Institute of Architecture of Application Systems

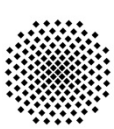
RoSE: Reoccurring Structures Detection in BPMN 2.0 Process Models Collections

Marigianna Skouradaki[1], Vasilios Andrikopoulos[1], Oliver Kopp[2], Frank Leymann[1]

1. Institute of Architecture of Application Systems,
University of Stuttgart, Germany
2. Institute of Parallel and Distributed Systems,
University of Stuttgart, Germany

BibTeX:

```
@inproceedings {SAKL_2016,
  author = {Marigianna Skouradaki and Vasilios Andrikopoulos and Oliver Kopp and Frank
            Leymann},
  title = {{RoSE: Reoccurring Structures Detection in BPMN 2.0 Process Model Collections}},
  booktitle = {OTM Confederated International Conferences ''On the Move to Meaningful Internet
            Systems"},
  publisher = {Springer International Publishing},
  pages = {263--281},
  year = {2016},
  doi = {10.1007/978-3-319-48472-3_15}
}
```

University of Stuttgart
Germany

RoSE: Reoccurring Structures Detection in BPMN 2.0 Process Model Collections

Marigianna Skouradaki[1], Vasilios Andrikopoulos[1], Oliver Kopp[2], and Frank Leymann[1]

[1]IAAS, [2]IPVS, University of Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de

Abstract The detection of structural similarities of process models is frequently discussed in the literature. The state-of-the-art approaches for structural similarities of process models presume a known subgraph that is searched in a larger graph, and utilize behavioral and textual semantics to achieve their goal. In this paper we propose an approach to detect reoccurring structures in a collection of BPMN 2.0 process models, without the knowledge of a subgraph to be searched, and by focusing solely on the structural characteristics of the process models. The proposed approach deals with the problems of subgraph isomorphism, frequent pattern discovery and maximum common subgraph isomorphism, which are mentioned as NP-hard in the literature. In this work we present a formal model and a novel algorithm for the detection of reoccurring structures in a collection of BPMN 2.0 process models. We then apply the algorithm to a collection of 1,806 real-world process models and provide a quantitative and qualitative analysis of the results.

Keywords: BPMN 2.0; Process similarity; Graph matching; Structural similarity; Business process management

1 Introduction

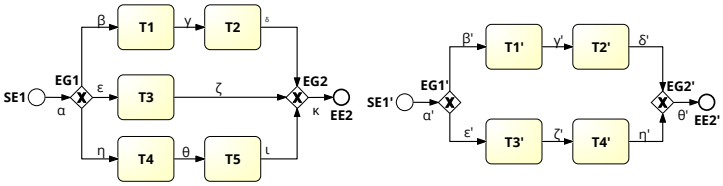
Business process model similarities is a topic frequently discussed in the literature [2,6]. It finds application in multiple scenarios, as for example the comparison or integration of business processes [2]; the validation of processes models [2]; the adjustment of the process models to different target groups [2]; the detection and refactoring of duplicates (clones) in process model repositories [7, 9]; the detection of different versions of the same process models [21]; or the generation of process model collections out of re-occurring fragments [32]. The "BenchFlow" project[1] has created an additional use case scenario that requires the structural comparison of process models. The scope of the project is to create the first standard benchmark for Workflow Management Systems (WfMS) that are compliant with the Business Process Model and Notation (BPMN 2.0) [16] language. In order to determine a reliable and representative benchmark, special attention

[1] http://www.iaas.uni-stuttgart.de/forschung/projects/benchflow.php

needs to be paid in the definition of the test scenarios (i. e., workload) [27]. The usage of non-representative workload might result in misleading results [10]. A prominent role in the workload of a WfMS benchmark play the process models that constitutes it. Namely, the process models that are given as input to the performance tests [11]. For defining a set of representative process models, we have contacted research and industrial partners to collect real-world process models. Since the process models constitute a corporate asset for the stakeholders, in most of the cases they were reluctant to share them. In order to overcome this burden and foster the process models sharing, we have developed tools, that anonymize the process models, while they keep their structural and, if present, their behavioral characteristics. This effort, has resulted in the collection of 11,151 real-world BPMN 2.0 business process models, some of which are anonymized, and some of which are reference, non-executable process models [27].

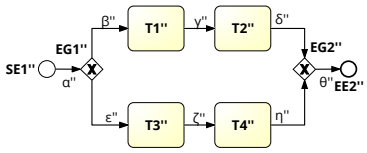
The analysis of the process models in order to define the workload is twofold. Firstly, we need to define process models with representative structures that are also capable to stress the WfMS in terms of performance. Secondly, we need to artificially simulate a realistic behavior for the defined process models. The later can be handled with the utilization of existing techniques in process mining [3] and workload characterization [10]. On the contrary, the first task requires the structural analysis of our collection, in order to determine the recurring structures. Given the existence of anonymized and reference models, an approach is required that should *exploit solely the structural information of the process models* (Objective 1). Addressing the Objective 1 can generally be reduced to the very well known challenge of subgraph isomorphism [28] which is discussed as NP-complete in the literature [13]. However, as the well-structured BPMN 2.0 process models are special types of graphs, i. e., Series Parallel Graphs [12] the challenge of subgraph isomorphisms can be solved in polynomial time [13]. The reduction to polynomial complexity indicates the feasibility of developing and applying a technique that targets BPMN 2.0 process models.

Although many existing approaches are solving the subgraph isomorphism problem [4,13,28,30] they assume a given graph whose occurrence is searched in a larger graph. In our case, however, *a known subgraph does not exist. Instead, we need to identify and extract it based on the structural similarity among a set of process models* (Objective 2). To this edge and in order to target both of the defined objectives, we are reaching the challenge of *frequent pattern discovery without candidate generation* which can be seen as a variant of the subgraph isomorphism [31]. Breuker et al. [2] have applied an extensive research on the performance of publicly available algorithms for frequent pattern matching without candidate generation. More particularly, Breuker et al. [2] ran experiments for the gSPAN [31] and Gaston [19] algorithms to discover frequently occurring patterns in a collection of thousands of Event-Driven Process Chain (EPC) process models. In the experiments, the textual semantics are omitted, and the comparison is applied only with respect to the structural semantics of the process models (cf. Objective 1). In this case, the authors report quick failure of both algorithms when applied to a collection of thousands of models.

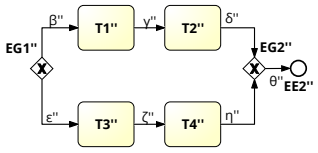


(a) Model A (b) Model B

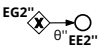
Figure 1: BPMN 2.0 Example Models



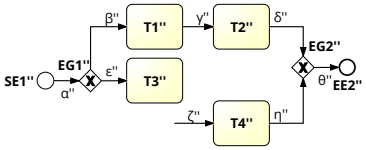
(a) Re-occurring Structure 1



(b) Re-occurring Structure 2



(c) Re-occurring Structure 3



(d) Re-occurring Structure 4

Figure 2: Reoccurring Structures in the Models of Fig. 1

This work exploits and combines existing techniques to propose a novel approach for the efficient detection of frequently reoccurring structural patterns. The targeted use case scenario is a collection of BPMN 2.0 process models. However, the proposed approach may be generalized to any process modelling language, by adapting the proposed formal model. More particularly, the contributions of this work can be summarized as follows; we present: 1) the formal model of our approach, 2) a novel algorithm for detecting and extracting reoccurring structures (i. e., structural similarities) from a collection of BPMN 2.0 process models and 3) a qualitative and quantitative validation of our approach. The rest of this paper is structured as follows: Section 2 discusses a motivating example for introducing the objectives of our approach; Sect. 3 sets the formal model of the proposed approach; Sect. 4 presents the algorithm that detects and extracts the reoccurring structures in a collection of BPMN 2.0 process models. In Sect. 5 we validate our approach, and we demonstrate and discuss the qualitative and quantitative results. The paper closes with related work in Sect. 6, and conclusions and future work in Sect. 7.

2 Motivating Example

In this work we target the challenge of detecting reoccurring structures in a collection of BPMN 2.0 process models. Our approach has a complete focus on

handling BPMN 2.0 process models as special types of graphs in order to detect their structural similarities. Hence, textual semantics or behavioral similarities of the process models are currently not considered by our approach. Nevertheless, the proposed approach may be combined with existing approaches on textual and behavioral similarities [6]. In this way, the detected recurring structures will be also similar from a content point of view.

Figure 1 shows two examples of BPMN 2.0 process models for which we detect the reoccurring structures. At a first glance, the presented process models are very similar to each other. Thus, a quick observation leads to the detection of many reoccurring structures. Figure 2 shows some of such reoccurring structures in the process models of Fig. 1a and Fig. 1b. The demonstrated structures are only some possible results and not the complete set of reoccurring structures between these two process models. Focusing on the structures shown in Fig. 2a and Fig. 2b we observe that the only difference between these two structures is their starting point. Namely, the structure of Fig. 2a begins with a start event, while the structure of Fig. 2b with an exclusive gateway. Both structures stem from exactly one starting point (i. e., *SE1* and *SE1′* matched to *SE1″* and *EG1*, and *EG1′* matched with *EG1″* respectively). As seen in Fig. 2a and Fig. 2b, both structures constitute maximal common reoccurring structures between the compared models. Which means that there is not any other larger common structure between models A and B (Fig. 1) stemming from these starting points.

In order to satisfy the purposes of our approach not all possible reoccurring structures are to be considered. For example, the structure of Fig. 2c consists of two BPMN 2.0 elements connected with each other. Since this structure is a minimal building block for any BPMN 2.0 model we consider it of no particular structural interest and exclude it from our results. Another more complex structure that is not considered by our approach is shown in Fig. 2d. In this case, all the BPMN 2.0 elements of this structure can be mapped to exactly one BPMN 2.0 element of either Fig. 1a or Fig. 1b. However, the structure cannot exactly be characterized as reoccurring, as it is not fully contained in Fig. 1b. This is because the ζ'' sequence flow is not connected to *T3″*. Moreover, not every element of the structure in Fig. 2d is connected to a starting point. Only all elements of the structures in Fig. 2a and Fig. 2b can be reached through the starting points *SE1″* and *EG1″*, respectively. For the purposes of this work we therefore focus only on identifying the *non-trivial, reoccurring structures of maximum size* which contain *exactly one starting point*, out of which every participating BPMN 2.0 element of the structure is *connected* to the source and can be reached by a common graph traversal algorithm.

3 Formal Model

A BPMN 2.0 model can be seen as a directed, attributed graph $G = (\boldsymbol{V}, \boldsymbol{E})$ [12] with $n = |\boldsymbol{V}|$ vertices (denoting the activities) and $m = |\boldsymbol{E}|$ edges (denoting the sequence flow connectors). *Type of a vertex* v_i is called the function $type : \boldsymbol{V} \rightarrow \boldsymbol{L_a}$

that assigns an element from a set $\boldsymbol{L_a}$ to each $v_i \in \boldsymbol{V}$. The set $\boldsymbol{L_a}$ contains element types from the BPMN 2.0 set as described by the BPMN 2.0 standard [16].

Zur Muehlen and Recker [18] show that most of the BPMN 2.0 elements are not actually applied in practice. What is more, a preliminary statistical analysis in our collection has indicated that approximately a 12% of the contained process models contain cyclic structures. Since our approach targets the detection of *the most commonly reoccurring structures* of a BPMN 2.0 collection, the proposed approach supports the most commonly used BPMN 2.0 elements [18], while it does not support cyclic graphs. More particularly, we consider the elements of the set $\boldsymbol{L_a} = \boldsymbol{Evt} \cup \boldsymbol{Gt} \cup \boldsymbol{Tsk}$ only, where:

- ***Evt*** is the set of all events types as defined in the BPMN 2.0 standard (e. g., start event, end event etc.)
- ***Gt*** is the set of all gateways types in the standard (e. g., parallel gateway, exclusive gateway, complex gateway etc.)
- ***Tsk*** is the set of all tasks types in the standard (e. g., manual task, script task, service task etc.)

The concept of checkpoints introduces "areas for investigation" in the process model, as the ***Evt*** and ***Gt*** elements of BPMN 2.0 are those differentiating the models structurally. In other words, without ***Evt*** and ***Gt*** elements, the process models consisted only of ***Tsk*** elements and this is not of any particular structural interest.

Definition 1 (Checkpoints) *Let $\boldsymbol{L_{ch}} = \boldsymbol{L_a} \setminus \boldsymbol{Tsk}$ be the set of all BPMN 2.0 element types excluding all tasks. The function $ch : \wp(G) \to \wp(\boldsymbol{V})$ creates a subset of the vertices of a graph G. The subset contains vertices having a type contained in $\boldsymbol{L_{ch}}$ only when $ch : G = (\boldsymbol{V}, \boldsymbol{E}) \mapsto \{v \mid v \in \boldsymbol{V},\ v \in \boldsymbol{L_a}\}$. In the following, we use $\boldsymbol{V}_{ch}^{G}$ as shortcut for $ch(G)$. The elements of a graph G contained in $\boldsymbol{V}_{ch}^{G}$ are called* checkpoints of graph G.

For example, the checkpoints of the models presented in Fig. 1 are the vertices $\boldsymbol{V}_{ch}^{ModelA} = \{SE1, EG1, EG2, EE2\}$ and $\boldsymbol{V}_{ch}^{ModelB} = \{SE1', EG1', EG2', EE2'\}$.

Following on, for any directed edge $e = (u, v) \in \boldsymbol{E}$, we say that e is *outgoing* from u and *incoming* to v. The functions $incoming : \boldsymbol{V} \to \wp(\boldsymbol{E})$ and $outgoing : \boldsymbol{V} \to \wp(\boldsymbol{E})$ are defined accordingly. A vertex $v \in \boldsymbol{V}$ is called a *source* when $incoming(v) = \emptyset$ and a *sink* when $outgoing(v) = \emptyset$. In our case we assume that any graph has a unique source, i. e., $|\{v \in \boldsymbol{V} \mid incoming(v) = 0\}| = 1$. In the following, we call this node of a graph G v_{source}^{G}. In addition, *path* of length k from a vertex u to a vertex w is a sequence $v_1, ..., v_k, v_i \in \boldsymbol{V}$ of vertices that are connected through a sequence fo edges. More particularly, for the vertices of a path from u to w it holds that $(v_i, v_{i+1}) \in \boldsymbol{E} \wedge i < k \wedge v_i = u \wedge v_k = w \wedge u \neq w$.

Definition 2 (Source connectivity) *A graph G is called* source connected *if there exists a path of length $k, k \in \mathbb{N}^+$ from its source v_{source}^{G} to any other vertex $v \in \boldsymbol{V}$.*

For example, the graph presented in Fig. 2d is not source connected, because the vertex *T4″* cannot be reached from the source vertex *SE1″*.

Definition 3 (Checkpoint-subgraph) *We call a source connected subgraph H of G a* checkpoint-subgraph *iff $v^H_{source} \in \boldsymbol{V}^G_{ch}$. The function $\mu : \boldsymbol{V}^G_{ch} \rightarrow \boldsymbol{C}^G_{ch}$ maps a checkpoint of a graph G to a* set $\boldsymbol{C}^G_{ch}$ *of* all *the checkpoint-subgraphs of G starting from a particular checkpoint.*

It can be seen that all subgraphs shown in Fig. 2, except for the Fig. 2d, are a subset of the checkpoint-subgraphs of the models shown in Fig. 1.

Moving now to the definition of *subgraph isomorphism* for our formal model we are using this provided by Valiente [29].

Definition 4 (Subgraph isomorphism [29]) *A* subgraph isomorphism *of the graph $G_1 = (V_1, E_1)$ to a graph $G_2 = (V_2, E_2)$ is an injection $M \subset \boldsymbol{V}_1 \times \boldsymbol{V}_2$ such that, for every pair of vertices $v_i \in \boldsymbol{V}_1 \wedge v_j \in \boldsymbol{V}_1$ and $w_i \in \boldsymbol{V}_2 \wedge w_j \in \boldsymbol{V}_2$ with $v_i \in M \wedge w_i \in M$ and $(v_j, w_j) \in M, (w_i, w_j) \in \boldsymbol{E}_2$ if $(v_i, v_j) \in \boldsymbol{E}_1$. In such a case, M is a subgraph isomorphism of G_1 into G_2 and we denote $G_1 \cong G_2$.*

In the following we adjust the definitions of *common subgraph isomorphism* and *maximum common subgraph isomorphism* provided by Valiente [29] to be consistent the formal model of our approach.

Definition 5 (Common Subgraph Isomorphism (CSI)) *A* common subgraph isomorphism *of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a graph $C = (V, E), V \subseteq V_1 \times V_2, E \subseteq E_1 \times E_2$ for which it holds: $C \cong G_1 \wedge C \cong G_2$, i. e., C is a subgraph isomorphism from both G_1 to G_2 and vice versa. In this case we write $C \triangleq (G_1, G_2)$.*

In other words, an isomorphic subgraph of two graphs is a CSI when its structure has a one-to-one mapping in both graphs. For example, the results shown in Fig. 2 constitute different common subgraph isomorphisms of the two graphs G_1 (Model A) and G_2 (Model B) presented in Fig. 1. We are interested in principle in the largest possible such graph, so for this purpose we define:

Definition 6 (Maximum Common Subgraph Isomorphism (MCSI)) *A CSI $C = (\boldsymbol{V}_C, \boldsymbol{E}_V)$ of two graphs G_1 and G_2 is the* maximum common subgraph isomorphism *iff $\nexists\, C' = (\boldsymbol{V}_{C'}, \boldsymbol{E}_{C'}) \triangleq (G_1, G_2)$ such that $|\boldsymbol{V}_{C'}| > |\boldsymbol{V}_C|$.*

It therefore follows that the graph shown in Fig. 2a is the MCSI of the models shown in Fig. 1. Bringing now the above definitions together, we have:

Definition 7 (Common checkpoint-subgraph) *If two graphs $G_{ch1} \in \boldsymbol{C}^{G_1}_{ch}$ and $G_{ch2} \in \boldsymbol{C}^{G_2}_{ch}$ are checkpoint-subgraphs of graphs G_1 and G_2, respectively, then any $C \triangleq (G_{ch1}, G_{ch2})$, $C \in \boldsymbol{C}^{G_1}_{ch} \wedge C \in \boldsymbol{C}^{G_2}_{ch}$ is called* common checkpoint-subgraph. *We then define the set $\mathscr{M}(\boldsymbol{C}^{G_1}_{ch}, \boldsymbol{C}^{G_2}_{ch})$ as containing all MCSIs of the common checkpoint-subgraphs of two graphs G_1, G_2.*

We are now ready to define the concept of *relevant process fragments* as the output of our proposed approach. More specifically:

Definition 8 (Relevant Process Fragment (RPF)) *A graph C is called a Relevant Process Fragment (RPF) of a process model collection $\boldsymbol{G_{coll}} = \{G_1, \ldots, G_n\}$ when it holds:*

1. $C = (\boldsymbol{V}_C, \boldsymbol{E}_C) \in \mathscr{M}(\boldsymbol{C}_{ch}^{G_i}, \boldsymbol{C}_{ch}^{G_j})$, $G_i, G_j \in \boldsymbol{G_{coll}}$, $i, j = 1, \ldots, n$ (p_1)
2. $|\boldsymbol{V}_C| \geq v_{min}$, *where* $v_{min} \in \mathbb{N}^+$ (p_2)
3. $\exists v \in \boldsymbol{V}_C : type(v) \in \boldsymbol{Tsk}$ (p_3)

In other words, as RPF we define all the maximum common checkpoint-subgraphs between any two graphs G_1 and G_2 of a graph collection (p_1). It also follows from Definitions 3 and 8 that an RPF will always have exactly one source that is of type checkpoint. As seen in Definition 8 the RPF also have two extra properties. The first extra property is that the RPF must have a minimum number of vertices v_{min} (p_2). Since our approach focuses into detecting the most frequently occurring structures in a collection we have considered the $v_{min} = 5$ vertices. This is the minimum threshold from which a BPMN 2.0 subgraph has some structural interest. Moreover, tasks have a key role to the substance of any BPMN 2.0 process model. Thus, we are interested into detecting these structures that contain at least one task (p_3).

Some examples of RPFs can be seen in Fig. 2. More specifically, Fig. 2a and 2b show RPFs as they are a) both checkpoint-subgraphs starting from *SE1″* and *EG1″* respectively; b) both MCSIs of Model A (cf. Fig. 1a) and Model B (cf. Fig. 1b) for the corresponding checkpoints; c) they have at least 5 vertices; and d) contain at least one task element. In contrast, Fig. 2c is not an RPF because although it is a common checkpoint-subgraph of Model A and Model B it does not satisfy the following requirements: a) it does not have 5 vertices; and b) it does not contain a task element. Finally, the subgraph of Fig. 2d is not an RPF because it is not source connected and it does not have one source, i. e., it is not a common checkpoint-subgraph of the two models. In the following, we discuss an algorithm that identifies RPFs in a collection.

4 RPF Detection Approach

Let $\boldsymbol{G_{coll}}$ be a collection of BPMN 2.0 process models and $\boldsymbol{G^*_{coll}}$ the set of pairs of process models to check for reoccurring process fragments. $\boldsymbol{G^*_{coll}}$ is constructed to be irreflexive and not to be symmetric to avoid two runs for the same pairs of graphs. $\boldsymbol{G^*_{coll}}$ is the maximum set such that: 1) $\boldsymbol{G^*_{coll}} \subset \boldsymbol{G_{coll}} \times \boldsymbol{G_{coll}}$, 2) $x \in \boldsymbol{G_{coll}} \implies (x, x) \notin \boldsymbol{G^*_{coll}}$, 3) $x, y \in \boldsymbol{G_{coll}}, x \neq y \implies (x, y) \in \boldsymbol{G^*_{coll}} \veebar (y, x) \in \boldsymbol{G^*_{coll}}$, where $\veebar$ denotes the exclusive disjunction. For every tuple $(G_1, G_2) \in \boldsymbol{G^*_{coll}}$ we are executing the RoSE algorithm (cf. Algorithm 1) to a) construct the sets $\mathscr{M}(\boldsymbol{C}_{ch}^{G_1}, \boldsymbol{C}_{ch}^{G_2})$ (cf. Definition 7) for all possible pairs of checkpoints of the process models and b) to detect the valid RPFs $\forall\, G \in \mathscr{M}(\boldsymbol{C}_{ch}^{G_1}, \boldsymbol{C}_{ch}^{G_2})$. The RoSE

Algorithm 1 Creates the table of the matched edges of the compared models

Input: $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$: The models to compare. It holds that $|E_1| \geq |E_2|$.
Input: $\boldsymbol{V}_{ch}^{G_1}$, $\boldsymbol{V}_{ch}^{G_2}$: the sets of checkpoints of the two models G_1, G_2
Output: $\boldsymbol{RPF}_{coll}$: The set with the discovered Relevant Process Fragments

```
1: function RoSE()
2:     RPF_coll ← ∅
3:     for each outgoing(V_ch^G1 as e1) do
4:         maps ← ∅
5:         map ← InitializeMapWithZero()
6:         for each outgoing(V_ch^G2) as e2 do
7:             map ← CreateMap(e1, e2, map)
8:             maps ← maps ∪ {map}
9:         end for
10:        RPFS ← DetectValidRPFFromMaps(maps)
11:        RPF_coll ← RPF_coll ∪ {RPFS}
12:    end for
13:    return RPF_coll
14: end function
```

algorithm takes as input two process models G_1 and G_2 for which we need to detect the RPFs. As a precondition the model with the most edges always corresponds to G_1.The sets of process model checkpoints $\boldsymbol{V}_{ch}^{G_1}$ and $\boldsymbol{V}_{ch}^{G_2}$ are also given as input to the RoSE algorithm. At the end it outputs the collection of the detected RPFs.

Before proceeding to a detailed description of the algorithm we need to explain the involved variables. The variable $\boldsymbol{RPF}_{coll}$ (line 2) stores the collection of the detected RPFs, which will be returned at the end. The variable *map* (line 7) is an $|E_1|\times|E_2|$ *incidence matrix* in which each cell corresponds to a pair of edges (e_1, e_2) where $e_1 \in E_1$ and $e_2 \in E_2$. It is essentially a sparse matrix with ones in the cells where the two edges "match", or zeroes otherwise. Since our approach focuses only on the structural characteristics of the process models, two edges $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ are considered to "match" when the incident vertices are of the same type, i. e., $e_1 \simeq e_2 \Leftrightarrow type(u_1) = type(u_2) \wedge type(v_1) = type(v_2)$. For example, in Fig. 1 it holds that $\alpha \simeq \alpha'$ (start events connected to exclusive gateways) and $\beta \simeq \beta'$ (exclusive gateways connected to tasks) match. The initial incidence matrix of the process models of Fig. 1 is shown in Fig. 3a and it contains all the possible edge matches between the two models. For example, $\alpha \simeq \alpha'$, $\beta \simeq \alpha'$, $\beta \simeq \zeta'$, etc.

As seen in Fig. 3a the matrix may contain more than one ones per row and per column (i. e., $\Sigma_R > 1$ or $\Sigma_C > 1$). However, RPFs contain only one-to-one matches of edges. This is because the definition of isomorphism they rely on (Definition 4) is an injective function. Therefore, our goal is to reduce the extra ones per row and column consistently and derive the resulting RPF. By a closer look on the incidence matrix we observe that multiple ones on the same row indicate alternative matches of the corresponding edge, e. g., $\beta \simeq \beta'$ and $\beta \simeq \zeta'$.

Algorithm 2 Runs a Depth First Traversal (DFS) on checkpoint-subgraphs of G_1 and G_2 starting from a specific checkpoint to compare all the visited checkpoints with each other and create the indidence matrix (*map*) accordingly

Input: e_1, e_2 it takes as input an outgoing edge of a checkpoint of graph G_1 and G_2 respectively.
Input: *map* represents the sparse incidence matrix, initialized with zeroes ("0").
Output: *map* the sparse indidence matrix that contains ones ("1") on the cells for which the edges matched.

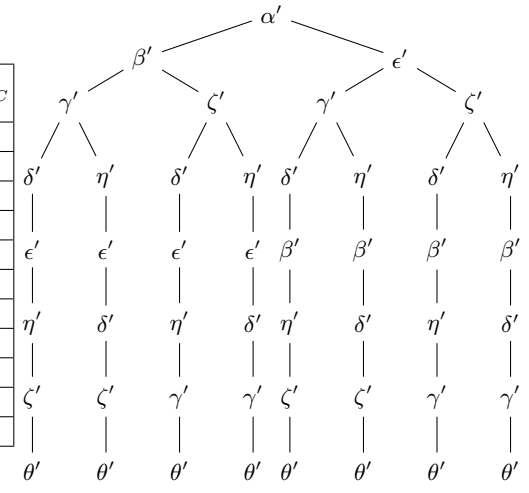
```
1: function CreateMap(e1, e2, map)          ▷ Let e1 = (u1, v1) and e2 = (u2, v2)
2:     if (e1 ≃ e2) then          ▷ If the two edges are of the same type (cf. Sect. 3)
3:         O_u1 ← outgoing(u1)
4:         O_u2 ← outgoint(u2)
5:         map[e1][e2] ← 1
6:         for each O_u1 as o1 do
7:             for each O_u2 as o2 do
8:                 map ← CreateMap(o1, o2, map)
9:             end for
10:        end for
11:    end if
12:    return map;
13: end function
```

For obtaining the one to one isomorphism we need to choose either $\beta \simeq \beta'$ or $\beta \simeq \zeta'$. Let us assume for example that we chose the first match (i. e., $\beta \simeq \beta'$). In this case we need to eliminate the $\beta \simeq \zeta'$ from the incidence matrix. We also need to eliminate the rest of the ones on the β column so that it will not be chosen for another edge (i. e., row). Similarly it can be concluded that the incidence matrix should contain at most one one in the Σ_R and Σ_C for any detected RPF. In other words, for the detection of an RPF it should hold that for each row $\Sigma_R \leq 1$ and for each column $\Sigma_C \leq 1$.

The challenge at this point is to eliminate the redundant ones of the incidence matrix consistently. Reducing the graph isomorphisms to tree searches is a well established technique [28]. To this effect, we are introducing a tree which represents all the possible isomorphic candidates indicated by the incidence matrix. By traversing each row of the matrix, we are gradually building the tree that represents all the possible choices (i. e., RPF candidates). The tree that maps to the incidence matrix of Fig. 3a is shown in Fig. 3b. It represents all the possible isomorphic choices that can be produced by the incidence matrix when gradually eliminating the ones of each row. The root of the tree is the source node of G_2 and each child of the root indicates the possible choices of ones of the next row. Each tree path from the root to the leaves indicates the set of edges that build a candidate RPF. For example, for the incidence matrix and corresponding decision tree of Fig. 3 we start traversing the matrix from the cell (α, α') which is the root. The next row (β) has ones in the positions (β, β') and (β, ϵ'), we can choose either of one of these. Thus, the root node α' on the

G_1 \ G_2	$\boldsymbol{\alpha'}$	$\boldsymbol{\beta'}$	$\boldsymbol{\gamma'}$	$\boldsymbol{\delta'}$	$\boldsymbol{\epsilon'}$	$\boldsymbol{\zeta'}$	$\boldsymbol{\eta'}$	$\boldsymbol{\theta'}$	Σ_C
$\boldsymbol{\alpha}$	1	0	0	0	0	0	0	0	1
$\boldsymbol{\beta}$	0	1	0	0	1	0	0	0	2
$\boldsymbol{\gamma}$	0	0	1	0	0	1	0	0	2
$\boldsymbol{\delta}$	0	0	0	1	0	0	1	0	2
$\boldsymbol{\epsilon}$	0	1	0	0	1	0	0	0	2
$\boldsymbol{\zeta}$	0	0	0	1	0	0	1	0	2
$\boldsymbol{\eta}$	0	1	0	0	1	0	0	0	2
$\boldsymbol{\theta}$	0	0	1	0	0	1	0	0	2
$\boldsymbol{\iota}$	0	0	0	1	0	0	1	0	2
$\boldsymbol{\kappa}$	0	0	0	0	0	0	0	1	1
Σ_R	1	3	2	3	3	2	3	1	

(a) Incidence Matrix



(b) Decision Tree for RPF Detection

Figure 3: Incidence Matrix and Corresponding Tree Search

tree has the children β' and ϵ' as they constitute alternative choices of the RPF. Proceeding now to the next row of the incidence matrix the γ is mapped to γ' or ζ'. This choice is represented in the tree by putting the γ' and ζ' as children of β' and ϵ', i. e., the alternatives produced by the edge β of the previous step. Similarly, the ones of the row δ will produce the children of the leaves γ' and ζ' produced in the previous step. We are now on the row ϵ where the ones are on the β' and ϵ'. The ones of these positions have already been used on the first step (edge α) of our procedure and are already placed as the children of the root. In this case, we cannot add both as children because each tree path must contain each edge only once. Therefore, for sub tree of the decision tree that starts from β' we will only add as leaf-children the ϵ', while for the sub tree of the tree that starts from ϵ' we will add as leaf-children the β'. We proceed likewise to build the rest of the decision tree. With respect the method described above, for the motivating example (cf. Fig. 1), the resulting RPF starting from the checkpoints (*SE1* and *SE1′*) is the model of Fig. 2a which is isomorphic to the complete model B (cf. Fig. 1b). Namely, the resulting RPF is a subgraph of model A (cf. Fig. 1a) and the complete model B (cf. Fig. 1b). In this case the tree is presenting different orderings of the same set of edges $(\alpha', \beta', \gamma', \delta', \epsilon', \eta', \zeta', \theta')$.

For detecting the RPFs of two process models the RoSE algorithm (Algorithm 1) exploits the aforementioned constructs of incidence matrix and its corresponding decision tree. The CreateMap function (Algorithm 2) is called by the RoSE algorithm to compare all possible combinations of the checkpoints of G_1 to the checkpoints of G_2 (lines 3 and 6). In this case the checkpoint helps us reduce the occurring comparisons (line 2). When the type of two checkpoints

do not match (e. g., comparing start event to an exclusive gateway) the comparison is terminated immediately and an empty incidence matrix is returned by the CreateMap function (cf. line 12). If the two checkpoints match, we start traversing all possible combinations of the edges of the two process models by using a Depth First Search (DFS) algorithm. During the DFS traversals (Algorithm 2) we are comparing all edges of the process models to each other and we gradually build the incidence matrix which is stored in the *map* variable (done in CreateMap, called in line 7). If there is a mismatch between two edges, the DFS traversal stops for the corresponding path and the map will contain zeroes to all the forthcoming edges.

When an incidence matrix of a comparison is constructed we insert it to a set of incidence matrices variables (*maps* in line 8). Then, through the function "DetectValidRpfFromMaps(*maps*)" (line 10) we are detecting all the valid RPFs that occur from each $map \in maps$ based on Definition 8, by applying the tree approach described above. In order to make sure we will choose the maximal isomorphism from each comparison we sort the paths of the tree according to their size. Then we validate them one by one against the RPF properties (cf. Definition 8). If we find a valid RPF then the searching on this tree search is stopped and the RPF is returned. Since the valid RPF contains the maximum number of discovered edges (as we are using the longest path), there will be no other RPF with more edges. The definition of the function "DetectValidRpfFromMaps(*maps*)" could not be provided due to space limitations. However, its prototypical implementation is available as open source[2].

The detected RPFs are inserted into an $\boldsymbol{RPF}_{coll}$ (line 11) which is the set of detected RPFs for the given pair of process models (G_1, G_2). Finally, the detected $\boldsymbol{RPF}_{coll}$ is returned by the RoSE algorithm (line 13). The RoSE algorithm is iteratively called for all the possible distinct pairs of model combinations in the collection. Each RPF detected for a distinct model combination is compared with the set of the previously detected RPFs. If the RPF is already detected by another comparison, it is not added to the results set but is counted as duplicate entry. In this way we manage to derive statistics for the frequency of occurrence of the detected RPFs. In previous work [25] we have introduced an initial approach for deriving the frequency of occurrence of the detected RPFs. In this work we have extended the presented approach to be compliant with the RoSE algorithm.

To sum up, the proposed methodology uses DFS traversal on the two models for discovering all possible isomorphisms between their edges and insert them into the incidence matrix (Fig. 3a). By construction the incidence matrix will contain all possible subgraph isomorphisms between the edges of the two models. We are then applying a tree search [28] to consistently produce all possible isomorphisms between the edges of the two process models. Our methodology combines well established techniques (DFS, incidence matrix, tree search) for the detection of RPFs between two process models. Thus, if there is one solution our methodology will detect it as it basically exhausts the search space. Consequently, we can argue that the proposed methodology is complete. In terms of complexity, due to the

[2] https://github.com/marigianna-iaas/RoSE

naïve nature of the decision tree construction in the worst case we are in the area of $O(m^n)$ where $m = |\boldsymbol{E}_1|$ and $n = |\boldsymbol{E}_2|$. Furthermore, we need to compare all possible pairs of graphs in the collection, denoting a factorial time $O(k!)$ where k is the size of the business process model collection. However, in practice and due to the sparsity of the mapping table and the low ratio of checkpoints per model the observed performance, as we will discuss in the following section, can be achieved in realistic times.

5 Evaluation

In order to validate and evaluate the proposed approach we implemented the RoSE algorithm in a proof-of-concept prototype that utilizes the EMF BPMN 2.0 metamodel[3] and is available in an open source model[4]. For measurement purposes we deployed our implementation on a virtual machine in a private cloud solution. The VM was configured with 8 MB memory and 4 CPUs, using the Ubuntu LTS 14.04 operating system. The complete collection of real-world BPMN 2.0 process models collected in the scope of "BenchFlow" project, contains 11,151 BPMN 2.0 process models coming from the: i) IBM Industry Process and Service Models[5], ii) the BPMN 2.0 standard, the research by Pietsch et al. [21], iii) the BPM Academic Initiative[6] and iv) other research and industrial partners. The BPM Academic Initiative provides 8,657 BPMN models. For deriving the models we have selected the "BPMN 2.0 Process" and "BPMN 1.1" 100% correctness, last revision and all available languages and sizes. The process models derived from the BPM Academic Initiative do not comply with the BPMN 2.0 standard serialization in the Extensible Markup Language (XML). Therefore, they currently cannot be parsed by the EMF BPMN 2.0 metamodel and consequently our algorithms. For this reason they are excluded from our collection and we resulted in a subset of our original collection, which contains 1,806 real-world BPMN 2.0 process models. Since the RoSE algorithm currently does not support cyclic graphs, we have also detected and extracted the cyclic process models from our initial collection. This has resulted to the analysis of 1,587 BPMN 2.0 acyclic process models, that are given as input for the validation of the RoSE algorithm. We have set the RoSE algorithm to detect RPFs with $size \geq 5$, namely the minimum size an RPF can have. The algorithm ran over all models of the collection compared pairwise, as discussed in the previous section.

For the aforementioned collection and infrastructure, the RoSE algorithm executed for 9 hours and 45 min, and detected 94 unique RPFs. For determining the most frequently occurring RPFs we calculate how many times it has been detected by RoSE (Fig. 4). The first five most frequently occurring RPFs of our collection and the number of comparisons that generated these RPFs are shown in Fig. 5. Figure 4 on the other hand shows the sizes of the detected RPFs and

[3] http://www.eclipse.org/modeling/mdt/?project=bpmn2
[4] https://github.com/marigianna-iaas/RoSE
[5] http://www-01.ibm.com/software/data/industry-models/
[6] http://bpmai.org/

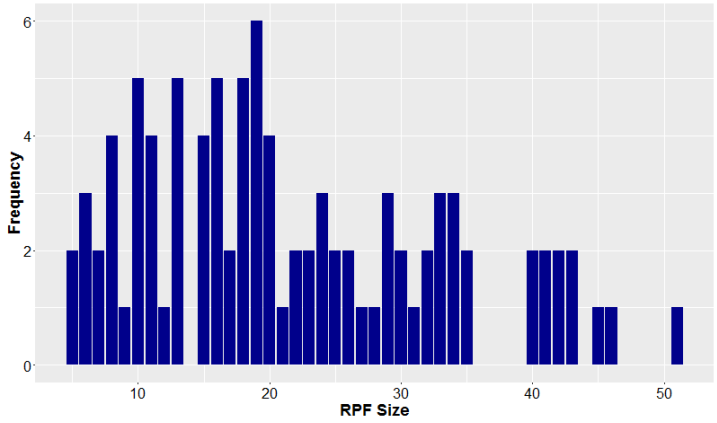
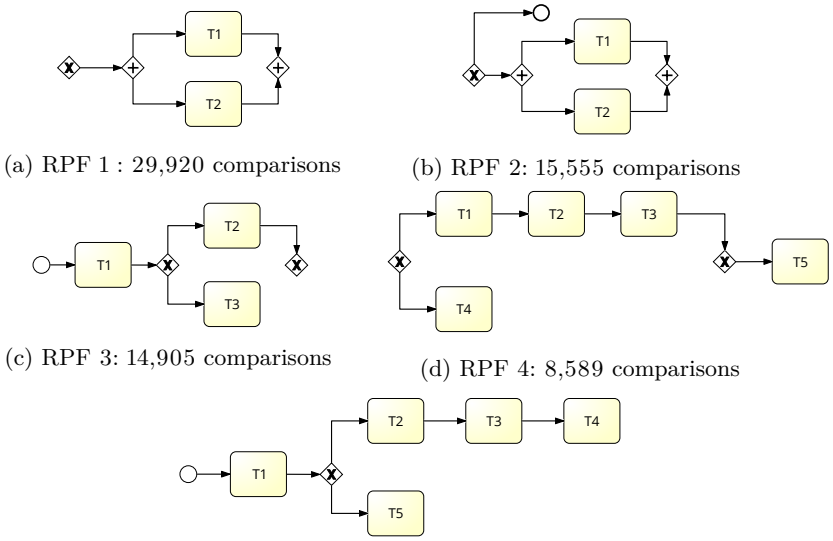


Figure 4: Frequency count per detected RPF size

their corresponding size frequencies. As it can be seen, most of the detected RPFs have a $5 \leq size \leq 30$, with the majority of them in the range from $size = 13$ to $size = 30$ (first and third quantile, respectively), with mean size $\overline{size} = 21.69$, median $\widetilde{size} = 19$, and standard deviation $\sigma_{size} = 11.28$. However, there are also many detected RPFs for even bigger sizes, up to the size 51 (22 of them to be precise). In most of the cases these RPFs are detected in different versions of the same process models. Furthermore, most of the models originate from the IBM Process and Service models collection. It can therefore be concluded that the modelers are reusing some complex structures as identified best practices. If the RoSE algorithm is applied on a bigger and more diverse sample, these results might vary, and more conclusions could be drawn on the modeling practices.

The sizes discussed in Fig. 4 relate to the whole set of detected RPFs. However, by the application of the RoSE algorithm we are also interested in the RPFs that were more frequently detected in the collection. These RPFs are shown in Fig. 5. As seen in Fig. 5 the most frequently occurring RPF are smaller structures. However, two more complex structures also appear in the top-5 results (Fig. 5d and 5e). The RPF shown in Fig. 5a is a sub-structure of Fig. 5b. Thus, the count 29,920 of Fig. 5a also contains the occurrences of the RPF of Fig. 5b. By these two RPFs we can conclude that in almost half of the cases the modelers choose to conditionally terminate the process. The third RPF (cf. Fig. 5c) indicates a preferred approach of initializing a process. Generally, the first three RPFs (Fig. 5a to 5c) basically confirm the widely accepted usage of the well established control flow workflow patterns and reveal interesting combinations of these [1]. The next two detected RPFs shown in Fig. 5d and 5e are more complex structures. Although these two RPFs were extracted almost half the times than the rest of the detected RPFs, they still reveal modeling techniques that could be exploited towards enabling process model modeling re-usability.

The RoSE algorithm provides an effective mean towards detecting reoccurring structures in a process model collection. While its efficiency can be definitely



(a) RPF 1 : 29,920 comparisons

(b) RPF 2: 15,555 comparisons

(c) RPF 3: 14,905 comparisons

(d) RPF 4: 8,589 comparisons

(e) RPF 5: 8,548 comparisons

Figure 5: Resulting RPFs of the BPMN 2.0 collection

improved, it needs to be taken into account that there are no "online" requirements for this process. Since this process is meant to run only once for each collection (or every time the collection is updated, which happens very infrequently) then the total running time per se is not an issue for its operation. Nevertheless, in future work we aim to improve the performance of the algorithm by using heuristics-based methods like the one discussed in [30].

6 Related Work

Process fragmentation is frequently discussed in the literature [8, 24] and is accepted to have an important role in the re-usability of process models. Process fragments are introduced as incomplete process parts that are extended by adding Business Process Management compliance features [24]. The definition of Relevant Process Fragments proposed in this work is an extension to the original definition [24] with particular focus on the reoccurrence of the fragment as well as its structural properties.

To the extent of our knowledge the approach introduced in this work is the first complete approach that detects and extracts the structural similarities from a collection of BPMN 2.0 process models by relying solely on the structural information of the process models. A preliminary version of this approach has been published by Skouradaki et al. [26] and evaluated through an artificial process models collection. The approach presented in Skouradaki et al. [26] has also been extended for counting the frequency of occurrence of the detected RPFs and validated again through the same collection of artificial models [25].

In the evaluation of this approach using real-life process models we have detected inconsistencies in the results which are solved by the presented version of the RoSE algorithm. The methodology for counting the frequency of occurrence of the detected RPFs preserves the same logic as the one presented in [25] but is extended to be compliant with the RoSE algorithm. Generally, the presented version of the RoSE algorithm can be seen as a more comprehensive approach than the one presented in the previous works. Hertis and Juric [14] conduct an analysis similar to ours for the Web Services Business Process Execution Language (WS-BPEL) [20]. In order to detect the reoccurring structures in a collection of thousands of WS-BPEL process models, they transform the process models to process trees. Afterwards, they apply tree mining algorithms to detect and extract the reoccurring structures. Although the ultimate goal of this work is similar to ours, the different nature of BPMN 2.0 language does not allow to apply the same tree mining techniques for similar structures detection. Moreover, in our case we do not use a given pattern or model to search against the collection, but we extract the detected reoccurring similar structures, that are originally unknown. A prototype for the comparison of BPMN 2.0 process models is introduced by Pietsch and Wenzel [21]. In this approach it is assumed that the compared process models are different versions of each other and heuristics based on textual semantics are used for the results. Although the approach seems very promising it is argued by the authors that it might not be efficient to large, complex real world process models.

As discussed in the introduction already, process model similarities is a major research stream that branches towards three directions: textual similarities, behavioral similarities, and structural similarities [6]. The textual similarities approaches base their comparisons on the labels of the process elements (e. g., task labels, event labels etc.), the behavioral similarities exploit the execution semantics of the process models, and the approaches on structural similarities compare the textual semantics of the process model as well as their topology [5]. Most of the approaches presented by Dijkman et al. [5] are expected to under perform for process models with $size > 20$. On the contrary, our approach has been applied to much bigger models and has even detected RPFs with $30 \leq size \leq 51$.

The APROMORE process model repository [17] is extensively used by the Business Process Management community for process models comparisons. Similarity search and pattern-based analysis are the features that are more relevant to the scope of this work. However, the similarity search indicates the percentage of similarity between two process models, without indicating the exact regions of similarities for the process models. Likewise, the pattern based analysis is searching the existince of a certain pattern in a collection of process models. Therefore, the features supported by the APROMORE [17] repository do not satisfy the objective of this work. La Rosa et al. [22] propose a method to detect similar parts of process models. The method relies on textual and behavioral similarities, thus it could not be applied to our collection.

Similarly, Ivanov et al. [15] present a prototype for detecting the similarities of BPMN 2.0 process models by basing on behavioral semantics. Although the

approaches mentioned above cannot be applied to our business process models collection, they can be potentially combined with our approach for a more accurate detection of reoccurring structures, that also considers the context of the process model. The Refined Process Structure Tree (RPST) is a technique for fragmenting the process model into single-entry-single-exit (SESE) regions, out of which we can build a hierarchical tree representation [23]. The RPST has been utilized for detecting mappings of process models [7, 9]. However, in our case the derived fragmentation will not produce all the possible substructures that should be examined as reoccurring. Therefore, the RPST is not used by our approach.

For the development of the proposed approach we have also investigated existing approaches in subgraph isomorphism and pattern matching techniques. During our research we were not able to find any approaches that accept graphs as an input and return the set of reoccurring subgraph isomorphisms, but in most of the cases a smaller sub-graph is searched and compared to the bigger graphs [4, 28, 30]. For the RoSE algorithm we have adopted existing techniques suggested by [28, 30] and customized them for the detection and extraction of RPFs.

7 Conclusions and Future Work

In this work we introduced the RoSE algorithm that detects reoccurring structures in a collection of BPMN 2.0 process models. The method has been developed under the needs of defining meaningful test scenarios for benchmarking BPMN 2.0 WfMS. However, structural similarities of process models is widely accepted to be useful and finds application in different scenarios of the Business Process Management, such as detection of difference in various in versions of the same process models [21], clone detection in process model repositories [7, 9] or generation of synthetic process models with respect to reoccurring fragments [32].

In contrast to most of the approaches of subgraph isomorphisms that base on a known subgraph that is searched against a bigger graph, our approach starts by comparing the bigger graphs (i. e., process models). The goal is to detect and extract the reoccurring subgraphs (RPFs), which are basically the reoccurring structures (or subgraphs) of the compared process models. Thus, in the scope of this work we first defined the underlying formal model of our approach and then we described the approach that detects the RPFs of the BPMN 2.0 collection. Our approach utilizes and customizes well established techniques of sub-graph isomorphism as for example this of Ullmann's algorithm [28] and we argued that it is complete as it exhausts the search space. The presented approach currently does not support cyclic structures. Therefore we detected the process models with cyclic structures in a collection of 1,806 real-world process models and validated the RoSE algorithm against the remaining 1,587 process models. Unlike similar efforts [2] the algorithm completed successfully in realistic times and discovered 94 RPFs, some of which are small and confirm the usage of the control-flow workflow patterns, while other are more complex and indicate modeling trends.

In future research we plan to extend the analyzed collection of process models by including the models of the BPM Academic Initiative and other diverse collections of real-world process models. For a more comprehensive approach, we will extend the RoSE algorithm to support cycles and combine it with existing techniques that apply textual or behavioral semantics, in order to detect reoccurring structures of similar context. Moreover, we envision to improve the performance of the algorithm by applying heuristic-based methods as these proposed by Valiente and Martínez [30] and execute comparative performance analysis against well known pattern discovery algorithms such as gSPAN [31] and Gaston [19]. Subsequently, we will employ the RPFs detected by the RoSE algorithm towards the generation of representative, executable BPMN 2.0 process models that will be used as a test scenario to the first standard benchmark of BPMN 2.0 compliant Workflow Management Systems.

Acknowledgments This work is funded by the "BenchFlow – A Benchmark for Workflow Management Systems" project DACH Grant No. 200021E-145062/1 and by the "SmartOrchestra" (01MD16001F) project funded by the BMWi.

References

1. van der Aalst, W.M.P., et al.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (Jul 2003)
2. Breuker, D., Delfmann, P., Dietrich, H.A., Steinhorst, M.: Graph Theory and Model Collection Management: Conceptual Framework and Runtime Analysis of Selected Graph Algorithms. Information Systems and e-Business Management 13(1), 69–106 (Feb 2014)
3. Burattin, A.: Process Mining Techniques in Business Environments. Springer International Publishing (2015)
4. Cordella, L.P., et al.: A (Sub)graph Isomorphism Algorithm for Matching Large Graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence 26(10), 1367–1372 (2004)
5. Dijkman, R.M., et al.: Similarity of Business Process Models: Metrics and Evaluation. Information Systems 36(2), 498–516 (Apr 2011)
6. Dijkman, R.M., et al.: A Short Survey on Process Model Similarity. In: Seminal Contributions to Information Systems Engineering. Springer (2013)
7. Dumas, M., et al.: Fast Detection of Exact Clones in Business Process Model Repositories. Information Systems 38(4), 619–633 (Jun 2013)
8. Ekanayake, C.C., et al.: Fragment-Based Version Management for Repositories of Business Process Models. In: OTM. Springer (2011)
9. Ekanayake, C.C., et al.: Approximate Clone Detection in Repositories of Business Process Models. In: BPM. Springer (2012)
10. Feitelson, D.G.: Workload Modeling for Computer Systems Performance Evaluation. Cambridge University Press (2015)
11. Ferme, V., et al.: A Container-centric Methodology for Benchmarking Workflow Management Systems. In: Proceedings of CLOSER '16. Springer (2016)
12. Grossmann, W., Rinderle-Ma, S.: Fundamentals of Business Intelligence. Data-Centric Systems and Applications, Springer (2015)

13. Gupta, A., Nishimura, N.: The Complexity of Subgraph Isomorphism: Duality Results for Graphs of Bounded Path- and Tree-Width. Tech. Rep. CS-95-14. Univ. of Waterloo, Comp. Sc. Dep., Waterloo, Ontario, Canada (1995)
14. Hertis, M., Juric, M.: An Empirical Analysis of Business Process Execution Language Usage. IEEE Transactions on Software Engineering 40(8), 738–757 (Aug 2014)
15. Ivanov, S., et al.: BPMNDiffViz: A Tool for BPMN Models Comparison. In: BPM (Demos). CEUR-WS.org (2015)
16. Jordan, D., Evdemon, J.: Business Process Model And Notation (BPMN) version 2.0. Object Management Group, Inc (January 2011), http://www.omg.org/spec/BPMN/2.0/
17. La Rosa, M., et al.: APROMORE: An Advanced Process Model Repository. Expert Systems with Applications 38(6), 7029–7040 (2011)
18. zur Muehlen, M., Recker, J.: How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In: Porceedings of CAiSE '08. Springer (2008)
19. Nijssen, S., Kok, J.N.: The Gaston Tool for Frequent Subgraph Mining. Electronic Notes in Theoretical Computer Science 127(1), 77–87 (Mar 2005)
20. OASIS: Web Services Business Process Execution Language Version 2.0 – OASIS Standard (2007)
21. Pietsch, P., Wenzel, S.: Comparison of BPMN2 Diagrams. In: Proceedings of BPMN '12. pp. 83–97. Springer (2012)
22. Pittke, F., et al.: Enabling Reuse of Process Models through the Detection of Similar Process Parts. In: Business Process Management Workshops. Springer (2013)
23. Polyvyanyy, A., et al.: Simplified Computation and Generalization of the Refined Process Structure Tree. In: WS-FM. Springer (2011)
24. Schumm, D., et al.: Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls. In: MKWI (2010)
25. Skouradaki, M., Leymann, F.: Detecting Frequently Recurring Structures in BPMN 2.0 Process Models. In: SummerSOC. IBM (2015)
26. Skouradaki, M., et al.: Application of Sub-Graph Isomorphism to Extract Reoccurring Structures from BPMN 2.0 Process Models. In: Proceedings of SOSE '15. IEEE (2015)
27. Skouradaki, M., et al.: On the Road to Benchmarking BPMN 2.0 Workflow Engines. In: IProceedings of ICPE '15 (2015)
28. Ullmann, J.R.: An Algorithm for Subgraph Isomorphism. ACM 23(1), 31–42 (1976)
29. Valiente, G.: Algorithms on Trees and Graphs. Springer, Berlin; New York (2002)
30. Valiente, G., Martínez, C.: An Algorithm for Graph Pattern-Matching. In: South American Workshop on String Processing. Carleton University Press (1997)
31. Yan, X., Han, J.: gSpan: Graph-Based Substructure Pattern Mining. In: Proceedings of ICDM '02. Institute of Electrical & Electronics Engineers (IEEE) (2002)
32. Yan, Z., Dijkman, R., Grefen, P.: Generating Process Model Collections. Software and Systems Modeling (Oct 2015)