Matching Heterogeneous Events with Patterns

Shaoxu Song, Yu Gao, Chaokun Wang, Xiaochen Zhu, Jianmin Wang, and Philip S. Yu

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. Recognizing the NP-hardness of the optimal event matching problem with patterns, we propose efficient heuristic. Finally, extensive experiments demonstrate the effectiveness of our pattern based matching compared with approaches adapted from existing techniques, and the efficiency improved by the bounding, pruning and heuristic methods.

Index Terms—Schema matching, event matching

1 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 [1]) in multiple sectors, identifying similar complex procedures (complex event processing [2]) in different branches, or obtaining a global picture of business processes (workflow views [3]) 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 [4] 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 [5]) or linguistic similarity (using dictionary of ontology like WordNet [6]) of event names are often unlikely to perform (see examples below).

To solve the matching problem with "opaque" names, graph based matching approaches [7] 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 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. Fig. 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., in Fig. 1a, 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)* or *DL(7)* in Trace 2 in $\mathcal{L}_2$).

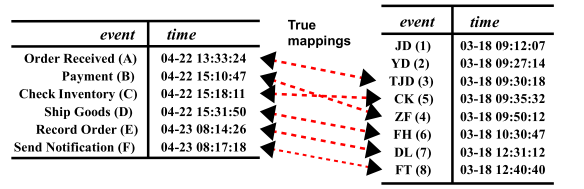
As shown between Figs. 1a and 1b, events in $\mathcal{L}_1$ and $\mathcal{L}_2$ have opaque names. According to our manual investigation, Ship Goods in $\mathcal{L}_1$ is found corresponding to an event namely FH (which is an abbreviation of Chinese phonetic representation) in $\mathcal{L}_2$. Such a 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 $\mathcal{L}_2$.

Figs. 1e and 1f 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 Fig. 1e for instance) indicates that they appear consecutively in at least one trace (e.g., Trace 1 in Fig. 1c). 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 percent traces of the event log.

Frequencies of individual events are obviously not discriminative for matching, e.g., events A,B,...,1,2,...share

- *S. Song, Y. Gao, C. Wang, X. Zhu, and J. Wang are with Tsinghua University, Beijing 100084, China. E-mail: {sxsong, nkugaoyu, chaokun, zhu-xc10, jimwang}@tsinghua.edu.cn.*
- *P.S. Yu is with the University of Illinois at Chicago, Chicago, IL 60607. E-mail: psyu@cs.uic.edu.*

Manuscript received 11 Sept. 2016; revised 18 Mar. 2017; accepted 26 Mar. 2017. Date of publication 4 Apr. 2017; date of current version 5 July 2017. Recommended for acceptance by F. Afrati.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2017.2690912



(a) Trace 1 of events in $\mathcal{L}_1$

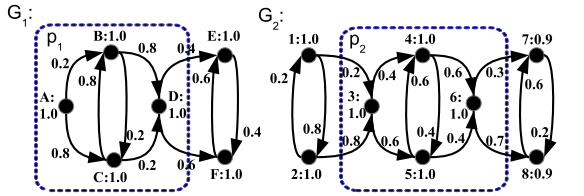
ID	Trace
1	<ABCDEF>
2	<ACBDEF>
3	<ACBDFE>
4	<ABCDFE>
5	<ACBDEF>
6	<ACBDEF>
7	<ACBDFE>
8	<ACBDFE>
9	<ACBDFE>
10	<ACBDFE>

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

(b) Trace 2 of events in $\mathcal{L}_2$

ID	Trace
1	<12354678>
2	<12345678>
3	<2135468>
4	<1235467>
5	<12345687>
6	<12345687>
7	<21354687>
8	<12345687>
9	<12354687>
10	<12354687>

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



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

Fig. 1. An example of heterogeneous event logs.

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 [2]) 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 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 Trace 1 in $\mathcal{L}_1$) matches with the pattern if an instance of the pattern $<$ ABCD $>$ appears as a substring of the trace.

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 [8], [9], [10] (see a discussion of choosing discriminative patterns in Section 2). 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*.

The main challenge of event matching originates from the large space of all possible mappings. We show in Theorem 1 that the problem of finding the optimal event matching with patterns is indeed NP-hard. 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.

1.1 Contributions

Our major contributions in this paper are summarized as:

- We formalize the event matching problem with patterns, show its hardness (Theorem 1) and identify efficiently solvable special case (Theorem 2).
- 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 propose efficient heuristic for event matching. In particular, the proposed algorithm returns the optimal solution in a certain case of vertex patterns (Proposition 6).
- We report an extensive experimental evaluation. It demonstrates that our proposed pattern based matching methods achieve higher accuracy compared with the state-of-the-art approaches. The advanced bounding function significantly reduces time cost (up to two orders of magnitudes improvement).

A preliminary version of this paper appears in [11].

2 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 1 lists the frequently used notations in this paper.

2.1 Uninterpreted Event Matching

Let V be a set of events. A trace 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 [7]) to event logs.

TABLE 1
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 $\delta(p)$
$g(M, U_1, U_2)$	normal distance of a partial mapping M
$h(M, U_1, U_2)$	upper bound of normal distances on unmapped events
$\theta(v_1, v_2)$	the estimated score of a match $v_1 \rightarrow v_2$

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 $M : V_1 \rightarrow V_2$. For an event $v_1 \in V_1$, $v_2 = M(v_1)$ is called the corresponded event of v_1, and $v_1 \rightarrow v_2$ is called a matching/corresponding event pair.

In general, the number of events between the two logs is not required to be the same. However, to illustrate the performance of the Heuristic algorithm in Proposition 6, the same number of events between two logs is assumed, i.e., $|V_1| = |V_2|$. For two logs with different $|V_1|$ and $|V_2|$, following the same line of the Kuhn-Munkres method [12], we can simply introduce some artificial events without actually appearing in the logs, in order to make two logs having the same number of events.

Owing to the absence of typographic or linguistic similarity, the uninterpreted schema matching method [7] 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*

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

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 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.

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

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 Figs. 1e and 1f. We can use normal distance to evaluate event mappings. For the true mapping $M = \{A \rightarrow 3, B \rightarrow 4, C \rightarrow 5, D \rightarrow 6, E \rightarrow 7, F \rightarrow 8\}$ illustrated in Fig. 1, we have normal distance $D^N_v(M) = 5.89$ ($E \rightarrow 7$ and $F \rightarrow 8$ 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 $M' = \{A \rightarrow 6,\ B \rightarrow 2,\ C \rightarrow 1,\ D \rightarrow 3,\ E \rightarrow 4,\ F \rightarrow 5\}$, 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.

2.2 Event Matching with Patterns

As mentioned in Section 1, complex patterns can be discriminative features in event matching. Following the convention of expressing complex event processing queries [2], 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;*
- *$\mathrm{SEQ}(p_1, p_2, \ldots, p_k)$ is an event pattern in which the patterns $p_i, i \in 1, \ldots, k$, occur sequentially;*
- *$\mathrm{AND}(p_1, p_2, \ldots, p_k)$ 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. That is, $\mathrm{AND}(p_1, p_2, \ldots, p_k)$ can be equivalently represented as $\bigvee_{(p'_1, p'_2, \ldots, p'_k) \in \mathcal{Q}} \mathrm{SEQ}(p'_1, p'_2, \ldots, p'_k)$, where $\mathcal{Q}$ is a set containing all distinct permutations of $p_1, p_2, \ldots, p_k$. It denotes that any order of $p_1, p_2, \ldots, p_k$ is acceptable. By recursively representing AND sub-patterns, we obtain $p = \bigvee_{(v'_1, v'_2, \ldots, v'_k) \in \mathcal{I}(p)} SEQ(v'_1, v'_2, \ldots, v'_k)$, where $\mathcal{I}(p)$ denotes all the allowed orders of events $v_1, v_2, \ldots, v_k$ specified by p.

An event pattern can naturally be represented as a directed graph, where each vertex corresponds to an event [13]. 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 $e_1, e_2, \ldots, e_k$ 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). *Consider a trace. Let p be an event pattern in graph form. We say that the trace* matches *with p, if there is a substring of the trace belonging to $\mathcal{I}(p)$.*

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*

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

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 Fig. 1e. 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 Fig. 1e.

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 [2]. 2) There are many existing methods for discovering event patterns in event log [8], [9], [10].

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 Fig. 1) may be less discriminative, since it has a high chance of mapping to many irrelevant patterns.

2.3 Hardness Analysis

The optimal event matching problem with patterns is thus: given a set of events pattern $\mathcal{P}$ over two event logs $\mathcal{L}_1$ and $\mathcal{L}_2$ with event sets V_1 and V_2, to find a mapping $M : V_1 \to V_2$ such that $D^N(M)$ is maximized.

Unfortunately, as shown below, even for the special case of very simple SEQ patterns, the problem is already hard.

Theorem 1. *For the case of edge patterns, i.e., each pattern is only in the form of SEQ$(v, u), v, u \in V_1$, the problem of determining whether there is a mapping M such that $D^N(M) \geq k$ is NP-complete.*

Proof. The problem is clearly in NP. Referring to Formula (1), for each pattern p, the frequencies $f_1(p)$ and $f_2(M(p))$ can be calculated simply by one pass through the event logs $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively.

To show NP-hardness, we build a reduction from the subgraph isomorphism problem, which is known to be NP-complete [14]. Let $G_1(V_1, E_1), G_2(V_2, E_2)$ be graphs. The subgraph isomorphism problem 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 $G_0 \cong G_1$, i.e., whether there exists an $m : V_1 \to V_0$ such that $(v, u) \in E_1 \Leftrightarrow (m(v), m(u)) \in E_0$.

Given two graphs $G_1(V_1, E_1), G_2(V_2, E_2)$, we create two event logs $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. Each edge (v, u) in E_1 or E_2 corresponds to a trace containing two events $\langle vu \rangle$. Moreover, we add traces with a single event to make the same size $|\mathcal{L}_1| = |\mathcal{L}_2|$. For each edge $(v, u) \in E_1$, an event pattern SEQ(v, u) is introduced. The transformation completes.

We show that there exists an $m : V_1 \to V_0$ such that $(v, u) \in E_1 \Leftrightarrow (m(v), m(u)) \in E_0$, if and only if mapping $M : V_1 \to V_2$ such that $D^N(M) \geq k$ where $k = |E_1|$.

First, if such a m exists, we consider m exactly as M. Both edges $(v, u) \in E_1$ and $(m(v), m(u)) \in E_0$ correspond to a trace in $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively, i.e., with frequency 1. By Formula (1), we have $D^N(M) = |\mathcal{P}| = |E_1| = k$.

Conversely, suppose that there is a mapping M with $D^N(M) \geq |\mathcal{P}| = |E_1| = k$. Referring to $0 \leq \delta(p) = 1 - \frac{|f_1(p) - f_2(M(p))|}{f_1(p) + f_2(M(p))} \leq 1$, it must be a mapping M with $D^N(M) = |\mathcal{P}| = |E_1| = k$, where each edge (pattern) is matched. It corresponds to $m : V_1 \to V_0$ such that $(v, u) \in E_1 \Leftrightarrow (m(v), m(u)) \in E_0$. □

Nevertheless, we identify a special case that can be efficiently solved.

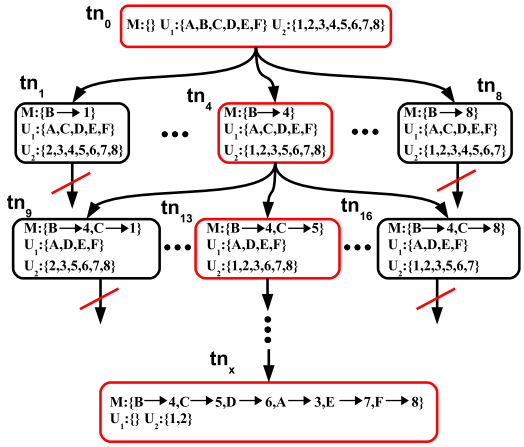


Fig. 2. An example of A* search tree.

Theorem 2. *For the case of vertex patterns, i.e., each pattern consists of only one single event, the optimal event matching problem can be solved in* $O(n^4 * |\mathcal{L}| * |\mathcal{P}|)$ *time, where* $n = \max(|V_1|, |V_2|)$.

Proof. We present Algorithm 3 for event matching. Referring to the complexity analysis in Section 5.3.2, and correctness in Proposition 6, the conclusion is proved. □

3 A Generic Event Matching Framework

The total number of distinct mapping M is $n(n-1)(n-2)\dots(n-m+1)$, 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.

3.1 Overview of A* Search

The process of A* search algorithm follows the growth of A* search tree, e.g., in Fig. 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 $a \to b$ 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 $U_1' = U_1 \setminus \{a\}, U_2' = U_2 \setminus \{b\}$ 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

Require: Event logs $\mathcal{L}_1$ and $\mathcal{L}_2$ with event sets V_1 and V_2
Ensure: 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: $Q := Q \setminus \{(M, U_1, U_2)\}$
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. Fig. 2 illustrates an example of conducting the A* search algorithm for matching $\mathcal{L}_1$ and $\mathcal{L}_2$ in Fig. 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.

3.2 Efficiently Computing the Normal Distance G

Let $\mathcal{P}_{M'}$ be the set of patterns whose events are all defined in the new partial matching M', and $\mathcal{P}_M$ be all the patterns corresponding to the previous M. It is evident that $\mathcal{P}_M \subseteq \mathcal{P}_{M'}$. We denote $\mathcal{P}_{\text{new}} = \mathcal{P}_{M'} \setminus \mathcal{P}_M$ 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

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

according to Formula (1).

Therefore, instead of recalculating g for each node, we use a more efficient three-step method to conduct the

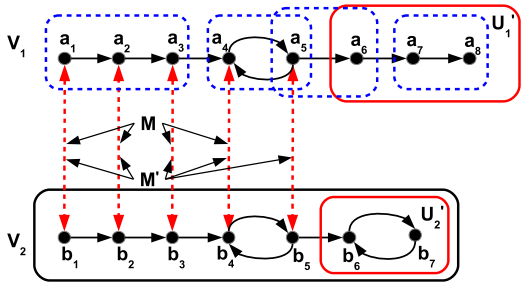


Fig. 3. Expanding a new mapping $a_5 \rightarrow b_5$ to M.

incremental computation of g of a child node from the g of its parent node. First, 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 $\delta(p)$.

3.2.1 Capturing Newly Introduced Patterns

As the first step of calculating g, we need to know which patterns are new introduced, i.e., $\mathcal{P}_{\text{new}}$. 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 $\mathcal{P}_{\text{new}} = (\bigcup_{v \in V_1 \setminus U'_1} I_p(v) \setminus \bigcup_{v \in U'_1} I_p(v)) \setminus (\bigcup_{v \in V_1 \setminus U_1} I_p(v) \setminus \bigcup_{v \in U_1} I_p(v))$. 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 $\mathcal{P}_M$).

3.2.2 Pruning No Contribution Patterns

For a pattern $p \in \mathcal{P}_{\text{new}}$ 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 3 (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 $p \in \mathcal{P}_{\text{new}}$, if $M(p)$ is not a subgraph of G_2, we directly conclude $\delta(p) = 0$ without computing over $\mathcal{L}_2$.

3.2.3 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 $v \in V_1$, corresponds to a list of traces 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 $f_2(M(p))$.

3.3 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. Fig. 3 illustrates an example of expanding a new mapping $a_5 \rightarrow b_5$ to a partial matching $M = \{a_1 \rightarrow b_1, a_2 \rightarrow b_2, a_3 \rightarrow b_3, a_4 \rightarrow b_4\}$. Then we have $\mathcal{P}_M = \{\text{SEQ}(a_1, a_2, a_3)\}$, $\mathcal{P}_{M'} = \{\text{SEQ}(a_1, a_2, a_3), \text{AND}(a_4, a_5)\}$, thus $\mathcal{P}_{new} = \{\text{AND}(a_4, a_5)\}$. 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 $M(\text{AND}(a_4, a_5))$ may exist in G_2 and cannot be pruned. At last, we have the remaining patterns $\text{SEQ}(a_7, a_8)$ and $\text{SEQ}(a_5, a_6)$.

3.4 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 $O(|V_1| * |\mathcal{L}| * |\mathcal{P}| * n!)$.

For average-case complexity, let r denote the branching factor, $0 \leq r \leq 1$. That is, only a proportion of rx branches among all the x branches for a node are expanded (in Fig. 2), while the other $(1-r)x$ branches are pruned by various upper bounds. Such a factor r can be determined empirically [15] by measuring the number of nodes expanded, N, and solving $N = nr + n(n-1)r^2 + \cdots + n!r^n$. The expected time complexity is thus $O(|V_1| * |\mathcal{L}| * |\mathcal{P}| * n!r^n)$, where $0 \leq r \leq 1$.

4 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 $M : V(p) \rightarrow U_2$.*

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 Fig. 3, we have $\Delta(\text{SEQ}(a_6, a_7, a_8), \{b_6, b_7\}) = 0$.

TABLE 2
Summary of Tighter Upper Bounds [11]

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

For general patterns, we compute the bounds by utilizing vertex and edge frequencies. Table 2 summarizes the upper bound $\Delta(p,U_2)$ for various patterns p by either using the maximum vertex frequencies f_n or edge frequencies f_e (see detailed derivation in [11]). 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 4. If not, the algorithm tries to estimate the upper bound via vertex frequency (bound 1 in Table 2) or via edge frequency (bounds 2-4 in Table 2), 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)

Require: p is a pattern in $\mathcal{L}_1$ and U_2 is the set of events in $\mathcal{L}_2$.
Ensure: The upper bound $\Delta(p,U_2)$ of $\delta(p)$ w.r.t. possible mappings $M: V(p)\to U_2$.
1: **if** $|V(p)| > |U_2|$ **then**
2: **return** 0
3: $f_n :=$ the highest frequency of events (vertices) in U_2
4: $f_e :=$ the highest frequency of edges in the subgraph of G_2 with vertices of U_2
5: **if** $f_n \geq \omega(p) f_e$ **then**
6: $f_{\mathsf{min}} := \omega(p) f_e$
7: **else**
8: $f_{\mathsf{min}} := f_n$
9: **if** $f_{\mathsf{min}} \leq f(p)$ **then**
10: **return** $1-\frac{f(p)-f_{\mathsf{min}}}{f(p)+f_{\mathsf{min}}}$
11: **else**
12: **return** 1.0

5 HEURISTIC APPROACH

Although effective pruning bounds are devised, referring to the complexity analysis in Section 3.4, the cost of exact algorithm is still high. Recognizing the hardness of the matching problem in Theorem 1, we study efficient heuristics below. A straightforward approach is rather than trying all the possible expansions in the A* search tree, we consider only one expansion, e.g., with the maximum g and h. That is, in Line 7 in Algorithm 1, it considers only one M' by expanding $a\to b$, which has the maximum $g(M',U_1',U_2')+h(M',U_1',U_2')$ among all the $b\in U_2$.

This simple heuristic falls short in two aspects. (1) Each step evaluates locally the next unmatched event and the corresponding patterns, without a global view of all the events and patterns. (2) Once an erroneous pair $a\to b$ is expanded in matching, it will not be changed and affect the matching of other events, in the subsequent steps.

5.1 Overview

To address the aforesaid deficiencies of simple heuristics, we follow two intuitions. (1) We estimate the contribution of each possible matching event pair to the pattern normal distance, and thus are able to have a global estimation during the generation of candidate mapping M. (2) Enlighten by the Hungarian algorithm [12] for bipartite matching, we propose to modify the previously determined matching M referring to the patterns.

5.1.1 Preliminaries

For $v_1\in V_1, v_2\in V_2$, we define the estimated score of a match $v_1\to v_2$ in the normal distance $D^N(M)$ of M as

$$\theta(v_1,v_2)=\sum_{p\in\mathcal{P},v_1\in p}\frac{1}{|p|}\left(1-\frac{f_1(p)-f_2(v_2)}{f_1(p)+f_2(v_2)}\right), \tag{2}$$

where $v_2=M(v_1)$, p is a pattern containing v_1, and $|p|$ is the total number of events in pattern p. Let $\Theta(M)=\sum_{v_1\in V_1}\theta(v_1,M(v_2))$ denote a global estimation of pattern normal distance. This estimated score has the following properties.

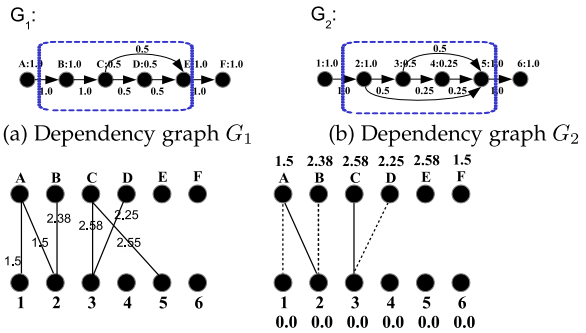
(1) If $f_2(v_2)$ perfectly estimates $f_2(p_2)$ for each possibly matched pattern p_2, where v_2 is contained in p_2, then we have $D^N(M)=\Theta(M)$, i.e., computes exactly the normal distance.
(2) When only vertex patterns are considered, having $|p|=1$ for each pattern, we have $D^N(M)=\Theta(M)$, i.e., exactly equals to vertex normal distance.

In order to obtain candidates of M that have large estimated score $\Theta(M)$, we consider an event labeling function $\ell: V_1\cup V_2\to\mathcal{R}$. A feasible labeling has $\ell(v_1)+\ell(v_2)\geq\theta(v_1,v_2),\forall v_1\in V_1,v_2\in V_2$. Thereby, the summation of labeling over all the events serves as an upper bound of $\Theta(M)$. We show below how the matching is augmented while the upper bound of $\Theta(M)$ by labeling is still valid.

A path $v_1,v_2\ldots,v_n$, denoted by v_1-v_n for simplicity, has $v_i\in V_{2-(i \bmod 2)}, i=1,\ldots,n$, i.e., $v_1,v_3,v_5,\ldots\in V_1, v_2,v_4,v_6,\ldots\in V_2$ and $\ell(v_j)+\ell(v_{j+1})=\theta(v_j,v_{j+1}), j=1,\ldots,n-1$. A path is alternating if $M(v_{2i+1})=v_{2i}, v_{2i+1}\in V_1, v_{2i}\in V_2, i=1,\ldots,\lceil\frac{n}{2}\rceil-1$. We represent multiple alternating paths starting from the same event v_1 as an alternating tree rooted at v_1. A path is augmenting if (1) it is an alternating path, and (2) both v_1 and v_n are unmatched. We augment M w.r.t. an augmenting path v_1-v_n to generating a new matching M' by setting $M'(v_{2i-1})=M'(v_{2i}), i=1,\ldots,\frac{n}{2}$. From M to M', the number of matched event pairs increases by one $|M|+1=|M'|$.

By gradually augmenting the matching, being aware of the upper bound of $\Theta(M)$ by labeling, a mapping over all the events will be generated. For simplicity, we assume $|V_1|=|V_2|$. (For $|V_1|\neq|V_2|$, such as $|V_1|<|V_2|$, to ensure the correctness, we can simply introduce some dummy events v_1 in V_1 to make their sizes equal, having $\theta(v_1,v_2)=0,\forall v_2\in V_2$.)

Example 7. Fig. 4 illustrates two dependency graphs transformed from two logs. Suppose that in addition to vertex and edge patterns, there are two other complex patterns



(a) Dependency graph G_1 (b) Dependency graph G_2

(c) Some estimated score (d) Alternating/augmenting paths

Fig. 4. An example of heterogeneous event logs.

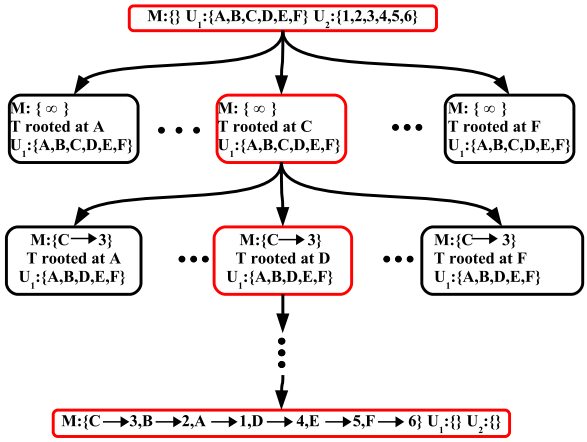


Fig. 5. An example of heuristic matching.

$p_1 = \text{SEQ}(B, C, D, E)$ and $p_2 = \text{SEQ}(B, C, E)$. We have 6 patterns containing the event C, i.e., one vertex pattern $\{C\}$, three edge patterns $\{BC, CD, CE\}$, and the aforesaid two complex patterns with the same frequency 0.5. Referring to Formula (2), it estimates $\theta(C, 3) = 2.58$ as the number presented between events C and 3 in Fig. 4c, given $f_2(3) = 0.5$. (We present only some estimated scores that are used in the following examples, and omit others to make the figure clear.)

Fig. 4d illustrates a possible feasible labeling ℓ. For each pair of events from V_1 and V_2, respectively, say $C \in V_1$ and $3 \in V_2$, we have $\ell(C) + \ell(3) = 2.58 + 0 \geq \theta(C, 3)$. Suppose that the current mapping is $M = \{A \to 2, C \to 3\}$. Then, $D, 3, C$ is an alternating path, denoted by $D - C$, where the solid line means a matching in M. Moreover, $B, 2, A, 1$ is an augmenting path, denoted by $B - 1$, where both B and 1 are unmatched.

5.1.2 Algorithm

Algorithm 3 presents the major steps of heuristic matching. Initially, we take a feasible labeling ℓ having $\ell(v_1) = \max_{v_2 \in V_2}\{\theta(v_1, v_2)\}, \forall v_1 \in V_1$, and $\ell(v_2) = 0, \forall v_2 \in V_2$.

In each iteration, we consider an unmatched $u_i \in U_1$. Line 4 in Algorithm 3 finds all the augmenting paths starting from u_i (by using Algorithm 4 introduced below), represented in an alternating tree $\mathcal{T}$.

For each augmenting path, we augment M to generate a possible M', such that the number of matched event pairs increases by one $|M| + 1 = |M'|$.

Among all the possible augmentations, referring to Line 3 in the exact Algorithm 1, we heuristically accept the augmentation M' with the maximum bound $g(M', U_1', U_2') + h(M', U_1', U_2')$.

Example 8. Fig. 5 illustrates an example of conducting the heuristic matching algorithm given the matching graphs G_1 and G_2 in Fig. 4. According to Algorithm 3, we initialize a feasible labeling ℓ as presented in Fig. 4d. To expand the mapping, each unmatched event u_i is selected from U_1. It generates a corresponding alternating tree rooted at u_i. For each alternating tree, it may has many augmenting paths. We will choose the new mapping with the maximal $g + h$. Suppose that the augmenting path $C, 3$ in the alternating tree rooted at C has the maximum $g + h$ score. The mapping is augmented, having $M = \{C \to 3\}$. In the second iteration, we consider alternating trees rooted at unmatched events A, B, D, E, F, respectively, among which there is an augmenting path starting from D with the maximum $g + h$. The expansion carries on till U_1 is empty. Finally, a mapping $M = \{C \to 3, \ldots, F \to 6\}$ is returned.

Algorithm 3. HEURISTIC MATCHING

Require: Two event logs $\mathcal{L}_1$ and $\mathcal{L}_2$ with event sets V_1 and V_2
Ensure: Event mapping M
1: initialize labeling ℓ
2: **repeat**
3: **for** each unmatched event $u_i \in U_1$ **do**
4: $(\mathcal{T}_i, \ell_i) :=$ ALTERNATINGTREE(u_i, ℓ)
5: **for** each augmenting path $u_i - u_j$ in the alternating tree $\mathcal{T}_i$ **do**
6: let M^{ij} be the new mapping by augmenting M w.r.t. $u_i - u_j$
7: $(M, \ell) := \arg\max_{(M^{ij}, \ell_i)} g(M^{ij}, U_1^{ij}, U_2^{ij}) + h(M^{ij}, U_1^{ij}, U_2^{ij})$
8: **until** $U_1 = \emptyset$
9: **return** M

5.2 Generating Maximal Alternating Tree

It is worth noting that any augmenting path $u - v$ always has $u \in V_1$ and $v \in V_2$. To consider more augmenting candidates, we consider all the possible augmenting paths with the same start event u and various end events from V_2, i.e., the maximal alternating tree rooted at u with at most $|V_2|$ possible augmenting paths. Algorithm 4 illustrates the major steps for generating the maximal alternating tree.

Given any labeling ℓ, we denote $E_\ell = \{(v_1, v_2) \,|\, \ell(v_1) + \ell(v_2) = \theta(v_1, v_2), v_1 \in V_1, v_2 \in V_2\}$ the edges between events. For any unmatched u, by either breadth-first or depth-first search, we obtain an alternating tree rooted at u.

Let $T_1 \subseteq V_1$ and $T_2 \subseteq V_2$ be the sets of events appearing in the alternating tree. Consider

$$\alpha_\ell = \min_{v_1 \in T_1, v_2 \in V_2 \setminus T_2} \{\ell(v_1) + \ell(v_2) - \theta(v_1, v_2)\}. \tag{3}$$

We update ℓ to introduce new edge

$$\ell'(v) = \begin{cases} \ell(v) - \alpha_\ell & \text{if } v \in T_1 \\ \ell(v) + \alpha_\ell & \text{if } v \in T_2 \\ \ell(v) & \text{otherwise .} \end{cases} \tag{4}$$

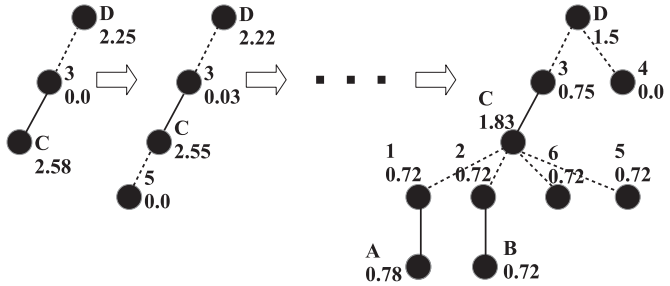


Fig. 6. Generating maximal alternating tree.

There is some $v_1 \in T_1, v_2 \notin T_2$, having $\ell'(v_1) + \ell'(v_2) = \theta(v_1, v_2)$. That is, the alternating tree is expanded (in Line 4 in Algorithm 4).

By gradually expanding all the events in V_2 into T_2 of the alternating tree $\mathcal{T}$, the maximal alternating tree $\mathcal{T}$ is generated and returned.

Algorithm 4. AlternatingTree

Require: unmatched u, labeling ℓ
Ensure: Maximal alternating tree $\mathcal{T}$
1: initialize $\mathcal{T}$ rooted at u w.r.t. ℓ
2: **while** $|T_2| < |V_2|$ **do**
3: update ℓ referring to Formula (4)
4: expand $\mathcal{T}$ rooted at u w.r.t. ℓ
5: **return** $\mathcal{T}, \ell$

Example 9. Let $M = \{A \to 2, B \to 1, C \to 3\}$ be the current mapping. Fig. 6illustrates an example of generating the maximal alternating tree rooted at event D. During the initialization, an alternating tree $\mathcal{T}$ with one path is generated, referring to the estimated scores and the current labeling ℓ in Fig. 4. Suppose that by using Formula (3), we obtain $\alpha_\ell = 0.03$. According to Formula (4), we have $\ell'(D) = \ell(D) - 0.03 = 2.22, \ell'(C) = \ell(C) - 0.03 = 2.55$, while $\ell'(3) = \ell(3) + 0.03 = 0.03$. The alternating tree $\mathcal{T}$ is thus expanded with augmenting path $D, 3, C, 5$. By recursively applying the updates on ℓ till $|T_2| = |V_2|$, the maximal alternating tree $\mathcal{T}$ containing all the events in V_2 is obtained.

5.3 Correctness and Performance Analysis

5.3.1 Correctness

First, we illustrate that the alternating tree can always be expanded in each iteration in Algorithm 4, and the updating of ℓ in Formula (4) is safe.

Proposition 4. *The updated ℓ' in Formula (4) is a feasible labeling and*

(1) *for $v_1 \in T_1, v_2 \in T_2$, if $\ell(v_1) + \ell(v_2) = \theta(v_1, v_2)$, then $\ell'(v_1) + \ell'(v_2) = \theta(v_1, v_2)$;*
(2) *for $v_1 \notin T_1, v_2 \notin T_2$, if $\ell(v_1) + \ell(v_2) = \theta(v_1, v_2)$, then $\ell'(v_1) + \ell'(v_2) = \theta(v_1, v_2)$.*

Proof. We consider four possible cases as follows.

(1) for $v_1 \in T_1, v_2 \in T_2$, referring to Formula (4), we have $\ell'(v_1) + \ell'(v_2) = \ell(v_1) + \ell(v_2) \geq \theta(v_1, v_2)$, i.e., still feasible.
(2) for $v_1 \notin T_1, v_2 \notin T_2$, the labeling on v_1, v_2 is not changed, having $\ell'(v_1) + \ell'(v_2) = \ell(v_1) + \ell(v_2) \geq \theta(v_1, v_2)$ as well.
(3) for $v_1 \in T_1, v_2 \notin T_2$ we have $\ell'(v_1) + \ell'(v_2) = \ell(v_1) + \ell(v_2) - \alpha_\ell$. Referring to the definition of α_ℓ, i.e., $\alpha_\ell \leq \ell(v_1) + \ell(v_2) - \theta(v_1, v_2)$ for $v_1 \in T_1, v_2 \in V_2 \setminus T_2$, it follows $\ell'(v_1) + \ell'(v_2) \geq \theta(v_1, v_2)$.
(4) for $v_1 \notin T_1, v_2 \in T_2$, it has $\ell'(v_1) + \ell'(v_2) = \ell(v_1) + \ell(v_2) + \alpha_\ell \geq \ell(v_1) + \ell(v_2)$, since $\alpha_\ell = \ell(v_1') + \ell(v_2') - \theta(v_1', v_2') \geq 0$ for some $v_1' \in T_1, v_2' \in V_2 \setminus T_2$. □

Moreover, we illustrated that the matching M can always be augmented in each iteration in Algorithm 3.

Proposition 5. *There always exists an augmenting path $u_i - u_j$ in the alternating tree $\mathcal{T}$ generated by Algorithm 4, having u_j unmatched.*

Proof. Referring to Line 2, Algorithm 4 generates an alternating tree $\mathcal{T}$ with all the events in V_2 involved. If all the events in V_2 are matched, according to $|V_1| = |V_2|$, all the events in $|V_1|$ should be matched as well, which is contradictory to existing some unmatched $u_i \in U_1$. In other words, there exist some unmatched $u_j \in V_2$ in the alternating tree $\mathcal{T}$ which forms augmenting path $u_i - u_j$. □

Consequently, by augmenting the matching in each iteration, Algorithm 3 always terminates and return a perfect matching M where each event is matched.

5.3.2 Complexity

Referring to Proposition 5, the size of matching M increases by one in each iteration in Line 2 in Algorithm 3. Let $n = |V_1| = |V_2|$ be the number of events. To generate the alternating tree with all the events in V_2 involved, the labeling ℓ over events may need to be updated n times in Algorithm 4, with cost $O(n^2)$. There are at most n augmenting paths in an alternating tree $\mathcal{T}$. For each augmenting M^{ij}, we evaluate its pattern score, with $O(n * |\mathcal{L}| * |\mathcal{P}|)$ time. Given at most n alternating trees, the cost of each augmenting iteration is $O(n^3 * |\mathcal{L}| * |\mathcal{P}|)$. Given at most n augmenting iterations, Algorithm 3 runs in $O(n^4 * |\mathcal{L}| * |\mathcal{P}|)$ time.

5.3.3 Special Case of Vertex Patterns

While Theorem 1 states that even for the special case of edge patterns, finding the optimal matching is NP-hard, the proposed heuristic algorithm performs well for the case of vertex patterns.

Proposition 6. *For the special case of vertex patterns, i.e., each pattern consists of only one single event, Algorithm 3 returns the optimal matching.*

Proof. The augmentation on M w.r.t. the augmenting path ensures $\ell(v_1) + \ell(v_2) = \theta(v_1, v_2)$ on the augmented $v_1 \in V_1, v_2 \in V_2, v_2 = M(v_1)$ in the augmenting path. Moreover, Proposition 4 guarantees that $\ell(v_1) + \ell(v_2) = \theta(v_1, v_2)$ retains after updating ℓ in Formula (4). Algorithm 3 returns a perfect matching M where each event is matched, i.e., having $D^N(M) = \Theta(M) = \sum_{v \in V_1 \cup V_2} \ell(v)$.

Referring to Kuhn-Munkres theorem [12], the summation of labeling over all the events $\sum_{v \in V_1 \cup V_2} \ell(v)$ serves as an upper bound of $\Theta(M') = D^N(M')$ w.r.t. vertex

TABLE 3
Characteristics of the Logs

Dataset	# traces	# events (vertices)	# edges	# patterns
real	3,000	11	57	3
synthetic	10,000	100	142	16
random	1,000	4	12	0

patterns for any matching M'. Since there is no other pattern, the solution M is optimal, i.e., $D^N(M) = \Theta(M)$ w.r.t. $\theta(v_1, v_2)$ over all $v_1 \in V_1, v_2 \in V_2$ is the maximum. □

6 Experiment

In this section, we report the experimental evaluations by comparing our proposed method with the state-of-the-art schema matching approaches [7], [16].

Data Set

We employ a real data set from a bus manufacturer. The event logs extracted from the ERP systems of two departments located at distinct industrial parks, respectively. There are up to 11 distinct events (analogous to schema size in schema matching) in these logs ranges, and the corresponding number of traces (instance size of the number of tuples) is 3,000. Table 3 shows the characteristics of the logs. In addition to traces, the numbers of vertices (events) and edges in the dependency graph are reported. The patterns are manually assigned by the guidelines discussed in Section 2. For edge patterns, referring to [7] with the consideration of both Vertex and Edge, all the edges appearing in the dependency graph are employed.

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 $F\text{-}Measure = 2 \cdot \frac{precision \cdot recall}{precision + recall}$.

6.1 Evaluating Exact Approach

We compare the accuracy of the proposed approach (Pattern) with existing approaches [7] (Vertex, Vertex+Edge) and [16] (Iterative). In particular, [7] employs the normal distance by considering Vertex or Vertex+Edge similarities. Instead of enumerating possible mappings and ranking the corresponding normal distances, [16] computes the vertex similarity in page-rank like iterative way.

Figs. 7a and 8a report the F-measure of all approaches by varying event set sizes and trace numbers. An event set with size x is determined by projecting the first x events appearing in the dataset, and similarly a number of y traces are determined by selecting the first y traces appearing in the event log. As shown, our pattern based approach outperforms others with the highest accuracies. Pattern-Simple shows exactly the same accuracy as Pattern-Tight, since both approaches return the exact results (with different pruning power). The accuracy drops from sizes 2 to 4 of event sets in Fig. 7a. 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.

We compare the time performance of all approaches in Figs. 7b and 8b, by varying event set sizes and trace numbers. To evaluate the pruning power of upper bounds of h, we test the proposed approaches with simple bound (Pattern-Simple) in Section 3.3 or tight bound (Pattern-Tight) in Section 4. 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 3. The time cost also rises along with the growth of trace number since it needs more time to compute the frequencies in traces.

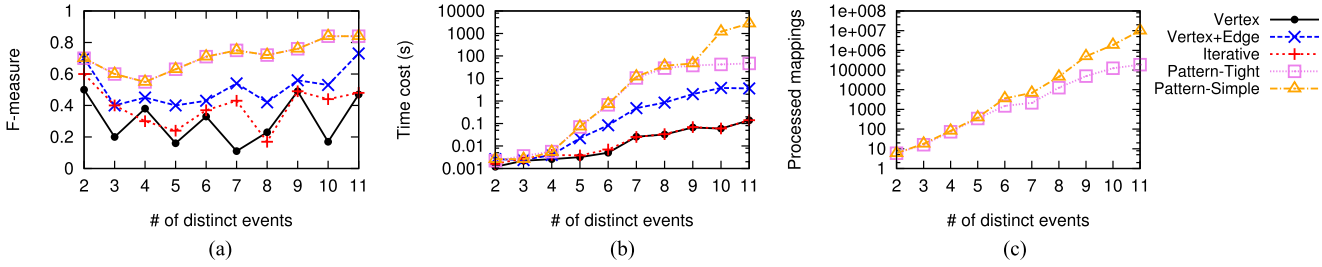


Fig. 7. Evaluation of exact approaches over various # of events.

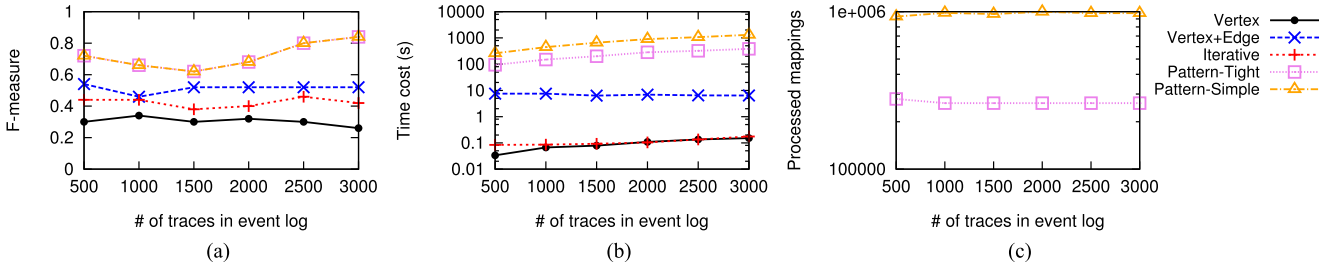


Fig. 8. Evaluation of exact approaches over various # of traces.

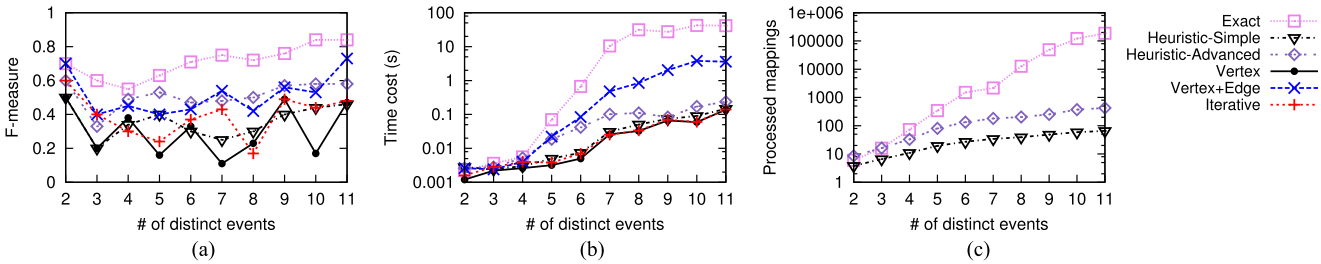


Fig. 9. Evaluation of heuristic approach over various # of events.

To evaluate the pruning power of tight upper bounds, in Figs. 7c and 8c, we observe the number of processed mappings M' in Line 7 in Algorithm 1, i.e., the number of processed tree nodes in the A^* search tree. As shown in Fig. 7c, the approach using tight bounding function expands less tree nodes during the A^* search, especially in large event set sizes. Consequently, as discussed in Fig. 7b, the approach with tight bound shows at most 2 orders of magnitudes improvement in time costs. Fig. 8c shows the growth of trace number does not affect the pruning power much.

6.2 Evaluating Heuristic Approach

To evaluate heuristic matching studied in Section 5, we consider two approaches: (1) Heuristic-Simple considers one expansion instead of all in the A^* search tree, as introduced at the beginning in Section 5; and (2) Heuristic-Advanced, i.e., Algorithm 3. For the Exact approach in comparison, we employ the aforesaid Pattern-Tight with lower time costs.

Figs. 9a and 10a report the accuracy of heuristic and exact approaches, again over various event set sizes and trace numbers. As shown, the Heuristic-Advanced method shows a clear improvement on accuracy compared to Heuristic-Simple, in all the tests. The results verify the analysis of simple heuristic deficiency at the beginning in Section 5, and the motivation of the advanced heuristic proposal presented in the first paragraph of Section 5.1.

Compared to the Exact approach, the heuristic methods show orders of magnitudes improvement in time performance, as illustrated in Figs. 9b and 10b. The rationale is that the heuristic approaches process significantly less mappings. Similarly, in Figs. 9c and 10c, we report the number of processed mappings M^{ij} in Line 6 in Algorithm 3. As shown, while Heuristic-Advanced processes a bit more mappings compared to Heuristic-Simple, it is still significantly lower than that of Exact. In summary, the Heuristic-Advanced method provides a good trade-off between matching accuracy and time costs, with F-measure closer to Exact and time costs comparable to Heuristic-Simple.

We also plot the results of baseline approaches, including Vertex, Vertex+Edge and Iterative. As shown in Figs. 9a and 10a, Heuristic-Advanced shows clear improvement in accuracy compared to Vertex and Iterative approaches, while their time costs are comparable in Figs. 9b and 10b. On the other hand, Heuristic-Advanced has accuracy comparable to Vertex+Edge (with improvement in some tests), the corresponding time costs of Heuristic-Advanced are significantly lower (more than one order of magnitude improvement). The results demonstrate again the trade-off by the Heuristic approach.

6.3 Evaluation Over Synthetic Data

6.3.1 *Larger Synthetic Data*

To generate larger data sets, we repeat the structures in Fig. 1 in Example 1 (with different event names) as illustrated in Figs. 11a and 11b for two event logs, respectively. The traces are randomly drawn from the sets in Figs. 11c and 11d, which are instances of event traces leading to the aforesaid dependency graphs. The total number of traces is 10,000. It is worth noting that events ABCD are executed in parallel (with an AND pattern), while events F,G,H,I are executed separately. Similar execution applies to 1234 and 6,7,8,9. Therefore, the ground truth should be $A \rightarrow 1, B \rightarrow 2, \ldots, F \rightarrow 6, G \rightarrow 7, \ldots$. However, their structures in dependency graphs in Figs. 11a and 11b are very similar, which leads to difficulty in matching as discussed in Example 1.

Fig. 12 presents the results over up to 100 events. We do not consider an even larger number of events in one experiment, since the number of events in real business processes is often bounded by about 60, according to the recent survey [17]. Indeed, referring to the business process modeling guidelines [18], processes should be decomposed if they have more than 50 events, so that they are easier to operate in practice. That is, hundreds or thousands of distinct events/items in an information system are decomposed into small components. The matching thus happens between these relatively small components.

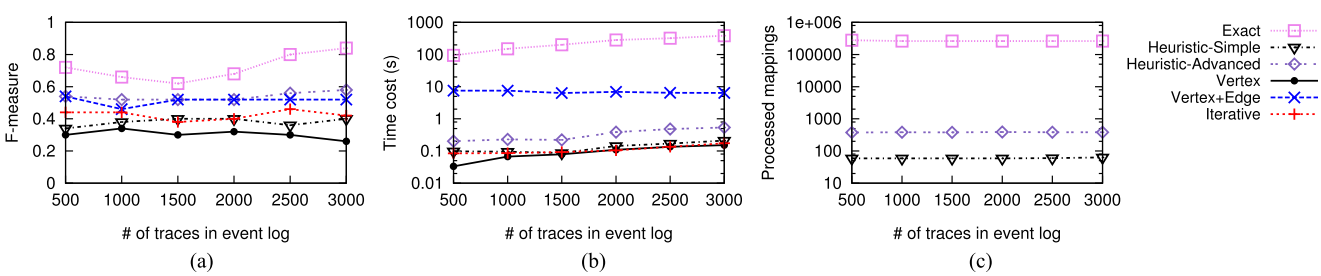


Fig. 10. Evaluation of heuristic approach over various # of traces.

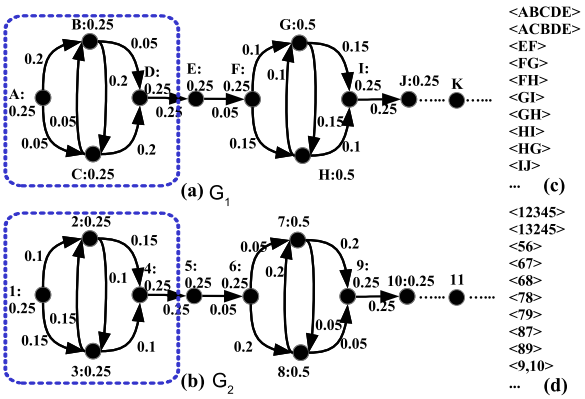


Fig. 11. Synthetic data generation.

As shown in Fig. 12, our proposed Exact method still have the highest accuracy. Unfortunately, this high cost Exact method does not scale well over a larger number of events (cannot return results over 20 events). Indeed, the Vertex+Edge approach cannot return results under 20 or more events either. Nevertheless, the proposed Heuristic algorithms still find a matching with higher accuracy (compared to Vertex and Iterative methods).

To compare with other non-graph-based methods, we consider the Entropy-only approach in [7]. Since we focus on un-interpreted matching, where the typographic or linguistic similarities are not available, only the frequency information of event appearance could be utilized. The Entropy-only approach considers the similarity of events on the uncertainty of whether the events appear in a trace, without exploiting the structural information among events. As shown in Fig. 12, compared to the Entropy-only approach, the matching accuracy of our proposal is higher, while the time cost of Entropy-only is lower. The comparison to the non-graph-based method shows the trade-off between accuracy and efficiency. In this sense, the proposed method is applicable to the scenarios having high matching accuracy requirements while the additional time costs are acceptable.

6.3.2 Random Data

To further evaluate the behavior of the algorithm, we consider two random logs over four events A, B, C, D and 1,2,3,4, respectively. The total number of randomly generated traces is 1,000. Obviously, no true mapping exists between these two random logs.

Table 4 presents the mapping results by applying the proposed methods, Exact, Heuristic-Simple and Heuristic-Advanced. As shown, all the 24 possible mappings may be returned in 1,000 tests, rather than finding the same mapping over multiple runs. We count the number of returned mappings in these tests. No method is clearly in favor of some particular results.

6.4 Discussion

When there is no complex patterns available, e.g., with only two events in Fig. 7a, the pattern-based approach shows no improvement compared to the edge-based method. Since edges are the only (special) patterns, these two approaches are equivalent in this case. Moreover, if the total number of events is large, such as 100 in Fig. 12, the high cost Exact matching cannot return a result. Indeed, all the methods

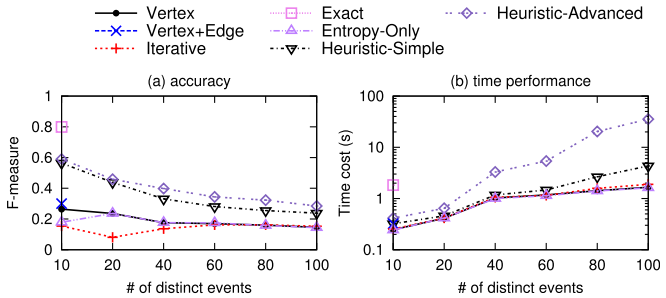


Fig. 12. Varying # of events over larger synthetic data.

shows low matching accuracy in such cases. The reason is that events are more similar with each other when there are more events under consideration. That is, distinguishing events becomes harder, and thus the accuracy of all the methods is low.

7 Related Work

The quality of event data has recently been highlighted. Rather than addressing inconsistencies existing in event logs [19], [20], in this study, we focus on matching the heterogeneous events collected from different sources.

7.1 Structure Based Matching

Madhavan et al. [21] 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. [22] 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. [16] proposed a method by calculating the vertex similarity between two graphs through iterative

TABLE 4
Counts of Returned Results Over Random Logs in 1,000 Tests

	Mapping Result	Exact	Heuristic-Simple	Heuristic-Advanced
1	$A \to 4, B \to 3, C \to 2, D \to 1$	51	42	43
2	$A \to 4, B \to 1, C \to 2, D \to 3$	48	36	35
3	$A \to 3, B \to 4, C \to 2, D \to 1$	35	39	39
4	$A \to 3, B \to 1, C \to 2, D \to 4$	32	41	43
5	$A \to 4, B \to 2, C \to 1, D \to 3$	41	34	45
6	$A \to 4, B \to 3, C \to 1, D \to 2$	40	46	49
7	$A \to 1, B \to 3, C \to 4, D \to 2$	39	35	37
8	$A \to 2, B \to 1, C \to 3, D \to 4$	46	47	48
9	$A \to 2, B \to 4, C \to 3, D \to 1$	53	50	41
10	$A \to 1, B \to 2, C \to 4, D \to 3$	32	45	39
11	$A \to 2, B \to 1, C \to 4, D \to 3$	41	43	39
12	$A \to 2, B \to 3, C \to 4, D \to 1$	42	39	33
13	$A \to 3, B \to 1, C \to 4, D \to 2$	36	52	45
14	$A \to 3, B \to 2, C \to 4, D \to 1$	44	37	47
15	$A \to 4, B \to 2, C \to 3, D \to 1$	42	41	39
16	$A \to 4, B \to 1, C \to 3, D \to 2$	43	31	41
17	$A \to 2, B \to 3, C \to 1, D \to 4$	39	45	49
18	$A \to 1, B \to 4, C \to 2, D \to 3$	48	39	48
19	$A \to 1, B \to 3, C \to 2, D \to 4$	36	54	36
20	$A \to 2, B \to 4, C \to 1, D \to 3$	48	39	46
21	$A \to 1, B \to 4, C \to 3, D \to 2$	45	44	39
22	$A \to 1, B \to 2, C \to 3, D \to 4$	30	42	36
23	$A \to 3, B \to 2, C \to 1, D \to 4$	35	43	41
24	$A \to 3, B \to 4, C \to 1, D \to 2$	54	36	42

computations. According to the experimental results in Section 6, the iterative method [16] shows lower matching accuracy than our proposed pattern based matching. Kang et al. [7] 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. [23] 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.

7.2 Capturing Structure Information Among Events

Graph is often employed to represent the structural information among events [24]. 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. [25], Cook et al. [26], and Ferreira et al. [27] 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. In this paper, we employ the dependency graph proposed in [7] by weighting vertices and edges with normalized frequencies, since it distinguishes the significance of distinct edges, and is easy to interpret.

7.3 Discovering Event Pattern

The discovery of complex event pattern has been studied in complex event processing (CEP) [28]. Agrawal et al. [8] and Mannila et al. [9] 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. [10] 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 2 on choosing discriminative patterns for matching.

8 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. Recognizing the NP-hardness of the optimal event matching problem with patterns, we propose efficient heuristic. 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 heuristic method significantly reduce the time costs.

It is not supported in this study that an event is mapped to multiple events, known as 1-to-n matching, or more generally m-to-n matching. Since one event might be mapped to multiple events, patterns in two logs may have different structures. Pruning w.r.t. distinct patterns becomes more complicated. We leave this interesting yet challenging problem in the future study.

Acknowledgments

This work is supported in part by the National Key Research Program of China under Grant 2016YFB1001101, China NSFC under Grants 61572272, 61325008, 61370055, and 61202008, and the Tsinghua University Initiative Scientific Research Program. Chaokun Wang is the corresponding author.

References

[1] P. Sun, Z. Liu, S. B. Davidson, and Y. Chen, "Detecting and resolving unsound workflow views for correct provenance analysis," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 549–562. [Online]. Available: http://doi.acm.org/10.1145/1559845.1559903

[2] L. Ding, S. Chen, E. A. Rundensteiner, J. Tatemura, W. Hsiung, and K. S. Candan, "Runtime semantic query optimization for event stream processing," in *Proc. 24th Int. Conf. Data Eng.*, 2008, pp. 676–685. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2008.4497476

[3] O. Biton, S. C. Boulakia, S. B. Davidson, and C. S. Hara, "Querying and managing provenance through user views in scientific workflows," in *Proc. 24th Int. Conf. Data Eng.*, 2008, pp. 1072–1081. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2008.4497516

[4] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB J.*, vol. 10, no. 4, pp. 334–350, 2001. [Online]. Available: http://dx.doi.org/10.1007/s007780100057

[5] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava, "Text joins in an RDBMS for web data integration," in *Proc. 12th Int. World Wide Web Conf.*, 2003, pp. 90–101. [Online]. Available: http://doi.acm.org/10.1145/775152.775166

[6] T. Pedersen, S. Patwardhan, and J. Michelizzi, "WordNet:: Similarity-measuring the relatedness of concepts," in *Proc. 19th Nat. Conf. Artif. Intell. 16th Conf. Innovative Appl. Artif. Intell.*, 2004, pp. 1024–1025. [Online]. Available: http://www.aaai.org/Library/AAAI/2004/aaai04–160.php

[7] J. Kang and J. F. Naughton, "On schema matching with opaque column names and data values," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2003, pp. 205–216. [Online]. Available: http://doi.acm.org/10.1145/872757.872783

[8] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proc. 11th Int. Conf. Data Eng.*, 1995, pp. 3–14. [Online]. Available: http://dx.doi.org/10.1109/ICDE.1995.380415

[9] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering frequent episodes in sequences," in *Proc. 1st Int. Conf. Knowl. Discovery Data Mining*, 1995, pp. 210–215. [Online]. Available: http://www.aaai.org/Library/KDD/1995/kdd95–024.php

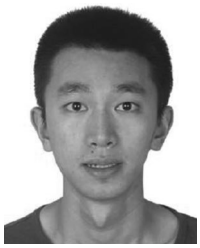
[10] C. Bettini, X. S. Wang, S. Jajodia, and J. Lin, "Discovering frequent event patterns with multiple granularities in time sequences," *IEEE Trans. Knowl. Data Eng.*, vol. 10, no. 2, pp. 222–237, Mar./Apr. 1998. [Online]. Available: http://dx.doi.org/10.1109/69.683754

[11] X. Zhu, S. Song, J. Wang, P. S. Yu, and J. Sun, "Matching heterogeneous events with patterns," in *Proc. IEEE 30th Int. Conf. Data Eng.*, 2014, pp. 376–387. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2014.6816666

[12] H. W. Kuhn, "The hungarian method for the assignment problem," *50 Years Integer Program. 1958-2008-From Early Years State-of-the-Art*, pp. 29–47, 2010. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68279-0_2

[13] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2006, pp. 407–418. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142520
[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.
[15] S. J. Russell and P. Norvig, *Artificial Intelligence-A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Pearson Education, 2010. [Online]. Available: http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-01360425% 97,00.html
[16] S. Nejati, M. Sabetzadeh, M. Chechik, S. M. Easterbrook, and P. Zave, "Matching and merging of statecharts specifications," in *Proc. 29th Int. Conf. Softw. Eng.*, 2007, pp. 54–64. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2007.50
[17] J. Wang, T. Jin, R. K. Wong, and L. Wen, "Querying business process model repositories-A survey of current approaches and issues," *World Wide Web*, vol. 17, no. 3, pp. 427–454, 2014. [Online]. Available: http://dx.doi.org/10.1007/s11280–013-0210-z
[18] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst, "Seven process modeling guidelines (7PMG)," *Inf. Softw. Technol.*, vol. 52, no. 2, pp. 127–136, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2009.08.004
[19] J. Wang, S. Song, X. Zhu, and X. Lin, "Efficient recovery of missing events," *Proc. VLDB Endowment*, vol. 6, no. 10, pp. 841–852, 2013. [Online]. Available: http://www.vldb.org/pvldb/vol6/p841-song.pdf
[20] J. Wang, S. Song, X. Zhu, X. Lin, and J. Sun, "Efficient recovery of missing events," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 11, pp. 2943–2957, Nov. 2016. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2016.2594785
[21] J. Madhavan, P. A. Bernstein, and E. Rahm, "Generic schema matching with cupid," in *Proc. 27th Int. Conf. Very Large Data Bases*, 2001, pp. 49–58. [Online]. Available: http://www.vldb.org/conf/2001/P049.pdf
[22] G. Jeh and J. Widom, "SimRank: A measure of structural-context similarity," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2002, pp. 538–543. [Online]. Available: http://doi.acm.org/10.1145/775047.775126
[23] X. Dong, A. Y. Halevy, and J. Madhavan, "Reference reconciliation in complex information spaces," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2005, pp. 85–96. [Online]. Available: http://doi.acm.org/10.1145/1066157.1066168
[24] J. Wang, S. Song, X. Lin, X. Zhu, and J. Pei, "Cleaning structured event logs: A graph repair approach," in *Proc. 31st IEEE Int. Conf. Data Eng.*, 2015, pp. 30–41. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2015.7113270
[25] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining process models from workflow logs," in *Proc. 6th Int. Conf. Extending Database Technol. Advances Database Technol.*, 1998, pp. 469–483. [Online]. Available: http://dx.doi.org/10.1007/BFb0101003
[26] J. E. Cook and A. L. Wolf, "Event-base detection of concurrency," in *Proc. ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 1998, pp. 35–45. [Online]. Available: http://doi.acm.org/10.1145/288195.288214
[27] D. R. Ferreira and D. Gillblad, "Discovering process models from unlabelled event logs," in *Proc. 7th Int. Conf. Bus. Process Manage.*, 2009, pp. 143–158. [Online]. Available: http://dx.doi.org/10.1007/978–3-642-03848-8_11
[28] D. Luckham, "The power of events: An introduction to complex event processing in distributed enterprise systems," in *Proc. Int. Symp. Rule Representation Interchange Reasoning Web*, 2008, Art. no. 3. [Online]. Available: http://dx.doi.org/10.1007/978–3-540-88808-6_2

Shaoxu Song is an associate professor in the School of Software, Tsinghua University, Beijing, China. His research interests include data quality and complex event processing. He has published more than 20 papers in top conferences and journals such as SIGMOD, VLDB, ICDE, etc.



Yu Gao is working toward the ME degree in the School of Software, Tsinghua University, Beijing, China. His current research interests include event data quality and cleaning.

Chaokun Wang is an associate professor in the School of Software, Tsinghua University. He has published more than 60 refereed papers, received two best paper awards at international conferences and held 12 patents. His current research interests include social network analysis, graph data management, and music computing.

Xiaochen Zhu is working toward the PhD degree in the School of Software, Tsinghua University, Beijing, China. His current research interests include event data management and schema matching.

Jianmin Wang is a professor in the School of Software, Tsinghua University. His current research interests include unstructured data management, workflow and BPM technology, benchmark for database system, information system security, and large-scale data analytics.

Philip S. Yu's main research interests include data mining (especially on graph/network mining), social network, privacy preserving data publishing, data stream, database systems, and Internet applications and technologies. He is a professor in the Department of Computer Science, UIC and also holds the Wexler chair in Information and Technology.