Inf Syst Front
DOI 10.1007/s10796-016-9662-x

Mining variable fragments from process event logs

Asef Pourmasoumi[1] · **Mohsen Kahani**[1] · **Ebrahim Bagheri**[2]

Abstract Many peer-organizations are now using process-aware information systems for managing their organizational processes. Most of these peer-organizations have shared processes, which include many commonalities and some degrees of variability. Analyzing and mining the commonalities of these processes can have many benefits from the reusability point of view. In this paper, we propose an approach for extracting common process fragments from a collection of event logs. To this end, we first analyze the process fragment literature from a theoretical point of view, based on which we present a new process fragment definition, called *morphological fragments* to support composability and flexibility. Then we propose a novel algorithm for extracting such morphological fragments directly from process event logs. This algorithm is capable of eliciting common fragments from a family of processes that may not have been executed within the same application/organization. We also propose supporting algorithms for detecting and categorizing morphological fragments for the purpose of reusability. Our empirical studies show that our approach is able to support reusability and flexibility in process fragment identification.

Keywords Process fragments · Morphological fragments · Event logs · Cross organizational mining · Reusable fragments

✉ Mohsen Kahani
kahani@um.ac.ir

[1] Web Technology Laboratory, Ferdowsi University of Mashhad, Mashhad, Iran

[2] Department of Electrical and Computer Engineering, Ryerson University, Toronto, ON, Canada

1 Introduction

Many organizations have replaced their traditional processes, which would be executed and monitored manually, with the so-called Process Aware Information Systems (PAIS) (van der Aalst 2009). Business Process Management (BPM) technology lies at the core of PAIS and promotes effectiveness and efficiency of business processes (Dumas et al. 2013). In the relative short evolution lifetime of BPM technology, the number of already deployed process models within organizations has grown exponentially (Weske 2012).

With the growth of the deployment of business process models, new challenges have emerged. One of the challenges of interest to our work in this paper is the co-existence of multiple variants of the same business process or its fragments within the same or different peer organizations (Milani et al. 2013). For example, in municipalities, many processes are driven by legislation, e.g., the process for determining and calculating property sales tax rates is in most cases highly regulated. As such, regulated processes would have very high resemblance to each other even if executed and deployed in different departments or organizations. However, while legislation is establishing the important fundamentals, some degree of freedom is often given to the executing units regarding the concrete implementation of such processes. Therefore, different departments could in practice be implementing similar processes that have been only slightly modified based on local requirements and preferences (Larosa et al. 2013a).

Given this fact that it is possible to find notable similarity between the process fragments within similar organizations, consolidating families of process variants can assist organizations in improving their operations and to design more efficient process models. By detecting similar fragments, it would be possible to create repositories of process fragments that can facilitate reusability (Uba et al. 2011). Reusability has many

Published online: 20 June 2016

benefits for organizations (especially large-scale organizations). During the design phase, organizations can use process fragmentation approaches for extracting common fragments from the repository of process models. Reusing these extracted process fragments can decrease design for new process models and significantly accelerate the design process.

Many existing work identify process fragments and variabilities through comparison of structural similarities of business process models (Larosa et al. 2013b; Valenca et al. 2013). In (Milani et al. 2015), authors review the state of the art on approaches that manage variability in process models. In our work, we will focus on the challenging problem of identifying process fragments when the business process models are not available by only analyzing *process event logs*. This is mainly driven by the fact that the process models that are initially designed are often different from the ones that are actually being executed due to the gradual changes that are made on the models during execution (TFPM – ieee task force on process mining 2012). The advantage of directly using event logs is that we can identify and handle exceptions and rare patterns in process execution, which would otherwise be harder to identify by only looking at the business process model itself (Pourmasoumi 2015).

The main objective of our work is to identify *flexible* and *composable* process fragments from process event logs. In other words, we would like to identify fragments from process event logs such that they can be used within the process of automated customization or personalization of business processes. The major challenge with our endeavour is the need to strike a balance between *composability* and *flexibility*. There are two classes of work in the literature: *i*) those that identify similar fragments that have some degree of resemblance to each other (similar fragments) and *ii*) those that identify and extract exact fragment matches that can be used interchangeably (exact fragments). The identification of *similar fragments* allows for the development of variants of the same business process with some flexibility; while the exact fragments enhance composability and automated process composition. Our objective is to define a specific form of process fragments that allows for both composability and flexibility.

To this end, we first identify and review various well-known definitions within the process fragment domain. To the best of our knowledge, most of the existing process fragment definitions have only considered the practical implications of their work and little, if any, have performed substantial theoretical analysis (Mancioppi and Danylevych 2012). In this paper, we investigate the use of ontological theories for the theoretical analysis of process fragment definitions. An ontological theory defines necessary constructs for describing processes and structures of the world in general (Pourmasoumi et al. 2015a). Ontological theories have been used to evaluate modeling languages in terms of the correspondence of ontological concepts and modeling constructs. Bunge's ontology is a widely used ontological theory that has been used in particular to evaluate conceptual modeling languages such as BPMN and workflow nets (Jan et al. 2006; Evermann and Wand 2005). Bunge's ontology has also been used for representing process models. In (Soffer et al. 2010), a generic model, called GPM, is derived from Bunge's ontology for semantically describing process models. GPM gives a formal abstract view of a process model in terms of state transitions that occur rather than using common notions such as control flows and activities (Soffer et al. 2010). In the other words, GPM can be viewed as a mapping between process models and the real world. GPM also has been used in several applications. For example, in (Ghattas and Soffer 2009), the authors propose a generic model based on GPM to describe the inter-organization shared processes without forgoing privacy and autonomy of each organization. Since GPM is an ontological representation of real world process models, we presume that it is appropriate for analyzing the fragments of processes models that represent real world domains. Our theoretical analysis of existing process fragment definitions will be based on GPM.

Based on the theoretical analysis of the process fragment literature, we propose a new process fragment definition, called *morphological fragments* that support composability and flexibility as defined later in the paper. We propose a novel algorithm for extracting such morphological fragments from process event logs. We also further propose supporting algorithms for detecting and categorizing morphological fragments for the purpose of reusability.

In this paper, we provide the following concrete contributions:

- In our previous works, we proposed a formal definition for process fragmentation that supports both composability and flexibility, called *morphological fragments* (Pourmasoumi et al. 2015a, 2014). In this paper, we propose an algorithm that enables the extraction of such morphological fragments with a manageable time complexity. Additionally, in the current paper, we prove that our numerical representation for extracting common fragments is unique for each morphological candidate (Lemma1).
- Most fragment detection methods are limited to finding exact common fragments. In this paper, we extend our earlier work and define a degree of variation, called *degree of morphologicality*, which considers how much two fragments have structural similarity. We present a clustering algorithm for eliciting fragments, which have the same degree of morphologicality.
- In our earlier work, we evaluated our proposed approach with a simple case study. In this paper, we comprehensively evaluate our improved fragments extraction algorithm and also the approach proposed for extracting

morphological fragments with varying degree of morphologicality. We have used the standard BIT 2009 collections and also used the tools proposed in (Pourmasoumi et al. 2015b) for generating test data. In order to evaluate the results, the *refactoring gain* measure as proposed in (Uba et al. 2011) is used.

The rest of this paper is organized as follows. Section 2 reviews the related works and Section 3 presents a running example. Section 4 establishes the basic terminology and definitions that are adopted in this work. In Section 5, we analyze the various fragment definition. In Section 6, we described the proposed approach for extracting process fragments. Section 7 outlines our evaluation plan. In Section 8 some discussions on the proposed approach is presented and Section 9 concludes the paper.

2 Related work

Process fragmentation is referred to the act of categorizing process model elements such as activities, data flows, and control flows into groups (Mancioppi and Danylevych 2012). The created groups are known as *process fragments*. *Process fragmentation* is the basis for techniques supporting reusability, parallel execution, management and analysis of process models (Tan and Fan 2007). It is also known as *process decomposition* or *process modularization* (Reijers et al. 2011). Many process fragmentation approaches have been proposed in the past. A survey of fragmentation algorithms can be found in (Mancioppi and Danylevych 2012). In (Mancioppi and Danylevych 2012), a classification framework investigates the characteristics of fragmentation techniques for process models and the properties of the resulting process fragments. The classification criteria presented in (Mancioppi and Danylevych 2012) provides (1) a basis for classifying existing fragmentation techniques, and (2) a "check-list" of what authors should consider in their fragmentation approaches.

Reijers et al. (2011) propose three types of criteria for fragmentation namely: a) *the block-structuredness of the sub-process*, b) *the connectedness of nodes in the sub-process*, and c) *the similarity of the labels of the nodes in the sub-process*. A process is called block-structured if it has a single *entry* and a single *exit* (SESE) (Leemans et al. 2014). There are multiple representations for block-structured process models, such as BPEL (Andrews et al. 2003) and refined process structure tree (Vanhatalo et al. 2009). In (Uba et al. 2011), Uba et al. propose an indexing structure (RPSDAG) for identifying exact SESE fragment clones that can be refactored into shared sub-processes. The RPSDAG index combines a method for decomposing (the refined process structure tree decomposition) process models into fragments with a method for generating a unique string from a labeled graph. In this approach, first, the processes are converted into a structure tree. Then, the common sub-graphs are detected and the repository is built. The structural restriction of this approach is that the input processes should be block structured.

In terms of connectedness, a collection of nodes (activities) is considered to be connected if the nodes in the collection are more strongly connected by arcs to each other than to nodes outside this collection (Reijers et al. 2011). Graph analysis techniques can be used for extracting connected nodes (Schumm et al. 2010). The third fragmentation criterion from (Reijers et al. 2011) builds on the idea that activities that have more similar labels can be considered to have a higher probability of belonging to a fragment than activities that have very different labels. Several approaches have been proposed in the literature based on this criterion. Most of them use natural language processing techniques for eliciting label similarity (Smirnov et al. 2010). In (Dijkman et al. 2013), several metrics for determining similarity of two labels such as edit distance or semantic similarity has been investigated.

In (Rosa et al. 2015), an approach for extracting approximate clones in repositories of business process models is proposed. In this approach, using process similarity methods, clones of similar sub-processes are identified. This approach has wide applicability in the real world because it identifies clones with some degree of variability while many of the other clone detection techniques are only limited to finding exact clones. Similarly, Gao et al. (2014) extract common sub-processes using process similarity methods. They use fragmentation and clustering techniques for fragmenting input processes. Then they merge the extracted fragments of each cluster into a master fragment. Although Gao et al. eventually consider a collection of process models as input, but their proposed fragmentation approach only considers one process model at a time instead of working with the whole collection simultaneously.

There are also some approaches that propose the use of existing methods from other domains for process fragmentation. For example, in (Ivanovic et al. 2010), a fragment identification approach based on *sharing analysis* is presented. Sharing analysis is a static analysis technique that interprets a program by mapping sets of variable values into abstract domains, together with data operations. The authors have used Horn clauses designed to adequately enforce sharing between inputs and outputs of workflow activities. A limitation of this approach might be that describing data objects does not influence sharing analysis, and consequently, the obtained fragments would be the same as when data objects are not taken into consideration. The work presented in (Zemni et al. 2012) is another approach that analyses process fragmentation from another unique point of view. In this work, the authors propose an approach based on *Formal Concept Analysis* (*FAC*) for generating privacy-aware reusable process figments. Since

each business process model may contain personal data, privacy concerns might be raised during process fragments reuse. FCA is a mathematical formalism, which is usually used to check the affinities between objects (Ganter and Wille 1999). The authors first translate a process model into the FCA Formal Context where objects are activities and data objects are attributes. Then they incorporate privacy constraints in FCA logic. The focus of this paper is on process models, not event logs and is therefore different from the focus of our paper.

The idea of process fragmentation has also been considered from the perspective of process distribution and parallelization. The main idea here is to identify process fragments that can either be executed in parallel or in a distributed fashion. The authors in (Tan and Fan 2007) address this challenge and dynamically separate an integrated workflow model into smaller partitions at runtime and allocate them to different servers to be executed. Similarly, in (Atluri et al. 2007), a decentralized workflow model, known as self-describing workflows is introduced. The self-describing workflow allows each execution unit to receive a self-describing workflow. Then, the execution agents execute their tasks and forward it to the next agent. This approach does not consider any restrictions on the model structure. The process fragmentation is done by the combination of the reachable successive activities.

To the best of our knowledge, most of the existing fragmentation methods as discussed above and reviewed in (Mancioppi and Danylevych 2012), are based on the process model structure and not on event logs. Our focus is on identifying process fragments from process event logs for two main reasons: 1) during their lifetime formal business processes can become stale and outdated (TFPM – ieee task force on process mining 2012); therefore, would not be representing the actual processes that are being executed in practice. 2) In each process model there might be several exception points that might only rarely occur. These exceptions cannot be easily detected from process models; however, they can be observed within event logs (van der Aalst 2011, 2012). For these practical reasons, our focus in this paper will be to identify process fragments from event logs.

3 A running example

In this paper, we employ a running example from a "purchasing order" process for better explaining the various aspects of the paper. Figure 1 presents the structure of the purchasing order process. Figure 1a shows the flow of this process model. It starts by "filling request form" and ends with "send material" or "request rejection" activities. After the "filling request form" activity, the request will be checked and if it is confirmed then two actions can be done simultaneously: the stock will be checked and the financial calculation will be done. If the requested material exists in stock, then the user account will be checked and payment will be processed. Afterwards, the receipt will be printed and sent along with the material to the customer. This process is a simplified purchasing order process and is used just for the purpose of demonstrating the details of our work and does not necessarily show the complexity of the real world process models.

4 Preliminaries

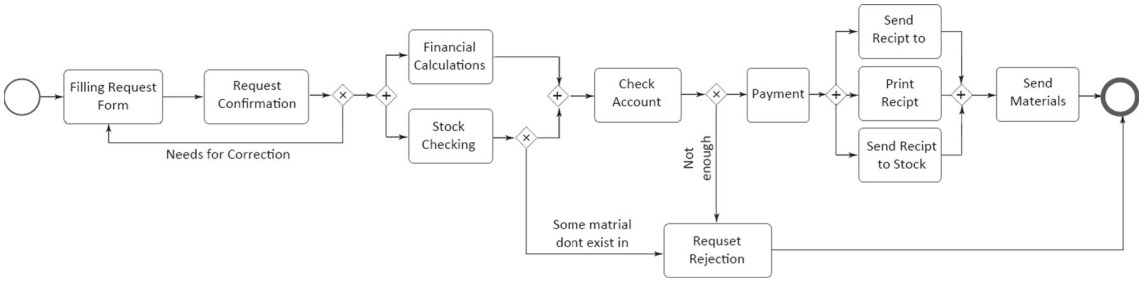
4.1 Bunge's ontology

As a foundation for our work, we are interested in the theoretical analysis of the various aspects of different process fragment definitions through Bunge's ontology. Bunge, well adopted in information systems research (Wand and Weber 1995), which models the world, as a world of systems (Bunge 1977). In Bunge's ontological model, the "*world is made up of substantial things which possess properties*" (van der Aalst 2012). Since Bunge's ontology provides concepts for representing real world phenomena, it seems appropriate to be used for analyzing process models in real software systems (Bunge 1977).

Bunge's ontological model contains four essential concepts: thing, property, state and event. Table 1 provides an overview of its fundamental ontological constructs (Wand and Weber 1995). Things are elementary units and can be specific instances of person, building, car, and book. A "*property*" of a thing can be any intrinsic or mutual (meaningful only in the context of two or more things) feature of it such as height, color, weight, and shape. A "*state*" is the vector that contains the values of all property functions of a thing. A "*state law*" makes a restriction on the values of the thing's properties to a subset that is considered lawful. The set of things states that conform to the state laws of the thing are called the "*lawful state space*". An "*event*" occurs when a change in the state of a thing can be seen. The set of all possible events of a thing is called its "*event space*". A set of things that possess a common property is termed a "*class*". Wand et al. have extended Bunge's ontology with 28 (real-world) constructs (Wand and Weber 1995). The fundamental Bunge's ontological constructs for our running example are shown in Fig. 1b.

4.2 Generic process model (GPM)

The Generic Process Model (GPM) provides process specification semantics based on ontological constructs (Soffer et al. 2010). GPM is considered to be a framework for reasoning about process models according to their real-world meaning based on Bunge's ontology. GPM has already been used for



(a) An example of purchasing process model.

Things	Properties (attributes)	State Space	Lawful Event Space
Specific instances of Request, Stock, Material, Employee, Applicant, Account	Request(ID, Date, Materials) Stock(ID, Materials, No), Material(ID, Name, Price), Employee(ID, Name, Grade), Applicant(Code, Requests), Account(ID, Credit, ApplicantID)	Request-Status(Filled, Confirmed, Rejected, Audited, Manager Confirmed) Material-Status (Available, Not Exist) Account-Status(A, B, C, D)	Request{ (Filled→Confirmed), (Filled→Filled), (Confirmed →Audited), (Confirmed→ Stock Checked), (Stock Checked→ Rejected), (Stock Checked →Manager Confirmation), (Audited →Manager Confirmation)}, Account-Status{(A→B), (B→C), (C→ B), (B →A)}

(b) Ontological representation of mentioned purchasing process.

Fig. 1 An example of purchasing process

various purposes such as analyzing the validity of process models, describing the conception of goals in business processes, and for interpreting control flow elements (Soffer et al. 2010).

GPM focuses on the notion of *domain* through the use of Bunge's ontology concepts. A domain is represented by a set of state variables. The state of the domain can change for two reasons: first, due to the internal events, which occur within the domain and second by external events, which are stimulated from outside the domain (Soffer et al. 2010). A state that changes due to actions in the domain is called "*unstable state*" and a state that only changes due to the actions of the domain's environment is called "*stable state*" (Soffer et al. 2010). A complete definition of GPM can be found in (Soffer et al. 2010).

In (Soffer and Yehezkel 2011), a process model is defined based on GPM as follows:

Definition 1 (Soffer and Yehezkel 2011): *a process model is a tuple <I , G , L , E> where,*

I: *is a subset of unstable states of the domain* (*initial states*),

G: *is a subset of stable state* (*goal set*),

L: *is the set of state transitions*,

E: *is a set of relevant external events.*

One of the advantages of this definition is that GPM explicitly addresses the goal of a process and checks the validity of a process design against its defined goal (Soffer and Yehezkel 2011). In this paper, we will use GPM for theoretically analyzing process fragments.

Table 1 Fundamental ontological constructs in the Bunge-Wand-Weber representational model (Larosa et al. 2013a)

Fundamental Onto-logical Construct	Description
Thing	The basic unit in the Bunge's ontological model is thing and can be in two types: simple and compound. A compound thing is made up of other things.
Property	Each property can be described using a function (called attribute function) that maps the thing into some values. Things can have several properties.
State	The state of a thing is a vector of values for each property functions of that thing.
Transformation	A mapping from a domain containing states into a co-domain containing states is called transformation.

4.3 Process fragment definitions

In order to perform our theoretical analysis, we first systematically review some of the well-known process fragment definitions from the literature and discuss their pros and cons within the context of Bunge's ontology and GPM. By identifying the strengths and weaknesses of different process fragment definitions, we attempt to propose a definition that would cover the pros of the existing definitions and address their cons.

In order to identify the main work in process fragmentation, we first started by focusing on the main survey papers in this domain (Mancioppi and Danylevych 2012) (Reijers and Mendling 2008). We then gathered additional papers by searching for the keywords mentioned in these survey papers

on reliable databases including *CiteSeerX*, *ScienceDirect*, *ACM Portal*, *SpringerLink* and *IEEEXplore*. We then extracted and classified these papers based on their fragment definition and then selected only those papers that had been cited more than 10 times according to Google Scholar. The result is shown in Table 2.

In order to analyze these definitions and identify their pros and cons, we required some comparative criteria. In (Mancioppi and Danylevych 2012), several classification criteria for process fragmentation techniques are provided. These criteria provide a foundation for the classification of process fragmentation algorithms and can be useful for the evaluation of these techniques. For example, some of the criteria state *why is the process model fragmented*, *how is the fragmentation performed*, *who performs the fragmentation*, *when is the fragmentation performed in the process model lifecycle*, among others. Most of these criteria are independent of the fragment *definition* and are with respect to different aspects of the fragmentation *algorithms*. Therefore the criteria introduced in (Mancioppi and Danylevych 2012) cannot be directly applied in our classification, which focuses primarily on fragment definitions and not the algorithms.

Therefore, we consider three criteria, *structural restrictions for input/output process models*, *ambiguity* and *determinism.* We do not claim that these criteria are complete, but they can highlight some of the main weaknesses of the fragment definitions. These criteria are presented as research questions in the following:

Q1: *Does the fragment definition impose any structural restrictions on the input process model or the output fragments*?

Some definitions consider limitations on the input or output processes. For example, some require the process fragment to have no cycles or that all transitions be reachable from the start node.

Q2: *Does the fragment definition have any elements of ambiguity or leave room for different interpretations*?

Given some of the definitions do not have a theoretical representation and are written in natural language, there might be room for different interpretations of the definitions. For example, some definitions just state that the portions of process models that are suitable for reusability are fragments. Such definitions have ambiguity and therefore, could result in different interpretations.

Q3: *Is the definition precise and deterministic*?

This criterion specifies whether the definition will always guarantee the extraction of the exact same process fragments for the same input process model or not. This criterion is different from ambiguity. In the case of determinism, an implementation of a non-deterministic definition can be viewed as being not a *function* and for the same input, could produce different outputs. So, a definition might not be ambiguous but be non-deterministic.

In Table 3, the results of the evaluation of existing process fragment definitions based on the three criteria are shown.[1] As seen, most of the definitions are non-deterministic and have no structural restrictions. For example, the definition in Group 1 has no ambiguity and is clear, but is non-deterministic, because in this definition any part of a process can be considered as a fragment and for the same input process, different fragments can be produced. Group 2 has a more precise definition and places structural restriction on process inputs and requires them to be *connected*. This definition is clear and has no ambiguity. Since it uses precise label similarity, it is deterministic. The definition in Group 3 does not place a clear structural restriction on the input process and the definition is not clear which parts of the input process can be considered to be fragments. The definition in Group 4 states that fragments should be connected have no cycles, and have at least one activity and a single control flow linking up two distinct activities. This definition is unambiguous but is non-deterministic because it does not work as a function. The definition in Group 5 is unambiguous and has no structural restriction but again it is non-deterministic, since for the same input process different process fragments can be generated.

The definitions in Groups 6 and 7 are descriptive and contain ambiguity. Finally, in the definition of Group 8, a fragment is a connected graph (structural restriction) that has a start and end node and at least one activity and is not necessarily directly executable. So, this definition has no ambiguity. It is non-deterministic because with the same process we can have lots of fragments that have start and end nodes and be also connected.

We further analyzed the various fragment definitions and identified the top six features that they had in common as shown in Table 4. These features represent the characteristics of the process fragments that are described by the definitions. Feature F_1 is the most widely seen feature for fragments. This feature implies that each fragment must have a single input and single output. Features F_2 and F_6 are concerned with the concept of connectivity. A node belongs to a fragment if it has a connection with at least one node within that fragment. Feature F_3 has a semantics point of view. It implies that the nodes that have more similar labels have a higher probability of belonging to the same fragments (Reijers et al. 2011).

[1] The evaluation was conducted by investigating the coverage of the criteria for each definition. In the cases that there were uncertainties for the coverage, they have been discussed between the authors to reach conclusions about the coverage.

Table 2 A review of process fragment definitions

Group	References	Description
G1	(Leymann 1996; Basu and Blanning 2003; Vanhatalo et al. 2007, 2008)	Components with single input and single output control flow arc (SESE).
G2	(Reijers et al. 2011)	SESE components which have connected nodes and their nodes have high label similarity.
G3	(Seidita et al. 2010)	A portion of design process adequately created and structured for being reused during the composition and enactment of new design processes.
G4	(Eberle et al. 2009; Seidita et al. 2011; Zemni et al. 2012; Assy et al. 2013)	A connected portion of a process intended for reuse and contains no cycles and a single control flow linking up two distinct activities. It is made of at least one activity and several controls.
G5	(Tan and Fan 2007)	A partition of a workflow model, and consists of a source transition, all the transitions are reachable from the source transition, and all the linking places of these transitions.
G6	(Hens et al. 2014; Khalaf et al. 2008)	A logically different, smaller model part of input process model which extracted with the intention to distributing over different execution and controlling partners.
G7	(Mancioppi and Danylevych 2012)	A part of process models that contain process's elements such as activities, data flows, and controls.
G8	(Schumm et al. 2010)	A connected graph with significantly relaxed completeness and consistency criteria compared to an executable process graph which contains a process start or end node and at least one activity and is not necessarily directly executable.

Natural language processing (NLP) tools can be used for detecting the similarity of the labels of the nodes in the fragments. Feature F_4 creates a structural limitation on the fragments that they should not contain cycles. Feature F_5 implies that in each fragment at least one action needs to be done.

In Table 5, we show each feature against each definition group. In other words, Table 5 shows for each definition group, what the features of their fragments are. In this table, the rows show each definition group's id (from Table 3) and the columns show the features (from Table 4). From this table, it can be understood that Features F_1 and F_2 are the most frequent features for process fragments within the literature.

Now, we exploit the features of Table 4 in order to provide a formal definition for a process fragment as follow:

Definition 2 *A process fragment F is a directed graph described as* $F(A, G, R, s, e)$ *where*:

1) $R \sqsubseteq (A \times G) \cup (G \times A) \cup (G \times G)$,
2) $|A| \geq 1$,
3) $\forall\ t \in (A \cup G)\ \exists\ v_0 = s, v_1, v_2, \ldots, v_k = t\ ((v_{i-1}, v_k) \in R, 1 \leq i \leq k)$,
4) *if* $\ni n_1, n_2\ (n_1 = s, n_2 = s)$ *then* $n_1 = n_2$,
5) *if* $\ni n_1, n_2\ (n_1 = e, n_2 = e)$ *then* $n_1 = n_2$,
6) $\nexists t \in (A \cup G)\ ((v_t, v_i)$ *and* $(v_k, v_t) \in R)$ *and* $((v_j, v_{j+1}) \in R, i \leq j < k)$

In Definition 2, A is the set of activities, G represents gateways, R is the set of control flow relations, s is single input and e is single output nodes. In this formalism, $|A| \geq 1$ (line 2) ensures feature F_5 and the expression in line 3 enforces F_2 and F_6. Two if-clauses in this formalism (lines 4 and 5) together represent F_1 and the later expression is equal to F_4. In this definition we did not include feature F_3. The reason is that label similarity is based on the assumption that process names are always selected meaningfully and consistently (Reijers et al. 2011). For small processes or cross-organizational processes, this assumption is not necessarily always true.

Table 3 Analyzing various definitions of process fragments based on three criteria: structural restriction, ambiguity, and determinism

Group	Structural Restriction	Ambiguity	Non-Determinism
G1	–	–	■
G2	■	–	–
G3	–	■	■
G4	■	–	■
G5	–	–	■
G6	–	■	■
G7	–	■	■
G8	■	–	■

Table 4 All features extracted from various definitions of process fragments

F#	Features
F_1	Process fragment has single input and single output control flow arc (SESE)
F_2	Process fragment must be connected.
F_3	Fragment's node must have label similarity.
F_4	Process fragment should contain no cycles.
F_5	It is made of at least one activity and, of several control (dangling or not) and data flows.
F_6	All the transitions must be reachable from the source transition and all the linking places of these transitions.

Table 5 Extracted features against the various definitions

Definition Group ID	F1	F2	F3	F4	F5	F6
G1	■	–	–	–	–	–
G2	■	■	■	–	–	–
G3	–	–	–	–	–	–
G4	–	■	–	■	■	–
G5	■	–	–	–	–	■
G6	–	–	–	–	–	–
G7	–	–	–	–	–	–
G8	■	■	–	–	■	–

5 Analyzing process fragments using Bunge's ontology

In Definition 2, we included the most prominent unambiguous features identified in the definition groups for process fragments. This definition is unambiguous in that it does not contain any ambiguous feature, has no structural limitation on the input and output and is deterministic. Nevertheless, it would be quite difficult to employ it for identifying process similarity or multi-process analysis. This is due to the following reason: in Definition 2, two fragments within two process variants are shared, if and only if they are identical (Pourmaoumi et al. 2014). For example, in Fig. 2, fragments 1, 2, 3 and 4 are not identical (Based on all of the definitions in Table2). Based on *Definition 2*, since the *R* set of these four fragments are not identical, they cannot be captured as common fragments between two process models. Therefore, in these definitions, common fragments are either complete matches or not a match at all.

In the example shown in Fig. 2, all fragments try to check a "*purchasing request*"; however, each achieves this in a different way. All fragments start with "*filling a request*" activity and end with "*management confirmation*" activity. As seen in Fig. 2, activities "*financial calculation*" and "*stock checking*" can be placed in different relations to each other. This difference can be probably due to various branch managers' choices. In practice, these fragments are performing a similar task and would be considered to be very similar fragments that only have minor structural variances.

In the next section, we will theoretically define the notion of process fragments using GPM and derive a new conceptual definition for a process fragment that would address the above issue, among others.

5.1 Process fragments based on GPM

Using Bunge's ontology, it can be inferred that there is a mapping between information systems and concepts in the real world. The concepts (domain) in the real world are made of sub-concepts (sub-domain). The structures and processes of the real world can be represented by constructs in the ontological models. So, we can define sub-processes or fragments of a process model based on its ontological mapping in the real world. In (Soffer et al. 2010), a sub-domain is defined based on GPM as follows:

Definition 3 (sub-domain) (Soffer et al. 2010): *A sub-domain is part of the domain described by a subset of the set of domain state variables.*

It must be noted that using this definition, there might be many ways to divide a domain into sub-domains and not all of them will be meaningful. In the real world, partitioning of a domain into independent sub-domains is possible. Partitioning of a domain into independently-behaving sub-domains is the result of different actors existing in the domain. In (Soffer et al. 2010), an independent sub-domain is defined as follows:

Definition 4 (*independent sub-domain*) (Soffer et al. 2010): *A sub-domain will be called independently behaving* (*or independent*) *in a given state* (*of the sub-domain*) *if the law projection on the sub-domain is a function for this state.*

In this case, the law projection is a function that depends only on the sub-domain's state variables. In the other words, the meaning of *Definition 4* is that each sub-domain behaves

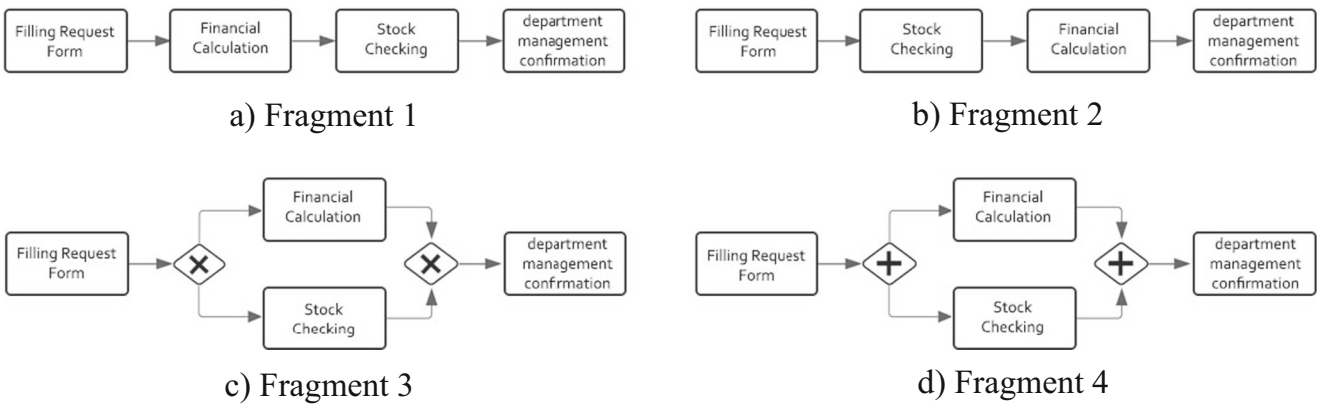


Fig. 2 Four fragments of *purchasing process* with different structures which operationally are identical

independently and ends on a stable state of the sub-domain. This stability can lead to the stability of the whole domain in the goal states or other states (Soffer et al. 2010). So, in this definition the stable states (initial and goal states) of the sub-domain are important. We use *Definition 4* for creating a mapping between sub-domains in the real world and process fragments in information systems. So we can define process fragments based on GPM as follows:

Definition 5 (*process fragment*): *A subset of a process model is called a process fragment if it starts and ends with stable states and there exists at least one transformation inside it.*

Now, using GPM-based definition of a process fragment (*Definition 5*), we can develop a model for process fragments that does not necessarily require a complete and exact match for finding similar fragments. As we will show, this allows us to assess the similarity of two process fragments beyond a binary match or no-match. In the next section, the new notion for process fragments, building on Definitions 2 and 5, is introduced.

5.2 Morphological fragments

In *Definition 5*, a process fragment is defined based on GPM. The main focus of this definition is on the stable states of fragments as a point of separability. Unstable states inside the sub-domain are responsible for the behavior of the sub-domain. It can be understood that two fragments f_1 and f_2 have equal behavior if they have equal stable states ($I_1 = I_2$ and $G_1 = G_2$) and also have equal *transformation* sets (In (Soffer et al. 2010) transformation set of a process is equal to its activities set). The order of transformation sets is equal to the way in which processes execute. In the other words, with equal stable states and transformation sets, different transformation orders (law) for two fragments show that the fragments do similar tasks in different ways. In other words, they represent the same behavior but not necessarily the same structure. Now, we can define some measures for extracting common fragments from a family of process variants based on this observation.

Definition 6 *Two fragments* $f_1\left(A_{f_1}, G_{f_1}, R_{f_1}, s_{f_1}, e_{f_1}\right)$ *and* $f_2\left(A_{f_2}, G_{f_2}, R_{f_2}, s_{f_2}, e_{f_2}\right)$ *are behaviorally similar, called morphological fragments, iff*:

$$s_{f_1} = s_{f_2} \;\&\; e_{f_1} = e_{f_2} \;\&\; A_{f_1} = A_{f_2}$$

In this definition, the start and end points are equal to the stable states in the *Definition 5* and the relation between internal nodes is ignored given the above explanation. The reason is that two fragments that have equal start/end points and have equal activities sets, and performing similar tasks, could be considered to be very similar fragments that only have structural variances hence, the term *morphological fragment*. For example in Fig. 2, there are four fragments that check the purchasing request of a customer in different ways. These differences can affect the efficiency and effectiveness of the whole process. So detecting these fragments as common fragments among a family of process variants can increase the reusability and lead to added value for organizations. All of these four fragments have equal start/end points and their internal activities are identical. So they are morphologically identical. The different relationships among internal activities show the different ways of performing the same task.

Definition 6 allows the identification of process fragments that are behaviorally similar but not structurally identical. It is now possible to define the degree of morphological similarity based on the degree of the transformation spaces similarity of the fragments. But before that, first we use the definitions mentioned in (Asadi et al. 2012) for a set of general similarity patterns between two sets of phenomena (By phenomena we refer to any possible observation that can be made about the domain or part of it). Assume $A = \{a_1, a_2, \ldots, a_n\}$ is a set of phenomena belonging to domain D_1 and $B = \{b_1, b_2, \ldots, b_m\}$ is a set of phenomena belonging to domain D_2. We can see one of the following situations with respect to similarity between these two sets:

Definition 7 (Equivalent Set of Phenomena) (Asadi et al. 2012): *Phenomenon* A_1 *is equivalent to* A_2 (*denoted as* $A_1 \equiv A_2$), *if and only if there is a unique mapping between elements in* A_1 *and elements in* A_2.

Definition 8 (Similar Sets of Phenomena) (Asadi et al. 2012): *Phenomenon* A_1 *is similar to* A_2 *with respect to p (denoted as* $A_1 \equiv A_2$*) if and only if there is a subset of* A_1 *(i.e.,* $A_1' \subset A_1$*) and of* A_2 *(i.e.,* $A_2' \subset A_2$*) which are equivalent* $A_1' \equiv A_2'$. *p is the equivalent subset i.e.* $p = A_1' = A_2'$.

Definition 9 (Completely Dissimilar Set of Phenomena) (Asadi et al. 2012): *Phenomenon* A_1 *is completely dissimilar to* A_2 *(denoted as* $A_1 \neq A_2$*) if and only if there are no subsets of* A_1 *(i.e.,*$A_1' \subset A_1$*) and of* A_2 *(i.e.,* $A_2' \subset A_2$ *) that are equivalent.*

Based on Definitions 7–9, we can define the following similarity patterns between two different sets of phenomena:

- *Full similarity double side*: when the sets A_1 and A_2 are *equivalent* (i.e., $A_1 \equiv A_2$).
- *Full similarity one side*: when the sets A_1 and A_2 are *similar* (i.e., $A_1 \cong_p A_2$) and when we have either $A_1' \subset A_1$ and $A_1' \equiv A_2$ or $A_2' \subset A_2$ and $A_2' \equiv A_1$.
- *Partial similarity*: when the sets A_1 and A_2 are *similar* (i.e., $A_1 \cong_p A_2$) and there is no subset of one set that is equivalent to the other set.
- *Complete Dissimilarity*: when two sets are *completely disjoint*.

All of the above similarity patterns can occur between any two sets in the real world. In the case of *partial similarity* we can define the amount of similarity as:

$$S_p(A_1, A_2) = \frac{A_1 \cap A_2}{A_1 \cup A_2} \quad (1)$$

The value S_p is a positive value between 0 and 1 where values of S_p closer to 1 represent higher similarity between A_1 and A_2.

In Table 6, we show all of the similarity patterns that can happen between the two subsets of processes based on fundamental ontological constructs. We just show *stable state space* and *transformation space* constructs based on GPM, because we intend to analyze behavioral similarities between process fragments and structural similarity is not our concern in this paper.

In Table 6, the first row is equivalent to our definition for morphological fragments. It shows that two fragments are behaviorally identical if their *stable state space* and *transformation sets* are equivalent (have *full similarity double side* pattern). In other words, the order of events does not matter. The only important point is that their internal event space (*transformation set*) is equivalent and they have equal start and end nodes (*stable state space*). So, the first row of highlighted part of Table 6 describes full similarity. Also, we can define some degree of similarity between morphological process fragments. If T_1 and T_2 be *transformation* sets of process fragments F_1 and F_2 respectively, then we can define the degree of similarity between two process fragments by:

$$D_m(T_1, T_2) = 2 \times \frac{\left|T_1 \cap T_2\right|}{\left|T_1\right| + \left|T_2\right|} \quad (2)$$

The two fragments F_1 and F_2 are fully similar if their degree of the similarity, called degree of morphologicality in this paper, is equal to 1; $D_m(T_1, T_2) = 1$. The reasons for why we do not consider the other parts of Table 6 as morphological fragments are discussed in Section 8.1.

Figure 3 shows four structurally different fragments that deal with the purchasing process. All fragments start and end with the same activities "*filling request form*" and "*department manager confirmation*", respectively. Regardless of their order or composition, the set of internal activities of all these fragments are identical. However, the order and relationships of them are different. All of them check the validity of the purchase request and check the existence of the goods in stock, each in their own way. These differences can be due to various reasons and can affect efficiency and performance of the process. For example, one manager may decide that checking the stock and doing the financial calculations should be done in parallel and another might decide that it should be done sequentially. Undoubtedly this decision will affect various execution characteristics of the process, e.g. time to completion. By identifying similar morphological fragments, one can determine various quantifiable measures for these fragments and use them for the purpose of process improvement. For instance, given two fragments that start and end with the same activities, one can decide to replace one morphological fragment with another similar morphological fragment in the hopes to reduce the time to completion. As well, other criteria such as complexity, and cost can be used for optimizing processes through morphological fragments. It should be noted that existing definitions of process fragments as given in Table 2 do not support for this important point pertaining to composability, which is supported by morphological fragments.

Table 6 Similarity patterns between two sub-processes

Stable State	Transformation Space	Class Name
Full Similarity-Double Side	Full Similarity-Double Side	Full Similarity-Double side among Process
	Full Similarity-One Side Partial Similarity	Partial Similarity among Process
	Dissimilarity	Complete Dis-similarity among process
Full Similarity-One Side	Full Similarity-Double Side Full Similarity-One Side	Complete Dissimilarity among process
	Partial Similarity	
	Dissimilarity	
Partial Similarity	Full Similarity-Double Side Full Similarity-One Side	Complete Dissimilarity among process
	Partial Similarity	
	Dissimilarity	
Dissimilarity	Full Similarity-Double Side Full Similarity-One Side	Complete Dissimilarity among process
	Partial Similarity	
	Dissimilarity	

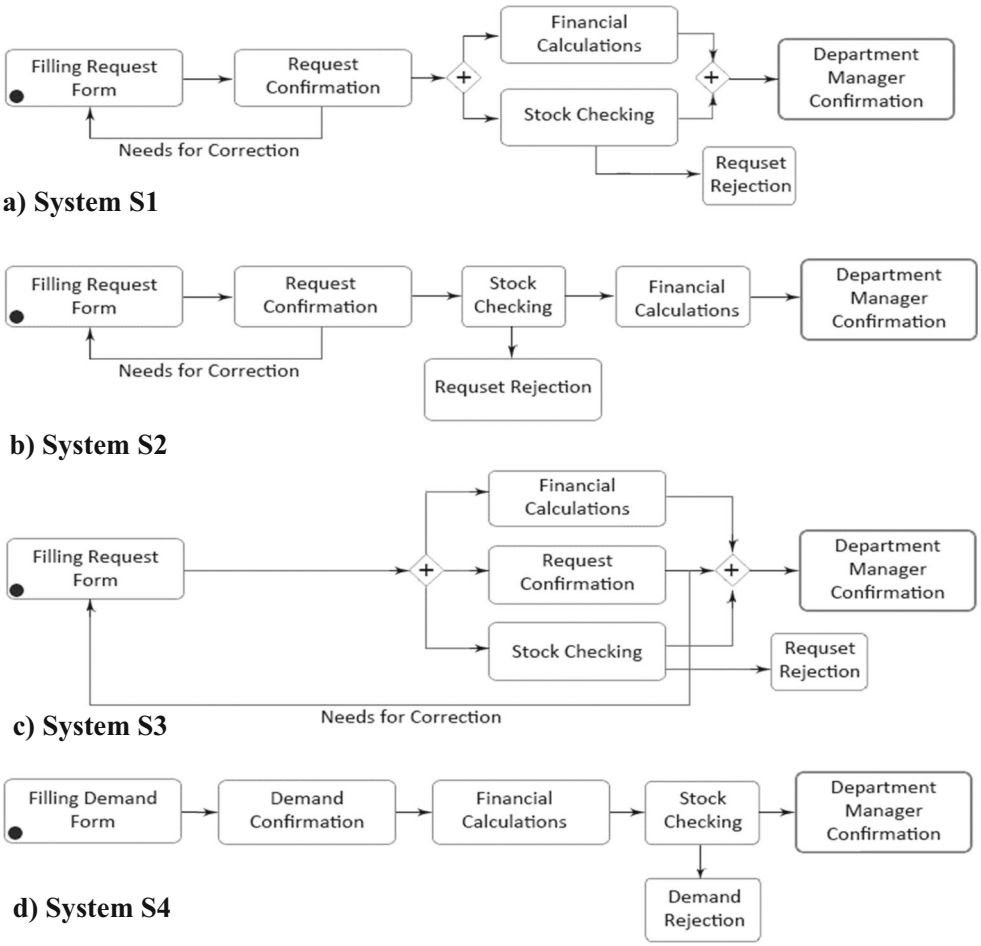


Fig. 3 Four similar fragments corresponding to purchasing systems

6 Extracting morphological fragments

Now that an unambiguous and deterministic definition for morphological fragments have been proposed, we present an algorithmic approach for mining such morphological fragments in a process family from a collection of event logs. As mentioned in the related works section, most of the existing fragmentation algorithms are based on the structure of the process models and not based on event logs. We are interested in identifying fragments directly from a collection of event logs. Our goal is twofold. First, we are interested in reducing the detection error rate, i.e., the number of misidentified fragments. In each process model, there might be some exceptional or rare parts. Working with event logs directly can identify exceptional/rare activities and based on the occurrence frequency of such activities, we can place less importance on the less likely activity sequences. Second, we would like to improve the fragmentation identification speed of the algorithm. Given intermediate node structures is not important in the morphological fragment definition, if fragments could be directly extracted from unstructured event logs, there is no need to consider activity relationships and therefore it can significantly speed up the extraction process.

We divide our proposed algorithm into three main phases: *i*) preprocessing; *ii*) generating successive L-grams[2] from each set of event logs and *iii*) finding common L-grams between different organizations' event logs. In the next section, a detailed description of these phases is given.

6.1 Preprocessing

Each process variant may contain millions of event traces, most of which are similar. We denote the finite non-empty set of traces of a process variant of organization i as TS^i. In this paper, we assume that process related data is available. In (Li et al. 2015), the authors have proposed an intelligent approach for extracting this data. Utilizing all of the redundant records in all steps of extracting common fragments is not efficient. Therefore, as the first step, we extract distinct traces and calculate their occurrence frequency. Having the occurrence frequency of each trace, one can easily filter out noise or rare patterns. We refer to the occurrence frequency of trace t as

[2] This is the same as the concept of n-grams in computational linguistics but we use the term L-grams as we reserve variable n to represent the number of traces in an event log.

local frequency, and denote it by LF_t^i. The higher the local frequency of a trace is, the more important it would be and similarly the lower the local frequency of a trace, the more probable it would be that it is noise or just a rare pattern.

Now, if an activity is observed more than once within a trace, then there might be a loop inside that process model (Wen et al. 2007). So, loops can be detected by observing repetitive activities inside traces. After detecting loop regions, we detach each repetition and create a new trace using that part. In Fig. 4.a, an example process model, which is selected from the IBM BIT[3] dataset is shown. This example process model contains a loop. In Fig. 4.b a part of related event logs of this process after performing the first phase of preprocessing is shown. The occurrence frequency of each trace is separated by colon. Figure 4.c shows the final extracted event logs set after executing the preprocessing step. As seen in the figure, this set contains four different executions of the example process model (Fig. 4.a). So, we successfully reduced thousands of records within the event log into four traces.

Giving a finite set of traces TS^i, the preprocessing phase will produce a much smaller set of traces, which we refer to as TS_p^i.

6.2 Generating *L*-grams from each set of event logs

For the sake of simplicity and without loss of generality, we explain the method with an example. Imagine there are four organizations each of which executes the purchase order process. Each organization has its own event log dataset. This is shown in Fig. 5. For instance, we assume that all of the organizations have operationally identical fragments, but each performs it in slightly different ways. The corresponding fragment of each traces set is shown below them in the fig. We will first execute the preprocessing steps on TS^i $(1 \le i \le 4)$. Then using the obtained $TS_p^i (1 \le i \le 4)$, we generate all L-grams $(2 \le L \le ||t_{\max}||)$ for each TS_p^i. Here, we consider the minimum size of 2 for L (we consider minimum size of 2 for each fragment) and the maximum value equal to the length of longest trace ($t_{\max}$). We take a window of size L around all event traces and generate all possible L-grams. This is shown in Fig. 6. In this figure, three traces of O_1 for "*purchasing process*" are shown: EHABCDFG, HEABCDFG and HEABCDGF. Here, A is "*filling request form*", B is "*financial calculation*", C is "*stock checking*" and D is about "*payment*". Red numbers are numerical values, which have been mapped to each 4-g. Other characters E, F, G and H represent other activities such as "*print receipt*" or "*send receipt to finance*". Taking a window of 4 ($L = 4$) around EHABCDFG, we can generate five 4-grams: EHAB, HABC, ABCD, BCDF and CDFG. In a similar vein, we can generate five 4-grams for HEABCDFG: HEAB, EABC, ABCD, BCDF and CDFG. Likewise, five 4-grams for HEABCDGF: HEAB, EABC, ABCD, BCDF and CDFG. All of the 4-g for the four organizations' event logs are shown in Fig. 6. It should be noted that since the relations between the intermediate activities do not matter in our morphological fragment definition, we do not need to calculate 4-combinations or 4-permutations in our work. With this approach we are able to generate L-grams with any size L.

6.3 Detecting common *L*-grams between organizations' event logs

After generating all L-grams $(2 \le L \le ||t_{\max}||)$, we will now need to detect shared L-grams between process variants that respect the properties outlined in Definition 2. We present a fast and accurate method for identifying such common L-grams. For the sake of simplicity, we will follow the previous running example. Assume that there are 8 activities in all process variants named: A, B, C, D, E, F, G and H. We map each activity to a numerical representation, which is a power of 10 as shown in Table 7. Now, we assign each trace t to a numerical value using the following formula:

$$\text{value}(t) = \varepsilon \times A_I + \left(\sum_{j \neq I, j \neq O} A_j \right) + \kappa \times A_O \tag{3}$$

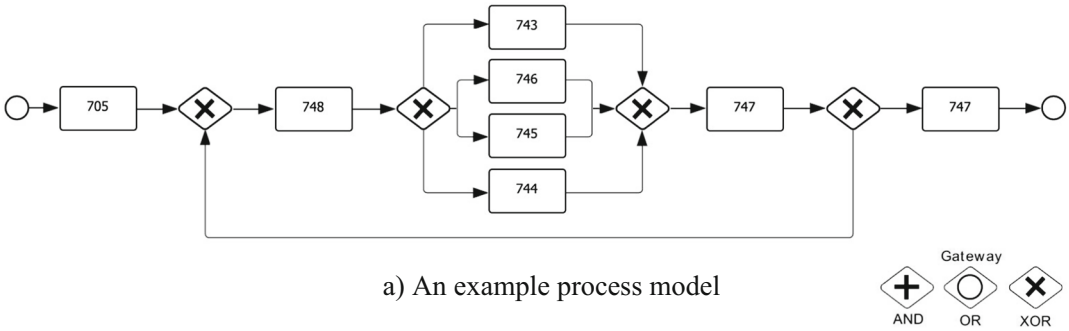
where A_I is the numerical representation of the first activity of the trace and A_O is the numerical representation of the last activity and $A_j (j \neq I, j \neq O)$ is the numerical representation of other activities in the trace.

Lemma 1 Using Eq. 1, we are able to develop a unique numerical representation for each L-gram.

Proof According to Eq. 3, there are two constants ε and κ which distinguish the start and end of an L-gram. Since the other parts of the L-gram (all activities except start and end) have a constant coefficient of 1, the two constants ε and κ which can be any distinct numbers between 2 and 9 distinguish the start and end of L-gram. On the other hand, according to the definition of morphological fragments, the presence of the exact set of intermediate activities is required but their order can be ignored. Therefore, the second term of Eq. 3 ensures that the same set of activities are present in the morphological fragment while the first and last terms of the equation ensure that the start and end of the morphological fragments are identical.

According to Definition 2, ABCD and ACBD traces, which can be considered operationally similar, will have the same value. The reason is that each activity regardless of its position in the trace has a unique value (a power of 10). We

[3] BIT-Process Library-Release 2009



a) An example process model

```
[705, 748, 744, 747, -->, 748, 744, 747, -->, 748, 745, 747, -->, 748, 743, 747, 706]:2
[705, 748, 745, 747, -->, 748, 745, 747, -->, 748, 745, 747, -->, 748, 744, 747, -->, 748, 743, 747, 706]:1
[705, 748, 745, 747, -->, 748, 744, 747, -->, 748, 745, 747, -->, 748, 746, 747, 706]:3
[705, 748, 743, 747, -->, 748, 744, 747, -->, 748, 744, 747, -->, 748, 746, 747, -->, 748, 743, 747, 706]:2
[705, 748, 746, 747, -->, 748, 745, 747, -->, 748, 744, 747, -->, 748, 744, 747, -->, 748, 743, 747, 706]:1
[705, 748, 746, 747, -->, 748, 746, 747, -->, 748, 746, 747, -->, 748, 746, 747, 706]:4
[705, 748, 743, 747, -->, 748, 743, 747, -->, 748, 744, 747, -->, 748, 744, 747, 706]:2
[705, 748, 744, 747, -->, 748, 745, 747, -->, 748, 743, 747, -->, 748, 746, 747, -->, 748, 746, 747, 706]:2
[705, 748, 744, 747, -->, 748, 746, 747, -->, 748, 743, 747, -->, 748, 743, 747, -->, 748, 746, 747, 706]:1
[705, 748, 746, 747, -->, 748, 744, 747, -->, 748, 744, 747, -->, 748, 745, 747, 706]:4
[705, 748, 746, 747, -->, 748, 743, 747, -->, 748, 746, 747, -->, 748, 743, 747, 706]:2
[705, 748, 744, 747, -->, 748, 744, 747, -->, 748, 745, 747, -->, 748, 744, 747, -->, 748, 743, 747, 706]:3
```

b) Part of event logs after preforming first phase of preprocessing

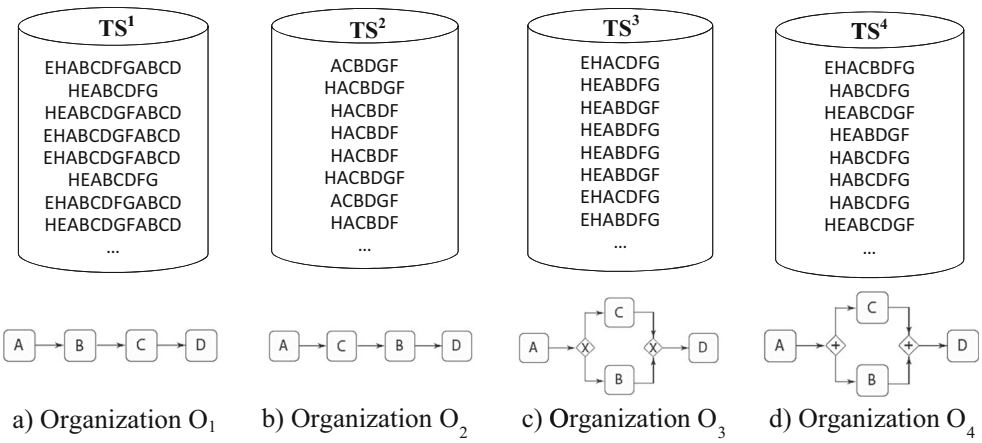
```
[705, 748, 743, 747, 706]:4424
[705, 748, 746, 747, 706]:4583
[705, 748, 744, 747, 706]:4544
[705, 748, 745, 747, 706]:4449
```

c) Final trace set which extracted after performing preprocessing

Fig. 4 An example of performing preprocessing steps

have added two different coefficients for the first and last activities. The important point is that the coefficients should make a difference between the first and last positions and the other positions in a trace. An example of mapping traces to numerical values is shown in Table 8 by assuming ε is 2 and κ is 3. The two constants ε and κ can be any value as long as they distinguish between the first and last positions and the rest of the positions and they should not be a power of 10. As shown in Table 8, ABCD and ACBD traces have equal numerical values. DBCA has a different value compared to ABCD because their first and last activities are different.

Algorithm 1 shows our proposed approach for extracting morphological fragments from multiple event logs. Lines 1 to 4 are preprocessing steps. In line 1 and 2, redundant traces are detected and removed. For this purpose, at first we calculate local frequency LF_t^i of each trace t in its own process variant event logs i. The traces that have $LF_t^i < \beta$ would be considered as noise and would be removed. In line 3 and 4 loops are handled. Line 5 generates all L-grams ($3 \leq L \leq |t|$) for each trace t. Line 6 generates a value for each L-gram according to Eq. 3. In line 7, we calculate the global frequency of each trace among all variants' event logs. Finally, in line 8 we



a) Organization O_1 b) Organization O_2 c) Organization O_3 d) Organization O_4

Fig. 5 Event logs from four different organizations

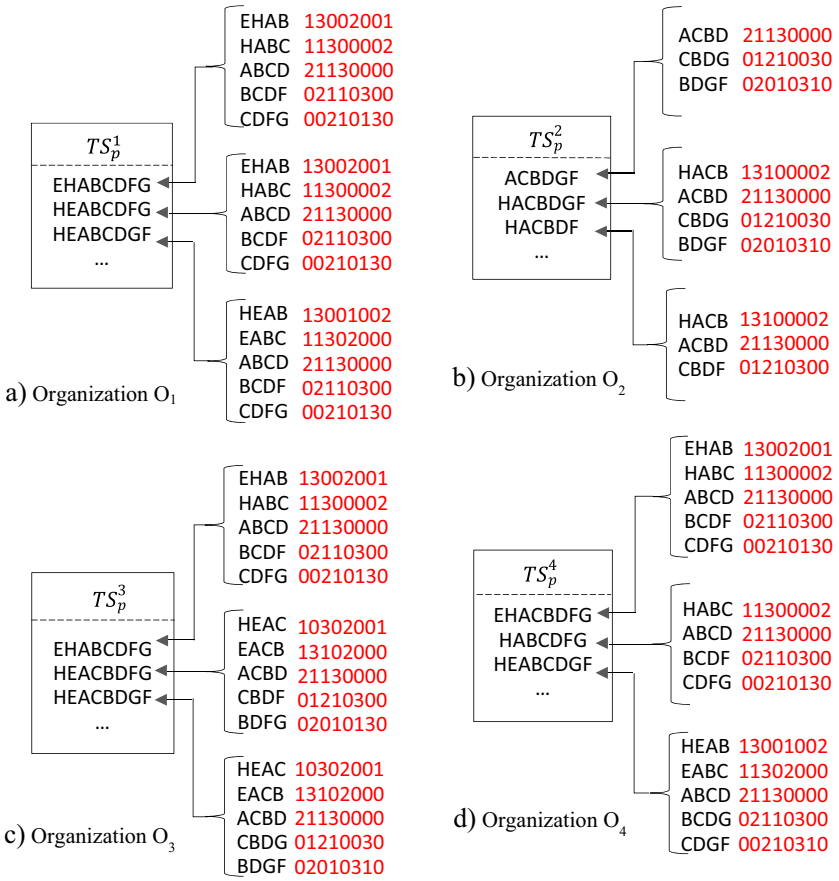


Fig. 6 Generating 4-g for traces existing in organization event logs set for *purchasing process*. In this example, *A* is "*filling request form*", *B* is "*financial calculation*", *C* is "*stock checking*" and *D* is about "*payment*". Red numbers are numerical value which have been mapped to each 4-g. Other characters *E*, *F*, *G* and *H* represent other activities such as "*print receipt*" or "*send receipt to finance*"

return all *L*-grams that have global frequency of higher than 1 as morphological fragments. The higher the global frequency of the extracted morphological fragments are, the more reusable they will be.

Algorithm 1. Proposed Algorithm for Extracting Morphological Fragments

Input: A collection of event logs TS^i.
Output: Set of morphological fragments.

1 For dataset *i*, extract distinct traces *t* and calculate their occurrence frequency LF_t^i.
2 In each dataset *i*, delete traces *t* where $LF_t^i < \beta$.
3 Detect loop regions by observing repetitive activities inside traces.
4 For each dataset *i*, create TS_p^i by detaching each repetition area and create a new trace by using that part.
6 For each trace *t* in TS_p^i, generate all *L*-grams LG_t^i where $3 \leq L \leq |t|$.
7 For each *L*-gram *L* in LG_L^i, generate a value using Eq. 3: $value(L) = \varepsilon \times A_I + \Sigma_{j=I,I=I} A_j + k \times A_O$.
9 For each *L*-grams calculate its global frequency GF_L among all LG_L^j that have the same value.
11 Return all *L* which $GF_L > 1$.

At some point in time, a process might have multiple overlapping fragments, each of which might be shared with other processes. For example, consider two traces "ABCDEF" and "ABCHBCD". With a quick glance, one can find a common fragment "ABC" between these two traces. By taking a closer look, one can also see that there is another common fragment, "BCD", which will not be identified if "ABC" is immediately extracted as a common fragment. Since in our approach we first generate all possible *L*-grams for each trace, using the global frequency of each candidate fragment, we can select the candidate that has the higher global frequency. So in this example, between "ABC" and "BCD", we can select the candidate, which has the higher commonality with the other processes.

6.4 Extracting common fragments with flexible degree of morphologicality

Given the flexibility of the morphological fragment definition, the identification of such fragments can lead to insights for organizations that cannot be otherwise obtained if a strict fragment definition is employed. Under real world scenarios, the number of *exact* fragments that can be mined across different

Table 7 Mapping of activities to a unique value as a power of 10

Activity	A	B	C	D	E	F	G	H
Value	10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0

Table 8 An example of mapping traces to numerical values

Trace	Mapping Calculation	Numerical Value
ABEG	$2 \times 10^7 + 10^6 + 10^3 + 3 \times 10^1$	21,001,030
ABCD	$2 \times 10^7 + 10^6 + 10^5 + 3 \times 10^4$	21,130,000
ACBD	$2 \times 10^7 + 10^5 + 10^6 + 3 \times 10^1$	21,130,000
DBCA	$2 \times 10^4 + 10^6 + 10^5 + 3 \times 10^7$	31,120,000

organizations may not be numerous; therefore, one can use the degree of morphologicality (α in Eq. 2) in order to relax the requirements to some extent. Two fragments f_i and f_j are morphologically *identical* if $D_m(f_i,f_j)$ is equal to 1. Likewise, two fragments are completely dissimilar if $D_m(f_i,f_j) = 0$. Otherwise, there is a degree of morphological similarity where degrees closer to 1 represent higher similarity between the identified fragments.

In order to identify morphological fragments that have the same degree of morphologicality, we need to extend our proposed method in Algorithm 1 to support for some degree of variation. To this end and in order to identify morphological fragments that have a certain degree of morphologicality, we need to measure the similarity between the identified fragments and cluster the fragments into groups of similar fragments. Clustering techniques are generally divided into three types: partitional, hierarchical and overlapping (so-called non-exclusive, alternative clustering or multi-view clustering) (Hruschka et al. 2009). In partitional clustering, each object belongs to exactly one cluster. Hierarchical clustering is a nested sequence of partitional clustering in which each object that belongs to a child cluster also belongs to the parent cluster. Overlapping techniques can be soft where each object can belong to more than one cluster or fuzzy in which each object may belong to one or more clusters with different degrees of membership (Hruschka et al. 2009).

Regardless of the type of the clustering algorithm, the main goal of clustering is to maximize the homogeneity within each cluster and the heterogeneity amongst clusters (Hruschka et al. 2009). Therefore, most of the clustering algorithms are based on the distances of data objects. In our case, we should additionally consider the structural similarity (degree of morphologicality) of process fragments. Therefore, it is not possible to employ an existing clustering technique for our purpose. This is primarily due to the fact that the clustering method that we require needs to consider fragment similarity as well as structural constraints when building the fragment clusters. For example, consider a scenario where we are interested in identifying all the morphological fragments that have a degree of morphologicality of more than 90 % from the processes in Fig. 7. In this example, Fragment 2 does not include one activity of Fragment 1 (*request confirmation*). So, the degree of morphologicality of $f_1$and f_2 is: $D_m(f_1,f_2) = 2 \times 5/(5+6) = 0.909$ and they can form a cluster. Likewise, Fragments 1 and 3 have a high degree of morphologicality because their degree of morphologicality is $D_m(f_1,f_3) = 2 \times (5/5+6) = 0.909$ and hence can constitute a cluster. However, Fragments 2 and 3 have a lower degree of morphologicality since they have two differences on activities *request confirmation* and *activity ask for payment* and their degree is $D_m(f_2,f_3) = 2 \times 3/(5+5) = 0.6$. Therefore f_2 and f_3 cannot be placed together with Fragment 1 in the same cluster with the equal degree of morphologicality. This example shows that the degree of morphologicality is not a transitive relation and therefore the adoption of traditional clustering techniques that assume inherent transitivity of the distance function is not applicable in our work.

Algorithm 2. Proposed Clustering Algorithm for Fragment Extraction

Input: Set of all L-grams L^*.
Output: Set of clusetrs C^* that contain fragments with same degree of morphologicality.

1 For each L-grams $g_i, g_j \in L^*$ calculate their degree of morphologicality $D_m(g_i, g_j)$.
2 Create a cluster C_i for each L-gram $g_i \in L^*$.
3 For each $g_i \in L^*$ and $C_k \in C^*$ if $\forall g_j \in C_k$, $D_m(g_i, g_j) > \alpha$ then create a new cluster C_{new} and add g_i and C_k to C_{new}
4 Resolve cluster C_i in C_j when $C_i \subseteq C_j$.
5 Remove clusters C_i which $|C_i| \leq 1$ (clusters which contain just one fragment).

The following definition provides the characteristics of our desirable clustering technique:

Definition 10 *A set of L-grams* C *is a* cluster of fragments with the degree of morphologicality α *iff*:

$$\forall g_i, g_j \in L^*, D_m\left(g_i, g_j\right) \geq \alpha$$

In order to develop a clustering technique that has the characteristics of Definition 10, we have proposed Algorithm 2 for clustering the common morphological fragments, which have a degree of morphologicality of higher than α. In the first step (Line 1), we create one cluster for each L-gram g_i. Once the clusters are created for each L-gram, we calculate the pairwise degree of morphologicality $D_m(g_i, g_j)$ of all pairs of L-grams g_i and g_j (Line 2). Once the pairwise degrees of morphologicality are calculated, our objective is to merge those clusters that have degrees more than α. For each $g_i \in L^*$and each cluster $C_k \in C^*$ if the degree of morphologicality of $g_i$with all of existing L-grams in C_k is more than α, then we create a new cluster C_{new} and add g_i and C_k to C_{new} (Line 3). Finally, we identify and remove those clusters which are complete subsets of other clusters (Lines 4 and 5). The result clusters set C^*, contains clusters which have morphological fragments with a degree of morphologicality of more than α.

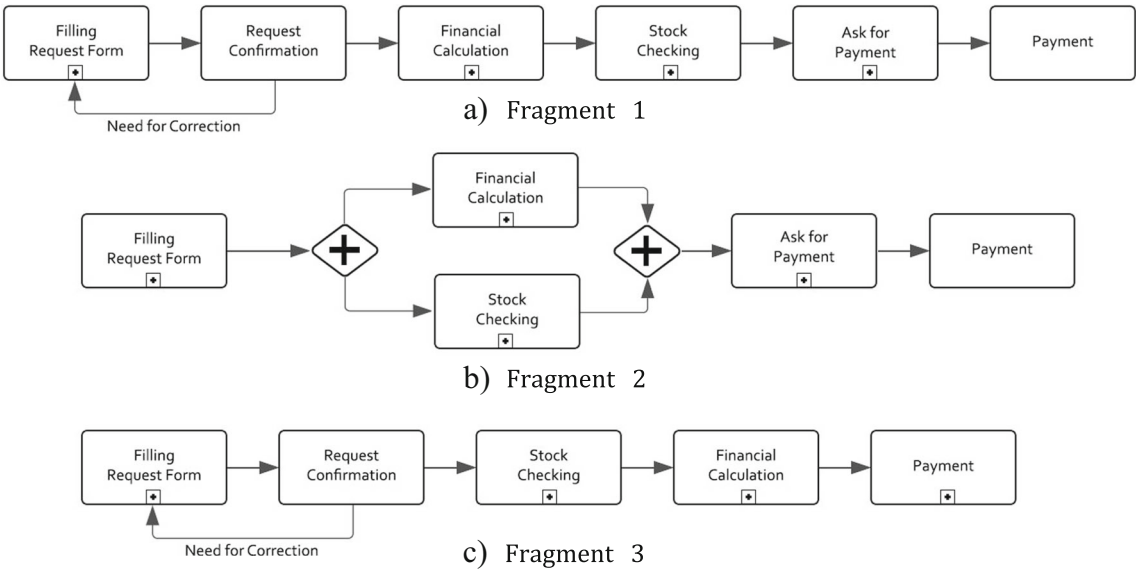


Fig. 7 Sample fragments that show a transitive morphological similarity between fragments does not exist

7 Evaluation

7.1 Dataset

We evaluated our proposed algorithm using two sets of datasets: two collections of the IBM BIT process library (Fahland et al. 2011), namely collections A and B3, as well as process logs that were generated with the process variant generator tool (Pourmasoumi et al. 2015b). There are 5 collections A, B1, B2, B3 and C in the IBM BIT library. In (Fahland et al. 2011), these collections were analyzed and it was concluded that collections B1 and B2 are earlier versions of B3, and collection C is a mix of models from different sources and as such it does not contain any clones. Based on the conclusions reported in (Fahland et al. 2011) and as suggested in that paper, we selected A and B3 from the IBM BIT process library. According to (Fahland et al. 2011) and also our own analysis, the BIT collection A contains 269 models ranging from 5 to 47 nodes and collection B3 contains 247 models with 5 to 42 nodes.

In addition to the IBM BIT collection, we also used synthesized process logs. In (Pourmasoumi et al. 2015b), we proposed an approach and its supporting toolset for generating process variants, called it PVG. A screen shot of the process variant generator tool is shown in Fig. 8. Based on the technique and tool, a collection of process variants can be randomly created according to a probabilistic distribution and based on a user-defined variation rate parameter. The tool is provided within the PLG toolset (Burattin and Sperduti 2010). PLG generates random process models using context-free grammars by employing 5 basic workflow patterns: single activity, loop, sequence, XOR split-join and AND split-join. Moreover, the user can select from three probability distribution functions: Uniform, Gaussian and Beta, which will be used for generating the number of branches for AND/XOR split-join patterns. After generating a random process model, PLG is capable of generating its execution logs by traversing the generated process graph.

The synthetic dataset consists of 500 process variants and 5000 event logs are generated for each process variant. We generated these variants for 10 randomly generated process models (50 variants for each process model). The randomly generated input process models have different sizes. The overall specification of the whole 500 variants is shown in Table 9. For the process variant generation, we set the variation rate to 30 %.

7.2 Experiments

7.2.1 Refactoring gain

For evaluating the proposed approach, we use the criterion proposed in (Uba et al. 2011). As previously mentioned, one of the major applications of process fragmentation is reusability. In order to evaluate the usefulness of fragmentation approaches for reusability, the *refactoring gain* measure as proposed in (Uba et al. 2011) is used. The refactoring gain of a process fragment is the reduction in the number of nodes obtained by embedding that fragment into a separate sub-process, and replacing every occurrence of the fragment with a task that invokes this sub-process.

Definition 11 (Uba et al. 2011). *Let S be the size of a fragment, and N the number of occurrences of this fragment. Since all occurrences of a fragment are replaced by a single occurrence plus N sub-process invocations, the refactoring gain is defined as* $G_f = S \times N - S - N$.

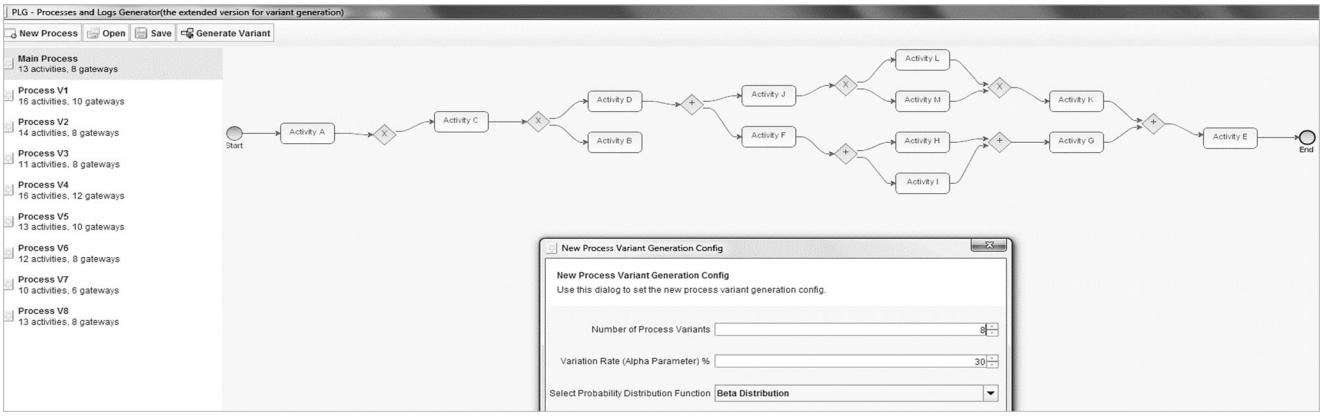


Fig. 8 A screen shot of process variant generator tool

In this definition, the number of existing activities in a process fragment trace is considered as its size *S*. Based on Definition 11, we calculate the refactoring gain of the extracted fragments in each data collections. In our experiments, we only consider *L*-grams with length greater than 2 as candidate fragments. The reason is that fragments with a length of 2 can be easily modeled by a business modeler and do not add too much value in terms of reuse (Uba et al. 2011).

Table 10 summarizes the statistics of the extracted fragments for each data collection. This table shows the minimum, maximum and average size of the detected fragments. It also shows the minimum, maximum and average occurrence of the extracted fragments. Based on these results, it can be seen that a significant number of common fragments is elicited. The average occurrence of common fragments in the collection A is 3.05 and the maximum is 10. In collection B, these measures are 3.91 and 21. The largest fragment detected in collection A has a size of 14 activities and in collection B the largest fragment has a size of 8 activities. The result of running experiments on the dataset generated using the PVG tool is also shown on the third row of Table 10. The average occurrence of common fragments is 4.35, the maximum occurrence is 25 and the largest detected fragment has a size of 17 activities.

The refactoring gain of each data collection is shown in Table 11. The *total refactoring gain* is the sum of the refactoring gains of non-trivial fragments (size ≥3) in the collection (Uba et al. 2011). We have also shown the maximum, minimum and average refactoring gain of extracted fragments in Table 11.

The total refactoring gain of the extracted fragments from the process variants generated by PVG tool (Pourmasoumi et al. 2015b) is more than the refactoring gain of fragments extracted from the other data sets. This is due to the fact that the process variants generated by the PVG tool are limited to a specific deviation rate (In our experiments they are limited to a variation rate of 30 %) from the main input process models. However, the other data sets cover a wider range of deviation. For example, collection A of IBM BIT dataset contains variations with size 3 to sizes of 34 activities.

For comparing our proposed approach with other works, the most similar work, which is comparable, is (Uba et al. 2011). In this work, the authors proposed a clone detection approach and evaluated their work on BIT A and B3 collections. The authors used refactoring gain measure for evaluating their work. However, their proposed approach is on process models, not event logs. They report that they were able to find 57 fragments in collection A with their first proposed approach with a refactoring gain of 195 (4.3 % compared to 9.8 % by our method) and 174 with their second approach with a refactoring gain of 384 (8.4 % compared to 9.8 % by our method). They also extracted 19 fragments with refactoring gain of 208 (6.5 % compared to 7.1 % by our method) from collection B3 using their first approach and 49 fragments with refactoring gain of 259 (6.6 % compared to 7.1 % by our method) from collection B3 using their second approach. In the absence of comparable methods that work specifically on event traces and given the fact that our method works on event traces, we believe that the refactoring gain metric could be considered an acceptable

Table 9 Statistics of used dataset

Dataset Collection Name	#Process Variants	#Log Traces	Activities			Gateways		
			Min	Max	Avg	Min	Max	Avg
IBM BIT Collection A	269	100,00	3	34	12.01	0	13	3.07
IBM BIT Collection B3	247	5000	3	32	8.92	0	11	2.21
Data Generated By PVG tool	500	5000	10	40	13.04	0	15	3.12

Table 10 Statistics of the extracted fragments

Dataset Collection Name	Size			# Occurrence		
	min	max	avg	min	max	avg
IBM BIT Collection A	3	14	3.01	2	10	3.05
IBM BIT Collection B3	3	8	2.97	2	21	3.91
Dataset Generated Using PVG tool	3	17	3.9	2	25	4.35

comparison metric with methods that work with process models. This is because the refactoring gain metric is based on the 'percentage' of size reduction as explained earlier. So, in comparison to the approach proposed in (Uba et al. 2011), it can be seen that there is an increase in the refactoring gain of the extracted fragments by our proposed work. Table 12 shows 16 % and 7.5 % improvement in refactoring gain of collection A and B3, respectively.

7.2.2 Impact of rate of morphologicality

In Section 6.4, we introduced the degree of morphologicality for increasing flexibility. We also proposed a clustering algorithm for extracting clusters of common fragments with the same degree of morphologicality. In this section, we intend to evaluate the effect of the degree of morphologicality on the flexibility of the proposed fragmentation approach. To this end, we extract clusters which have degree of morphologicality $\alpha = 85\%$. The reason that we consider $\alpha = 85\%$ is that we want to evaluate the effect of the minimum changes in morphological fragments. So, based on the average rounded size of fragments which is equal to 4 activities according to (Uba et al. 2011), if we miss at most one activity, the degree of similarity would be $2 \times 3/(3+4) = 85.7\%$.

Table 12 shows the refactoring gain of the extracted clusters with degree of morphologicality of greater than 85 %. In this experiment, for calculating the refactoring gain metric, we need to calculate the center of each cluster. Within each cluster, we consider the fragment, which has the highest similarity with the other fragments in that cluster as center of the cluster. The refactoring gain metric is calculated as follows:

Definition 12 *The refactoring gain of a cluster of fragments is the reduction in the number of nodes obtained by embedding all fragments inside the cluster into a separate sub-process, and replacing every occurrence of each fragment inside the cluster with a task that invokes this sub-process. The refactoring gain can be calculated by*:

$$G_f = \sum_{j=1}^{K} \left(\sum_{i=1}^{M} \left(|f_i^j| \times N_i^j \right) - N_i^j \right) - |f_c^j| \quad (4)$$

In this equation, $|f_i^j|$ is the size of fragment f_i within cluster j, N_i^j is the occurrence frequency of fragment i, $|f_c^j|$ is the size of centric fragment of cluster j and K is the number of clusters.

Table 12 shows the number of extracted clusters. Collection A contains 108, B3 contains 92 and the dataset generated using PVG has 226 clusters. As explained previously, these clusters might have overlapping fragments. For example two clusters may contain a fragment F_1. So, for calculating the 'true' refactoring gain, we consider F_1 only once in our computations. We exclude shared fragments from our calculation after extracting clusters and consider the presence of shared fragments only in one cluster. Table 13 shows the size of each cluster in terms of the number of fragments. The average size for A, B3 and PVG dataset is 2.7, 2.32 and 3.01 respectively.

According to Table 12 the refactoring gain of collections A, B3 and PVG dataset shows a significant increase of 13 %, 16 % and 18 %, respectively. The experimental results in Tables 12 and 13 show that with a little flexibility allowed through the degree of morphologicality, it is possible to extract a considerably higher number of common fragments with slight variations. These variations might be for negligible reasons such as different design styles of different designers. However, we do not contend that all extracted fragments within a cluster can be

Table 11 Refactoring gain of each collection

Dataset Collection Name	#Fragments	Refactoring Gain			
		total	max	avg	std
IBM BIT Collection A	185	9.8 %	13	2.9	3.2
IBM BIT Collection B3	130	7.1 %	28	5.7	9.2
Dataset Generated Using PVG tool	410	12.4 %	39	3.94	4.01

Table 12 Refactoring gain of each collections with degree of morphologicality of 85 %

Dataset Collection Name	#Clusters	Total Refactoring Gain of Clusters
IBM BIT Collection A	108	11.04 %
IBM BIT Collection B3	92	8.3 %
Dataset Generated Using PVG tool	226	14.6 %

Table 13 Statistics of extracted clusters with degree of morphologicality of 85 %

Dataset Collection Name	Size of Clusters in Terms of Number of Fragments		
	min	max	avg
IBM BIT Collection A	2	5	2.7
IBM BIT Collection B3	2	4	2.32
Dataset Generated Using PVG tool	2	7	3.01

used interchangeably in reuse applications and there is need for an expert review for a complete compliance check.

8 Discussions

In this section, we will discuss the underlying reasons why we do not consider the other parts of Table 6 as morphological fragments and only focus on full similarity-double side of stable states. We then provide a formal analysis of the computational complexity of the proposed approach in Section 8.2. We will show that the time complexity of our proposed approach is $O(n^2)$, where n is the number of traces.

8.1 Stable states of morphological fragments

There are three reasons why we only consider the first row of Table 6 and not the other three rows in our morphological fragment definition:

- First, the primary goal of eliciting morphological fragments is business process improvement and/or reusability. For this purpose, the most important property of morphological fragments is *composability*. Therefore, having the same input and output activities for morphological fragments can significantly facilitate replacement and composition of process fragments.
- Morphological fragments do not necessarily guarantee full goal compatibility but rather they point to sub-processes that are likely to be *related* to similar goals/objectives. It should be noted that we would relax attention to accuracy and full compatibility for the sake of finding more potential matches.
- In many cases, if there is partial similarity between two fragments, it is possible by taking a window of smaller size on the main process, to reach to *full similarity-double side*. For example, imagine that we have two fragments: $F_1 = <A, B, C, F, E, D, G, H>$ and $F_2 = <M, H, C, D, E, F, G, A>$. These fragments are not morphological fragments because their stable states are not equals. However, if we take a window of smaller size on these fragments, these morphological fragments can be achieved: $F_1' =< C, F, E, D, G >$ and $F_2' =< C, D, E, F, G >$.

8.2 Computational complexity analysis

The proposed approach has a very lightweight implementation, has polynomial time complexity and converges to a global solution. In this section, the time complexity of the proposed approach for the main parts of the algorithm is explained. This time complexity is determined independent of the implementation technique.

8.2.1 Time complexity of preprocessing

Assume that we have S organizations, each of which contains on average N traces of event logs, $|TS^i|_{AVG} = N$, $1 \le i \le S$. In the first phase of the preprocessing step (Section 6.1), we calculate the local frequency of traces and remove the duplicates. The time complexity of this step is $O(SN^2)$. The next phase in preprocessing step is detecting loops. The time complexity of detecting loop for each trace t with length $l = |t|$ is $O(l^2)$. Since after performing the first phase of the preprocessing step the number of traces reduce significantly and can be considered as a constant N_p; therefore, the time complexity of detecting all loops for all organizations event logs would be $O(SN_pl^2)$. Given that N_c and l are finite and $N_p \ll N$, $l \ll N$, so the time complexity of preprocessing step is $O(SN^2)$.

8.2.2 Time complexity of generating L-grams

We can calculate the number of created L-grams as follows:

$$\begin{aligned} \#_{L\text{-}gram} &= \sum_{i=1}^{S} \sum_{j=1}^{N_p} \sum_{k=1}^{l} (l-k+1) \\ \#_{L\text{-}gram} &= S \times N_p \times \sum_{k=1}^{l} (l-k+1) \\ \#_{L\text{-}gram} &= S \times N_p \times \left(\frac{l^2+l}{2} \right) \end{aligned} \tag{5}$$

As seen in Eq. (5), the time complexity of generating L-grams is $O(SN_pl^2)$.

8.2.3 Time complexity of detecting common L-grams

The time complexity for calculating numerical values corresponding to each trace is a small constant C. Hence, the time complexity for calculating numerical value of all traces is equal to the number of them. Likewise, the time complexity for detecting common fragments is in the order of the number of traces. Therefore, the time complexity of detecting common L-grams and the proposed method as a whole is $O(N_pl^2)$.

Considering the time complexity of each part, we can infer that the total time complexity of the proposed algorithm for detecting common morphological fragments is $O(SN^2)$. As the number of organization is limited, the values of S can be ignored. Thus, the time complexity is $O(N^2)$.

8.2.4 Time complexity of extracting common fragments with flexible degree of morphologicality

For extracting common fragments with flexible degree of morphologicality (Section 6.4), we have to perform the preprocessing step and also generate L-grams. So, this part would have a time complexity of $O(SN^2)$. For every two L-grams

g_i and g_j with length l, $D_m(g_i, g_j)$ can be calculated with $O(l^2)$. The time complexity of calculating $D_m(g_i, g_j)$ for all existing L-gram ($SN_p l^2$) is $O((SN_p l^2)^2 \times l^2)$. The number of operations for detecting clusters is equal to the number of l-grams multiplied by the number of initial clusters (which we set to be equivalent to the number of L-grams). So, the time complexity of extracting clusters is $O((SN_p l^2)^2)$. Since we have $l \ll N$, $S \ll N$ and $N_p \ll N$, again we can infer that the time complexity of the algorithm when considering the degree of morphologicality is also equal to $O(N^2)$.

9 Conclusion and future works

In this paper, we assess various existing definitions of process fragments based on Bunge's ontological model and its process representational model, GPM and compare them based on various criteria. We then present a new definition for process fragments, called *morphological fragments*, which has practical implications on composability and reusability. On this basis, we propose a novel algorithm for extracting common morphological fragments from a collection of event logs. Furthermore, we propose a supporting algorithm for clustering fragments with some degree of variation. According to the experimental results, our proposed approach supports reusability and flexibility in identifying process fragments. Another positive feature of our proposed approach is that it extracts process fragments directly from event logs, which would be useful in many domains of process mining. As future work, we are interested in exploring the applications of our work in composing optimized business process models. We intend to extract and use other features of event logs such as cost or complexity and propose methods for automatically composing the best configurable process model according to user's preferences. To this end, we would like to focus on reusing morphological fragments for composing optimized process models given an input set of constraints.

References

Andrews, T., et al. (2003). *Business process execution language for web services*, version1.1, Technical Report, Microsoft, IBM, Siebel Systems, SAP, BEA.

Asadi, M., Gašević, D., Wand, Y., Hatala, M. (2012). *Deriving Variability Patterns in Software Product Lines by Ontological Considerations*, In Proceedings of the 31st International Conference on Conceptual Modeling, Florence, Italy, pp. 397–408.

Assy, N., Chan, N.N., Gaaloul, W. (2013). *Assisting Business Process Design with Configurable Process Fragments*. In: IEEE SCC, pp. 535–542.

Atluri, V. S. A., Chun, S. A., Mukkamala, R., & Mazzoleni, P. (2007). A decentralized execution model for inter-organizational workflows. *Distributed and Parallel Databases, 22*(1), 55–83.

Basu, A., & Blanning, R. (2003). Synthesis and decomposition of processes in organizations. *Information Systems Research, 14*(4), 337–355.

Bunge, M. (1977). *Treatise on basic philosophy*, *Ontology I: The Furniture of the World* (Vol. 3). Boston: Reidel.

Burattin, A., & Sperduti, A. (2010). *PLG*: *A Framework for the Generation of Business Process Models and Their Execution Logs*. In Business Process Management Workshops, pages 214–219. Springer.

Dijkman, R. M., Dongen, B. F., Dumas, M., Garcia-Banuelos, L., Kunze, M., Leopold, H., et al. (2013). *A short survey on process model similarity*, *Seminal Contributions to Information Systems Engineering* (pp. 421–427). Heidelberg: Springer.

Dumas, M., Rosa, M., La, M. J., & H.A., R. (2013). *Fundamentals of business process management* (p. 414). Berlin: Springer-Verlag.

Eberle, H., Unger, T., & Leymann, F. (2009). Process fragments, in On the Move to Meaningful Internet Systems: OTM 2009, ser. *Lecture Notes in Computer Science, 2009*(5870), 398–405.

Evermann, J., & Wand, Y. (2005). Ontology based object-oriented domain modelling: fundamental concepts. *Requirements Engineering, 10*(2), 146–160.

Fahland, D., Favre, C., Koehler, J., Lohmann, H., Volzer, N., & Wolf, K. (2011). Analysis on demand: instantaneous soundness checking of industrial business process models. *Data and Knowledge Engineering, 70*(5), 448–466.

Ganter, B., & Wille, R. (1999). *Formal concept analysis, mathematical foundations*. Berlin: Springer.

Gao, X., Chen, Y., Ding, Z., Wang, M., Zhang, X., Yan, Z., Wen, L., Guo, Q., Chen, R. (2014). Process Model Fragmentization, *Clustering and Merging*: *An Empirical Study*. Lecture Notes in Business Information Processing Volume 171.

Ghattas, J., & Soffer, P. (2009). Evaluation of inter-organizational business process solutions: A conceptual model-based approach. *Information Systems Frontiers, 11*(3), 273–291.

Hens, P., Snoeck, M., De Backer, M., & Poels, G. (2014). Process fragmentation, distribution and execution using an event-based interaction scheme. *Journal of Systems and Software, 89*, 170–192.

Hruschka, E. R., Campello, R. J. G. B., Freitas, A. A., & de Carvalho, A. C. P. L. F. (2009). A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews, 39*, 133–155.

Ivanovic, D., Carro, M., and Hermenegildo, M. (2010). Automatic fragment identification in workflows based on sharing analysis. In Mathias Weske, Jian Yang, Paul Maglio, and Marcelo Fantinato, editors, Service-Oriented Computing –ICSOC.

Jan R., Marta I., Michael R., and Peter G. (2006). *How Good is BPMN Really*? *Insights from Theory and Practice*. Proceedings 14th European Conference on Information Systems, Goetborg, Sweden.

Khalaf, R., Kopp, O., & Leymann, F. (2008). Maintaining data dependencies across bpel process fragments. *International Journal of Cooperative Information Systems, 17*, 259–282.

Larosa, M., Dumas, M., Uba, R., & Dijkman, R. M. (2013a). Business process model merging: An approach to business process consolidation. *ACM Transactions on Software Engineering and Methodology, 22*, 2,11.

Larosa, M., Dumas, M., Uba, R., & Dijkman, R. M. (2013b). Business process variability modeling: a survey. *ACM Transactions on Software Engineering and Methodology, 22*(2), 11.

Leemans, S. J. J., Fahland, D., & van der Aalst, W. M. P. (2014). *Discovering block-structured process models from incomplete event logs* (Vol. 8489, pp. 91–110). Heidelberg: Springer. In PETRI NETS 2014. LNCS

Leymann, F. (1996). Workflows make objects really useful. *EMISA Forum, 6*(1), 90–99.

Li, J., Wang, H. J., & Bai, X. (2015). An intelligent approach to data extraction and task identification for process mining. *Information Systems Frontiers, 17*(6), 1195–1208.

Mancioppi, M., & Danylevych, O. (2012). *Towards classification criteria for process fragmentation techniques* (pp. 1–12). Heidelberg: Springer Berlin. in: Proc. BPM Work.

Milani, F., Dumas, M., & Matulevičius, R. (2013). *Decomposition driven consolidation of process models*, advanced information systems engineering. *Lecture Notes in Computer Science, 7908*, 193–207.

Milani, F., Dumas, M., Matulevičius, R., & Ahmed, N. (2015). Criteria and heuristics for business process model decomposition: review and comparative evaluation. *Business & Information Systems Engineering, 58*(1), 7–17.

Pourmaoumi A., Kahani, M., Bagheri, E., Asadi, M. (2014). *Mining Common Morphological Fragments from Process Event Logs*. In Center for Advance Studies Conference (IBM CASCON), Best Student Paper Award, Toronto, Canada.

Pourmasoumi, A. (2015). *Mining configurable process models from event logs*. In Business Process Management, Doctoral Symposium, Innsbruck, Austria.

Pourmasoumi, A., Kahani, M., Bafgeri, E., Asadi, M. (2015a). Process fragmentation: *An ontological perspective*. In Proceedings of the 2015 International conference on business process modeling, development, and support (BPMDS'15), Springer, Stockholm.

Pourmasoumi, A., Kahani, M., Bagheri, E., Asadi, M. (2015b). On Business Process Variants Generation, CAiSE FORUM.

Reijers, H.A., Mendling, J. (2008). *Modularity in process models*: *review and effects*, in: BPM, Springer.

Reijers, H. A., Mendling, J., & Dijkman, R. M. (2011). Human and automatic modularizations of process models to enhance their comprehension. *Information Systems, 36*(5), 881–897.

Rosa, M., Dumas, M., Ekanayake, C., García-Bañuelos, L., Recker, J., & Hofstede, H. M. (2015). Detecting approximate clones in business process model repositories. *Information Systems, 49*, 102–125.

Schumm, D., Leymann, F., Ma, Z., Scheibler, T., Strauch, S. (2010) *Integrating Compliance into Business Processes*: *Process Fragments as Reusable Compliance Controls*. Proc. of the MKWI.

Seidita, V., Cossentino, M., Hilaire, V., Gaud, N., Galland, S., Koukam, A., et al. (2010). The meta model: a starting point for design processes construction. *International Journal of Software Engineering and Knowledge Engineering, 20*(4), 575–608.

Seidita, V., Cossentino, M., Chella, A. (2011) *A proposal of process fragment definition and documentation*, 9th European Workshop on Multi-agent Systems (EUMAS).

Smirnov, S., Dijkman, R. M., Mendling, J., & Weske, M. (2010). Meronymy-based aggregation of activities in business process models. *Conceptual Modeling–ER, 2010*, 1–14.

Soffer, P., & Yehezkel, T. (2011). *A state-based context-aware declarative process model*, *In: Enterprise, Business-Process and Information Systems Modeling* (pp. 148–162). Berlin: Springer.

Soffer, P., Kaner, M., & Wand, Y. (2010). Assigning ontological meaning to workflow nets. *Journal of Database Management, 21*(i3), 35.

Tan, W., & Fan, Y. (2007). Dynamic workflow model fragmentation for distributed execution. *Computers in Industry, 58*(5), 381–391.

TFPM – ieee task force on process mining (2012). *Process mining manifesto*, *In BPM Workshops. Lecture Notes in Business Information Processing Series* (Vol. 99). Berlin: Springer-Verlag.

Uba, R., Dumas, M., Garc'ıa-Ba˜nuelos, L., & La Rosa, M. (2011). In S. Rinderle-Ma, F. Toumani, & K. Wolf (Eds.), BPM 2011. LNCS *Clone detection in repositories of business process models* (Vol. 6896, pp. 248–264). Heidelberg: Springer.

Valenca, G., Alves, C., Alves, V., & Niu, N. (2013). A systematic mapping study on business process variability. *International Journal of Computer Science & Information Technology, 5*(1), 1–21.

van der Aalst, W. (2009). *Process-aware information systems: Lessons to be learned from process mining*. T. Petri Nets and Other Models of Concurrency, 2:1–26.

van der Aalst, W. (2011). *Process mining: Discovery, conformance and enhancement of business processes*. Berlin: Springer-Verlag.

van der Aalst, W. (2012) *Process mining: Overview and opportunities*. ACM Transactions on Management Information Systems, Vol. 99, No. 99, Article 99.

Vanhatalo, J., Volzer, H., Leymann, F. (2007). *Faster and more focused control-flow analysis for business process models through sese decomposition.* Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC).

Vanhatalo, J., V¨olzer, H., Koehler, J. (2008) *The refined process structure tree.* In Proceedings of BPM, Lecture Notes in Computer Science, Springer, 5240, 100–115.

Vanhatalo, J., V¨olzer, H., & Koehler, J. (2009). There fined process structure tree. *Data & Knowledge Engineering, 68*(9), 793–818.

Wand, Y., & Weber, R. (1995). On the deep structure of information systems. *Information Systems Journal, 5*, 203–223.

Wen, L., Wang, J., & Sun, J. (2007). *Mining invisible tasks from event logs*, *APWeb/WAIM 2007. LNCS* (Vol. 4505, pp. 358–365). Heidelberg: Springer.

Weske, M. (2012). *Business process management architectures*, *in Business Process Management* (pp. 333–371). Berlin Heidelberg: Springer.

Zemni M., Hadj-Anouane, N., Yeddes, M. (2012). *An Approach for Producing Privacy-Aware Reusable Business Process Fragments*, Proceedings of the 2012 I.E. 19th International Conference on Web Services, p.659–661, June 24–29.

Asef Pourmasoumi is Ph.D candidate, research assistant at Ferdowsi university of Mashhad. He is also member of the Laboratory for Systems, Software and Semantics (LS^3) at Ryerson University, Canada. His research interest includes process mining (especially on organizational mining), natural language processing and software engineering. He can be reached at *asef.pourmasoumi@stu.um.ac.ir*

Mohsen Kahani is a professor of computer engineering, IT director and head of Web Technology Lab. at Ferdowsi University of Mashhad. His research interests includes semantic wen, software engineering, natural language processing and process mining. He can be reached at kahani@um.ac.ir

Ebrahim Bagheri is a Canada Research Chair in Software and Semantic Computing, Associate Professor, and the Director of the Laboratory for Systems, Software and Semantics (LS^3) at Ryerson University. He has been active in the areas of social semantic web and software engineering, and, for the past several years, Dr. Bagheri's research has focused on devising *empirically tested* methods that enhance the systematic large-scale reuse of software assets with a focus on the software product line engineering paradigm. He can be reached at bagheri@ryerson.ca