Log-based Process Fragment Querying to Support Process Design

Karn Yongsiriwit
Telecom SudParis
UMR 5157 Samovar, France

Nguyen Ngoc Chan
Université de Lorraine
UMR 7503 Loria, France

Walid Gaaloul
Telecom SudParis
UMR 5157 Samovar, France

Abstract

In recent years, many approaches have been proposed to support business process design, for instance, by providing reference models, retrieving similar business processes or querying process fragments. However, these approaches are still labor-intensive, error-prone and time-consuming. Moreover, they have not yet fully exploited process event logs, which contain useful information about the real execution of business processes. In this paper, we present an innovative approach that extracts information from event logs to develop a useful tool to support the process design. Concretely, we extract the execution order of activities to build a neighborhood context for each activity. We match both activities’ labels and their neighborhood contexts to compute the similarity between them. Finally, we propose a query language as a practical tool that allows process designers to query activities and their involved log-based process fragments based on the computed similarity. We developed an application to validate our approach as a proof of concept. We also performed experiments on a large public dataset and experimental results show that our approach is feasible and efficient.

1 Introduction

Process design is a crucial phase in the life-cycle of business process management [27]. It enables companies to sketch out their business process, overview the execution process, plan resources, identify new opportunities and foresee risks. In addition, it assists companies to flexibly adapt their business to different markets by developing process variants. Many approaches have been proposed to support this phase such as developing reference models [25, 12], computing similarity between business processes [33, 28, 16] or querying business processes [24, 6, 14, 3]. Existing approaches, however, mainly study conceptual process models and take into account entire business process topology. They have not yet examined useful information hidden in process event logs.

On the one hand, conceptual process models do not always exist in large-scale information systems such as ERP, CRM, or workflow management systems [30], even when process logs are available. They often describe business processes in a coarse-granular way without requiring any technical detail of activities [4]. Moreover, they do not explicitly show the importance of activities and execution orders, which is essential for identifying important business execution paths. This information also presents more precisely the behavior of activities in term of their relation to neighbors.

On the other hand, dealing with entire business process topology is very time-consuming. In some cases [33, 16], a compromise between the computational complexity and the quality of results needs to be found. In addition, large-scale, complex process models are not handy for a designer who is looking for pick a specific piece of functionality from them to integrate into their designed process. In this context, it is desirable to focus on process fragments forming a small but fine-grained, well-selected set of activities.

In this paper, we present an approach that allows process designers to query executable process fragments that are similar to the process fragment surrounding a selected activity on a designed process. We examine activity execution orders and their frequency recorded in process event logs instead of using conceptual process models. We accumulate activity execution orders to create a so-called log-based process model. We define the notion of a neighborhood context of an activity as a fragment of the log-based process model that contains the considered activity and relations to its neighbors. We compute the similarity between activities based on the semantic matching of their labels and their neighborhood contexts. Finally, we propose a query language to assist process designers to find similar process fragments.

As activities are frequently labeled according to their functionality and neighborhood contexts expose real behavior (in term of relation to neighbors) of the centric activities, our approach is able to retrieve activities that are similar to a selected activity on both functionality and behavior. By exploiting process event logs, focusing on process fragments
and providing a query language, our objective is threefold: (i) to discover and utilize valuable knowledge that is hidden in process event logs, (ii) to retrieve fine-grained, well-selected sets of activities for process design and (iii) to provide a useful tool to support the development of business processes.

Our paper is organized as follows: the next section presents an example to motivate our approach. We present some basic concepts in section 3 and detail our proposed query language in section 4. We elaborate the similarity computation in section 5. Implementation and experiments are presented in section 6. Finally, we present related work in section 7, and conclude our work in section 8.

2 Motivating Example

We use a simple loan process of a financial institute which was presented in [8] (Figure. 1) to motivate our approach. This process is designed as follows: when a customer submits a loan request, activity A sends a confirmation e-mail to the applicant. Next, three activities B, C, D are executed in parallel: B checks the customer’s credit history with a third control agency. C computes the customer’s loan capacity, and D checks whether the customer is already registered in the bank. Thereafter, a decision is made as either accepted (activity E) or rejected (activity F). Finally, a notification is sent to the customer by activity G.

Consider a process designer who is about to develop a new variant of this process to be adopted in different environments or cross-organizations. He wants to find activities that are similar to C, i.e. which are able to compute customer’s loan capacity, and their possible executions, i.e. which activities are executed before and after them.

We suppose that there is a process event logs repository that records the relation between activities based on their execution order and frequency. Table 1 presents typically 3 process event logs of 3 other related loan application processes. Each process instance is recorded as a trace. Each trace is a sequence of activities, which presents their execution order within a process instance. In a trace, the following activity (activity on the right) is performed after the followed activity (activity on the left). For example, consider the trace $\sigma = ABCDEG$, activity B is performed after A and before C.

Fig. 2 elaborates steps to query similar log-based process fragments. Firstly, we construct the neighborhood context of C based on the logs of the simple loan process (Table 1a). This neighborhood context is presented as a graph, in which nodes represent the activities with their unique identifiers and activities’ names (labels) and directed edges present the execution order. Edge weights depict their repetition in the logs. Then, we match this neighborhood context with contexts of other activities constructed by event logs of other processes (Table 1b, 1c) to compute their similarity. We combine this similarity with the semantic similarity between activities labels to compute relevant activities. As a result, C2 and C3 together with their involved neighborhood context are selected for the designer’s request.

Process event logs are adequate as they reflect more precisely to the behavior of a process. By taking into account logs, our approach is very useful as we can retrieve relevant business process fragments without requiring any information about a-priori conceptual process models. In addition, the semantic similarity between activity labels enable the retrieval of activities that have similar functionality. These query results would assist the designer to develop new process variant.

3 Preliminaries

In this section, we present basic definitions that are used in our approach. They include definitions about logs and traces (section 3.1), log-based business process (section 3.2) and log-based neighborhood context (section 3.3)
3.2 Log-based Business Process

The sequence of activities in a log trace \( \sigma = a_1a_2 \ldots a_n \in A^* \) presents their ordering relations. A relation between an activity \( a_i \) and its followed activity \( a_{i+1} \) in the trace \( \sigma \), \( 1 \leq i \leq n-1 \) can be presented as a directed edge from \( a_i \) to \( a_{i+1} \). The activity relations in a business process log \( L \) can be presented as a weighted directed graph where the edge’s weight presents the number of times that the edge was repeated in the log \( L \). This graph is called log-based business process graph (Definition 3.3).

Definition 3.3 (Log-based business process graph) A log business process graph is a weighted directed graph \( G_L = (V_L, E_L, w) \) built from a business process log \( L \in (P(A^*), m) \) where:

- \( V_L = A = \{a_1, a_2, \ldots, a_n\} \).
- \( E_L = \{(a_i, a_j) \in A \times A : a_i > L a_j \} \subseteq A \times A \).
- \( w \) is a weight function from \( E_L \) to \( N \):
  \[
  w : (a_i, a_j) \rightarrow |a_i > L a_j|
  \]
  \( |a_i > L a_j \) is number of times that \( a_i > L a_j \) comes about in the log \( L \).
  \( w(a_i, a_j) = 0 \) if \( \exists \sigma = a_1a_2 \ldots a_n \), \( k \in \{1, 2, \ldots, n-1\} : a_k = a_i \land a_{k+1} = a_j \)

3.3 Log-based Neighborhood Context

We define the log-based neighborhood context as a directed labeled graph that presents the shortest path from an activity to its neighbors. Intuitively, the closeness between activities is presented by the paths connecting them. The shortest path between activities presents their closest relation. The log-based neighborhood context of an activity presents the best relations between the activity and its neighbors.

In a log-based neighborhood context graph, each vertex is associated to a number that indicates the shortest path length from it to the associated activity. Vertices that have the same shortest path length are considered to be located on the same layer around the associated activity. Thus, we name the number associated to each activity in a neighborhood context graph layer number. The layer number of an activity \( a \) is denoted by \( l(a) \). The area limited between two adjacent layers is called zone. The edge connecting two vertices in a neighborhood context graph belongs to a zone as the vertexes are on the same or adjacent layers. We assign to each edge a number, so-call zone number, which determines the zone that the edge belongs to.

The edge connecting \( a_j, a_k \) in the neighborhood context graph of an activity \( a_i \) is assigned a zone number \( z(a_j, a_k) = \)

---

Table 1: Log-based neighborhood context graphs

<table>
<thead>
<tr>
<th>Query neighborhood context</th>
<th>Process event logs repository</th>
<th>3. Retrieving similar log-based neighborhood context</th>
</tr>
</thead>
</table>

1. Constructing

2. Neighborhood context matching with other activities

Figure 2: Query to find activities having similar log-based neighborhood context to C

### 3.1 Business Process Log

**Definition 3.1 (Log trace, business process log, \( L \))** Let \( A \) be a non-empty set of activities, each activity has a set of attributes (\( \text{attr}(a) \)). \#id(a) refers to the value of the attribute id for the activity \( a \in A \). We require that each activity has at least the following two attributes:

- \#id(a) refers to the unique identifier of the activity.
- \#label(a) refers to the label of the activity.

\( A^* \) denotes the set of finite sequences over \( A \) and \( \sigma = a_1a_2 \ldots a_n \in A^* \) is a log trace. \( L \in (P(A^*), m) \) is a business process log.

For example, assume that event logs of the loan process and another process are reported in Table 1. The first column present the traces, which is a process instances recored in the log (subset of \( P(A^*) \) Definition 3.1). The second column presents the repetition of traces (\( m \) in Definition 3.1).

**Definition 3.2 (Log-based ordering relation, \( >_L \))** Let \( L \) be a business process log over \( A \), i.e., \( L \in P(A^*) \). Let \( a, b \in A \). \( a >_L b \) if \( \exists \sigma = a_1a_2 \ldots a_n, i \in \{1, 2, \ldots, n-1\} : \sigma \in L \land a_i = a \land a_{i+1} = b \).

For example, from the logs given in Table 1, we have \( A >_L B, B >_L C, C >_L D, D >_L E \), and so on.
min(l(a_j), l(a_k)) + 1. This means, if a_j and a_k are located on two adjacent layers, the edge (a_j, a_k) will belong to the zone limited by l(a_j) and l(a_k). In the case that a_j and a_k are located on the same layer, the edge connecting them belongs to the outer zone of their layer, which is limited by layers l(a_j) and l(a_k) + 1.

**Definition 3.4 (Activity neighborhood context graph)**

The neighborhood context graph of an activity a_i, denoted by Gc(a_i), is an extension of G_L = (V_L, E_L, w) with vertex layer numbers and edge zone numbers. The layer number of an vertex a_j, denoted by l(a_j)_{Gc(a_i)}, is the shortest path length from a_j to a_i, and the zone number of an edge (a_j, a_k), denoted by z(a_j, a_k)_{Gc(a_i)}, has value \(\min(l(a_j)_{Gc(a_i)}, l(a_k)_{Gc(a_i)}) + 1\):

1. \(l(a_j)_{Gc(a_i)} = \text{ShortestPathLength}(a_j, a_i)\).
2. \(z(a_j, a_k)_{Gc(a_i)} = \min(l(a_j)_{Gc(a_i)}, l(a_k)_{Gc(a_i)}) + 1, a_j \geq_L a_k \quad a_k \geq_L a_j\).

For example, the neighborhood context graphs of C and C2 according to the event logs presented in Table 1 are depicted in Fig. 3. In the neighborhood context graphs of C, F belongs to the 1st layer \((l(F)_{Gc(C)} = 1)\), while G belongs to the 2nd layer \((l(G)_{Gc(C)} = 2)\). The edge from C to F belongs to the 1st zone \((z(C, F)_{Gc(C)} = 1)\), while the edge from F to G belongs to the 2nd zone \((z(F, G)_{Gc(C)} = 2)\) and so on.

## 4 Querying Process Fragments

The query in our approach does not only help to search for relevant activities but also allows process designers to specify constraints over the requested context in order to filter the searching results. It consists of three parameters, which are: associated activity identifier, constraint, and radius. The associated activity is the event whose neighborhood context is taken into account to match with other contexts. The constraints are used to filter the query results according to the user’s needs. The radius is the number of connection layers taken into account for the neighborhood context matching.

We present in the following the grammar of our proposed query (section 4.1) and steps to achieve querying results (section 4.2).

### 4.1 Query Grammar

We present our proposed query’s grammar using the Extended Backus-Naur Form [32]. The context-free grammar are illustrated in Table 2. We reuse EBNF notations in our query grammar, for example, comma (,) is used for concatenation; semicolon (;) is used for termination; vertical bar (|) is used for alternation; [...] is used for option; (...) is used for grouping; [...] is used for repetition and ‘...’ is used for terminal string.

The grammar is explained as follows.

- **Line 1**: the query is defined with three parameters separated by ‘.’. The constraint is optional. It can be defined to filter the query result.
- **Line 2**: the activity identifier is defined as a string of characters or digits.
- **Line 3**: we define the constraint. A constraint can be included/excluded activities, data conditions (see lines 6-9) or neighborhood context conditions (see line 10). It can also include different items and operators.
• Line 4 defines a ‘Term’ as an ‘Item’ or another constraint with the ‘+’ and ‘−’ operators. (e.g. +B +D in the query example)

• Line 5 defines ‘Item’ as an activity id, a data condition or a neighborhood context condition that is included/excluded in the query.

• Line 6 defines the rule for data conditions which is presented within ‘[’ and ‘]’ signs. It includes a string of an activity and its data constraints.

• Line 7, ‘DataConss’ is defined to allow single or multiple data conditions over an activity. (e.g. B: ‘creditApproved’ = ‘true’ in the query example)

• Line 8, 9, we define two types of data constraints and the property identifier of the data associated to the activity is defined as a string of characters or digits within the single quotation symbol.

• Line 10, we define the rule of neighborhood context conditions which is presented within ‘[’ and ‘]’ signs. It includes a string of an operator that describe the neighborhood context of two events.

• Lines 11 to 17: basic operators and elements are defined.

For example, suppose that the process designer wants to find activities that are similar to C (see our motivating example in section 2), excludes A, E, G and includes B, D, F in their context such that the credit and loan capacity are approved and the customer has not yet registered in the system. The designer can specify the query as follows.

‘calculate capacity’: = ‘send e-mail’ + ‘check credit’ + ‘check system’ = ‘accept’ + ‘reject’ = ‘send notification’ + [‘check credit’: ‘creditApproved’ = ‘true’] + [‘calculate capacity’: ‘capacityApproved’ = ‘true’] + [‘check system’: ‘isRegistered’ = ‘false’]; 3

Fig. 2 shows possible results of this query (see section 2). Other examples of our query are given in Table 3.

4.2 Querying Process

On using our querying approach, the process designer needs to specify required parameters complying the query grammar. Once obtaining a query request, we perform the following steps to return the results.

• Step 1: we capture the log-based neighborhood context of the associated activity. This neighborhood context is identified by the associated activity and connection flows to its neighbors (see section 3.3).

• Step 2: we match the log-based neighborhood context of the associated activity to the contexts of other activities in other business processes (see section 5). During the matching, we compute the semantic similarity between activities’ labels if their unique identifiers are not identical. We restrict the number of activities by selecting only activities whose log-based neighborhood contexts satisfy the query’s constraints.

• Step 3: we sort the selected activities based on the matching values and pick up top-N activities.

• Step 4: we return the top-N activities together with the log-based business process fragments around them. The largeness of these fragments is specified by the radius parameter.

In the next section, we present in details our approach to compute the similarity between activities based on their log-based neighborhood contexts and labels.

5 Neighborhood Context Matching based on similar activity labels

5.1 Zone Matching

This section introduces our neighborhood context matching based on similar activity labels. We apply the vector space model (VSM) to compute the matching on edges in each zone of two log-based neighborhood context graphs. VSM is a common technique used in Information Retrieval to compute the similarity between two items. It presents items in vectors and compute their similarity based on the cosine of the angle between the two corresponding vectors. In our approach, we present each zone as a vector who elements are edges and values are their corresponding weights. Then, we align elements connecting the same activities in the same layers. We present these vectors in the same space (the unaligned element have zero value). We consider Finally, we compute the cosine value of these two zone-vectors, and we calculate the semantic similarity between activities’ labels if their unique identifiers are not identical.

Particularly, in the first zone, we match the edges that connect the two associated activities to the same activities in the first layer. We define the two associated activities as root activities and name them r0.

Concretely, assume that Pr and Pq are two log-based business processes constructed from event logs Lr and Lq. Let Ar, Aq be sets of activities of Pr and Pq. We compute the similarity between activities a ∈ Ar and b ∈ Aq by applying VSM as following.

Let Ek−z(a) and Ek−z(b) be the sets of edges in kth-zone of a ∈ Pr and b ∈ Pq. Let e(a), e(b) be vectors of weights of
these edges.

\[ E_k^1(a) = \{(x,y) : z(x,y) = k, x, y \in A_p\} \]
\[ = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\} \]
\[ \bar{e}(a) = (w(x_1, y_1), w(x_2, y_2), \ldots, w(x_m, y_m)) \]
\[ E_k^1(b) = \{(e, f) : z(e, f) = k, e, f \in A_q\} \]
\[ = \{(e_1, f_1), (e_2, f_2), \ldots, (e_n, f_n)\} \]
\[ \bar{e}(b) = (w(e_1, f_1), w(e_2, f_2), \ldots, w(e_n, f_n)) \]

Let \( N_k(a, b) \) be the set of the most similar activity mapping as tuples of the activities pair with their similarity value \((a_i, b_j, sim(a_i, b_j))\) of a and b in the \( k^{th} \) layer where \( a_i \) is a neighbor activity of a on layer \( k \), \( b_j \) is a neighbor of b on layer \( k \), \( k > 0 \). We will explain in more details how we create this mapping using the sim function in the following section 5.2.

As we define the two associated activities as root activities and name them \( r_0 \), we have: \( N_k^0(a) = a = r_0 \), \( N_k^0(b) = b = r_0 \) and \( N_k^0(a, b) = r_0 \)

Let \( E_k^1 \) be the set of common edges of a and b in \( k^{th} \)-zone.

\[ E_k^1 = \{(r, t) : r \in N_{k-1}^1(a, b), t \in N_k^1(a, b) | r > l_a, t \land r > l_b, t \}
\cup \{(r, t) : r \in N_{k-1}^1(a, b) | r > l_a, t \land r > l_b, t \}

Let \( \bar{e}(a) \), \( \bar{e}(b) \) be vectors of weights of these common edges.

\[ \bar{e}(a) = (w(r_1, t_1), w(r_2, t_2), \ldots, w(r_z, t_z)) \in E_k(A_p), \quad 1 \leq i \leq z = (a_1, a_2, \ldots, a_z) \]
\[ \bar{e}(b) = (w(r_1, t_1), w(r_2, t_2), \ldots, w(r_z, t_z)) \in E_k(A_q), \quad 1 \leq i \leq z = (b_1, b_2, \ldots, a_z) \]

By applying VSM, the similarity between activities \( a \) and \( b \) in the neighborhood context of the \( k^{th} \)-zone is given by Equation 1 where \( r_a, r_b \) are root activities, \( t_a, t_b \) are target activities of common edges between \( \bar{e}(a) \) and \( \bar{e}(b) \).

\[ M_{\text{sim}}(a, b) = \sum_{i=1}^{z} \frac{\text{sim}(r_a, r_i) \times \text{sim}(t_a, t_i) \times w(r_a, t_a) \times w(r_b, t_b)}{||e(a)|| \times ||e(b)||} \tag{1} \]

### 5.2 Activities’ Labels Matching

Function \( \text{sim} \) computes the similarity of two labels. We use Stanford Part-of-Speech (POS) [19, 26] for stemming string and removing function words. We modify the bag-of-words similarity with label pruning technique [20] to prune words from the longer label then measure the similarity of two labels based on pruned words.

The similar label matching between label \( l_1 \) and \( l_2 \) is defined as follows:

\[ \text{sim}(l_1, l_2) = \frac{\sum_{i=1}^{w_1} \text{max}_{\omega} \text{sim}_w(\text{pr}_1(\omega), \text{pr}_2(\omega)) + \sum_{j=1}^{w_2} \text{max}_{\omega} \text{sim}_w(\text{pr}_1(\omega), \text{pr}_2(\omega))}{2 \times \min(w_1, w_2)} \tag{2} \]

Let \( l_1 \) and \( l_2 \) be the labels of the activities \( a_1 (\text{#label}(a_1)) \) and \( a_2 (\text{#label}(a_2)) \), and \( \omega_1 = (l_1), \omega_2 = (l_2) \) are tokenized lists of words contained in the labels by POS technique. Further, \( pr_1 = \text{pru}(\omega_1, \omega_2) \) and \( pr_2 = \text{pru}(\omega_2, \omega_1) \) are the pruned list of words. Function \( \text{sim}_w \) computes the similarity between two words using existing approaches such as Lin metric ontology matching technique [22]. We define a threshold \( \beta_1 \) for \( \text{sim} \). Two labels are considered to be functionally similar iff \( \text{sim}(l_1, l_2) \geq \beta_1 \). Let \( \text{pru} : P(W) \times P(W) \rightarrow P(W) \) be a generic function. It returns a set of words extracted from its input. \( \text{pru}(l_1, l_2) \) is \( \omega_1 \) iff \( |\omega_1| \leq |\omega_2| \), or a subset of \( \omega_1 \) of size \( |\omega_2| \) otherwise. The similarity scores of all word pairs, as well as the maximum score for each word are calculated in \( |\omega_1| \). \( \text{pru} \) returns the \( |\omega_2| \)-top-scoring words from \( \omega_1 \).

In order to illustrate our approach, we present the similarities between labels of the activities in the \( 1^{st} - z \)-zone of the neighborhood context of \( C \) and \( C' \) using \( \text{sim} \) (see our motivating example). We define the threshold as \( \beta_1 = 0.5 \). The result after descending sort are depicted in Table 4. Due
to the space restriction, we do not add full detail of computation here. we refer to [1] for the more example of computation.

We create the matching between the most similar activities by the ranking the similarity values. For example, B2 is matched to B with \( \text{sim}(B2, B) = 1.000 \) and D2 is matching to D with \( \text{sim}(D2, D) = 0.543 \).

\[
N^1(C, C2) = \{(B2, B, 1.000), (D2, D, 0.543)\}
\]

Using \( \text{sim} \), the log-based neighborhood context matching is described in Equation 3. We weight the similar pair of activities based on their labels similarity.

\[
M^e_{\text{sim}}(a, b) = \sum_{\langle i, j \rangle \in c} \text{sim}(l_{i}, l_{j}) \times \text{sim}(l_{i}, l_{j}) \times w(r_{i}, r_{j}) \times w(r_{i}, r_{j})
\]

\[
M^e_{\text{sim}}(a, b) = \frac{\sum_{\langle i, j \rangle \in c} \text{sim}(l_{i}, l_{j}) \times \text{sim}(l_{i}, l_{j}) \times w(r_{i}, r_{j}) \times w(r_{i}, r_{j})}{|e(a)| \times |e(b)|}
\]  

(3)

For example, using the label similarities from Table 4, we have \( N^1(C, C2) = \{(B2, B, 1.000), (D2, D, 0.543)\} \). So, \( \bar{e}_C(C) = (w(B, C), w(C, D)) = (54, 46) \), \( \bar{e}_C(C2) = ((B2, C2), (C2, D2)) = (70, 70) \) and their matching in the 1\( ^{st} \) -zone is:

\[
M^e_{\text{sim}}(C, C2) = \frac{1 \times 1 \times 54 \times 70 + 1 \times 0.543 \times 46 \times 70}{\sqrt{4^2 + 6^2 + 54^2 + 46^2 + 42^2 + 13^2 + 33^2} \times \sqrt{70^2 + 70^2}} \approx 0.615
\]

Table 5 shows the similarity values between activity C and other activities in process variant 2 with \( k^{th} \)-zone values varied from 1 to 5.

### 6 Validation

#### 6.1 Implementation

We implemented a log-based process fragment querying plug-in and integrated it into ProM. This plug-in interacts with database to retrieve recommendations based on the query in the ProM interface. This application was developed to validate our approach as a proof of concept.

The screen-shot of our application is shown in Figure 4. It consists of 4 areas\(^2\). Area 1 allows selecting a working process from a selected repository. The designer can enter the query in area 2. Area 3 shows the log-based neighborhood context of the query (the selected activity from the query is highlighted). Area 4 shows Top-10 similar activities to the log-based neighborhood context of the query. The designer may select an activity in area 4 in order to view its log-based neighborhood context in area 5 (the selected activity is highlighted).

#### 6.2 Dataset

To evaluate the scalability of our approach, we have occasionally used simulated logs. Indeed, getting real logs from big size workflow examples that are enough various turns out to be a difficult task. The advantage in using simulated logs is that it is easier to fix and vary external factors ensuring a better diversity of the examples and a better and more accurate validation. The scaling issue of our validation test is consequently better dealt with simulated logs that enable us to cover a qualitatively and quantitatively various set.

In order to do this, we use a log simulating tool [13] which creates random XML logs by simulating already designed workflow processes based on CPN tools\(^3\). These tools support the modeling, the execution and the analysis of colored Petri nets [18]. They enable to create simulated

---

\(^2\)Due to the lack of space, more details about our tool can be found at: http://www.inf-it-sudparis.eu/SIMBAD/tools/eventLogsQuery/

\(^3\)http://wiki.daimi.au.dk/cpntools/cpntools.wiki
logs conforming with the XML structure proposed in [29]. Modifications were brought to these tools to call predefined functions that create logs for each executed workflow instance. This stage implies modifications in the modelling level of CPN workflow declarations, particularly in the actions and the transition input/output levels. These functions indicate in particular the place, the prefix and the extension of the XML files that CPN tools create for each executed workflow instance. Thereafter, we have used ProMimport\(^4\) to group or gather the simulated workflow logs.

We performed experiments on synthesized logs. The logs were generated from a large public dataset [23] which consists of 186 different SAP reference models in EPC Markup Language (EPML). We transformed the collected business process models from EPML format to Colored Petri Nets (CPN) format. Then, we simulated the execution of these business process models using CPN Tools application. Consequently, process event logs in XES files were generated\(^5\). Table 6 gives more details about the generated logs.

### 6.3 Experimental results

We evaluate the feasibility and the accuracy of our approach by performing two experiments. In the first experiment, we vary the \(k\)th-zone values from 1 to 5 and make statistics on the number of recommended activities for each selected activity. Figure 5 shows the percentage of activities that have at least 1 recommended activity with the similarity value greater than 0, 0.5 and 0.8.

In the second experiment, we evaluate the accuracy of our approach based on Precision and Recall metrics. We consider activity identifiers as ground truth data in computing Precision and Recall. Concretely, consider a selected activity \(a\) in a log-based business process \(P\). Assume that \(a\) appears in \(n\) log-based business processes. The recommendations for this selected position consist of \(l\) activities, in which \(t(\lt l)\) activities are \(a\). Precision and Recall of these recommendations are:

\[
\text{Precision} = \frac{t}{l}; \quad \text{Recall} = \frac{t}{n};
\]

The primary objective of the experiment is to retrieve a small share of activities that are likely irrelevant (high precision). It is of secondary importance to retrieve the full range of potentially relevant activities (moderate recall) in order to avoid the designer being overwhelmed.

In our experiment, we compute the Precision and Recall values with \(N\) (the number of activities recommended for selected activity). We performed the experiment with \(k\)th-zone equals to 1 and on activities that appear in at least 2, 5, 7 and 10 different business processes.

The results (see Figure 6) show that we obtained good Precision values (from 0.61 to 0.68) in case of recommending 1 activity for each selected activity. These values decrease when we increase the number of recommended activities. On the other hand, activities that appear in more processes will have greater Precision values. It also shows that the Recall values increase when \(N\) changes from 1 to 10. This means that we can retrieve more relevant activities when the number of recommended activities increase. The highest Recall value in our experiment is 0.484 when \(N\) equals to 4.

Figure 7 shows the average Precision and Recall values of our approach with different \(k\)th-zone values. It shows that our approach achieved much better results than an approach that generates recommendations randomly (in average, 114.4 times greater than Precision value and 44.6 times

<table>
<thead>
<tr>
<th>Table 6: Dataset details</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>No. of process instances</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>No. of events in process instances</td>
</tr>
<tr>
<td>No. of events in an instance</td>
</tr>
<tr>
<td>No. of start events in an instance</td>
</tr>
<tr>
<td>No. of end events in an instance</td>
</tr>
</tbody>
</table>

\(^4\)www.promimport.sourceforge.net

\(^5\)Our dataset in EPML, CPN and XES format are published at: http://www-inf.it-sudparis.eu/SIMBAD/tools/eventLogsQuery/dataset/
new kth-zone number

Figure 6: Precision and Recall values with different kth-zone

Figure 7: Precision and Recall values with different kth-zone

greater than the Recall value).

The Precision and Recall values showed that our approach retrieve not only the right activities but also the new relevant activities for each selected position and the number of new activities increases when the number of recommended activities increases. These results showed that our approach can be applied in real use-cases as we can provide the designer the flexibility in using existing process fragments or designing new process variants.

7 Related Work

Many process querying approaches have been proposed to assist the business process designer. For example, Balan et. al. [5] have proposed a tool suite based on the BPEL standard, to offer a uniform, query-based, user-friendly interface for BP analysis. This tool allows gracefully combining the analysis of process specifications, monitoring of runtime behavior, and posteriorly querying of execution traces (logs), for a comprehensive process management.

Beheshti et. al. [7] have presented a framework, simple abstractions and a language for the exploitative querying and understanding of BP execution from various user perspectives which enable an analyst to group related events or find the correlation between events. They implemented a language, so-called FPSPARQL, by extending the SPARQL graph query language. Another querying approach has been proposed by Awad et. al. [2] with a language named BPMN-Q. It allows process designers to search for process fragments that are limited by two given activities by performing the perfect matching between activity labels. Different from these approaches, we retrieve activities that are similar to a selected activity together with their involved contexts. In addition, we compute the semantic similarity between labels instead of perfect matching.

Other approaches that aim at supporting the process design without building a query language includes business process searching [33, 16] and business process similarity measuring [28, 15, 21, 31, 17]. They provide techniques to help the designer find process models that are similar to a designed one. Some of them also provide techniques to discover process models from a collection of process event logs. In our approach, we do not study conceptual process models. We do not also take into account entire BPs. Instead, we exploit process event logs and focus on retrieving log-based business process fragments.

Klinkmüller et. al. proposed bag-of-words similarity with label pruning technique [20] for activity label matching which improves recall without sacrificing precision. Cardoso introduced an approach [9] to measure the similarity between two web service by computing the geometric distance between the similarity of the properties lists. We adopt these two approaches to create mapping between activities based on their labels.

The work present in this paper is an extension of our recent work presented in [11, 10]. In the previous work, we have introduced the activity neighborhood context. However, we did not deal with the semantic similarity between activity labels. We did not also integrate it into a query language includes business process searching [33, 16] and business process similarity measuring [28, 15, 21, 31, 17]. They provide techniques to help the designer find process models that are similar to a designed one. Some of them also provide techniques to discover process models from a collection of process event logs. In our approach, we do not study conceptual process models. We do not also take into account entire BPs. Instead, we exploit process event logs and focus on retrieving log-based business process fragments.

In the experiment, we are not able to compare quantitatively our approach to existing approaches as they barely present their tools and most of them do not provide experimental results. Therefore, we can only analyze and argue results achieved by our approach.

8 Conclusion

In this paper, we propose to assist the designer with a log based query language. Such logs capture the real behavior of processes which cannot be derived from their designed models. Most of the existing process mining tech-
niques aim to perform on a collection of event logs toward discover a conceptual process model, not to develop a new process variant. We present an approach that effectively utilizes knowledge extracted from business process logs. This approach is based on a notion of activity neighborhood and a corresponding calculation of similarity. Our approach has been implemented as two plug-ins for Prom and evaluated using real processes.

In future work, we aim to extend the similarity calculation with properties such as, actors, resources, etc. We also plan to assist the business process design with configurable process fragments by proposing an approach to mine configurable process fragments. The mined fragment is defined as the neighborhood context of an activity that shows the interaction of the activity with its context in multiple variants through configurable elements. Since a configurable fragment captures a generalized behavior, we aim also at proposing a dynamic guideline-driven approach to assist the configuration process.

References


