# Similarity Search over Personal Process Description Graph

Jing Xu, Hye-young Paik, and Liming Zhan

The University of New South Wales, Sydney, NSW, Australia
{jxux494, hpaik, zhanl}@cse.unsw.edu.au

**Abstract.** People are involved in various processes in their daily lives, such as cooking a dish, applying for a job or opening a bank account. With the advent of easy-to-use Web-based sharing platforms, many of these processes are shared as step-by-step instructions (e.g., "how-to guides" in eHow and wikiHow) on-line in natural language form. We refer to them as *personal process descriptions*. In our early work, we proposed a graph-based model named Personal Process Description Graph (PPDG) to concretely represent and query the personal process descriptions. However, in practice, it is difficult to find identical personal processes or fragments for a given query due to the free-text nature of personal process descriptions. Therefore, in this paper, we propose an idea of similarity search over the "how-to guides" based on PPDG. We introduce the concept of "similar personal processes" which defines the similarity between two PPDGs by utilizing the features of both PPDG nodes and structure. Efficient and effective algorithms to process similarity search over PPDGs are developed with novel pruning techniques following a filtering-refinement framework. We present a comprehensive experimental study over both real and synthetic datasets to demonstrate the efficiency and scalability of our techniques.

**Keywords:** How-to guides, Personal Process Description Graphs, Similarity Search

## 1 Introduction

People are engaged in all kinds of *processes* all the time, such as cooking a dish, applying for a bank account or filing a tax return. Although the expertise in the area of workflow management and business process management (BPM) [4] has produced solutions for modelling, automating and managing much of the business and organizational workflows, still significant portion of the processes that people experience daily exist outside the realm of these technologies.

However, with the advent of easy-to-use Web-based sharing platforms, people often share their experience/knowledge about a process on the Web, in the form of how-to guides or step-by-step instructions. Although these are primarily describing a process, without the modelling expertise, they are normally written in natural language as a sequence of sentences forming the process steps. In order to distinguish these texts from the conventional business workflow models, we refer to them as *personal process descriptions*. A plethora of examples on personal process descriptions can be found in cooking recipes, how-to guides or Q&A forums.

The texts in natural language format are not precise enough to be useful in utilizing the process information presented in them. For example, the state-of-the-art for search technologies over the existing personal process descriptions are still a keyword/phrase-based search and users would have to manually investigate the results. A *process-aware* technique should be able to (i) produce the overall activity structure, (ii) show the dependencies between data and actions, and (iii) compare and contrast different paths to accomplish the goal.

In our previous work [13], we proposed a simple structured query language designed to perform exact-match search over the personal process descriptions. The language is supported by a graph-based, light-weight process model called PPDG (Personal Process Description Graph) which concretely represents the personal process description texts.

In this paper, we extend our query technique to return *similar* process descriptions to a query input. This technique is particularly important because the PPDGs are obtained from texts, and being able to cope with semantic similarity in words as well as the similarity in the structure of the activities in a process can return more relevant results. Our contributions are summarized as follows:

– We formally define the *similarity over PPDG*. PPDG is a graph-based process description language that presents both control-flow and data-flow in a personal process description.
– We propose effective and efficient algorithms to perform similarity search over PPDGs.
– We further improve the performance of the algorithms by utilizing novel pruning techniques.

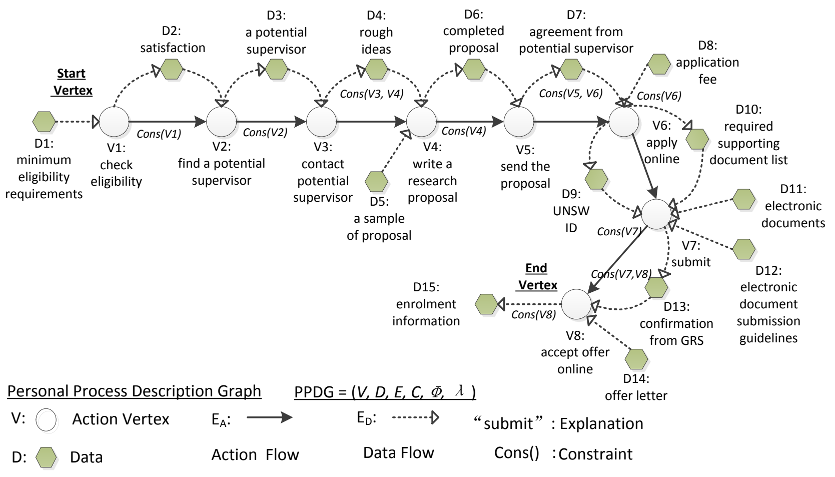
The paper is organized as follows: Sections 2 and 3 define the preliminary concepts and the problems. Section 4 describes the efficient algorithms to process the PPDG similarity search. Then we present the experiment results in Section 5. The related work is discussed in Section 6 followed by a conclusion in Section 7.

## 2 Preliminaries

In this section, we briefly introduce the personal process description graph (PPDG) and querying PPDGs. The full descriptions of these concepts are presented in [13].

### 2.1 Personal Process Description Graph

A PPDG represents a personal process description as a labeled directed graph. It describes the whole process of performing a personal process placing *equal emphasis* on both actions and input/output data relating to each action. Figure 1 depicts a PPDG of a PhD admission process experienced by an international applicant. Actions can be an one-off action, repeated action, or duration action. Data elements are represented by hexagonal nodes. A data element can be either basic or composite (i.e., composition of basic data). In order to make the



Start Vertex
D1: minimum eligibility requirements
V1: check eligibility
Cons(V1)
D2: satisfaction
V2: find a potential supervisor
Cons(V2)
D3: a potential supervisor
V3: contact potential supervisor
D4: rough ideas
Cons(V3, V4)
D5: a sample of proposal
V4: write a research proposal
Cons(V4)
D6: completed proposal
V5: send the proposal
D7: agreement from potential supervisor
Cons(V5, V6)
D8: application fee
Cons(V6)
V6: apply online
D9: UNSW ID
D10: required supporting document list
D11: electronic documents
D12: electronic document submission guidelines
Cons(V7)
V7: submit
Cons(V7,V8)
D13: confirmation from GRS
End Vertex
D15: enrolment information
Cons(V8)
V8: accept offer online
D14: offer letter
Personal Process Description Graph
PPDG = (V, D, E, C, Φ, λ)
V: Action Vertex
EA: Action Flow
ED: Data Flow
"submit" : Explanation
D: Data
Cons() : Constraint

**Fig. 1.** How to apply for PhD admission at UNSW by an international applicant

visualization of the graph simple, all types of actions are represented by using the same notation. The details are stored in the schema associated with each PPDG. The same principle applies to the data.

The data elements and actions are connected to form 'action flow' and 'data flow'. Action flows, represented by solid lines, describe temporal sequence of the actions. For example, in Figure 1, '$V_1$: *check eligibility*' takes place before '$V_2$: *find a potential supervisor*'. Data flows, represented by dotted lines, keep track of data sources and denote the relationships between the data and actions. For example, '$V_4$: write a research proposal' takes two data inputs '$D_4$: *rough idea*' and '$D_5$: *a sample of proposal*' and produces one data output '$D_6$: *completed proposal*'.

A PPDG also stores constraints/conditions relating to an action, data or the flows. For example, a condition may specify a location or the time an action takes place. We define PPDG more formally as follows.

**Definition 1.** *A personal process description graph PPDG is a tuple* $PPDG := (A, D, E_A, E_D, C, \phi, \lambda)$ *where:*

- *A is a finite set of nodes* $a_0$, $a_1$, $a_2$,... *depicting the starting action* ($a_0$) *and actions* ($a_1$, $a_2$, ...).
- *D is a finite set of nodes* $d_0$, $d_1$, $d_2$,... *depicting the data input/output of an action.*
- $E_A$ *is a finite set of directed action-flow edges* $ea_1$, $ea_2$,..., *where* $ea_i = (a_j, a_k)$ *leading from* $a_j$ *to* $a_k$ ($a_j \neq a_k$) *is an action-flow dependency. It reads* $a_j$ *takes place before* $a_k$. *Each node can only be the source/target of at most one action-flow edge :* $ea = (a_i, a_j) \in E_A : ea' = (a_k, a_l) \in E_A \setminus ea : a_i \neq a_k$ *and* $a_j \neq a_l$.
- $E_D$ *is a finite set of directed data-flow edges* $ed_1$, $ed_2$,..., *where* $ed_i = (a_j, d_k)$ *leading from* $a_j$ *to* $d_k$ *is a data-flow dependency. It reads* $a_j$ *produces* $d_k$. $ed_l = (d_m, a_n)$ *leading from* $d_m$ *to* $a_n$ *is a data-flow dependency. It reads* $a_n$ *takes* $d_m$.
- *C is a finite set of conditions* $c_1, c_2$,... *with* $c_i = (< name, descr >, x_j)$ *being associated to* $x_j \in \{A, D, E_A, E_D\}$ *and having name and description of the condition.*
- $\phi$*: a function that maps Action Label to action nodes.*
- $\lambda$*: a function that maps Data Label to data nodes.*

In this work, we have not yet considered constraints in PPDG. For simplicity, we remove $C$ (conditions) from PPDG here on.

**Definition 2 (PPDG Query Graph).** *A PPDG query graph is a tuple* $PPDG\text{-}Q = (QA, QD, QE_A, QE_D, Q_\phi, Q_\lambda, QP, \Delta)$ *where:*

- $QA$ *is a finite set of action nodes in a query.*
- $QD$ *is a finite set of data nodes in a query.*
- $QE_A \subseteq QA \times QA$ *is the action flow relation between action nodes in a query.*
- $QE_D \subseteq QA \times QD$ *is the data flow relation between action nodes and data nodes in a query.*
- $Q_\phi$*: a function that maps Action Label to action nodes.*
- $Q_\lambda$*: a function that maps Data Label to data nodes.*
- $QP$ *is the path relation between action nodes which includes data nodes and data edges corresponding to each action node in query.*
- $\Delta$*:* $QP \rightarrow \{$*true,false*$\}$

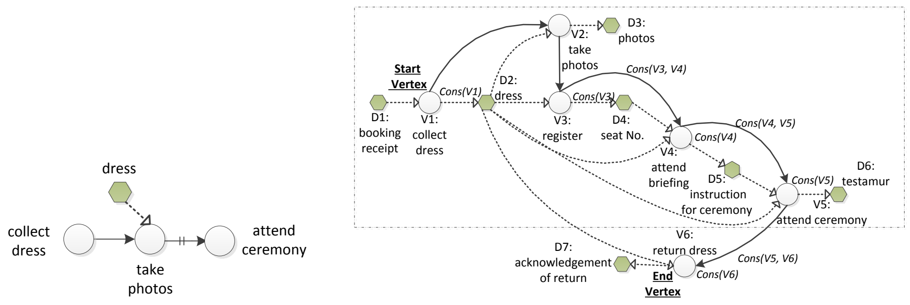


**Fig. 2.** An example of PPDG query input and output (an exact match)

Figure 2 shows an example of PPDG query input and output. A PPDG query graph consisting of "*collect dress*" action, immediately followed by "*take photos*" action with input data "*dress*", followed by a *path query* edge leading to "*attend ceremony*" action. The symbol "‖" is used to represent a *path query* between action nodes. In the above example, the path will match any action as well as connected data nodes from "*take photos*" to "*attend ceremony*". The subgraph inside dotted box in Figure 2 is the result of the query.

However, in practice, it is difficult to find out the identical personal processes by a given query graph due to natural property of personal process. The similarity search technique proposed in this paper will see the query results expanded to include subgraphs that are similar to the query graph.

## 2.2 Constructing PPDG and PPDG Repository

The PPDGs are constructed via *PPDG Builder*, a prototype system is implemented as part of a companion project which aims to establish a Web-based repository of PPDGs. The system relies on the Stanford's NLP parser and POS (Part-Of-Speech) tagger[1] to extract potential pairs of action and data (e.g., {book, academic dress}, {pay, cash}). We then build a smart *error correction*

[1] `http://nlp.stanford.edu/software/`, Stanford NLP Group

*and management* layer by using the well-known knowledge acquisition technique, RDR (Ripple Down Rules). The rules are incrementally built and managed to correct the extraction errors from the NLP parser and tagger outputs. Based on this, we are currently developing a graphical tool, *PPDG Editor*. The editor provides the graphical notations for the syntax elements in PPDG. For each personal process description, the users will see suggested action/data listing from the PPDG Builder. Although we do not assume that the PPDG construction process can be totally automated, our aim is to automate the mapping of the suggested action/data to PPDG nodes and labels as much as possible over time.

## 3 Problem Definition

The query techniques proposed in our earlier work implemented an exact match between a query graph and PPDGs. In this paper, we investigate the problem of similarity search over PPDGs. This is an important extension to PPDG queries because the construction process of PPDGs from the text-based sources generates nodes and labels that are semantically 'similar', but appear different. In similarity search, for a given PPDG query $Q$, our system returns similar PPDGs ranked by similarity scores from the PPDG repository. Without loss of generality, we assume the size of $Q$ is not larger than that of any PPDG $P \in \mathcal{P}$. We define the similarity between $Q$ and $P$, denoted as $Sim(Q, P)$, in this section.

To get the similarity score $Sim(Q, P)$, we separately consider the nodes (both action and data), and the directed edges linking the nodes[2]. Hence, we compute two separate similarity scores: one based on the labels of the nodes, the other on the edges from/to the nodes. We then combine the two to obtain $Sim(P, Q)$.

We note that, different from a PPDG, a PPDG query $Q$ may include a *path* between two action nodes. If a *path* exists in $Q$, that is $\Delta = true$, we consider the *path* as a special action edge and $Q$ as a special PPDG. $Q$ with *path* is special case and we will discuss it in Section 4. For the sake of simplicity, from here on, let us treat $Q$ as a PPDG. Now we formally define the similarity between two PPDGs $P$ and $P'$ as follows.

***Graph Nodes Similarity.*** First, we consider the similarity between the nodes in $P$ and $P'$. Since the label of each node $n$ in a PPDG is composed by a set $W$ of words, we can calculate the label similarity between two nodes by comparing the corresponding words sets.

**Definition 3 (Label Similarity).** *Given two nodes $n_1, n_2$, let $\omega$ be the function to separate the label of a node into a set of words[3], we get $W_1 = \omega(n_1)$ and $W_2 = \omega(n_2)$. Let $\mathcal{M}$ be a function that returns the number of exact matching words, $\mathcal{S}$ be a function that returns the number of synonymous words, we define the label similarity between the two nodes as follows:*

$$lSim(n_1, n_2) = \frac{2(\mathcal{M}(W_1, W_2) + \mathcal{S}(W_1, W_2))}{|W_1| + |W_2|} \tag{1}$$

[2] Throughout the paper, we sometimes refer to the directed edges to/from nodes as simply *graph structure*.

[3] Note that the common auxiliary words, such as "a", "for" and "of", are not included.

*Example 1.* Considering the two nodes $n_1$, $n_2$, their labels are "seek a potential supervisor" and "search for an supervisor", we get the two word sets $W_1$={seek, potential, supervisor} and $W_2$={search, supervisor}. Suppose we know "seek" and "search" are synonymous, then according to Equation 1, the $lSim(n_1, n_2) = \frac{2(1+1)}{3+2} = 0.8$.

According to Definition 3, we can obtain all label similarity scores between the nodes of $P$ and nodes of $P'$. For a node $n_P \in P$, we choose the best matching node $n'_P \in P'$, where the label similarity $lSim(n_P, n'_P)$ is the highest among all similarity scores between $n_P$ and all nodes in $P'$. Then we are able to define the graph nodes similarity $nSim(P, P')$ as follows.

**Definition 4 (Graph Nodes Similarity).** *Given two PPDGs $P = (A, D, E_A, E_D, \phi, \lambda)$ and $P' = (A', D', E'_A, E'_D, \phi', \lambda')$ with $|A| \leq |A'|$, let $a_i$, $d_j$, $a'_k$, and $d'_l$ represent the nodes of $A$, $D$, $A'$, and $D'$, respectively. Then the graph nodes similarity between $P$ and $P'$ is:*

$$nSim(P, P') = \frac{u \cdot \sum_{i=1}^{|A|} \max_{k=1}^{|A'|}(lSim(a_i, a'_k)) + v \cdot \sum_{j=1}^{|D|} \max_{l=1}^{|D'|}(lSim(d_j, d'_l))}{u \cdot |A| + v \cdot |D|} \tag{2}$$

*where $u$ and $v$ are weights for action nodes and data nodes, and $u, v \in [0, 1]$ .* The weight $u$ and $v$ are defined by users to indicate which type of node is more important for them (i.e., action or data).

***Graph Structure Similarity.*** From the processing of graph nodes similarity, we obtain the similar nodes mapping of $P$ and $P'$. Then we consider the similarity of the directed edges between the similar nodes of $P$ and $P'$. Particularly, we compute the similarity of the edges from/to the action nodes separately from that of the data nodes. Hence, the similarity score is computed as follows:

**Definition 5 (Graph Structure Similarity).** *Given two PPDGs $P = (A, D, E_A, E_D, \phi, \lambda)$ and $P' = (A', D', E'_A, E'_D, \phi', \lambda')$ with $|A| \leq |A'|$, the similar nodes mapping of $P$ and $P'$ : $A \leftrightarrow A'$ and $D \leftrightarrow D'$, we get the edges matching $M_{E_A} : E_A \leftrightarrow E'_A$ and $M_{E_D} : E_D \leftrightarrow E'_D$. Then the structure similarity between $P$ and $P'$ is:*

$$sSim(P, P') = \frac{u \cdot |M_{E_A}| + v \cdot |M_{E_D}|}{u \cdot |E_A| + v \cdot |E_D|} \tag{3}$$

*where $u$ and $v$ are weights of the two types of matchings.*

***Graph Similarity.*** Based on the two types of similarities between $P$ and $P'$, we utilize the *Harmonic Mean* to compute the graph similarity between $P$ and $P'$.

**Definition 6 (Graph Similarity).** *Given two PPDGs $P$ and $P'$, we obtain their nodes similarity $nSim(P, P')$ and structure similarity $sSim(P, P')$. Then the graph similarity between $P$ and $P'$ is computed by the Harmonic Mean as:*

$$Sim(P, P') = \frac{2 \cdot nSim(P, P') \cdot sSim(P, P')}{nSim(P, P') + sSim(P, P')} \tag{4}$$

***Problem Statement.*** Given a query graph $Q$, a set $\mathcal{P}$ of PPDGs, and a threshold $\eta$ , a PPDG similarity search returns all PPDGs from the set $\mathcal{P}$, such that the similarity between $Q$ and $P \in \mathcal{P}$ is no less than $\eta$, i.e. $Sim(Q, P) \geq \eta, P \in \mathcal{P}$.

## 4 Similarity Search

In this section, we present the efficient algorithms to process the PPDG similarity search. Given a PPDG query $Q$ and a set $\mathcal{P}$ of PPDGs, the straightforward approach to find the similarity graphs to $Q$ is to compute the graph similarity between $Q$ and each $P \in \mathcal{P}$. Firstly, we join the node set of $Q$ and $P$ to get a set of node pairs - each pair $\{n_Q, n_P\}$ contains one node from $Q$ and the other from $P$, and calculate the label similarity between the nodes in each pair by Equation 1. Then for each $n_Q$, we find the *most similar node pair* with highest label similarity among all pairs containing $n_Q$. Next, after obtaining all the *most similar node pairs* between $Q$ and $P$, we compute the nodes similarity $nSim(Q, P)$ by Equation 2. Thirdly, based on the *most similar node pairs*, we find the matched edges of $Q$ and $P$ and compute the structure similarity $sSim(Q, P)$ by Equation 3. Finally, we get the graph similarity $Sim(Q, P)$ by Equation 4. The naive approach is costly because we need to iterate all the nodes and edges of one PPDG. Therefore, we propose efficient and effective techniques to process similarity search over PPDGs following a filtering-refinement framework. We first perform the nodes similarity to get a set $\mathcal{C}$ of candidate PPDGs, and then refine $\mathcal{C}$ by structure similarity to get the search results. Section 4.1 presents the filtering technique with efficient and effective pruning rules to get the candidate set. Section 4.2 refines the candidates by computing graph structure similarity.

### 4.1 Filtering by Graph Nodes

For the query $Q$ and a PPDG $P$, the computing of graph nodes similarity $nSim(Q, P)$ is based on the label similarities between their nodes. In this subsection, we first propose the technique to compute label similarities between the nodes of $Q$ and $P$. Then the graph nodes similarity $nSim(Q, P)$ is obtained according to the label similarities.

To compute the label similarities between the nodes of $Q$ and $P$, we first decompose the label of each node into a set of words, and store the words set in corresponding node, i.e. $n_Q.W$ for one node $n_Q \in Q$ and $n_P.W$ for one node $n_P \in P$. Then we create a set of nodes pairs, each consisting of two nodes - one from $Q$ and the other from $P$, denoted as $\{n_Q, n_P\}$. In each pair, $n_Q$ and $n_P$ are the same type of nodes. Recall that there are two types of node - action node and data node - existing in PPDG. For each pair, we can compute the label similarity score between the two nodes.

To identify whether two words are synonymous, we use the electronic lexical database - WordNet [5] to match words. We also build a local dictionary to store the synonymous words to save the processing time of our algorithm. In this paper, we assume the dictionary is already built locally and it contains all the synonymous words in all graphs.

**Theorem 1.** *Given two nodes $n_1$ and $n_2$, let $W_1$ and $W_2$ to be their words sets respectively, then the similarity between the two nodes is:*

$$lSim(n_1, n_2) \leq \frac{2 \cdot \min(|W_1|, |W_2|)}{|W_1| + |W_2|} \tag{5}$$

*which is the upper bound of $lSim(n_1, n_2)$, denoted as $lSim(n_1, n_2)^{up}$.*

*Proof.* According to Definition 3, if all words are matched, the similarity score is $\frac{2\cdot\min(|W_1|,|W_2|)}{|W_1|+|W_2|}$. □

Theorem 1 gives an upper bound of the label similarity between two nodes, and we can use the bound to prune some node pairs without calculating their label similarities.

**Algorithm 1:** ***Filtering by Nodes Similarity*** **$(Q, \mathcal{P}, \eta, u, v)$**

```
   Input  : Query $Q$, A set $\mathcal{P}$ of PPDGs, Threshold $\eta$, Weight $u, v$
   Output: Candidate set $\mathcal{C}$
1  $\mathcal{C} := NULL$, $\tau := \eta/(2-\eta)$;
2  $N_Q \leftarrow$ All nodes in $Q$, $n_Q.W$ stores words extracted from the label of $n_Q$;
3  for each $P \in \mathcal{P}$ do
4      $pairSet := NULL$, $M_A := |Q.A|$, $M_D := |Q.D|$, $pruned := false$;
5      $N_P \leftarrow$ All nodes in $P$, $n_P.W$ stores words extracted from the label of $n_P$;
6      for each $n_Q \in N_Q$ do
7          $H := NULL$;
8          for each $n_P \in N_P$ and $n_Q$, $n_P$ are the same type of nodes do
9              $T.pair := \{n_Q, n_P\}$;
10             $T.lsimUp := lSim(n_Q, n_P)^{up}$;
11             $H$.push($T$);
12         $R.lsim := 0$;
13         while $H \neq NULL$ do
14             $T := H$.pop();
15             if $T.lsimUp < R.lsim$ then                         // Pruning rule 1
16                 break;
17             else if $lSim(T.pair) > R.lsim$ then
18                 $R.pair := T.pair$;
19                 $R.lsim := lSim(R.pair)$;
20         if the nodes in $R$ are action nodes then
21             $M_A := M_A - 1 + R.lsim$;
22         else
23             $M_D := M_D - 1 + R.lsim$;
24         $nSimUp := (u * M_A + v * M_D)/(u * |A| + v * |D|)$;
25         if $nSimUp < \tau$ then                                  // Pruning rule 2
26             $pruned := true$;
27             break;
28         $pairSet \leftarrow R$;
29     if !$pruned$ then
30         $P.pairSet := pairSet$; $P.nsim := nSim(Q, P)$;
31         $\mathcal{C} \leftarrow P$;
32 return $\mathcal{C}$;
```

***Pruning rule 1.*** For a node $n_Q \in Q$, we want to find a *matching* node $n_P \in P$ with the highest label similarity score. Once we get the label similarity $lSim(n_Q, n_{Pi})$ for one node pair $\{n_Q, n_{Pi}\}$, then any node pair $\{n_Q, n_{Pj}\}$

with $lSim(n_Q, n_{Pj})^{up} < lSim(n_Q, n_{Pi})$ can be pruned safely. Therefore, we perform the nodes similarity between $Q$ and $P$ as following steps. 1) we first extract words from the label of each node in $Q$ and $P$. 2) For each node $n_Q \in Q$, we join it with all nodes in $P$ that are of the same type as $n_Q$, to create a set of node pairs. Recall that there are two types of node - action node and data node - in one PPDG. 3) All pairs are sorted by the upper bound of their label similarities in descending order, and stored in a max heap $H$. 4) We compute the label similarity of node pairs from the top of $H$ one bye one, and the pair with highest label similarity score is kept in a tuple $R$. If the upper bound of the label similarity of the pair in the top of $H$ is smaller than the label similarity in $R$, the remaining pairs in $H$ can be pruned safely and $R$ is the *most similar node pair* from $P$ to $n_Q$. 5) After all *most similar node pairs* are obtained, the nodes similarity between $Q$ and $P$ can be calculated by Definition 4.

**Theorem 2.** *Given a PPDG query $Q$, a PPDG $P$, and a threshold $\eta$, to ensure $Sim(Q, P) \geq \eta$, the nodes similarity $nSim(Q, P)$ is:*

$$nSim(Q, P) \geq \frac{\eta}{2 - \eta} \tag{6}$$

*which is the threshold of the nodes similarity, denoted as $\tau$.*

*Proof.* According to Definition 6, we assume the structure similarity between $Q$ and $P$ is equal to 1, i.e. $sSim(Q, P) = 1$, then we have $nSim(Q, P) \geq \frac{\eta}{2-\eta}$. □

***Pruning rule 2.*** When the nodes similarity between $Q$ and $P$ is calculated, we can compare it with threshold $\tau$, obtained by Theorem 2. It is clear that if $nSim(Q, P) < \tau$, the graph $P$ can be pruned safely. Note that the computing of nodes similarity is based on the label similarities of all *most similar node pairs*. Therefore, after getting the label similarity of each *most similar node pair*, we can compute the upper bound $nSim(Q, P)^{up}$ of the nodes similarity between $Q$ and $P$ by assuming the label similarities of remaining unprocessed nodes are equal to 1. If $nSim(Q, P)^{up} < \tau$, $P$ can be pruned without further processing. Finally, all un-pruned graphs are stored in a candidate set $\mathcal{C}$ for the refinement step.

Algorithm 1 illustrates the details of the filtering step. To enable computing the nodes similarity in an iterative fashion, we use a tuple $T$ to process the query. $T$ is employed to maintain a node pair with the upper bound of its label similarity. Particularly, *T.pair* stores a node pair, one node from the query $Q$ and the other from a PPDG $P$, and *T.lsimUp* stores the upper bound of the label similarity of the pair. We first initiate $\tau$ by Theorem 2 in Line 1 and store all the nodes of $Q$ in a set $N_Q$ in Line 2. Each node $n_Q$ has a word set attribute $n_Q.W$ to store its words extracted from its label. Next, we iterate the graphs $P \in \mathcal{P}$ one by one to compute the nodes similarity $nSim(Q, P)$ from Line 3 to Line 31. To begin with, a pair set *pairSet* is initiated to store all the *most similar node pairs* between $Q$ and $P$. For each $P$, we first assume all action nodes and data nodes in $Q$ exactly match the nodes in $P$, and let $M_A$, $M_D$ equal to the number of action nodes and data nodes in $Q$, respectively, in Lines 4. Then we also get all the nodes of $P$ with their words and store them in a set $N_P$ in Line 5.

From Line 6 to Line 11, we join each node $n_Q \in N_Q$ with all nodes in $N_P$ to get node pairs and compute the upper bound of their label similarities, which are stored in *T.pair* and *T.lsimUp*, respectively. We use a max heap $H$ to store $T$. The pairs are sorted in $H$ by *T.lsimUp* in descending order. Line 12 initializes a tuple $R$ to store the node pair which has the highest label similarity score after all pairs in $H$ are iterated. From Line 13 to Line 19, we compute the label similarity of the pairs from the top of $H$ and store the pair with the highest label similarity score in $R$. According to the Theorem 1, if the upper bound of the label similarity *T.lsimUp* of a pair *T.pair* is smaller than *R.lsim*, the iteration is stopped in Line 13, and $R$ contains the *most similar node pair* with its label similarity. After we get one *most similar node pair R.pair*, $M_A$ or $M_D$ is updated based on its label similarity *R.lsim* and the upper bound *nSimUp* of the nodes similarity is computed in Lines 20-24. If *nSimUp* is smaller than $\tau$, $P$ can be pruned (Line 25). All *most similar node pairs* are stored in the set *pairSet* (Line 28). If $P$ is not pruned, after getting all the *most similar node pairs* between $Q$ and $P$, the nodes similarity *nSim* is computed by Equation 2 and $P$ is put into the candidate set $\mathcal{C}$ in Lines 29-31.

### 4.2 Refinement by Graph Structure

After processing $Q$ with all PPDGs by Algorithm 1, we obtain a candidate set $\mathcal{C}$ of PPDGs. In this subsection, we utilize the graph structure to refine the candidates.

In the filtering step, we obtain the *most similar node pairs* stored in *P.pairSet* for each candidate $P \in \mathcal{C}$. Considering two pairs $\{n_Q, n_P\}$ and $\{n'_Q, n'_P\}$ in *P.pairSet*, there is an edge $e_Q$ between $n_Q$ and $n'_Q$. If there is also an edge $e_P$ between $n_P$ and $n'_P$, and $e_Q$ and $e_P$ are the same type of edges with the same direction, we can determine $e_Q$ and $e_P$ are matched. The straightforward approach is to check whether any two pairs in *P.pairSet* have matched edges. However, the processing cost is very high, because we need check every two pairs. Even if there is no edge between $n_Q$ and $n'_Q$, we still need to check whether the edge exists on $Q$ or not.

We can use the graph structure to reduce the processing time. When we pick a pair $\{n_Q, n_P\}$ from *P.pairSet*, we get all the connected nodes of $n_Q$ from graph $Q$. Next, for each of connected node $n'_Q$, we get pair $\{n'_Q, n'_P\}$ from *P.pairSet*. If there is an edge $e_P$ between $n_P$ and $n'_P$, and $e_Q$ and $e_P$ are the same type of edge with the same direction, we say the two edges are matched. Particularly, after one pair is iterated, the pair is removed to avoid the reverse checking from $\{n'_Q, n'_P\}$.

**Theorem 3.** *Given a PPDG query $Q$, a PPDG $P$, and a threshold $\eta$, we get the nodes similarity $nSim(Q,P)$. To let graph similarity $Sim(Q,P) \geq \eta$, the structure similarity is:*

$$sSim(Q,P) \geq \frac{nSim(Q,P) \cdot \eta}{2 \cdot nSim(Q,P) - \eta} \tag{7}$$

*which is the threshold of the structure similarity, denoted as $\epsilon$.*

*Proof.* According to Definition 6, it is clear that the inequality holds. □

**Algorithm 2:** ***Refinement by Structure Similarity*** $(Q, \mathcal{C}, \eta, u, v)$

```
Input  : Query $Q$, Candidate set $\mathcal{C}$, Threshold $\eta$, Weight $u, v$
Output: search result $\mathcal{R}$
1  $\mathcal{R} := NULL$;
2  for each $P \in \mathcal{C}$ do
3      $pruned$:=false;
4      $\epsilon = (P.nsim * \eta)/(2 * P.nsim - \eta)$;
5      $M_{EA}$:=$|Q.E_A|$, $M_{ED}$:=$|Q.E_D|$;
6      $unvisited := P.pairSet$;
7      for each most similar node pair $\{n_Q, n_P\} \in P.pairSet$ do
8          $unvisited$.remove($\{n_Q, n_P\}$);
9          $neighbors \leftarrow$the connected nodes of $n_Q$ with edge types;
10         for each $n'_Q \in neighbors$ do
11             Find $\{n'_Q, n'_P\}$ in $unvisited$;
12             if there is no same type of edge between $n_P$ and $n'_P$ then
13                 if $n'_Q.E$ is an action edge then
14                     $M_{EA} := M_{EA} - 1$;
15                 else
16                     $M_{ED} := M_{ED} - 1$;
17         $sSimUp := (u * M_{EA} + v * M_{ED})/(u * |Q.E_A| + v * |Q.E_D|)$;
18         if $sSimUp < \epsilon$ then                         // Pruning rule 3
19             $pruned$:=true;
20             break;
21     if !$pruned$ then
22         $P.sim \leftarrow$ the graph similarity;
23         $\mathcal{R} \leftarrow P$;
24 return $\mathcal{R}$;
```

***Pruning rule 3.*** From the filtering step, we gain the graph nodes similarity $nSim(Q, P)$ for each candidate $P \in \mathcal{C}$. Then we compute the threshold $\epsilon$ of the graph structure similarity according to Theorem 3. Like pruning rule 2, we assume the edges of $Q$ and $P$ are all matched in the beginning, and obtain the upper bound of the structure similarity $sSim(Q, P)^{up}$. When we process the graph structure similarity search, if one edge of $Q$ cannot match any edge of $P$, $sSim(Q, P)^{up}$ decreases due to the unmatched edges. If $sSim(Q, P)^{up}$ is less than $\epsilon$, $P$ can be pruned safely without further processing.

Algorithm 2 illustrates the details of the refinement step. We iterate all PPDGs in the candidate set $\mathcal{C}$. For each PPDG $P$, we first compute the threshold $\epsilon$ of its structure similarity based on its nodes similarity $P.nsim$ and graph similarity threshold $\eta$ in Line 4. Then, we assume all edges are matched and set $M_{EA}$ and $M_{ED}$ as the number of action edges and data edges of query $Q$, respectively, in Line 5. Line 6 initializes a pairs set *unvisited* to be filled with $P.pairSet$. Once a pair is visited, it is removed from *unvisited* to avoid reverse checking. From Line 7 to Line 20, we iterate all *most similar node pairs* to match the edges between $Q$ and $P$. Line 9 finds all neighbors of the node $n_Q \in \{n_Q, n_P\}$ with their edges, and store them into *neighbors*. For each neighbor $n'_Q \in neighbors$, we search *unvisited* to get the corresponding pair $\{n'_Q, n'_P\}$ (Line 11), and then check the PPDG $P$ to identify if there is a same type of edge between $n_P$ and $n'_P$ and update $M_{EA}$ and $M_{ED}$ based on the result (Line 12-16). Next, we compute the upper bound of the graph structure similarity from the updated $M_{EA}$ and $M_{ED}$ and prune $P$ if the upper bound is smaller than $\epsilon$ (Lines 17-20). Finally, if $P$ is un-pruned, we compute its similarity score and put it in result set $\mathcal{R}$ (Line 22-23).

When the query $Q$ has *path* edges, we consider it as a special PPDG. To process it, we make a little adaption on Algorithm 2 to refine the candidates. If there is a *path* from $n_Q$ to $n'_Q$ in $Q$, we obtain the pairs $\{n_Q, n_P\}$ and $\{n'_Q, n'_P\}$. Then we traverse the PPDG $P$ from $n_P$ to check whether $n'_P$ can be reached, and the result of *path* searching is passed to the judging condition in Line 12 of Algorithm 2 to check whether the two *paths* are matched.

## 5 Experiments

Now we present the results of a comprehensive performance study to evaluate the efficiency and scalability of our proposed techniques. Following algorithms are evaluated.

- **NAIVE:** Techniques in Section 4 but without any pruning rule.
- **P1:** Techniques in Section 4 but using Pruning Rule 1 only.
- **P12:** Techniques in Section 4 but using Pruning Rule 1 and 2 only.
- **SIM:** Techniques presented in Section 4 to process similarity search with all pruning techniques.

***Datasets*** We have evaluated our similarity search techniques on both synthetic and real datasets.

The synthetic datasets were generated by randomization techniques. We create a word set containing 100 words: 50 different words and 25 pairs of synonymous words. A dictionary is built to store the mapping of synonymous words. Then we randomly choose $n$ action nodes and $[0, 2n]$ data nodes to assemble $p$ process graph. For each node, $w$ words are randomly selected to make the label. After a graph is built, we make several small changes, such as changing the labels of nodes and adding/deleting nodes, to obtain 99 similar process graph. The number $p$ varies from $2K$ to $50K$ (default value = $10K$). The number $n$ of action nodes in each process is randomly chosen in a range varying from $[5, 10]$ to $[35, 40]$ (default value = $[15, 20]$). The number of data nodes is randomly chosen in $[0, 2n]$. For each node, there are up to $w$ words randomly to be selected. The $w$ varies from 10 to 25 (default value = 15). By the default setting, the total number of nodes is up to $600K$ in our experiment. The threshold $\eta$ varies from 0.2 to 0.8 (default value = 0.6). We choose 100 process graphs and get their subgraphs to make 100 query graphs, which are used in the experiment. The size $s$ of query, i.e. number of action nodes in query, varies from 3 to 9 (default value = 5). The average processing time of the 100 queries on each dataset represents the performance of our query processing mechanism. The weight $u$ and $v$ are set to 1 in all experiments.

The real dataset consists of 42 PPDGs collected from surveys conducted with research students and alumni at UNSW. We asked 26 volunteers to provide personal process descriptions on processes such as research degree admission, scholarship applications and attending graduation ceremony. In this dataset, the queries are chosen manually.

All algorithms are implemented in C++ and compiled by Cygwin GCC 4.3.4. The experiments are conducted on a PC with Intel i7 2.80GHz CPU and 8G memory on Windows 7 Professional SP1. All algorithms are run in main memory.

## Performance Evaluation

We evaluate the performance of the four algorithms (NAIVE, P1, P12, SIM) in the experiment.

***Real vs Synthetic.*** In the first experiment, we evaluate the performance of NAIVE, P1, P2 and SIM over the real and synthetic data. Due to the limited quantity of real process graphs, we magnify the result on the real data by 200 times in Figure 3. It is shown that our techniques give the similar pruning power on both datasets, and each pruning rule is very effective and reduces the processing time. Particularly, pruning rule 2 has the best performance among the three pruning rules.

***Impact of Threshold $\eta$.*** We evaluate the processing time of our algorithms as a function of the threshold $\eta$ which varies from 0.2 to 0.8. Figure 4 shows the cost of P12 and SIM are reduced significantly when $\eta$ increases, because the two techniques utilize $\eta$ to perform the pruning. Comparing with P12, the processing time of SIM does not improve a lot when $\eta$ is 0.8. The reason is that pruning rule 2 prunes a mass of graphs and leaves a few candidate graphs for the refinement step.

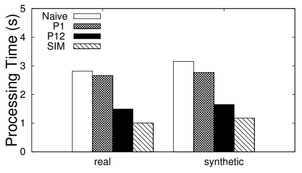


**Fig. 3.** Real vs Synthetic

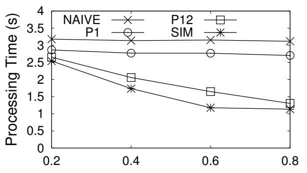


**Fig. 4.** Varying $\eta$

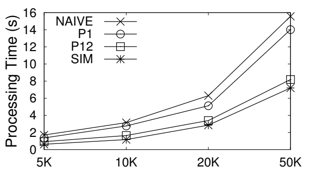


**Fig. 5.** Varying $p$

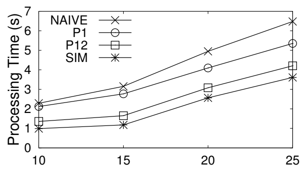


**Fig. 6.** Varying $w$

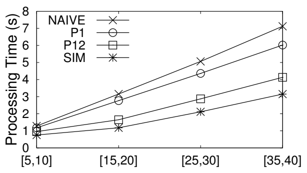


**Fig. 7.** Varying $n$

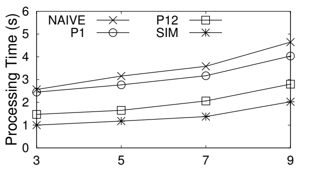


**Fig. 8.** Varying $s$

***Evaluating Impacts by Different Setting.*** We study the scalability of our algorithms with regards to the different number of process graphs ($p$), number of words ($w$) in one graph node, number of action nodes ($n$), and the query size ($s$) in Figure 5-8. The processing time increases with the increase of the four parameters. However, the results also demonstrate that each pruning rule is effective and reduces the processing time in all settings. Clearly, the dataset size increases with the number of process graphs and action nodes thus the filtering and refinement processing becomes more expensive. Longer word size makes it difficult to process the label similarity match, which increases the processing time. When the query size grows, the processing cost increases because more nodes and edges are involved in the similarity search.

## 6 Related Work

One of the closely related work to ours is Cooking Graphs [11]. A cooking graph describes a cooking process with cooking actions and relevant ingredients information. However, cooking graphs are specialized to represent one domain and the action/data concepts are not as generic as PPDG. Also, the focus of the work is with implementing a graph mining technique to recognize cooking process patterns (by considering graph structures) and recommend a suitable cooking recipes for a user. Through PPDG and PPDG querying techniques, we aim to provide a platform to support various analysis tasks, not limited to recommendation. Besides, PPDG considers similarity matching in labels as well as the graph structures.

There are several work we can refer to in the area of BPM with regards to querying processes. In these work, queries are processed over BPMN (Business Process Modelling Notation) or equivalent notations. The main purpose of the languages is to extract actions (i.e., control flows). For example, The Business Process Query Language (BPQL) in [2] works on an abstract representation of BPEL[4] files. The BPMN-Q is a visual language to query repositories of BPMN models [1, 8]. It processes the queries by converting both the query and BPMN to graphs. PPDG describes personal processes directly as graph (although mapping from/to BPMN is possible) and uses a query paradigm which takes both actions and data nodes into consideration with their labels and directed edges.

Recently, BPMN label matching techniques for querying similar process models is presented in [7]. However, the authors only consider action node labels. Also, they do not consider the directed edges (i.e., graph structure) present in the process models and simply refer to a process model as a set of activities.

In [3], the authors deal with the problem of retrieving process models in the repository that most closely resemble a given process model. This paper proposes three similarity metrics: (i) label matching similarity - it measures similarity based on words in the labels of business process model elements, (ii) structural similarity - it measures similarity based on graph edit distance of business process model, (iii) behavior similarity - it measures similarity based on the intended behavior (operation semantic) of process models. At present, authors have focused on developing the metrics rather than efficient implementation of algorithms.

Besides the BPM area, more general applications of graph similarity search have received considerable attention, such as Closure-Tree [6], $K$-AT [10], and SEGOS [12]. Specially, subgraph similarity search is to retrieve the data graphs that approximately contain the query. Grafil [14] proposes the problem, where

[4] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

similarity is defined as the number of missing edges regarding maximum common subgraph. GrafD-index [9] deals with similarity based on maximum connected common subgraph. [15] studies the problem of graph similarity search with edit distance constraints. However, they only consider the similarity of graph structure. Therefore, techniques proposed in [14, 9, 15] cannot be directly applied to PPDGs.

## 7 Conclusion

In this paper, we have investigated similarity search over Personal Process Description Graph (PPDG). We formally define the similarity between two PPDGs as a harmonic mean of two similarity scores: graph nodes similarity and graph structure similarity. By utilizing the features of PPDG nodes and structure, we develop effective and efficient algorithms with novel pruning techniques following the *filtering and refinement paradigm*. A comprehensive experimental study over both real and synthetic datasets demonstrates the efficiency and scalability of our techniques.

## References

1. A. Awad, S. Sakr, M. Kunze, and M. Weske. Design by selection: A reuse-based approach for business process modeling. In *ER*, pages 332–345, 2011.
2. C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes. In *PVLDB*, pages 343–354, 2006.
3. R. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling. Similarity of business process models: Metrics and evaluation. *Inf. Syst.*, 36(2):498–516, 2011.
4. M. Dumas, M. La Rosa, J. Mendling, and H. Reijers. *Fundamentals of Business Process Management*. Springer-Verlag Berlin Heidelberg, 2013.
5. C. Fellbaum. *WordNet: An Electronic Lexical Database*. Language, speech, and communication. MIT Press, 1998.
6. H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *ICDE*, page 38, 2006.
7. C. Klinkmüller, I. Weber, J. Mendling, H. Leopold, and A. Ludwig. Increasing recall of process model matching by improved activity label matching. In *BPM*, pages 211–218, 2013.
8. S. Sakr and A. Awad. A framework for querying graph-based business process models. In *WWW*, pages 1297–1300, 2010.
9. H. Shang, X. Lin, Y. Zhang, J. X. Yu, and W. Wang. Connected substructure similarity search. In *SIGMOD*, pages 903–914, 2010.
10. G. Wang, B. Wang, X. Yang, and G. Yu. Efficiently indexing large sparse graphs for similarity search. *IEEE Trans. Knowl. Data Eng.*, 24(3):440–451, 2012.
11. L. Wang. *CookRecipe: towards a versatile and fully-fledged recipe analysis and learning system*. PhD thesis, City University of Hong Kong, 2008.
12. X. Wang, X. Ding, A. K. H. Tung, S. Ying, and H. Jin. An efficient graph indexing method. In *ICDE*, pages 210–221, 2012.
13. J. Xu, H. Paik, A. H. H. Ngu, and L. Zhan. Personal process description graph for describing and querying personal processes. In *ADC*, 2015.
14. X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. In *SIGMOD*, pages 766–777, 2005.
15. X. Zhao, C. Xiao, X. Lin, W. Wang, and Y. Ishikawa. Efficient processing of graph similarity queries with edit distance constraints. *VLDB J.*, 22(6):727–752, 2013.