}}) + 1.0$.

B. Incremental Computation

When Proposition 10 is not applicable, we cannot guarantee that the previous optimal event matching is still the optimal. Nevertheless, the previous optimal mapping M_{old} and the queue Q containing all partial mapping results provide important information that can be utilized to reduce the recalculation of the new optimal mapping.

In the following, we propose two incremental computing strategies for different scenarios. The former approach utilizes the queue Q in Algorithm 1 and the latter approach utilizes only the previous optimal mapping M_{old}.

Continuing A* algorithm from Q: When a new pattern p comes, we do not have to recompute the whole A* search tree. As illustrated in Figure 2, any possible mapping (including the new optimal mapping) can be obtained by expanding the tree nodes in Q (i.e., $\{tn_1, tn_8, tn_9, tn_{16}\}$). Recomputing normal distances for processed nodes (that have been expanded, e.g., tn_0, tn_4 and tn_{13}) is not necessary. We can update *g* and *h* for the nodes in Q, and then continue the search in A* algorithm. It is notable that we do not have to recompute both *g* and *h* for each node (M, U_1, U_2). If all the events of the new pattern p' are already included in M, we only need to update the *g* value. Otherwise, we just recompute the *h* value for p'.

This strategy favors certain scenarios. It is not surprising that if the new optimal mapping M_{new} has many common corresponding event pairs as M_{old}, this approach may save time. On the contrary, if M_{new} deviates far from M_{old}, it is unworthy to recompute *g* and *h* and expand the nodes in Q. Hence, we develop another matching strategy for the latter scenario.

Restart A* algorithm with pruning by the previous optimal matching: In this method, we conduct the A* algorithm again from the root node. In particular, we utilize the updated normal distance $D'^N(M_{\mathsf{old}}) = D^N(M_{\mathsf{old}}) + \delta(p')$ of the previous optimal matching for pruning the nodes in A* search tree. When a new node is expanded, we compare *g*+*h* with $D'^N(M_{\mathsf{old}})$. If $g + h \leq D'^N(M_{\mathsf{old}})$, this node can be safely pruned (with no further expanding) since it can never yield results better than the previous optimal matching.

Unfortunately, we do not know ahead on whether M_{new} is similar with or deviates far from M_{old} before conducting the A* algorithm. We evaluate both strategies on real datasets in experiments.

VI. On Top-K Matching Results

While precisely matching heterogeneous events is not always possible by an uninterpreted approach, instead of returning only one optimal matching result, we produce a list of event mappings with the top-k maximum normal distances. Such top-k matching is naturally supported in the A* search algorithm, by altering Line 9 in Algorithm 1 to return a list of k matching results rather than only one matching result. A result is added to the list when either $U_1 = \emptyset$ or $U_2 = \emptyset$.

Top-k meets pay-as-you-go: Our pay-as-you-go matching approaches can be easily adapted to support top-k matching as well. Let us first reconsider Proposition 10. Given a new pattern p', we can renew the normal distances for the previous top-k optimal matchings w.r.t. p', and compare each updated normal distance with the highest $g + h$ among all the intermediate results remaining in Q.

Suppose that the updated normal distances of the previous top-k matching are $D'^N(M^1_{\mathsf{old}}), D'^N(M^2_{\mathsf{old}}), ..., D'^N(M^k_{\mathsf{old}})$ in the descending order. According to Proposition 10, if we have $D'^N(M^i_{\mathsf{old}}) \geq g+h+1.0$, for some $i \in 1, \ldots, k$, the mappings $M^1_{\mathsf{old}}, M^2_{\mathsf{old}}, ..., M^i_{\mathsf{old}}$ are still the top-i optimal matching after introducing the new pattern p. Consequently, we only need to find the $(i+1)^{th}, (i+2)^{th}, ..., k^{th}$ optimal matching instead of recalculating all the top-k matching.

Otherwise, for the general case, we review two pay-as-you-go matching strategies proposed in Section V-B. The former approach continuing from Q is directly supported as top-k A* search. We adapt the later strategy to top-k matching by replacing the pruning bound $D'^N(M_{\mathsf{old}})$ with $\min_{j \in 1,...,k} D'^N(M^j_{\mathsf{old}})$, i.e., the minimum updated top-k score that we know so far from the previous optimal top-k matching.

VII. Experiment

In this section, we report the experimental evaluations by comparing our proposed method with the state-of-the-art schema matching approaches [13], [17]. The programs are implemented in Java and all the experiments were performed on a computer with Intel(R) Core(TM) i7-2600 3.40GHz CPU and 8 GB memory.

Data set: We employ a real data set from a bus manufacturer. There are 38 event logs extracted from the ERP systems of two departments located at distinct industrial parks, respectively. The numbers of distinct events (analogous to schema size in schema matching) in these logs ranges from 2 to 11, and the number of traces (instance size of the number of tuples) ranges from 5 to 3000. For each different event set size, two corresponding event logs of distinct departments are used. The patterns are manually assigned by the guidelines discussed in Section II.

Moreover, in order to evaluate the scalability of our approach and the performance on another system, we generate

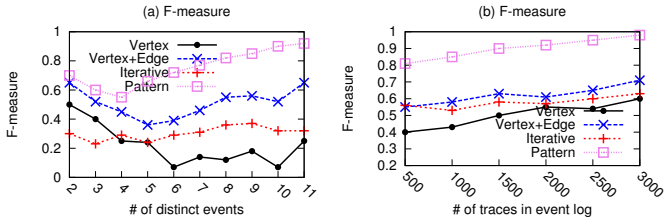


Fig. 5: Effectiveness of matching real data

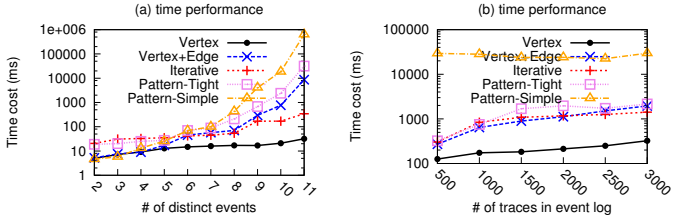


Fig. 6: Efficiency of matching real data

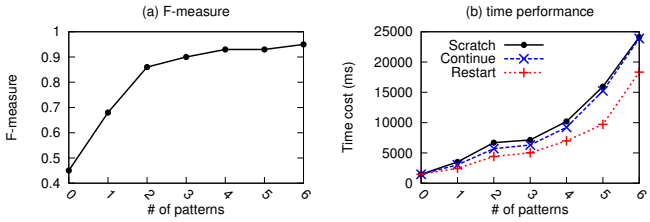


Fig. 7: Performance on pay-as-you-go matching

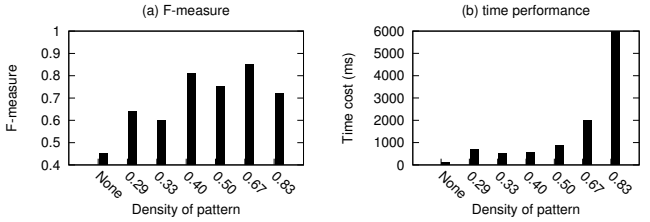


Fig. 8: Effectiveness and efficiency w.r.t. various pattern densities

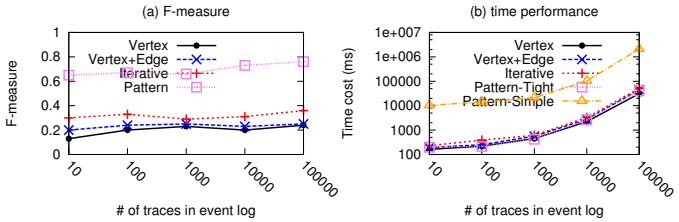


Fig. 9: Scalability on synthetic data

a synthetic data set by employing process specifications (in corresponding pairs) from a boiler manufacturer. Event logs are generated based on these process specifications with trace numbers up to 100000, by using the log generator of BeehiveZ [12]. Each trace is generated as a topological sort on the process specification graph. The generator randomly chooses a branch when multiple alternatives exist.

Criteria: Besides time performance, we also evaluate the effectiveness of our event matching approach, by using F-measure of precision and recall, which is widely used in text retrieval community. Let truth be the ground truth of event mapping discovered manually, and found be the event corresponding relation found by our method, we have $precision = \frac{|\text{found} \cap \text{truth}|}{|\text{found}|}$, $recall = \frac{|\text{found} \cap \text{truth}|}{|\text{truth}|}$ and $F\text{-}Measure = 2 \cdot \frac{precision \cdot recall}{precision + recall}$.

Exp. on effectiveness: We compare the accuracy of the proposed approach (Pattern) with existing approaches [13] (Vertex, Vertex+Edge) and [17] (Iterative). In particular, [13] employs the normal distance by considering Vertex or Vertex+Edge similarities. Instead of enumerating possible mappings and ranking the corresponding normal distances, [17] computes the vertex similarity in page-rank like iterative way.

Figure 5 reports the F-measure of all approaches by varying event set sizes and trace numbers. As shown in the figure, our pattern based approach outperforms others with the highest accuracies. Note that the accuracy drops from sizes 2 to 4 of event sets in Figure 5 (a). The rationale is that not many patterns can be employed in such a small number of events. Moreover, one error will significantly debase the F-measure when the event set size is small. The accuracy increases along with the increase of trace number, since more distinct events or patterns become more discriminative in a larger trace number.

Exp. on efficiency: We compare the time performance of all approaches in Figure 6. To evaluate the pruning power of upper bounds of h, we test the proposed approaches with simple bound (Pattern-Simple) in Section III-C or tight bound (Pattern-Tight) in Section IV. Figures 6 (a) and (b) report time costs of all approaches by varying event set sizes and trace numbers. The time costs of approaches increases fast when the event set size is large. It is not surprising owing to the large number of possible mappings, i.e., factorial of the event set size, as mentioned at the beginning of Section III. The time cost also rises along with the growth of trace number since it needs more time to compute the frequencies in traces.

To evaluate the pruning power of tight upper bounds, we observe the number of processed tree nodes in Figures 6 (c) and (d). As shown in Figure 6 (c), the approach using tight bounding function expands less tree nodes during the A* search, especially in large event set sizes. Consequently, as discussed in Figure 6 (a), the approach with tight bound shows at most 2 orders of magnitudes improvement in time costs. Figure 6 (d) shows the growth of trace number does not affect the pruning power much.

Exp. on pay-as-you-go matching: In this experiment, we gradually add a number of patterns from 0 to 6, and observe both the accuracy and time performance in Figure 7. As illustrated in Figure 7 (a), compared with no pattern, there is a significant improvement of accuracy after considering 1-2

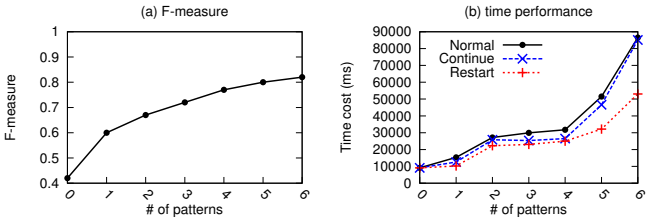


Fig. 10: Pay-as-you-go on synthetic data

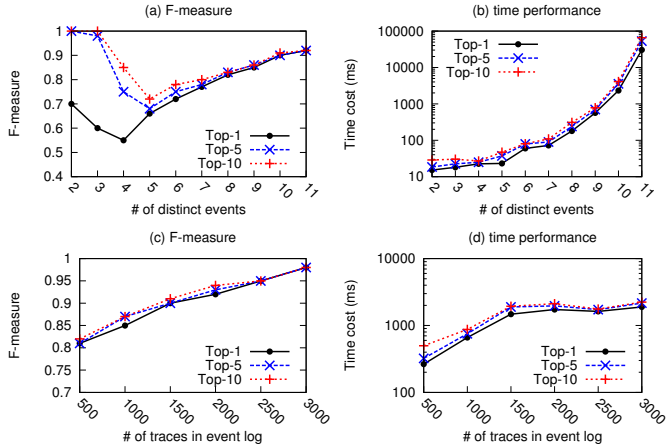


Fig. 11: Evaluation of top-k matching

patterns. Since the accuracy has already been high (larger than 0.8), the improvement of effectiveness by further introducing patterns is not as significant as the previous ones.

Figure 7 (b) reports the time cost of three approaches, where Scratch stands for recomputing the optimal matching from scratch when new patterns come, Continue (form Q) and Restart (with pruning) denote two pay-as-you-go strategies proposed in Section V. Compared with the Scratch method, both the pay-as-you-go approaches show less time to obtain the new optimal matching. In particular, the Restart strategy performs better than the Continue approach, since the new optimal matching results often vary from the previous one (as discussed in Section V).

Figure 8 studies how different complex patterns in various density will affect the accuracy and the time cost of matching. Let n_v, n_e be the numbers of vertices and edges of a pattern p, respectively. The density of p is $n_e/n_v(n_v-1)$, i.e., the number of edges in pattern divides the number of edges in complete graph with the same vertices in p. Intuitively, the higher the density is, the more complex a pattern would probably be. According to Figure 8 (a), patterns with higher density perform generally better in accuracy. As discussed at the end of Section II, a more complex pattern is probably more discriminative than a simple pattern (e.g. a path). The results verify our discussion on choosing discriminative patterns. The complex patterns also need more time to match with traces in event log, according to the result of time performance reported in Figure 8 (b).

Exp. on synthetic data set: Figures 9 and 10 reports the experimental results on the larger synthetic data set as introduced at the beginning of this section. In general, similar results as in the real data set are observed. First, in Figure 9, the proposed method still outperforms other methods in accuracy on this data set. The approach with tight bound again has a considerable pruning power compared with the approach with simple bound especially when the trace number is extremely large. The accuracy increases along with the growth of pattern amount in the pay-as-you-go experiment reported in Figure 10 (a) Finally, the Restart with pruning strategy has the best time performance in Figure 10 (b), which is similar with the result reported in Figure 7.

Exp. on top-k extension: Finally, we report the experimental results on top-k matching. Figures 11 (a) and (c) present the highest F-measure of k matching results. As illustrated, the top-5 and top-10 matching can improve the accuracy compared with top-1 matching, especially on small event set sizes. However, when the event number is large, the top-1 result already shows good performance and top-5 or 10 result can hardly improve the accuracy further. It is also observed that top-5 and top-10 matching need more time to obtain the top-k best matchings in Figures 11 (b) and (d). To sum up, it is suggested to consider top-k matching on small event set sizes, where the accuracy can be significantly improved without paying much more overhead.

VIII. Related Work

The quality of event data has recently been highlighted. Rather than addressing inconsistencies existing in event logs [22], in this study, we focus on matching the heterogeneous events collected from different sources.

Structure based matching: Owing to opaque event names, the event matching problem relies on the structure based uninterpreted matching methods.

Madhavan et al. [15] proposed an approach for matching XML schema tree based on label similarities among XML nodes and attributes, which cannot be applied in uninterpreted matching with opaque event names. Jeh et al. [11] proposed an approach named Simrank which calculates the similarity of graph vertices. However, it can only be applied on vertices within one graph.

Nejati et al. [17] proposed a method by calculating the vertex similarity between two graphs through iterative computations. According to the experimental results in Section VII, the iterative method [17] shows lower matching accuracy than our proposed pattern based matching. Kang et al. [13] proposed an uninterpreted approach for matching database schema (attributes) by using dependency graphs with frequency information on vertices and edges. As analyzed as well as experimental evaluated, these information are not discriminative enough in matching event data. Xin Dong et al. [7] proposed a graph-based approach which exploits the similarities among attributes of tuples to identify those data instances that represent the same real-world entity. However, such graph is hard to apply on event data since the attributes among events(names, operators) have very low similarities due to the heterogeneity.

Capturing structure information among events: Graph is often employed to represent the structural information among events. While vertices usually denote events, the edges in the graph are associated with various semantics exploited from event logs in different perspectives. Agrawal et al. [1], Cook et al. [5], and Ferreira et al. [8] use a graphical form of Markov transition matrix whose edges are weighted by the conditional probability of one event directly followed by another. However, the conditional probability cannot tell the significance of the edge. Weijters et al. [23] proposed another edge-weighted graph, namely D/F-graph, which uses several heuristic rules to calculate the "causality" between two events, which is not discriminative enough (many edges have the same causality), and also has a high computational complexity. In this paper, we employ the dependency graph proposed in [13] by weighting vertices and edges with normalized frequencies, since it distinguishes the significance of distinct edges, and is easy to interpret.

Discovering event pattern: The discovery of complex event pattern has been studied in complex event processing (CEP) [14]. Agrawal et al. [2] and Mannila et al. [16] study the problem of discovering frequent event patterns, i.e., the frequency of a subsequence is higher than a support degree. The discovery algorithm is starting with simple subpatterns and incrementally build larger pattern candidates. Bettini et al. [3] improve the efficiency of the discovery algorithm, and has ability to discover more complex patterns. As mentioned, the discovery or design of patterns is not the focus of this study. Instead, we directly utilized the given/discovered patterns. Nevertheless, heuristics are discussed in Section II on choosing discriminative patterns for matching.

IX. Conclusions

In this paper, we study the problem of matching heterogeneous events. Owing to opaque event names, we consider the structure based uninterpreted matching of events. Besides individual events and dependency relationship between events, complex event patterns are introduced as discriminative feature in matching. To support efficient pruning, we propose an A* search like framework for computing the optimal matching. Two indices are developed for accelerating the computation of normal distance. Furthermore, we devise a tight bounding function which can prune more non-optimal mappings as early as possible. Since the event patterns are often gradually explored, we adapt our algorithm to support matching in a pay-as-you-go style, i.e., to incrementally and efficiently update the best matching after the arrival of a new pattern. Finally, we extend the approaches to answer top-k best matching. Experimental results demonstrate that our proposed approach shows significantly higher accuracy than the state-of-the-art structure based matching approaches. Moreover, the advanced bounding function and the pay-as-you-go matching significantly reduce the time costs.

As future work, it is interesting to consider the more complicated $1 : n$ matching for the events with different granularity in distinct processes. Moreover, we may exploit other attributes of events besides the sequential dependencies for matching.

Acknowledgment: This work is supported in part by China NSFC under Grants 61073005, 61325008, 61202008 and 61370055; China 973 Program under Grant 2012-CB316200; US NSF through grants CNS-1115234, DBI-0960443, and OISE-1129076; and US Department of Army through grant W911NF-12-1-0066.

References

[1] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *EDBT*, pages 469–483, 1998.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In P. S. Yu and A. L. P. Chen, editors, *ICDE*, pages 3–14. IEEE Computer Society, 1995.

[3] C. Bettini, X. S. Wang, S. Jajodia, and J.-L. Lin. Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Trans. Knowl. Data Eng.*, 10(2):222–237, 1998.

[4] O. Biton, S. C. Boulakia, S. B. Davidson, and C. S. Hara. Querying and managing provenance through user views in scientific workflows. In *ICDE*, pages 1072–1081, 2008.

[5] J. E. Cook and A. L. Wolf. Event-base detection of concurrency. In *SIGSOFT FSE*, pages 35–45, 1998.

[6] L. Ding, S. Chen, E. A. Rundensteiner, J. Tatemura, W.-P. Hsiung, and K. S. Candan. Runtime semantic query optimization for event stream processing. In *ICDE*, pages 676–685, 2008.

[7] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD Conference*, pages 85–96, 2005.

[8] D. R. Ferreira, D. Gillblad, and D. Gillblad. Discovering process models from unlabelled event logs. In *BPM*, pages 143–158, 2009.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[10] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an rdbms for web data integration. In *WWW*, pages 90–101, 2003.

[11] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *KDD*, pages 538–543, 2002.

[12] T. Jin, J. Wang, and L. Wen. Efficiently querying business process models with beehivez. In *BPM (Demos)*, 2011.

[13] J. Kang and J. F. Naughton. On schema matching with opaque column names and data values. In *SIGMOD Conference*, pages 205–216, 2003.

[14] D. Luckham. The power of events: An introduction to complex event processing in distributed enterprise systems. In *RuleML*, page 3, 2008.

[15] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB*, pages 49–58, 2001.

[16] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In U. M. Fayyad and R. Uthurusamy, editors, *KDD*, pages 210–215. AAAI Press, 1995.

[17] S. Nejati, M. Sabetzadeh, M. Chechik, S. M. Easterbrook, and P. Zave. Matching and merging of statecharts specifications. In *ICSE*, pages 54–64, 2007.

[18] T. Pedersen, S. Patwardhan, and J. Michelizzi. Wordnet: : Similarity - measuring the relatedness of concepts. In *AAAI*, pages 1024–1025, 2004.

[19] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[20] A. D. Sarma, X. Dong, and A. Y. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD Conference*, pages 861–874, 2008.

[21] P. Sun, Z. Liu, S. B. Davidson, and Y. Chen. Detecting and resolving unsound workflow views for correct provenance analysis. In *SIGMOD Conference*, pages 549–562, 2009.

[22] J. Wang, S. Song, X. Zhu, and X. Lin. Efficient recovery of missing events. *PVLDB*, 2013.

[23] A. Weijters and W. van der Aalst. Process mining discovering workflow models from event-based data. In *ECAI Workshop on Knowledge Discovery and Spatial Data*, pages 283–290, 2001.

[24] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD Conference*, pages 407–418, 2006.