Application of Sub-Graph Isomorphism to Extract Reoccurring Structures from BPMN 2.0 Process Models

Marigianna Skouradaki, Katharina Görlach, Michael Hahn, Frank Leymann

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{skouradaki, goerlach, hahn, leymann}@iaas.uni-stuttgart.de

Application of Sub-Graph Isomorphism to Extract Reoccurring Structures from BPMN 2.0 Process Models

Marigianna Skouradaki , Katharina Görlach, Michael Hahn, Frank Leymann
Institute of Architecture of Application Systems
University of Stuttgart
Universitaetsstrasse 38, 70569 Stuttgart, Germany
{skourama,goerlach,hahnml,leymann}@iaas.uni-stuttgart.de

***Abstract*—The state-of-art approaches in structural similarities of process models base their operations on behavioral data and text semantics. These data is usually missing from mock-up or obfuscated process models. This fact makes it complicated to apply current approaches on these types of models. We examine the problem of the automated detection of re-occurring structures in a collection of process models, when text semantics or behavioral data are missing. This problem is a case of (sub)graph isomorphism, which is mentioned as NP-complete in the literature. Since the process models are very special types of attributed directed graphs we are able to develop an approach that runs with logarithmic complexity. In this work we set the theoretical basis, develop a configurable approach for the detection of re-occurring structures in any process models collection, and validate it against a set of BPMN 2.0 models. We define two execution scenarios and discuss the relation of the execution times with the complexity of the comparisons. Finally, we analyze the detected structures, and propose the configurations that lead to optimal results.**

***Keywords*-structural similarities, process models, BPMN 2.0, process fragments, subgraph isomorphism**

I. Introduction

Re-usability of process models is frequently discussed in the literature [1]–[5]. An efficient approach of re-using parts of the process models will improve the engineering of process models. It will increase the performance of the designers, because they will not need to start everything from scratch [2], and reinforce the usage of best design practices [6].

In order to determine these parts of a process model that can be re-used, one needs to analyze large collections of process models, and identify similar labels, structures, or behavior. In these terms the emerging research has done great steps towards the process models similarities by focusing on three different streams: 1) Text semantics, 2) Structural analysis and 3) Behavioral analysis of the process models [7].

It is however a usual phenomenon that large collections of process models (e.g. IBM Industry Models, Process Models from research projects etc.) contain non-executable or obfuscated models. These means that text semantic data or behavioral information are not available. This lack of data basically burdens the application of text semantics or behavioral similarity approaches.

One would conclude that the appropriate choice in this case is the structural similarities approach. However, the current approaches [7]–[10] are highly dependent to text semantics, and behavioral similarities. So, it is expected that there can be a limited application on such collections of models. In these terms the only option left is the application of similarity comparisons with a unique focus on the structural topology of the process models. This process is defined in graph theory as (sub) graph isomorphism. It is currently widely applied on different fields of computer science, such as networks, bio-informatics, and semantic web. However, it is rarely met in the field of Business Process Management, and we were not able to locate any approach with a focus on BPMN 2.0 process models.

In this sense, we define a configurable approach of sub-graph isomorphism to automate the detection of re-occurring structures in a collection of process models. Although the method that we present can be applied to process models expressed in any workflow-language we make a focus on BPMN 2.0 process models for this work. More particularly, the main contributions of this paper are to:

- Set the theoretical framework that will be used as a basis to our approach. In these terms, we extend the definition of “Process Fragments” presented in [2] and introduce the concept of “Checkpoints”.
- Define a configurable approach of sub-graph isomorphism that can be applied to any type of process models, without the usage of data information, or text semantics.
- Apply the approach on a collection of BPMN 2.0 process models.
- Define the metric of “Comparison Complexity” to support the evaluation of the approach.
- Evaluate the approach on a set of BPMN 2.0 process models.
- Discuss the corresponding execution times and the detected structures, and
- Suggest the optimal configuration options of our approach.

The rest of this paper is structured as follows: Section II-A motivates the goal of this work, section II-B describes the emerging related work, section II-C defines the concepts that will be used as a basis to our approach. Section III describes the detection of the re-occurring structures in a collection of BPMN 2.0 process models, section IV validates our approach against a collection of process models, examines the algorithm's behavior with respect to two execution scenarios and analyses the results. Section V gives a summary of our work and an outlook to our future plans.

II. Background

A. Problem Definition

In this work we address the problem of the automatic discovery of reoccurring structures in a collection of process models. The approach described in this paper will be applied in a collection of 5372 BPMN 2.0 [11] process models that were collected from the IBM Industry Models, BPM Academic Initiative [12], and research projects. Most of these process models are obfuscated or mock-ups. This means that we do not know their text semantics, and we cannot execute them. We therefore examine the graph-nature of process models, and intend to apply approaches from the graph theory in order to find their reoccurring structures.

In its most basic format a graph G = *(V,E)* is composed of nodes and edges. We call V the set of nodes and $E \subset V \times V$ the set of edges of a graph G. If two nodes (u, v) are connected by an edge $e \in E$ this is denoted by $e = (u, v)$ and these two nodes are called *adjacent*. In the case where an edge has a direction, then it is called "directed", and the graph *"directed graph"*. Graph's elements can also contain information such as labels, or other attributes. In this case the graph is called *attributed graph* [13].

In graph theory the task of discovering similar structures is expressed as sub-graph discovery problem. There we need to find sub-graphs that reoccur in a set of bigger graphs. Sub-graph discovery is a sub-category of the general problem of subgraph isomorphism which is proven to be NP-Complete [14], [15]. However there are special cases of graphs and matching problems that are proven to be of lower complexity [16].

Figure 1 presents two process models expressed in BPMN 2.0. We can observe in the figure 1 that the models comply with the above definition of attributed directed graphs. There are nodes (in BPMN 2.0 tasks, events and gateways), directed edges (in BPMN 2.0 sequence flows), and labeling (BPMN 2.0 language semantics on events and gateways, and names on tasks). Hence, process models are a special type of directed attributed graphs. Our goal is to define a method for automatic discovery of reoccurring structures in a collection of process models, with a complexity lower than NP-Complete.

Figure 1 shows four pairs of structures that reoccur in two BPMN 2.0 process models. These structures are not the complete set of reoccurring structures in these two process models. However we considered these as representative cases as they demonstrate the following attributes:

1) Structures can be nested within each other. Example structures of this attribute are marked with the red solid and the blue dash-dot line.
2) Reoccurring structures can appear in different positions of the process model. As position we define the number of nodes that we need to traverse from the model's start node until we discover the first node of a marked structure. Beginning from the model's first node. This attribute applies to all represented structures.
3) Structures can be "partially" similar. This means that some of the outgoing edges of a node can lead to matching mappings. Structures marked with the orange dash line, the green dot line are some examples of this attribute.

It is also possible that a structure demonstrates more than one of the above attributes. For example the structure marked with the blue dash-dot line is nested (attribute 2) and appears in different positions of the diagram (attribute 3).

In order to discover the structures with the aforementioned attributes we need to specify techniques to traverse and fragment the models. Going back to the basics, for graph traversals we backdated to the approaches of Depth-First-Search (DFS) or Breadth-First-Search (BFS) [17]. However, it was soon clear that a straightforward application of these approaches would not lead to the discovery of all reoccurring structures. In BFS and DFS each edge of the model is "discovered" only once. While we need to traverse the models in parallel and check all possible pairs of edges between the two models for matching. This means that a node should be traversed more than once. For example, the structures that start with the node marked with $\mathcal{C}$ in Model A, and $\mathcal{B}'$ in Model B match. This is because the lower part of the structure in Model A has the same sequence of node types (exclusive gateway, task, task, exclusive gateway) with the upper part of the structure marked in Model B. These structures are marked with green dotted line in figure 1). In order to cover cases like that, we use DFS algorithm and customize it to traverse and compare all possible path combinations of the two models. This approach is explained in detail in section III.

As mentioned before, a fragmentation methodology also needs to be defined. Extracting arbitrary parts out of complex process models is an essential technique [18]. In [19], [20] and [21] a methodology to decompose the structure of a process model into single-entry single-exit (SESE) regions is defined. In [22] it is explained how to detect sub-graphs within a BPMN models through the identification of SESE regions. These approaches focus on the definition of the Refined Process Structure Tree (RPST) which is basically an hierarchical structure of sub-graphs that comply with structural constraints. We considered to create the RPSTs of

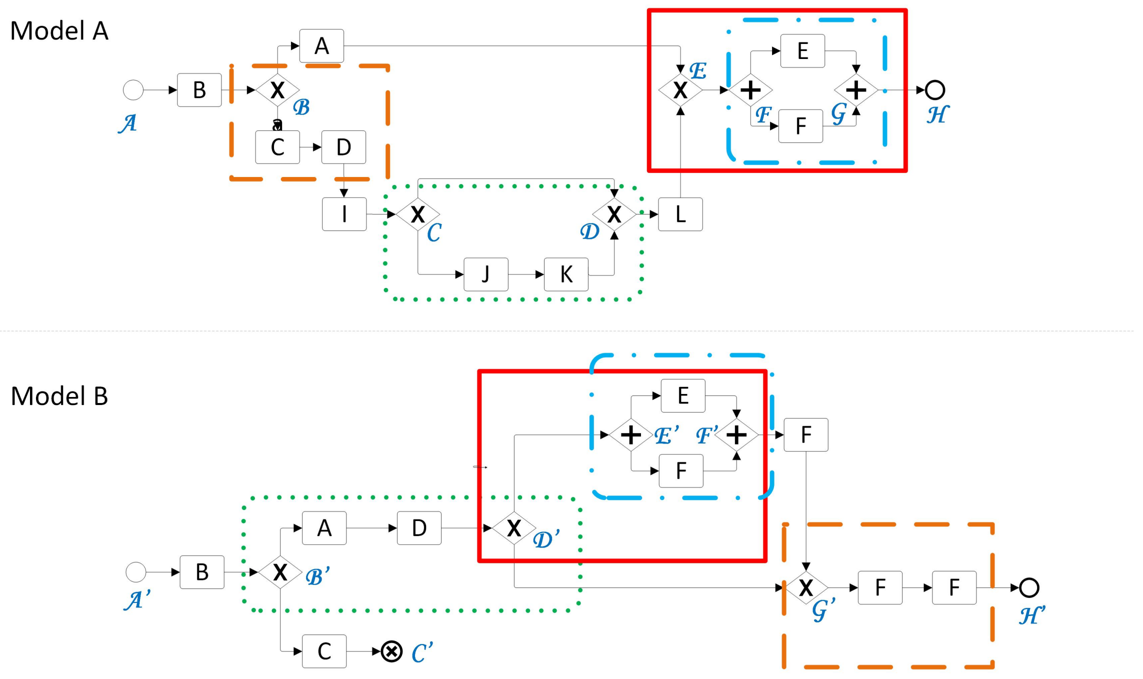

Figure 1: Manually Recognized Relevant Process Fragments

the process models, to compare them, and then to apply tree comparison techniques to export the reoccurring structures. However, the methodology of RPSTs divides the process model into SESE regions, and results in a subset of regions, out of which only a small subset of the existing re-occurring structures would be discovered (e.g. only the structure marked with the blue dash-dot line in figure 1). In order to create the complete set of sub-regions that should be examined as possible re-occurring structures, it is imperative to define a new methodology of fragmentation. This is described in detail in section II-C.

B. *Related Work*

We consider our work related to the following major fields of research: process fragmentation, process model similarities, and graph matching approaches. *Process fragmentation* plays a big role in the reuse of process models [1]–[5]. In [4], [5] "Process Fragments" are introduced as incomplete process parts that can be dynamically glued together during runtime. This approach has been extended by [2] where they add Business Process Management compliance features to the definition of process fragments. For our work we used this definition of [2] as a base, but as explained in section II-A we extended it according to our needs.

Process model similarities is a topic frequently discussed in the literature. According to [7] process similarities can be researched from three different perspectives:

- Text similarity: where comparisons are applied on the labels that appear in the elements of process models (e.g. task labels, event labels etc.).
- Structural similarity: a comparison of the topologies of the process models. Process models are seen here as graphs, and text similarities possibly also play a role in this case.
- Behavioral similarity: focuses on the execution semantics of the process models.

We could not find any approach that focuses on the detection of reoccurring structural parts in BPMN 2.0 models with a sole focus on their structural topology. In [7], [8] a set of algorithms for process model structural matching is presented. These algorithms do not focus on a specific workflow language. The presented methods rely on text information of process models, to detect similar mappings. It is also argued that these approaches will under perform or crash for process models that miss text information, or for models with a size bigger than 20 nodes.

A prototype for the comparison of BPMN 2.0 process models is introduced by [9]. The prototype is very promising, however it is argued that it might not be applicable to large, complex, real world models in its current state. A method to detect similar parts of process models is also proposed in [10]. With respect to our work, both of these approaches have a strong focus on text semantics, and it was therefore not possible to be further used. Our work focuses solely on the topologies of the process models, and is independent of text or behavioral information that is occasionally missing from process model collections (e.g. obfuscated models, mock-up models etc.). An approach with a goal similar to ours is found in [23]. This work focuses on BPEL processes, which are transformed to a BPEL process tree, in order to apply tree mining algorithms to discover reoccurring structures in a process. Although the further goal of this work is very similar to our goal, the different nature of BPMN 2.0 language does not allow to apply the same tree mining techniques for similar structures detection.

In order to achieve our goal we also needed to study current approaches in *graph matching*. To the extend of our research we were not able to find algorithms which take two graphs as an input, and return the set of matching structures as an output. In most of the cases, the input is a given sub-graph and the output is the mappings of this sub-graph in bigger graphs. The most relevant approach to our work, is considered the work of [24]. This approach (VF-2 algorithm) makes a propagating traversal in the graphs, and detects pairs that can be mapped to each other according to feasibility rules. These pairs are then returned as an output. In a broader sense our approach can be considered as a customization of VF-2 algorithm. As seen in section III we also apply a propagating traversal, and rules to check if two elements match with each other.

C. Basic Concepts

As already discussed, the first step towards the definition of our algorithm is to define the methodology of fragmentation that we are going to follow. In this context we extend the definition of "Process Fragment", initially given by [2], to match our needs. According to [2] a "Process Fragment" is a piece of process model with loose completeness and consistency. The existence of process graph elements (start, end, activities, context etc.) is possible but not imperative in a process fragment. However, a process fragment must have at least one activity and there must be a way to convert it to an executable process graph.

With respect to our needs this definition needs refinement in the following points:

1) the process fragment is not necessarily related to a complete process models
2) the starting point of a process fragment is not defined
3) the existence of a split or merge node is optional

Before we proceed to the extension of the 'process fragment" definition, we need to define the concept of a "Checkpoint". As *Checkpoint* we define any type of node that can be used as a starting point of our extended process fragments. The types of checkpoints are configurable from the user, and vary with respect to the process language that describes the models.

The extended definition of "Process Fragment" is called a "Relevant Process Fragment" because its detection is dependent to its existence at least K process models. We define a "Relevant Process Fragment" (RPF) as a "Process Fragment" that *exists in at least K process models* and satisfies the following structural requirements:

- starts with a checkpoint
- has at least N nodes included the checkpoint. Where N is a natural number and pre-configured from the user. We use $N \geq 3$.
- contains at least one activity

For this paper we focus on the detection of RPFs when $K = 2$, while in future work the threshold K may vary.

III. Implementation

This section describes the methodology we developed for the discovery of the RPFs. For a better comprehension the algorithm is separated in two main parts "Get Matching Path" and "Comparison". Algorithm 1 shows the main procedure of our algorithm. Algorithm 2 shows a complementary recursive procedure. Algorithm 1 starts the traversal of the model, calls Algorithm 2, and returns the discovered RPF. Algorithm 2 compares two models with each other and extracts the discovered similar structures.

As explained in Section II-A, it is required to traverse all possible sets of paths between the models. In order to address this requirement algorithm 1 takes as input one set for each model, that contains one entry for every outgoing sequence flows of each checkpoint. For example, see checkpoint $\mathcal{B}$ and checkpoint $\mathcal{E}'$ in figure 1. The corresponding sets given as input to the algorithm will be $S1 = \{\{exclusive\,gateway \rightarrow Task\,A\ \}, \{exclusive\,gateway \rightarrow Task\,C\ \}\}$ for Model A and $S2$ = *{{parallel gateway* $\rightarrow$ *Task E }, {parallel gateway* $\rightarrow$ *Task F }}* for Model B. These sets of checkpoint sequence flows are called checkpointFlows in lines 3-4 of algorithm 2.

For all possible pairs of checkpointFlows in the two checkpoint sets, namely for their Cartesian Product, we call algorithm 2 (line 6 of algorithm 1). It takes as parameters the current instances of checkpointFlows, and an empty fragment. By iteratively calling algorithm 2 for the different checkpointFlows we manage to check all possible checkpointFlows combinations as a possible start point of an RPF. The comparison procedure returns the calculated RPF. At this point we need to validate if the returned structure comprises a fragment or not. The validation rules applied are the requirements described in Section II-C. If the validation is successful, then the fragment is saved in a temporary data structure that holds all the discovered fragments (cf. variable"relevant Process Fragments" in algorithm 1)).

The functionality of algorithm 2 is to traverse and compare the models. The first step in this procedure is to check if the sequence flows that are given as parameters have sources and targets that are of the same type (e.g. task, exclusive gateway, start event etc.). When this condition is satisfied we say that two sequence flows match. Since we are strictly focusing on the structural similarity of the models this decision can be taken only based on the type of the sequence flow's source and target node. However, the condition could be extended if we wanted to check the similarity regarding more parameters e.g. labels of the

Algorithm 1 Procedure that calculates the matching paths of two models

```
procedure getMatchingPath
    $i \leftarrow 0$
    for all $checkpointFlows\ chFlowA \in modelA$ do
        for all $checkpointsFlows\ chFlowB \in modelB$ do
            $tmpFragment \leftarrow \varnothing$
            $tmpFragment \leftarrow$ comparison(chFlowA.sequenceFlow,chFlowB.sequenceFlow,tmpFragment)
            if $tmpFragmentisValidFragment$ then
                $relevantProcessFragments.add$(tmpFragment)
            end if
        end for
    end for
end procedure
```

Algorithm 2 Procedure that compares the two models

```
procedure comparison($sequenceFlowModelA$ ,$sequenceFlowModelB$,$tmpFragment$)
    if ($SourceTypeOfSequenceFlowA$ equals $SourceTypeOfSequenceFlowB$) and
    ($TargetTypeOfSequenceFlowA$ equals $TargetTypeOfSequenceFlowB$) then
        $outgoingA \leftarrow getOutgoingA$
        $outgoingB \leftarrow getOutgoingB$
        $fragment.add$(sequenceFlowModelB)
        for all $outgoingA \in ModelA$ do
            for all $outgoingB \in ModelB$ do
                if $outgoingB \notin tmpFragment$ then
                    comparison(outgoingA,outgoingB,tmpFragment)
                end if
            end for
        end for
    else
        return fragment;
    end if
end procedure
```

nodes. When the sequence flows match we add the respective sequence flow to the a temporary fragment structure (cf. variable tmpFragment in Algorithm 2). Afterwards we get all the outgoing sequence flows of the previously considered target nodes, aka. make a step forward to the traversal, and call recursively the comparison algorithm for all pairs of outgoing sequence flows. The terminate condition of the recursion is two sequence flows that do not match. If this condition is satisfied the algorithm returns the set of matched sequence flows until this point of execution (cf. variable tmpFragment in algorithm 2).

An implementation of our algorithm is available online at: http://www.iaas.uni-stuttgart.de/forschung/projects/benchflow/tools/BPMNCompare/.

IV. Evaluation of the Approach

This section evaluates our approach and discusses the resulting data. We ran our experiments on a local machine equipped with Intel Core i7-3520M CPU and 16GB RAM. The machine is running Windows 7. We realized the algorithm in a Java environment and used the BPMN 2.0 Meta Model of the Eclipse Modeling Framework (EMF) [25], [26].

The process model collection that we used for the experiments consists of 46 BPMN 2.0 models that originate from the datasets[1] also used in [9], and the BPMN 2.0 standard examples[2] [11]. Each model is compared against the rest of the models in the collection. In total, 2070 comparisons were calculated.

For the better comprehension of our results we need to have an overview to the complexity of our models. For this purpose we use the metric of Control-Flow-Complexity (CFC) described in [27] to describe the complexity of process models and we define the metric of *"Comparison Complexity"* to describe the complexity of the comparisons.

[1]http://pi.informatik.uni-siegen.de/qudimo/bpmn/
[2]http://www.omg.org/spec/BPMN/20100602/

The CFC measures all possible control flow decisions in a process model. According to [28] CFC cannot be considered as a metric to fully describe the model's structural complexity. Nevertheless, it was considered enough for giving the reader an overview of our collection. It is calculated by:

$$CFC(P) = \sum_{i\in\{XOR-splitsofP\}} CFC_{XOR-Split}(i) + \sum_{j\in\{OR-splitsofP\}} CFC_{OR-Split}(j) + \sum_{k\in\{AND-splitsofP\}} CFC_{AND-Split}(k) \quad (1)$$

The CFC functions $CFC_{XOR-Split}$, $CFC_{OR-Split}$ and $CFC_{AND-Split}$ return the following results for the node types that are represented by the corresponding subscripts:

- XOR-split: will have n outgoing transitions. In this case the designer has to consider all the possible outgoing transitions even though only one will be followed at the execution time. For this reason, each XOR-split will add the number n to the overall CFC metric, where n is the total number of its outgoing control flows.
- OR-split: the OR-split will also have n outgoing transitions. The OR-split must lead to at least one transition. The choice of at least one and at most n outgoing transitions is expressed by the possibility of 2^n-1. In this case the OR-split adds n outgoing transitions to the overall CFC metric of the model.
- AND-split: the execution of an AND-split will always result in a decision since all outgoing flows are followed. Therefore, the designer will only consider one possible state that will finally result in the execution of the AND-split. Hence, each AND split will add 1 unit to the overall CFC metric.

In our case we calculated the CFC metric for the compared BPMN 2.0 models. So, we adjusted the above functions to the BPMN 2.0 elements. More particularly:

1) Exclusive Gateway: Splits are computed the same way as the XOR split as they essentially correspond to the same functionality. So, each Exclusive Gateway adds n to the overall CFC metric where n is the number of the gateway's outgoing flows.
2) Inclusive Gateway: Splits are computed the same way as OR-splits, as they essentially correspond to the same functionality. So, each Inclusive Gateway: adds 2^n-1 to the overall CFC metric where n is the number of the gateway's outgoing flows.
3) Parallel Gateway: Splits are computed the same as AND-splits. Even though the functionality is not exactly the same, the control flows are executed concurrently and end up to the same task. So only one decision has to be taken in the control-flow. For this reason each Parallel Gateway adds complexity 1 in the CFC metric.
4) Complex Gateway: Is also assigned the same complexity as the OR-split because at least one path will be followed, but the designer needs ot think of all 2^n-1 possible paths. So each Complex Gateway adds 2^n-1 to the overall CFC metric where n is the number of the gateway's outgoing flows.
5) Event Based Gateway: Can be of type exclusive or parallel gateway. The complexity assigned to it is the complexity of the gateway that corresponds to this subtype.

The computed CFC of our models is shown in figure 2. There are 14 models with complexity 0 which means that there is not any existent gateway in the models, i.e. these models are sequential. There is one model with 1 complexity which means it contains one of the Gateways matched with the AND-split complexity, so a Parallel Gateway or an Event-Based Parallel Gateway. The rest of the models lay between the ranges of 2-8 which means there is a combination of a single number of different types of gateways. Finally, we can see a very complex model with $CFC = 14$.

Considering the context of the models, there are some models that represent an updated version of a source model, that also exists in the collection, but there are also models that have irrelevant context with each other. Here, it is important to point out that according to the suggested practices of process model engineering [23] we expect that also the real-world process models with which we will have to deal with will be of similar complexities.

For our experiments we have designed two scenarios. In the first scenario the types of {events,gateways} are set as types of checkpoints. That means that the checkpoint sets for the models shown in figure 1 correspond to the set *checkpointsA* = $\{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{H}\}$ for Model A, and the set *checkpointsB* = $\{\mathcal{A}', \mathcal{B}', \mathcal{C}', \mathcal{D}', \mathcal{E}', \mathcal{F}', \mathcal{G}', \mathcal{H}'\}$ for Model B. In the second scenario all available element types defined in the BPMN 2.0 standard [11] are set as checkpoint types, ie. {events, gateways, tasks}. Consequently, in this case all the existing nodes of the diagram play the role of a checkpoint and each node of a process model will be combined with all the other nodes of the model to compare. For both scenarios we have configured the detection to find RPFs with 3 or more nodes. According to our definition (cf. section II-C) these is the smallest possible size an RPF might have.

In order to conceptualize the results better, we have assigned a value to the complexity of each comparison. As seen in the algorithms 1 and 2 the number of checkpoints and their outgoing flows play an important role to the complexity of the algorithm, because the "for" loops are dependent to these values. Also, as explained in section III, for the realization of the algorithm, each outgoing sequence flow of a checkpoint, is associated with an entry to the checkpoint sets that are handled by the algorithm. Let us call S the set of the

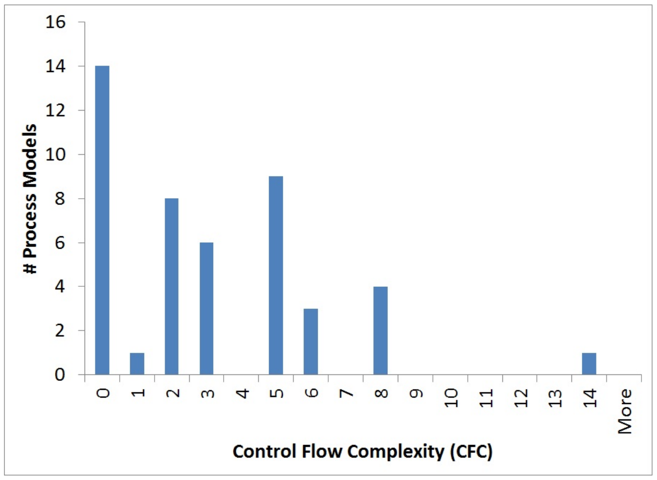

Figure 2: The control-flow-complexity (CFC) of the process models

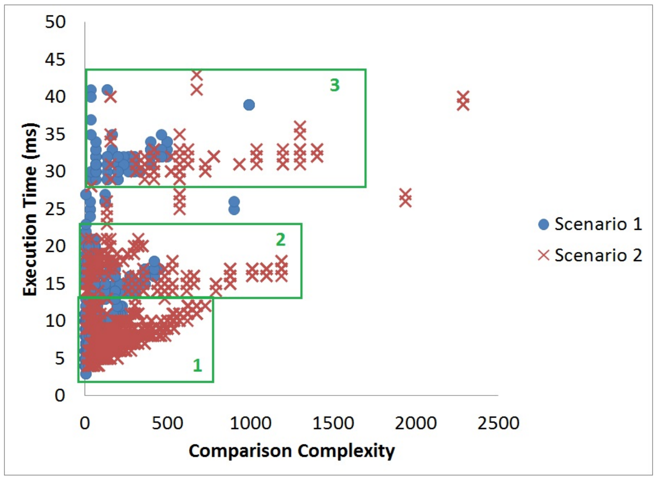

Figure 3: Execution times vs. comparison complexity for different comparison scenarios

outgoing sequence flows that correspond to checkpointFlows of model A and F the set of the checkpointFlows of model B. We calculate the complexity of the comparisons with the following equation

$$\text{Comparison Complexity} = \text{size(S)} * \text{size(F)} \tag{2}$$

By this way we manage to include the checkpoints and their corresponding outgoing sequence flows in the measurement of the comparison complexity.

Moving now to the execution of the experiments, we executed each scenario 20 times for all model combinations. The information about execution times, is collected with the use of JMeter [29].

The execution time needed for each comparison is depicted in figure 3. The vertical axis shows the values of execution times in milliseconds (ms), and the horizontal axis shows the comparison complexity of each comparison. The circles show the comparison complexities of the first scenario, while the X signs show the comparison complexities of the second scenario.

In a closer look, we can see that the results form three main clusters:

- Cluster 1: is formed with time ranges of [0-13] ms. In this cluster we can find comparison complexities up to 298 for the first scenario and 798 for the second scenario.
- Cluster 2: is formed with time ranges of [14-22] ms. In this cluster we can find comparison complexities up to 484 for the first scenario and 1188 for the second scenario.
- Cluster 3: is formed with time ranges of [29-41] ms. In this cluster we can find comparison complexities up to 990 for the first scenario and 1404 for the second scenario.

From the resulting clusters we conclude that the execution times are depending upon the comparison complexities. Nevertheless, there seem to be more reasons that affect performance. For example, there are bigger time values for 0 comparison complexity, or smaller time values for big comparison complexities (e.g. execution time 16 for comparison complexity 1188 in the second scenario). By a deeper analysis of the models we have observed that the execution times are also affected by the level of nesting and the total number of nodes participating in the models.

The number of RPFs detected in the process models does not seem to affect the execution times, but the similarity of the compared process models seemed to have a minor impact in the performance. We suspect, that similar models will need more time to compare, as the algorithm does more traversals. However, this hypothesis needs more data for investigation, and was left for future work.

Figure 5 shows the change of average time with respect to the average comparison complexities for both scenarios. It shows that the algorithm has complexity $O(n \log n)$ where n is the value of comparison complexity. The curve that the diagram shows between the times [10-15] for the first scenario, and [20-25] for the second scenario result from the small amount of data that we had in these times. This lack of data is also shown in figure 3. From figure 3 and figure 5 we conclude that despite the increase in the comparison complexity, there is minor difference in the execution times. We explain this stability of time by the fact that the process models are quite simple graphs [23].

As a conclusion to this discussion, in figure 4 we present the RPFs that were detected in the comparison of the process models of figure 1. As expected, we can see the RPFs that were manually detected in figure 4. For the first scenario the

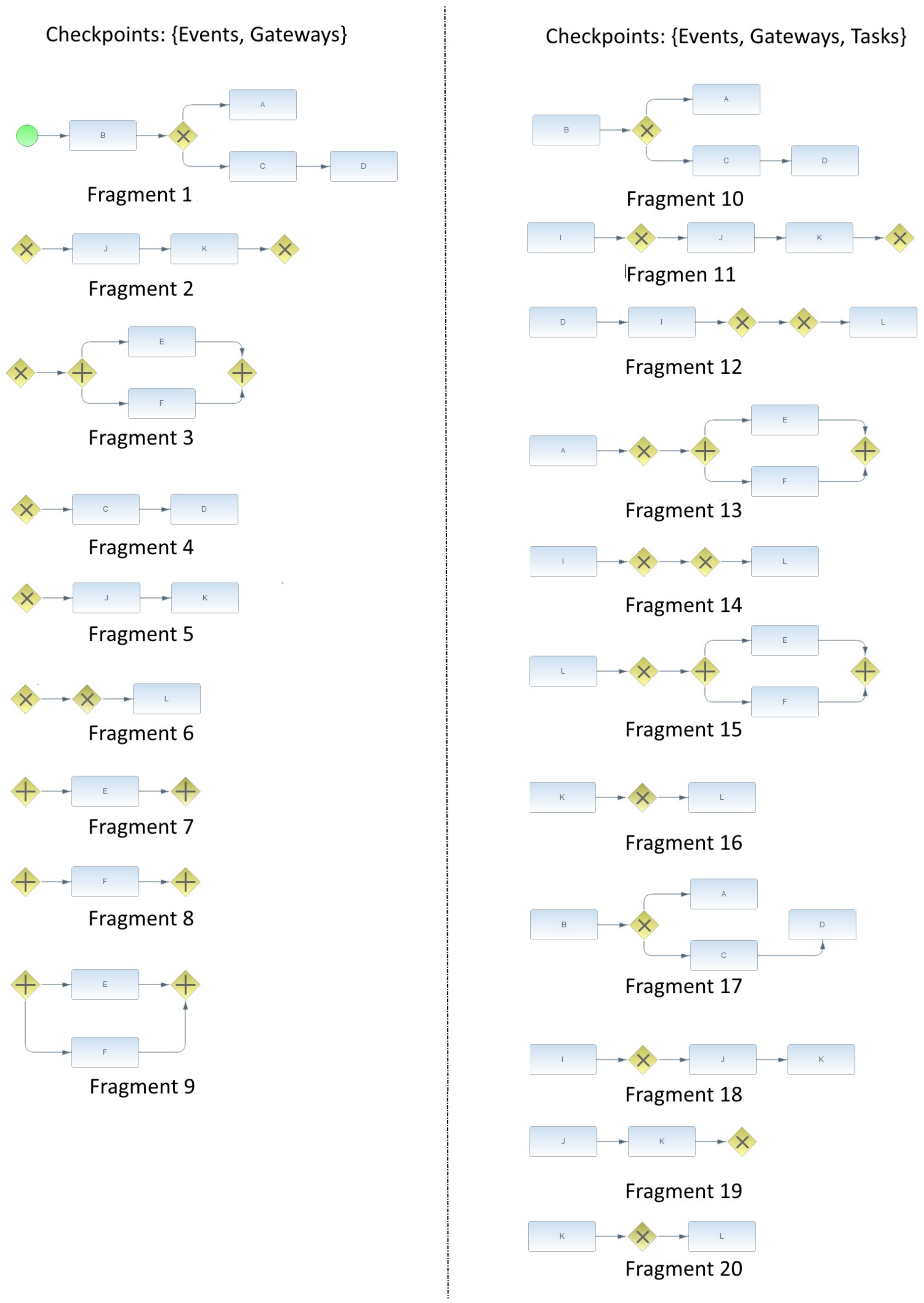

Figure 4: The detected process fragments for different comparison scenarios

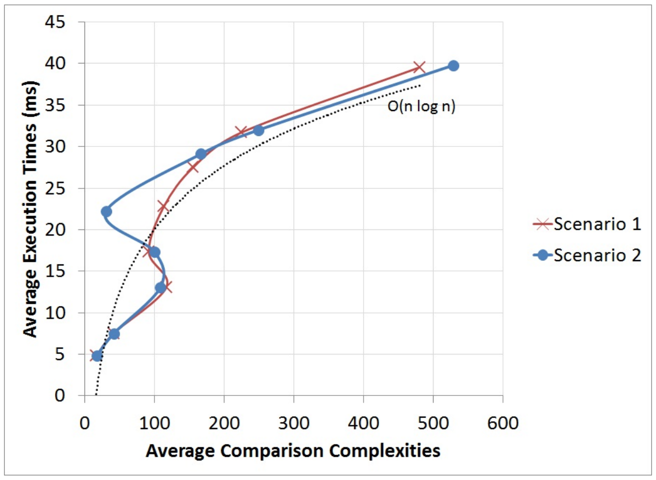

Figure 5: The average execution time vs. the average comparison complexity

algorithm detected five additional fragments that are shown in figure 4. As seen, there are also some duplicate entries returned (e.g. Fragment 4 and Fragment 5), which indicates that the development of results filtering is necessary. Moving to the second scenario where all nodes are calculated as RPFs, we can see 11 more RPFs detected. That means 122.23% increase of the results, and thus duplicate information as all of the RPFs presented in the second scenario are already included in the results of the first scenario, or can be easily reproduced by the addition of task sequences at the beginning or the end of the RPFs. For this reason we conclude, that even though the usage of checkpoints does not improve the performance in execution times, it should be a preferred option, as it reduces duplicate information in the exported RPFs and thus, reduces the efforts in filtering and analyzing the results.

V. Conclusion and Future Work

The work described in this paper is a first attempt to apply sub-graph isomorphism techniques on process models, for the automated discovery of reoccurring structures in a collection of process models. To the best of our knowledge, this work is the first attempt to apply graph isomorphism techniques to BPMN 2.0 process models. The approach presented at this work contributed (1) a theoretical basis of our approach, with the introduction of concepts such as "checkpoints" and "Relevant Process Fragments" and (2) an approach that automatically detects and exports the re-occurring fragments in a collection of process models.

We have evaluated our approach in a collection of 46 BPMN 2.0 process models, with respect to two execution scenarios that differ in the number of checkpoints to be compared. The resulting execution times were analyzed with respect to the "comparison complexities" a metric that we defined to describe how complex a comparison of two process models is. As seen from our results, the execution times do not differ much between the two executed scenarios, as process models are basically graphs of small or average complexity. We have also analyzed the RPFs that were detected by the application of our approach on two example process models. In this case the configuration of checkpoints types seems to play an imperative role, as the second scenario (larger checkpoint set) exported a lot of duplicate information. It is therefore suggested that the types of checkpoints are carefully configured in order to eliminate the unnecessary information, but also gain the distinct RPFs of interest.

The future work foresees to extend the algorithm to work for the complete set of BPMN 2.0, choreographies and process models with cycles. We also need to apply filtering methods on the results in order to eliminate the duplicate information, cluster the results, assign values to frequency of appearance on the detected RPFs, and detect RPFs for different threshold values (i.e. $K > 2$). Indexing techniques can also be applied when possible, in order to avoid comparing every model with all the other models of the collection, but efficiently detect models that are more possible to be similar to each other. When these optimizations are ready, we plan to run the approach on the complete collection of real-world models, and run a thorough analysis on the results.

Acknowledgment

The research leading to these results has received funding from Swiss National Science Foundation and the German Research Foundation with the BenchFlow - A Benchmark for Workflow Management Systems (DACH Grant Nr. 200021E-145062/1); and the German Research Foundation (DFG) within the Cluster of Excellence in Simulation Technology (EXC310) project.

References

[1] Z. Ma, "Process fragments: enhancing reuse of process logic in bpel process models," Ph.D. dissertation, Universität Stuttgart, 2012.

[2] D. Schumm, F. Leymann, Z. Ma, T. Scheibler, and S. Strauch, "Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls," in *MKWI'10, Göttingen, Germany, February 23-25, 2010*, Schumann/Kolbe/Breitner/Frerichs, Ed., Conference Paper, pp. 2125–2137.

[3] D. Schumm *et al.*, "Process Fragment Libraries for Easier and Faster Development of Process-based Applications," *JSI*, vol. 2, no. 1, pp. 39–55, January 2011.

[4] H. Eberle, T. Unger, and F. Leymann, "Process fragments," in *On the Move to Meaningful Internet Systems: OTM 2009*, ser. Lecture Notes in Computer Science, R. Meersman, T. Dillon, and P. Herrero, Eds. Springer Berlin Heidelberg, 2009, vol. 5870, pp. 398–405.

[5] H. Eberle, F. Leymann, D. Schleicher, D. Schumm, and T. Unger, "Process Fragment Composition Operations," in *Proceedings of APSCC 2010*. IEEE Xplore, December 2010, Conference Paper, pp. 1–7.

[6] P. Wohed, M. Dumas, A. H. M. T. Hofstede, and N. Russell, "Pattern-based analysis of bpmn - an extensive evaluation of the control-flow, the data and the resource perspectives," Tech. Rep.

[7] R. Dijkman, M. Dumas, and L. García-Bañuelos, "Graph matching algorithms for business process model similarity search," in *Proceedings of the 7th International Conference on Business Process Management*, ser. BPM '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 48–63.

[8] R. Dijkman, M. Dumas, B. van Dongen, R. Käärik, and J. Mendling, "Similarity of business process models: Metrics and evaluation," *Inf. Syst.*, vol. 36, no. 2, pp. 498–516, Apr. 2011.

[9] P. Pietsch and S. Wenzel, "Comparison of bpmn2 diagrams," in *Business Process Model and Notation*, ser. Lecture Notes in Business Information Processing, J. Mendling and M. Weidlich, Eds. Springer Berlin Heidelberg, 2012, vol. 125, pp. 83–97.

[10] F. Pittke, H. Leopold, J. Mendling, and G. Tamm, "Enabling reuse of process models through the detection of similar process parts," in *Business Process Management Workshops*, ser. Lecture Notes in Business Information Processing, M. La Rosa and P. Soffer, Eds. Springer Berlin Heidelberg, 2013, vol. 132, pp. 586–597.

[11] D. Jordan and J. Evdemon, "Business process model and notation (BPMN) version 2.0," Object Management Group, Inc, January 2011.

[12] M. Kunze, P. Berger, and M. Weske, "Bpm academic initiative - fostering empirical research." in *BPM (Demos)*, ser. CEUR Workshop Proceedings, N. Lohmann and S. Moser, Eds., vol. 940. CEUR-WS.org, 2012, pp. 1–5.

[13] E. Bengoetxea, "Inexact graph matching using estimation of distribution algorithms," Ph.D. dissertation, Ecole Nationale Supérieure des Télécommunications, Paris, France, Dec 2002.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[15] D. A. Basin, "A term equality problem equivalent to graph isomorphism," *Information Processing Letters*, vol. 51, 1994.

[16] R. M. Verma and S. W. Reyner, "An analysis of a good algorithm for the subtree problem, correlated," *SIAM J. Comput.*, vol. 18, no. 5, pp. 906–908, Oct. 1989.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA, USA: MIT Press, 2001.

[18] M. Mancioppi, O. Danylevych, D. Karastoyanova, and F. Leymann, "Towards classification criteria for process fragmentation techniques," in *Business Process Management Workshops*, ser. Lecture Notes in Business Information Processing, F. Daniel, K. Barkaoui, and S. Dustdar, Eds. Springer Berlin Heidelberg, 2012, vol. 99, pp. 1–12.

[19] J. Vanhatalo, H. Völzer, and F. Leymann, "Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition," in *Proceedings of Service-Oriented Computing (ICSOC 2007)*, ser. Lecture Notes in Computer Science, B. Krämer, K. Lin, and P. Narasimhan, Eds., vol. 4749. Springer-Verlag, Berlin, 2007, pp. 43–55.

[20] J. Vanhatalo, H. Völzer, and J. Koehler, "The refined process structure tree," in *Business Process Management*, ser. Lecture Notes in Computer Science, M. Dumas, M. Reichert, and M.-C. Shan, Eds. Springer Berlin Heidelberg, 2008, vol. 5240, pp. 100–115.

[21] A. Polyvyanyy, J. Vanhatalo, and H. Völzer, "Simplified computation and generalization of the refined process structure tree," in *Web Services and Formal Methods*, ser. Lecture Notes in Computer Science, M. Bravetti and T. Bultan, Eds. Springer Berlin Heidelberg, 2011, vol. 6551, pp. 25–41.

[22] L. García-Bañuelos, "Pattern identification and classification in the translation from bpmn to bpel." in *OTM Conferences (1)*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds., vol. 5331. Springer, 2008, pp. 436–444.

[23] M. Hertis and M. B. Juric, "An empirical analysis of business process execution language usage," *IEEE Transactions on Software Engineering*, vol. 40, no. 8, pp. 1–1, 2014.

[24] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1367–1372, Oct. 2004.

[25] R. Hille-Doering, "Making of the bpmn 2.0 meta model for eclipse: Merge and conquer," SAP, October 2010.

[26] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.

[27] J. Cardoso, "How to measure the control-flow complexity of web processes and workflows," in *Workflow Handbook 2005*, L. Fischer, Ed. Lighthouse Point, FL, USA: Future Strategies Inc., 2005, pp. 199–212.

[28] V. Gruhn and R. Laue, "Complexity metrics for business process models," in *in: W. Abramowicz, H.C. Mayr (Eds.), 9th International Conference on Business Information Systems (BIS 2006), Lecture Notes in Informatics*, pp. 1–12.

[29] E. Halili, *Apache JMeter*. Packt Publishing, 2008.