Detecting Approximate Clones in Business Process Model Repositories

Marcello La Rosa\textsuperscript{a,\textit{b}}, Marlon Dumas\textsuperscript{c}, Chathura C. Ekanayake\textsuperscript{a}, Luciano García-Bañuelos\textsuperscript{c}, Jan Recker\textsuperscript{a}, Arthur H.M. ter Hofstede\textsuperscript{a}

\textit{a} Queensland University of Technology, Australia
\textit{b} NICTA Queensland Lab, Australia
\textit{c} University of Tartu, Estonia

Abstract

Empirical evidence shows that repositories of business process models used in industrial practice contain significant amounts of duplication. This duplication arises for example when the repository covers multiple variants of the same processes or due to copy-pasting. Previous work has addressed the problem of efficiently retrieving exact clones that can be refactored into shared subprocess models. This article studies the broader problem of approximate clone detection in process models. The article proposes techniques for detecting clusters of approximate clones based on two well-known clustering algorithms: DBSCAN and Hierarchical Agglomerative Clustering (HAC). The article also defines a measure of standardizability of an approximate clone cluster, meaning the potential benefit of replacing the approximate clones with a single standardized subprocess. Experiments show that both techniques, in conjunction with the proposed standardizability measure, accurately retrieve clusters of approximate clones that originate from copy-pasting followed by independent modifications to the copied fragments. Additional experiments show that both techniques produce clusters that match those produced by human subjects and that are perceived to be standardizable.

Key words: Business process model, clone detection, model collection,
1. Introduction

Ample evidence suggests that duplication is a widespread phenomenon in software and model repositories [1, 2]. Not surprisingly, duplication is also found in repositories of business process models used in industrial practice [3]. Clones in process model repositories emerge for example as a result of copy-pasting, but also when multiple variants of a process co-exist and are described as separate models. For example, an insurance company typically runs multiple claims handling processes for different types of claims. Naturally, these process variants share commonalities, which manifest themselves in the form of clones.

Detecting clones in process models allows modelers to identify opportunities for standardization and refactoring. For example, consider the case of multiple variants of an insurance claims handling process, where each variant is captured as a separate process model. Given that disbursement of the insurance payout occurs in every variant (albeit differently depending on the type of claim), it is likely that these separate models will contain clones corresponding to disbursement activities. These clones can potentially be standardized\(^1\) and refactored as a shared subprocess. In this way, duplication is reduced and uniformity across process models is increased, to the benefit of model maintainability.

Standardization of clones however is only possible if the clones to be standardized are either exact clones or they are sufficiently similar that they can be replaced by a standardized fragment with minor changes to each original clone. Indeed, while some changes to a clone may be lexical (e.g. uniformizing the nomenclature of tasks), other changes may entail alterations to the underlying process, such as adding or skipping a task, leading to similar fragments that may or may not be standardizable depending on the business implications of the change.

The problem of clone detection has been widely studied in the field of software engineering, primarily in the context of source code clone detection, but also in the context of model clone detection (e.g. clones in Simulink models) [2, 5]. In this context, a distinction is made between four types of clones [2], which can be defined in the context of process models as follows:

\(^1\)We use the term *standardization* to refer to the act of replacing discrepant but similar process fragments with a single unified fragment. Other authors use the term *harmonization* [4] instead to emphasize that the unified fragment is not necessarily a “standard”.

• Type-1 (also called *exact clones*): Identical fragments except for layout variations and comments.

• Type-2: Syntactically identical fragments except for possible layout variations, comments and labeling variations (e.g. different task, event or data object labels with the same semantics).

• Type-3 (also called *approximate clones* [6] or *near-miss clones*): Copied fragments with further modifications such as changed, added or removed model elements in addition to variations allowed in Type-2 clones. Note that two Type-3 clones are not necessarily behaviourally equivalent.

• Type-4: Behaviorally equivalent fragments with syntactic differences (e.g. fragments with different combinations of gateways but same set of traces). Note that Type-4 clones are a superset of Type-2. While Type-2 clones only allow for one-on-one substitutions, Type-4 allow for any variation so long as behavior is preserved. On the other hand, Type-4 clones are not a superset of Type-3 clones nor vice-versa. Rather, Type-3 and Type-4 clones are alternative ways of extending the notion of Type-2 clones.

In previous work, we proposed a technique for identifying Type-1 (exact) clones in process models [7]. This technique can also be adapted to detect Type-2 clones by pre-processing the labels of model elements and replacing semantically equivalent labels with a standard label. However, this technique cannot detect Type-3 (approximate) clones, which are arguably likely to emerge in process model repositories when modelers copy-paste fragments across models – thus creating exact clones – and later on these exact clones evolve separately.

To address this gap, this article presents and compares two techniques for identifying Type-3 (approximate) clones in repositories of process models for the purpose of standardizing and refactoring them as shared subprocesses. The article also proposes and validates a measure of standardizability of a set of approximate clones, meaning a measure of the feasibility of replacing the clones with a single shared subprocess. This measure captures the tradeoff between the magnitude of changes required to achieve standardization and the simplification benefits that standardization yields.

---

2Type-4 clone detection in process models, while potentially relevant, deserves a separate treatment as it involves a very different set of techniques (behavioral equivalence checking).
The proposed techniques and standardizability measure are evaluated in a two-pronged manner. First, we evaluate the runtime performance and accuracy of the two techniques using a combination of real-life and synthetic datasets. Second, we report two experiments with human subjects in which we compare the proposed techniques in terms of: (i) their ability to retrieve groups of clones that human subjects perceive to be standardizable (that is, replaceable and refactored as a single shared subprocess); and (ii) their ability to replicate clusters produced by human subjects.

This article is an extended version of a previous conference paper on the subject [8]. The main extensions are the two empirical evaluations with human subjects (Section 6), as well as a more comprehensive discussion of related work, differences between the two techniques, limitations of the approach and threats to validity.

The article is organized as follows. Section 2 defines and justifies the notion of approximate clone adopted in this article and the proposed measure of standardizability. Next, Section 3 introduces techniques for process model parsing and exact clone detection, which are used as the basis for the proposed techniques. Section 4 presents the techniques. Next Sections 5 and 6 present the results of the evaluation while Section 7 discusses threats to the validity of the evaluation and limitations. Finally, Section 8 frames the contributions in relation to the literature while Section 9 concludes and discusses possible extensions of the research. The instruments used for the evaluation with human subjects are available as supplementary material attached to this paper.

2. Approximate Clones and Standardizability

This section defines the notion of similarity adopted in this paper and, on this basis, it defines a notion of approximate clone cluster and a measure of standardizability for approximate clone clusters.

2.1. Process Model Similarity

When designing an approximate clone detection method, a first step is to define what an approximate clone is. Generally, such a definition relies on a similarity or (equivalently) a distance metric. The similarity of process models specified in a graph-based notation can be measured on the basis of three complementary aspects: (i) the labels attached to tasks, events and other model elements; (ii) their graph structure; and (iii) their execution semantics. In this paper, we adopt a measure that combines structural and label similarity (distance) and that has been
shown to be correlated with perceived similarity [9]. We define this measure over an abstract representation of process models based on labelled graphs, as follows.

**Definition 1 (Simple process graph)** Let $\mathcal{L}$ be a set of labels. A process graph $H$ is a (weakly) connected labelled graph $(V, E, \lambda)$ where $V$ is the set of nodes, $E \subseteq V \times V$ is the set of edges, and $\lambda : V \to \mathcal{L}$ is a function from nodes to labels.

This definition is intended to capture only control-flow elements of process models (tasks, events and gateways). We discuss later how to extend this notion to non-control-flow elements of process models such as objects and roles.

In order to compare pairs of process graphs, we need to first be able to compare pairs of nodes belonging to these graphs. Accordingly, we assume given a node distance measure, or equivalently a similarity measure, as defined below.

**Definition 2 (Node distance, node similarity measure)** Let $V$ be a set of nodes. A node distance measure over $V$ is a function $\text{Dist}_V : V \times V \to [0..1]$ that satisfies the properties of identity, symmetry and triangle inequality, i.e. $\text{Dist}_V(x, x) = 0$, $\text{Dist}_V(x, y) = \text{Dist}_V(y, x)$ and $\text{Dist}_V(x, z) \leq \text{Dist}_V(x, y) + \text{Dist}_V(y, z)$. The dual of a distance measure $(1 - \text{Dist}_V(x))$ is a similarity measure.

When defined over simple process graphs, where each node is associated with a single label, we identify two complementary approaches to define a node distance measure, namely syntactic distance (similarity) and semantic distance (similarity) [9]. Syntactic distance (similarity) treats labels as strings. Here, we adopt a normalized version of the well-known string-edit distance [10].

**Definition 3 (Syntactic label distance)** Let $v_1, v_2$ be two nodes and $l_1 = \lambda(v_1), l_2 = \lambda(v_2)$ their corresponding labels. Furthermore, let $|l|$ be the length of a label $l$ and $\text{ed}(l_1, l_2)$ be the string-edit distance of labels $l_1$ and $l_2$. The syntactic distance of labels $v_1$ and $v_2$ is $\text{Dist}_{\text{ted}} = \frac{\text{ed}(l_1, l_2)}{\max(|l_1|, |l_2|)}$.

Meanwhile, a semantic node distance is one that that takes into account semantic relations, such as synonymity. To define such a measure, we use a function that calculates the semantic relatedness of two words. We compare two labels by identifying, for each word in one label, the most related word in the other label. A word in label $l_1$ that also appears in $l_2$ contributes a score of one to the semantic similarity. A word that appears in label $l_1$ but not in $l_2$ contributes a score equal to the semantic relatedness of the most related word in $l_2$, and vice-versa. The semantic distance between two nodes $\text{Dist}_{\text{sem}}(v_1, v_2)$ is one minus the normalized sum of the relatedness scores.
Definition 4 (Semantic label distance) Let \( v_1, v_2 \) be two nodes and \( l_1, l_2 \) their corresponding labels. Let \( \Omega \) be the set of all node labels and \( W \) be the set of all words. Let \( w : \Omega \rightarrow \mathcal{P}(W) \) be a function that separates a label into a set of words and \( \text{rel}(w, w') \) be a function that returns the semantic relatedness of two words. Furthermore, let \( lw_1 = w(l_1) \) and \( lw_2 = w(l_2) \). We define the semantic label distance of nodes \( v_1 \) and \( v_2 \), denoted \( \text{Dist}_{lsem}(v_1, v_2) \), as follows:

\[
1 - \frac{2 \cdot wi \cdot |lw_1 \cap lw_2| + \sum_{w \in lw_1 \setminus lw_2} \max_{w' \in lw_2} \text{rel}(w, w') + \sum_{w \in lw_2 \setminus lw_1} \max_{w' \in lw_1} \text{rel}(w, w')}{|lw_1| + |lw_2|}
\]

We use the Wordnet::Similarity package [11] to calculate the semantic relatedness between words. Before computing semantic distance between labels, we remove stop-words and apply stemming to the remaining words.

Following previous empirical results [9, 12], we adopt a node distance measure that combines syntactic and semantic distances. Specifically, we define the distance between two nodes as being the maximum of the semantic and syntactic distances i.e. \( \text{Dist}_l(x) = \max(\text{Dist}_{l_{\text{led}}}(x), \text{Dist}_{l_{\text{sem}}}(x)) \). Thus two nodes are similar if they are either syntactically or semantically similar (i.e. their syntactical or semantical distance is close to zero).

The above definitions of process graph and of node distance/similarity can be extended to capture non-control-flow elements such as objects and roles. One way of achieving this is by extending the definition of process graph with functions that map each (control-flow) node to a role and to a set of input and output objects. Node distance can then be defined as a weighted average of node label distance (e.g. as defined above), role distance and input/output objects’ distance, where distance between roles and objects is calculated using the same principles as label distance (combination of syntactic and semantic distance). Extended notions of node distance to cover non-control-flow elements are defined in [9]. In the rest of this article, we use the node label distance (i.e. \( \text{Dist}_l \)), since the models used in the evaluation did not contain sufficient information regarding objects and roles to take these into account in the computation of distance.

Given the distance measure over nodes \( \text{Dist}_l \), we define a distance measure over process graphs based on a well-known notion of graph-edit distance [13]. The graph-edit distance of two graphs is the minimal set of edit operations required to transform one graph into the other. There are three edit operations: node substitution, node insertion/deletion and edge insertion/deletion. A node substitution refers to the fact that a node \( v_1 \) in one of the graphs can be matched to a node \( v_2 \) in the other graph. A node substitution is only allowed if their label
distance $\text{Dist}_l(v_1, v_2)$ is below a user-defined threshold. We call such a node a “substituted node”. Whenever a node in a graph is not matched to any node in the other graph, it is considered as either inserted in one graph or deleted in the other one. We call this node a “skipped node”. Similarly, an edge insertion (or deletion) operation is applied to each edge that cannot be mapped to an edge in the other graph. Similarly, we call this edge a “skipped edge”. These notions are formalized below.

**Definition 5 (Normalized process graph edit distance [9])** Let $H_1 = (V_1, E_1, \lambda_1)$ and $H_2 = (V_2, E_2, \lambda_2)$ be two process graphs. Let $M : V_1 \rightarrow V_2$ be a partial injective mapping that maps nodes of $H_1$ to nodes of $H_2$. Moreover, let $\text{subv}$ be the set of substituted nodes, i.e., $\forall v \in \text{subv} : v \in \text{dom}(M) \cup \text{cod}(M)$, $\text{skipv}$ the set of skipped nodes, i.e., $\forall v \in \text{skipv} : v \notin \text{dom}(M) \cup \text{cod}(M)$, and $\text{skipe}$ the set of skipped edges, i.e., $\forall (v, w) \in \text{skipe} : v \notin \text{dom}(M) \cup \text{cod}(M) \lor w \notin \text{dom}(M) \cup \text{cod}(M)$. The normalized graph edit distance induced by mapping $M$ is:

$$\text{Dist}_{GED}^M(H_1, H_2) = \text{average}(f_{\text{skipv}}, f_{\text{skipe}}, f_{\text{subv}})$$

where $f_{\text{skipv}}$ is the fraction of skipped nodes, $f_{\text{skipe}}$ the fraction of skipped edges, and $f_{\text{subv}}$ the average distance between substituted nodes, i.e. $f_{\text{skipv}} = \frac{|\text{skipv}|}{|V_1| + |V_2|}$, $f_{\text{skipe}} = \frac{|\text{skipe}|}{|E_1| + |E_2|} m$ and $f_{\text{subv}} = \frac{2 \cdot \sum_{(v, w) \in M} \text{Dist}_l(\lambda_1(v), \lambda_2(w))}{|V_1| + |V_2|}$. 

Finally, the normalized graph-edit distance between $H_1$ and $H_2$, written $\text{Dist}_{GED}(H_1, H_2)$, is the smallest $\text{Dist}_{GED}^M(H_1, H_2)$ across all mappings $M$.

A $\text{Dist}_{GED}$ of 0 means that the process graphs are identical, while a $\text{Dist}_{GED}$ of 1 implies that the process graphs are completely dissimilar. Consider for example the two process fragments shown in Figure 1. In this case, one node appears in one fragment and not in the other (i.e. the node is counted as skipped), therefore $\text{skipv} = 1$. Two edges have been added (i.e. the two edges are counted as skipped), thus $\text{skipe} = 2$. There are two node replacements but their corresponding labels are identical, thus $\text{subv} = 2$ and $f_{\text{subv}} = 0$. Hence, $\text{Dist}_{GED} = \text{avg}(1/9, 2/6, 0) \sim 0.15$.

The problem of computing the graph-edit distance is NP-Complete [13]. In this paper, we adopt a greedy heuristic described in [14], since its computational

---

3The rationale for the factor of 2 in the definition of $f_{\text{subv}}$ is so that replacement of two nodes with completely different labels (i.e. $\text{Dist}_l(\lambda_1(v), \lambda_2(w)) = 1$) is equivalent to deletion of $v$ and insertion of $w$. 

7
complexity is much lower than that of an exhaustive algorithm, while having a high precision when emulating human judgement [15]. Still, despite the fact that we use a greedy heuristic, the computation of the \( Dist_{GED} \) is expensive – \( O(n^3) \) where \( n \) is the number of nodes of the largest graph.

The above definitions of process graph and process graph-edit distance assume that nodes are labelled but edges are not. Edge-labeled process models are not uncommon. For example in the standard Business Process Model and Notation (BPMN), edge labels are typically attached to arcs stemming from decision gateways. The above definitions can be extended to deal with edge labels as shown in [9, 12]. In a nutshell, this is achieved by introducing a term \( f_{sube} \) in the graph-edit distance definition in order to capture the fraction of “substituted edges”, meaning edges that are shared by the two models but with different labels. A pair of edges is said to be “substituted” if the distance between their labels is below a certain threshold. The weight assigned to the substituted edges is then equal to their label’s distance, in the same way as for substituted nodes.

2.2. Notion of Approximate Clone

Given the measure of distance defined above, we could simply postulate that two process model fragments are approximate clones if their graph-edit distance is below a given user-defined threshold. However, three additional issues ought to be considered when defining a notion of approximate clone. Firstly, any fragment \( g_1 \) is similar to any fragment \( g_2 \) such that \( g_2 \) contains \( g_1 \) or \( g_1 \) contains \( g_2 \), provided that the difference between \( g_1 \) and \( g_2 \) falls below the threshold. A definition that would consider two fragments as approximate clones merely because one contains the other would lead to many false positives (e.g. in the SAP reference model there are 8,876 fragments with 13,131 containment relations); this is an issue that has been widely discussed in the field of code and model clone detection [6]. Secondly, given the goal to identify approximate clones for the sake of refactoring them into subprocesses and given that subprocesses are invoked according to a
call-and-return semantics, it is necessary that the approximate clones we retrieve are Single-Entry, Single-Exit (SESE) fragments. Thirdly, we are not interested in trivial clones consisting of a single activity, since they do not represent an opportunity for subprocess extraction. These considerations are captured in the following definitions.

**Definition 6 (SESE Process Fragment)** Given a process graph $H = (V, E, \lambda)$, a SESE process fragment $F = (V', E', \lambda')$ of $H$ is a connected subgraph of $H$ such that $N'$ has a single source node (i.e. a single node without an incoming edge in $E'$) and a single sink node (i.e. a single node without an outgoing edge in $E'$).

**Definition 7 (Approximate Clone Pair)** Given a distance measure $\text{Dist}$ and a distance threshold $\tau$, two non-trivial, SESE process fragments $g_1$ and $g_2$ are approximate clones – written $\text{Approx}(g_1, g_2)$ – iff $g_1 \not\subseteq g_2$, $g_2 \not\subseteq g_1$ and $\text{Dist}(g_1, g_2) \leq \tau$.

Note that given a pair of approximate clones $F_1$ and $F_2$, it is possible that there exist sub-fragments $F'_1$ of $F_1$ and $F'_2$ of $F_2$, such that $F'_1$ and $F'_2$ are also approximate clone pairs. In other words, approximate clone pairs can occur at different levels of granularity. The above definition does not impose that a pair of approximate clones are maximal in their corresponding model(s). This is because smaller but more similar approximate clones ($F'_1$ and $F'_2$ in this example) may turn out to be suitable for standardization than larger but less similar approximate clones (e.g. $F_1$ and $F_2$).

Armed with the above definition, one can retrieve large amounts of approximate clone pairs [3]. However, if the goal is to help modelers to identify opportunities for standardization, retrieving all such pairs is of limited use. Instead, given the goal at hand, analysts need to identify clusters of fragments that can be standardized towards a single fragment with a bounded amount of changes on each fragment. Otherwise, some fragments would need to undergo changes during the standardization that would convert them into arbitrarily different fragments.

Given the goal to retrieve clusters of similar fragments suitable for standardization towards a single fragment, the next question is which fragment in the cluster would serve as the reference fragment towards which other fragments will be standardized. In this respect, we envisage two alternative approaches to standardize a given set of fragments:
A1. A set of fragments can be standardized by taking the “medoid”\(^4\) fragment as a reference, and standardizing all fragments towards this medoid. If we wish to bound the number of changes that need to be made to each fragment, the distance between the medoid and every fragment should thus be below the chosen bound.

A2. A set of fragments can be standardized by selecting any fragment in the set as a reference and standardizing all other fragments towards this reference fragment. If we wish to bound the number of changes that need to be made to each fragment, the distance between every pair of fragments should be below the bound, so that indeed any fragment can be selected as the reference fragment.

These observations lead us to the following definition.

**Definition 8 (Approximate Clone Cluster)** A set of SESE process model fragments \(C\) is a cluster of approximate clones iff one of the following properties holds:

1. \(\exists g \in C \forall g' \in C : \text{Approx}(g, g')\). In this case, \(g\) is called the cluster medoid.

2. \(\forall g, g' \in C : \text{Approx}(g, g')\).

**2.3. Measure of Cluster Standardizability**

Standardizing a cluster of approximate clones has costs and benefits and these should be taken into account when deciding which clusters of approximate clones are more amenable to standardization. We define the benefit-to-cost ratio (BCR) of a cluster as a measure of the cluster quality for the purpose of standardization, and use this notion to rank the retrieved clusters when presenting the list of clusters (which can be many) to the users. To operationalize the BCR, we thus need to define a cost measure and a benefit measure.

The cost (i.e. effort) of standardizing the fragments of a cluster into a single fragment is determined by many factors, some of them exogenous to the process models themselves. However, we contend that this cost is proportional to the

---

\(^4\)In data clustering, a medoid is a representative object of a cluster, i.e. an object whose average dissimilarity to all other objects in the cluster is minimal.
amount of elementary changes that will be made to the fragments in order to standardize them to one common subprocess. Indeed, each elementary change will require a certain amount of effort to validate (with business analysts and stakeholders involved in the process) that this change to the fragment is indeed meaningful and beneficial, and to ensure that the execution of the process is adapted to this change if required. This observation is supported by empirical evidence that shows that standardization effort is positively correlated with the amount of variation in the process [16].

Accordingly, we use the absolute GED \( \text{Dist}_{AGED}(H_1, H_2) \) defined in the same way as \( \text{Dist}_{GED}(H_1, H_2) \) in Definition 5 but replacing \( f_{skipv} \) and \( f_{skipe} \) with \( |\text{skipv}| \), \( |\text{skipe}| \) respectively, and removing the denominator in the definition of \( f_{subv} \). In other words, we count the actual number of edit operations as opposed to the fraction of edit operations relative to the total size. We do not use the normalized GED in this context (\( \text{Dist}_{GED} \)), because this is not reflective of the number of operations required to standardize the fragments. Instead, \( \text{Dist}_{GED} \) is reflective of the percentage difference shared between two models.

In the case of standardization approach A1 (cf. Section 2.2), the medoid fragment serves as reference. Thus, the cost of standardizing the cluster is the sum of the distances between each fragment in the cluster and the medoid \((m)\), i.e. \( \sum_{f \in C} \text{Dist}_{AGED}(f, m) \). In the case of standardization approach A2, every fragment in the cluster can potentially be used as the reference. Given that the aim is to maximize the benefit-to-cost ratio, i.e. to show first those clusters that are more amenable to standardization, we will pick as reference the fragment that yields the highest benefit-to-cost ratio.

The benefit of standardizing a cluster of approximate clones and replacing them with references to a shared subprocess, is proportional to the amount of reduction in duplication, which reflects itself in a reduction in size of the overall collection of process models. This size reduction is equal to the sum of the sizes of the fragments in the cluster (since they are removed) to which we subtract the size of the medoid – since this medoid becomes a new subprocess – and the number of fragments – since each cluster is replaced by a “call activity” to the subprocess. In other words, the benefit of standardizing a cluster is \( \sum_{f \in C} |f| - |m| - |C| \).

Given the above, we define the benefit-to-cost ratio of a cluster obtained with approach A1 as \( BCR(C) = \frac{\sum_{f \in C} |f| - |m| - |C|}{\sum_{f \in C} \text{Dist}_{AGED}(f, m)} \). In the case of standardization approach A2, we define the benefit-to-cost ratio of a cluster as the maximum of BCR(C) across all fragments in the cluster.
2.4. Graph-Edit Distance Threshold

The Graph-Edit Distance (GED) threshold is used to determine the sensitivity to differences between process model fragments when detecting approximate clones. This threshold must be in the interval (0,1). A value of 0 leads to Type 1 clones, viz. exact clones, and should thus be excluded when looking for approximate clones. Similarly, a value of 1 leads to fragments that are arbitrarily different, and cannot thus be used for standardization.

In practice, the choice of the GED threshold value depends on the standardization context. For example, if the standardization project has limited mandate, i.e. minor changes only are acceptable, a tighter GED threshold (e.g. 0.2) would be required to be sensitive to small changes only. On the other hand, in projects allowing major changes, for example in the case of company mergers or when the purpose is to increase simplification of operations across different units, a looser threshold should be used (e.g. 0.6). The GED threshold further depends on characteristics of the process model collection. In fact, for a heterogeneous collection, i.e. one where the various process models follow different modeling styles and conventions have not been enforced, one may want to use a high threshold to identify as many differences as possible, including “spurious” differences that result from a lack of homogeneity in the collection (e.g. two nodes being recognized as similar because their labels contain synonyms). Finally, different users may have different perceptions of when two process models are different.

3. SESE Fragment Extraction and Indexing

This section introduces two basic ingredients of the proposed techniques, namely the RPST and the RPSDAG, which allow us to efficiently identify SESE fragments from process models and index them in a way that allows us to identify exact clones and subsumed fragments.

3.1. RPST

The Refined Process Structure Tree (RPST) [17] is a parsing technique that takes as input a process model and computes a tree representing a hierarchy of single-entry single-exit (SESE) fragments. Intuitively, a process model, represented as a directed graph, is partitioned into sets of edges such that the subgraph induced by each set of edges is a SESE fragment. SESE fragments are organized by subset inclusion to form a rooted tree, where siblings are associated with disjoint sets of edges. As the process graph is partitioned into a set of edges, some
nodes may be shared in several SESE fragments. The RPST can be computed for any process model in linear time and it is unique [17].

A node in an RPST corresponds to a fragment of one out of four types: trivial, polygon, bond or rigid. A trivial consists of a single edge. A polygon represents a sequence of fragments. A bond corresponds to a subgraph where all child fragments are adjacent to the entry and exit nodes of the fragment. Any other case is a rigid fragment. We use the prefixes T, P, B and R to designate the type of fragment.

Figures 2(a)–(c) present sample process fragments extracted from models in the SAP Reference Model [18]. Each SESE fragment is delimited by a dashed rectangle. Figure 2(d) shows a tree representation of the RPST of each fragment in Figures 2(a)–(c). Consider specifically the process model shown in Figure 2(a). This model contains three bonds (B1, B2 and B3), two polygons (P1 and P2) and one rigid fragment (R1). The rigid fragment R1 is the root fragment, having B1, P1, and P2 as children. Polygon P1 is parent of bonds B2 and B3. Bond B1 appears in three different places (its occurrences are thus exact clones). Meanwhile, bonds B2 and B4 could be considered as approximate clones, depending on the user-defined distance threshold. Similarly, one level above, R1, R2 and R3 could also be considered as approximate clones.

3.2. RPSDAG

The RPSDAG [7] is an index structure designed for efficient and accurate identification of exact clones in a collection of process models. Conceptually, it can be thought of as the union of a set of RPSTs. A node in the RPSDAG corresponds to a SESE fragment of a model in the collection, whereas edges encode the containment relation among SESE fragments. Importantly, each fragment only appears once in the RPSDAG. Thus, if a fragment appears multiple times, in the same RPST or in different RPSTs, it is factored out and represented only once in the RPSDAG. For example, Figure 2(d) shows the RPSTs and the RPSDAG of the process fragments presented in Figures 2(a)–(c). Note that fragments B1 and P2 are represented only once in the RPSDAG. A node in the RPSDAG that has more than one parent is an exact clone fragment.

5 The sample process models in this paper are captured in the Event-driven Process Chain (EPC) notation because this is the original notation in which the models are captured. However, the presented techniques are notation-independent and can be applied for example to models captured using the standard Business Process Model and Notation (BPMN). The only assumption is that process models are represented as directed graphs with labeled nodes.
Figure 2: Sample process model fragments, their decomposition into RPSTs and corresponding RPSdag. SESE fragments are delimited by dashed rectangles and labelled “R” for rigid, “B” for bond and “P” for polygon followed by a number. Highlighted boxes represent exact clones.
The RPSDAG is built incrementally. When a new process model is added to the collection, the corresponding RPST is computed and merged into the existing RPSDAG. The RPSDAG implementation described in [7] incorporates several optimizations that make it scalable to real-life repositories of process models with hundreds of models. In addition to identifying exact clones, the RPSDAG allows us to determine if a process fragment is contained in another – a feature we will use during clustering.

4. Approximate Clones Clustering

In order to operationalize the standardization approaches discussed in Section 2.2, we propose to compute clusters of SESE fragments using adapted versions of existing clustering algorithms. To this end, we reviewed various clustering algorithms and selected two of them which allowed us, with some adaptations, to fit each of the two standardization approaches. These are the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [19] for the standardization approach A1, and the Hierarchical Agglomerate Clustering (HAC) [19] for approach A2.

Both algorithms assume that the distance between every possible pair of fragments is pre-computed and stored in a distance matrix. Below we discuss the calculation of this distance matrix, before presenting the algorithms themselves.

4.1. Distance matrix

Given a collection of process fragments of size N, the distance matrix of this collection is a symmetric matrix of size $N \times N$ such that $\text{Dist}_{\text{GED}}(s, p)$ is the graph-edit distance between fragment $s$ and $p$. As an optimization and given the cost of calculating $\text{Dist}_{\text{GED}}$ for two process graphs (cf. Section 2.1), the matrix only stores the distance $\text{Dist}_{\text{GED}}$ of Definition 5 for a pair of fragments if this is within the approximate clone threshold $\tau$ of Definition 7, and if the two fragments do not contain one another. For all other fragment pairs, it stores $\infty$.

As a further optimization, we first calculate a lower-bound of the GED. When this lower-bound is above threshold $\tau$, we do not need to compute $\text{Dist}_{\text{GED}}$ and instead store $\infty$. The lower bound is calculated using the following observations. First, we take the largest of the two graphs (i.e. the one with more nodes and more edges). Say that $H_2$ is larger than $H_1$ (otherwise we revert the roles). Now, assuming that $H_1$ is a subgraph of $H_2$, all vertices of $H_1$ can be substituted by vertices of $H_2$, all edges of $H_1$ are matched with edges of $H_2$, and no vertices are substituted. The only differences come from the vertices and edges of $H_2$ that
are not in $H_1$. Thus, $f_{skipv} = \frac{|V_1| - |V_2|}{|V_1| + |V_2|}$, $f_{skipe} = \frac{|E_1| - |E_2|}{|E_1| + |E_2|}$ and $f_{subv} = 0$. These are lower-bound values. If the assumption that $H_1$ is not a subgraph of $H_2$ is violated, then the GED will necessarily be greater because it entails additional differences. Thus, we conclude that $\text{Dist}_{GED}(H_1, H_2)$ is greater than the one obtained by feeding the above lower-bound values of $f_{skipv}$, $f_{skipe}$ and $f_{subv}$ into the equation for $\text{Dist}^M_{GED}(H_1, H2)$ in Definition 5. Note that if the graphs have equal size, the obtained lower-bound is zero—which is not useful.

We observe that the presence of roles and objects in process models will not increase the size of the distance matrix, since these elements are captured as node attributes of a process graph (cf. Section 2.1). Rather, it will typically lead to a less dense matrix, since any (small) difference in the attributes of two nodes to be compared contributes to increasing the distance between the two nodes.

4.2. Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

In standardization approach A1, we propose to standardize a set of clones towards a medoid fragment. Given a cluster, a medoid is an element of the cluster that is the closest to the center of the cluster. In order to avoid arbitrarily large changes to any given fragment, we need to bound the distance between the medoid and all other elements in the cluster by a certain threshold. A well-known algorithm that is built upon this principle is DBSCAN. DBSCAN creates clusters based on the density of neighborhoods. Given a set of objects $O$, the neighborhood of an object $o \in O$ is the set of fragments $N_o = \{o_i \in O \mid d(o, o_i) \leq \epsilon\}$, where $d(o, o_i)$ is a distance measure between $o$ and $o_i$ and $\epsilon$ is the neighborhood radius. A core object is an object whose $|N_o| \geq Size_{min}$, where $Size_{min}$ is the minimum cluster size (we observe that a core object is contained in its neighborhood since its distance with itself is 0). Thus, we have to specify two parameters for this algorithm: neighborhood radius and minimum cluster size. In our case, the neighborhood radius coincides with the user-defined distance threshold $\tau$, whereas we can fix $Size_{min}$ to 2 to retrieve clusters of at least two fragments. Here, we use the notion of graph-edit distance $\text{Dist}_{GED}$ as the distance measure between two objects as discussed in Section 2.1.

Standard DBSCAN identifies all core objects of a given dataset and considers their neighborhoods as initial clusters. If two core objects are within each other’s neighborhood, their neighborhoods are merged into a single cluster. On the other hand, if an object does not belong to the neighborhood of any core object, it is marked as noise. Our adaptation of DBSCAN is described in Algorithm 1. Given the set of process fragments $G$ extracted from the RPSDAG, the algorithm repeats...
the clustering process (Steps 2–14) until all fragments in $G$ have been checked whether they are core objects. At the beginning of each iteration, a random fragment $f$ is removed from $G$ and marked as “processed”. The neighborhood $N_f$ of $f$ is computed (Step 3), and if $f$ is a core object the fragments in $N_f$ are removed from $G$ and from Noise (Step 5), and added to a new cluster $C$ (Step 6). Otherwise $f$ is treated as noise and another fragment is extracted from $G$. The algorithm then expands cluster $C$ by checking whether there are core objects in $C$ whose neighborhoods can be merged with $C$. This is done by iterating over all fragments in $N_f$ except $f$, via a set $M_C$. For a fragment $m$ in $M_C$ that has not been processed, its neighborhood $N_m$ is computed (Step 8) to determine whether $m$ is itself a core object. If so, before merging its neighborhood with $C$, we check whether there is still a medoid $s$ whose distance with all other fragments of the combined cluster is within $\tau$ (Step 10), otherwise we will create clusters whose fragments are far apart from each other to be standardized. In case of merging, the fragments in $N_m$ are removed from $G$ and added, except $m$, to $M_C$ (Step 11), so that they can be checked whether they are core objects. If $N_m$ cannot be merged with $C$, $m$ is added back to $G$ so that it can be eventually processed again (Step 12). In fact, $N_m$ may form a cluster by itself or be merged with some other cluster.

A fragment’s neighborhood is constructed using the distance matrix. Given the non-containment relation enforced by this matrix, a fragment cannot be in the neighborhood of a core object that contains, or is contained by, it. We call two fragments where one is contained by the other “related” fragments. Still, it is possible to include two related fragments in a neighborhood if they are both sufficiently similar to the core object. To prevent this, we retrieve the set of all the ascendants and descendants of a fragment by computing the fragment’s transitive closure on the RPSDAG, and add to the neighborhood the fragment in the transitive closure that is the nearest to the core object (the original fragment may thus be discarded in favor of one of its ascendants or descendants). Further, we mark all other fragments in the transitive closure as “visited” for that cluster, so that these fragments are not included in any neighborhood of that cluster. These operations are performed as part of the neighborhood computation (Step 3) and are not explicitly shown in Algorithm 1 in the interest of brevity.

As an example, consider the collection of fragments shown in Fig. 4.2 and its corresponding distance matrix in Table 1. Let us assume that the threshold $\tau$ is 0.4 and that the minimum cluster size $Size_{min}$ is 3. One possible iteration of Algorithm 1 could be the following. First, sets Clusters and Noise are initialized to the empty set (Step 1). Let us now assume that fragment F20 is randomly selected as the one to be analyzed, that is $f$ in Step 2. The neighborhood of $N_f$
Algorithm 1: DBSCAN Clustering

**Input:** Set $G$ of process fragments.

**Output:** The sets of clusters ($Clusters$) and noise ($Noise$).

1. Initialize $Clusters$ and $Noise$ to empty sets.
2. Remove a fragment $f$ from $G$ and mark $f$ as “processed”.
3. Retrieve the neighborhood $N_f$.
4. If $|N_f| < Size_{min}$, add $f$ to $Noise$, then go to 2.
5. Remove $N_f$ from $G$ and from $Noise$.
6. Initialize a new cluster $C$ in $Clusters$ with $N_f$, and a new set $M_C$ to $N_f \{f\}$.
7. Remove a fragment $m$ from $M_C$.
8. If $m$ is not “processed”, mark $m$ as “processed” and retrieve $N_m$.
9. If $N_m \geq Size_{min}$
10. If there is a fragment $s \in C \cup N_m$ such that for all $p \in C \cup N_m$
11. $Dist_{GED}(s, p) \leq \tau$
12. Remove $N_m$ from $G$ and $Noise$ and add $N_m$ to $C$ and $N_m \{m\}$ to $M_C$.
13. Else, mark $m$ as “unprocessed” and add it to $G$.
14. If $M_C \neq \emptyset$ go to 7.
15. If $G \neq \emptyset$ go to 2.

<table>
<thead>
<tr>
<th></th>
<th>F20</th>
<th>F21</th>
<th>F22</th>
<th>F23</th>
<th>F24</th>
</tr>
</thead>
<tbody>
<tr>
<td>F20</td>
<td>0.00</td>
<td>0.20</td>
<td>0.09</td>
<td>0.42</td>
<td>0.46</td>
</tr>
<tr>
<td>F21</td>
<td>0.20</td>
<td>0.00</td>
<td>0.15</td>
<td>0.41</td>
<td>0.47</td>
</tr>
<tr>
<td>F22</td>
<td>0.09</td>
<td>0.15</td>
<td>0.00</td>
<td>0.35</td>
<td>0.41</td>
</tr>
<tr>
<td>F23</td>
<td>0.42</td>
<td>0.41</td>
<td>0.35</td>
<td>0.00</td>
<td>0.41</td>
</tr>
<tr>
<td>F24</td>
<td>0.46</td>
<td>0.46</td>
<td>0.41</td>
<td>0.41</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 1: Excerpt of the distance matrix associated with the collection of fragments from Fig. 4.2.

corresponds to set $\{F20, F21, F22\}$. As the size of $N_f$ is 3, it meets the minimum size required for a cluster to be kept (Step 4). A cluster $C$ is created with $\{F20, F21, F22\}$ and $M_C$ is set to $\{F21, F22\}$ (Step 6). Let us now assume that in Step 7 we select $m$ to be fragment F22. We observe that the neighborhood of F22 is $\{F20, F21, F22, F23\}$. Interestingly, the distance between F22 and the other fragments in the set $\{F20, F21, F22, F23\}$ is below 0.4. Thus, cluster $C$ is updated to $\{F20, F21, F22, F23\}$ with F22 as the cluster medoid. Since F22 is contained in F24, we cannot consider F24 and F22 in the same cluster. To avoid this possibility, we
use the special value $\infty$ in the distance matrix. The algorithm continues analyzing other fragments in the collection in the same way.

The complexity of Algorithm 1 is dominated by that of neighborhood computation (Steps 3 and 8), and by that of the merging condition (Step 10). Neighborhood computation for a fragment $f$ requires at most $|G| - 1$ lookups in the distance matrix. The exploration of the transitive closure of each neighbor of $f$ requires further $|G| - 1$ lookups (retrieving the transitive closure of an RPSDAG node is linear on the RPSDAG size, which is bounded by $|G|$). Similarly, the merging condition requires $|G| - 1$ lookups in the distance matrix for all members of a cluster. As the main loop is repeated $|G|$ times, the overall complexity of Algorithm 1 is $O(|G|^3)$. This is higher than the complexity of standard DBSCAN, which is $O(|G|^2)$ [19]. That said, the search space is greatly reduced by the cutoff conditions used when computing the distance of clusters, i.e. the distance threshold $\tau$ and the non-containment relationship. The result is that the distance matrix is sparse, but the sparsity depends on intrinsic characteristics of the process model collection. Further, we store each computed neighborhood so that it can be reused when reprocessing a core object whose neighborhood has not been merged.
4.3. Hierarchical Agglomerate Clustering (HAC)

In standardization approach A2 (cf. Section 2.2), a set of approximate clones can be standardized by selecting any fragment in the group as a reference and standardizing all other fragments towards this reference fragment. In other words, we require that every two fragments in a cluster have distance below the threshold $\tau$. This goal can be straightforwardly mapped to the strategy followed by the basic hierarchical agglomerative clustering method [19]. This clustering method starts with singleton clusters and iteratively merges the two clusters that are found to be the closest to each other, among all possible pairs. The process of merging continues until there is only one cluster left.

One key issue is the definition of the distance between two clusters, which needs to be recomputed after every cluster merging. Several possibilities are available, e.g., taking the smallest distance between fragments in one of the clusters to the fragments in the other one, known as single link, or taking the farthest distance, referred to as complete link. The complete link strategy suits well to the standardization approach A2, as it allows us to identify the cluster mergings that will not meet the requirement of keeping a distance below the threshold $\tau$. Note that the identification of such situation can be accomplished ahead of time. The intuition is captured in the following definition.

**Definition 9 (Distance of clusters under complete link strategy)** Let $C_i$ and $C_j$ be clusters in the dendrogram built by a hierarchical clustering algorithm, and $\tau$ be the similarity threshold among fragments of $C_i$ and fragments of $C_j$. Moreover, let $F(C)$ be a function that returns the set of fragments associated with $C$, inductively defined as follows: (BASE) if $C$ is a leaf node in the dendrogram, $C$ is a singleton and refers to a single fragment, say $f$, then $F(C) = \{f\}$; (STEP) if $C$ is an intermediate node then $F(C) = \bigcup_{c \in C} F(c)$. The distance of clusters $C_i$ and $C_j$, denoted as $\text{Dist}(C_i, C_j)$, can be defined as follows.

$$\begin{align*}
\text{Dist}(C_i, C_j) &= \max_{f \in F(C_i), g \in F(C_j)} \text{GED}(f, g) > \tau \\
&= \infty \text{ otherwise}
\end{align*}$$

The distance of two clusters is set to $\infty$ when there exists one fragment in the first cluster which is in containment relationship with another fragment in the second cluster. Moreover, when the farthest distance between fragments of both clusters is above the threshold $\tau$, the distance is set to $\infty$. In the two previous cases, we are meeting the constraints described in Definitions 7 and 8. The farthest
distance between the fragments of two clusters is reported as the distance of the clusters, only when the value is less or equal to the threshold $\tau$. Algorithm 2 corresponds to the modified version of the basic hierarchical agglomerative method adapted for clustering approximate clones.

**Algorithm 2:** Hierarchical Agglomerative Clustering

**Input:** Set $G$ of process fragments.

**Output:** The set of maximal clusters ($\text{TopClusters}$).

1. For each $f \in G$ create a singleton cluster. Initialize $\text{TopClusters}$ to contain all singleton clusters.

2. Using the distance matrix between fragments, calculate the initial distance matrix between clusters in $\text{TopClusters}$, i.e. $D[i, j] \leftarrow \text{Dist}(C_i, C_j)$, where $C_i, C_j \in \text{TopClusters}$.

3. In the distance matrix $D$, select two clusters $C_i, C_j \in \text{TopClusters}$ such that their distance is the minimum. Stop if no such pair exists, i.e. either all distances in $D$ are $\infty$ or $|\text{TopClusters}| = 1$.

4. Combine clusters $C_i$ and $C_j$ to form a new cluster $C_{ij}$. Remove clusters $C_i$ and $C_j$ from $\text{TopClusters}$. Add cluster $C_{ij}$ to $\text{TopClusters}$.

5. Update matrix $D$ by adding the distance between cluster $C_{ij}$ and all other clusters in $\text{TopClusters}$.

6. Go to 3.

Algorithm 2 can be divided into two parts. Step 1 and 2 initialize the set of singleton clusters, store them in $\text{TopClusters}$ and initialize the distance matrix between clusters (according to Definition 9). The remaining steps correspond to the main loop. In Step 3, two clusters are selected such that their distance is found to be the smallest among all possible pairs. If the distance of such fragments is $\infty$ or there is only one cluster left then the algorithm stops. In Step 4, a new cluster is created to hold the union of the previously selected pair. In Step 5, the distance matrix is updated (according to Definition 9), by removing the pair of clusters previously selected and adding the newly created cluster.

The algorithm starts with a working set of $|G|$ clusters. In every iteration, two clusters are removed and a new one is added. Hence, the size of the working set decreases monotonically. The algorithm stops when $|\text{TopClusters}| = 1$ or before if the entire distance matrix $D$ is filled with $\infty$.

Let us consider again the sample collection of fragments and distance matrix shown in Fig. 4.2 and Table 1. Steps 1 and 2 of Algorithm 2 create a copy of
the distance matrix with the only difference that the columns and rows are labeled with the corresponding singletons. In Step 3, the (singleton) clusters that are the closest to each other are selected. In our example, this corresponds to clusters \{F20\} and \{F22\}. Then, a new cluster is created by merging these clusters (Step 4) and the distance matrix is updated as shown in Table 2. In the updated distance matrix we can observe that the distance from each cluster to cluster \{F20, F22\} has been set to the maximum distance between that cluster and either F20 or F22. In a second iteration of Algorithm 2, clusters \{F20, F22\} and \{F21\} are merged and the distance matrix is updated as in Table 3. The resulting cluster cannot be further merged with the fragments shown in the distance matrix because the distance is above the threshold value of 0.4. One further remark is that DBSCAN and HAC retrieve different clusters on this example. This happens because DBSCAN uses \(\tau\) to set the maximum distance between each fragment and the cluster medoid, whereas HAC uses \(\tau\) as the limit on the distance between every two fragments in a given cluster.

The complexity of Algorithm 2 is dominated by the maintenance of the distance matrix (i.e., Steps 2 and 5), which has an initial size of \(O(|G|^2)\). As the main loop is repeated \(O(|G| - 1)\) times, the worst-case upper bound of the complexity is of \(O(|G|^3)\) [19]. The same simplifications of the search space that we used for DBSCAN apply to HAC (distance cutoff and non-containment). Also this algorithm has shown to be efficient in our experience.

<table>
<thead>
<tr>
<th></th>
<th>{ F20, F22 }</th>
<th>{ F21 }</th>
<th>{ F23 }</th>
<th>{ F24 }</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ F20, F22 }</td>
<td>0</td>
<td>0.2</td>
<td>0.42</td>
<td>(\infty)</td>
</tr>
<tr>
<td>{ F21 }</td>
<td>0.2</td>
<td>0</td>
<td>0.41</td>
<td>0.47</td>
</tr>
<tr>
<td>{ F23 }</td>
<td>0.42</td>
<td>0.41</td>
<td>0</td>
<td>0.41</td>
</tr>
<tr>
<td>{ F24 }</td>
<td>(\infty)</td>
<td>0.47</td>
<td>0.41</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Distance matrix after merging clusters \{F20\} and \{F22\}.

<table>
<thead>
<tr>
<th></th>
<th>{ F20, F21, F22 }</th>
<th>{ F23 }</th>
<th>{ F24 }</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ F20, F21, F22 }</td>
<td>0</td>
<td>0.42</td>
<td>(\infty)</td>
</tr>
<tr>
<td>{ F23 }</td>
<td>0.42</td>
<td>0</td>
<td>0.47</td>
</tr>
<tr>
<td>{ F24 }</td>
<td>(\infty)</td>
<td>0.47</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3: Distance matrix after merging clusters \{F20, F22\} and \{F21\}.
5. Performance and Accuracy Evaluation

We implemented the two techniques for approximate clone detection as a plugin of the Apromore platform [20]. This plugin allows users to identify, cluster, analyze and visualize approximate clones in the 2D space. A detailed presentation of the functionality of this plugin is provided in [21].

Using the Apromore plugin, we evaluated various aspects of the proposed techniques. First, we used the two techniques to detect approximate clones in process model repositories from practice, in order to measure their runtime performance, as well as the clusters characteristics in terms of distribution of size and benefit-cost ratio over the clusters retrieved. We use these results to provide an initial discussion on the differences between the two techniques (cf. Section 5.1). Second, we built a synthetic dataset to evaluate the accuracy of the two techniques in terms of correctly retrieving clusters of process fragments that have evolved via copy-pasting followed by independent modifications (Section 5.2). The results of these evaluations are reported in this section.

In the next section, we report on the results of two between-group experiments in order to evaluate: i) the perceived standardizability of the clusters produced by our techniques in comparison with those manually created by users, and ii) the perceived correctness of the clustering performed by our techniques in comparison to manual clustering.

5.1. Runtime performance and clusters characteristics

In this first evaluation, we used the two clustering techniques to examine the occurrence of approximate clones in practice, measure the runtime performance and check the characteristics of the clusters retrieved, in terms of distribution of clusters size and benefit-to-cost (BCR) ratio. For this purpose we looked at two industry-size datasets. The first dataset is the SAP R/3 reference model [18]. It contains 595 models with sizes ranging from 5 to 119 nodes (average 22.28). The second dataset is taken from an insurance company under condition of anonymity. It contains 363 models ranging from 4 to 461 nodes (average 27.12). We chose these two datasets because they exhibit different characteristics. The SAP dataset is a homogeneous dataset: it was developed over a relatively short period for a particular purpose (customization of SAP ERP deployments) by a team of consultants, and intentionally uses standardized terminology and procedures. Evidence

\footnote{Available at www.apromore.org}
of this is the high number of exact clones we retrieved, and their size w.r.t. the total size of the dataset: 563 clones corresponding to 17.1% of the total size [7]. On the other hand, the Insurance collection is an example of heterogeneous dataset: it was developed by different teams of analysts at different points in time and for different purposes (e.g. communication, requirements analysis, organizational restructuring), without a unified set of conventions being adopted by the various teams. As a result, this dataset exhibits an inconsistent use of model structures and labels (e.g. terms such as “mortgage” and “loan” are used interchangeably). The number of exact clones found in this dataset and their size w.r.t. the total size is much less, compared to the SAP dataset: 302 exact clones, corresponding to 9.5% of the total size [7].

We first computed the RPSDAG for both datasets and post-processed them by factoring out all exact clones using the technique presented in [7]. This yielded 2,238 fragments with at least 4 nodes for the SAP dataset (11.47 average size), and 2,037 fragments with at least 5 nodes for the insurance dataset (16.58 average size). We used a lower bound on the fragments’ size, corresponding to the size of the smallest process model in the evaluated datasets (4 nodes for SAP and 5 for the insurance dataset), in order to avoid introducing subprocesses that are smaller than those that process modelers would normally define themselves. Indeed, it might not be desirable to refactor out small clones, since this would add complexity and fragmentation in the model collection by introducing many small subprocesses and making the collection more difficult to navigate. We then applied the two clustering methods independently, having eliminated exact clones to avoid double-counting.

The clustering algorithms were run with a $Dist_{GED}$ threshold of 0.4, i.e. the two fragments need to be at most 40% different, or at least 60% similar. This threshold value was chosen because it corresponds to a relatively low similarity value, while being still sufficient to consider the approximate clones as “more similar than different”, such that standardization of the retrieved approximate clone fragments can be considered as an option. Therefore, this value allows us to evaluate the techniques in a very conservative setting. A higher threshold value, e.g. 0.6, would lead to larger clusters (i.e. more approximate clones), with less clusters overall. Likewise, a value lower than 0.4 would lead to smaller clusters, with more
clusters overall.\footnote{While we are not aware of a systematic empirical study on the GED threshold, in [12] (Fig. 12) we empirically showed that if we try to standardize two process models by taking their union (i.e. in a behaviorally preserving manner), at a distance of 0.4 the size of the standardized fragment is 80\% of the sum of the sizes of the two fragments to be merged, and that between 0 and 0.5 this size varies linearly. Thus, with 0.4 we achieve a good compromise between the size of the standardized fragment and the amount of differences we are sensitive to.}

The tests were run on a PC with a dual core Intel processor, 1.8GHz, 4GB memory, running Microsoft Windows 7 and Oracle Java Virtual Machine v1.6. The cluster computation is dominated by the computation of the distance matrix which took 26.3 mins for the SAP dataset and 2.69 hours for the Insurance dataset. The time for clustering itself is negligible in comparison. The longer time taken for the Insurance dataset is justified by the size of its fragments – much larger than those in the SAP dataset (e.g. the largest fragment in the Insurance dataset is a rigid with 461 nodes whereas the largest SAP fragment contains 117 nodes).

Figure 4 plots the histograms of distribution of cluster sizes for the two datasets. For the SAP dataset we retrieved 364 clusters with DBSCAN (with sizes ranging from 2 to 5 fragments per cluster) and 335 clusters for HAC (sizes between 2 and 13), while for the Insurance dataset we retrieved 243 clusters with DBSCAN (sizes between 2 and 6) and 309 clusters with HAC (sizes between 2 and 10). This confirms the intuition that real-life process model repositories contain a large number of approximate clone clusters, and thus that copy/pasting of fragments across process models is a very common practice. Looking at the size distribution, for both datasets the majority of the clusters contain between 2 to 8 fragments. This suggests that copy/pasting is typically limited to 2-8 copies per fragment. This result holds despite the approach we choose to standardize the approximate clones clusters.

Figure 5 shows the histograms of BCR distributions for both datasets. For the SAP dataset, the great majority of clusters have a very low BCR (there are 294 clusters with a BCR below 2 for DBSCAN and 236 for HAC), with only a few clusters having very high BCR (there are only 3 clusters with BCR above 7 for DBSCAN and 22 for HAC). A similar trend is registered for the Insurance dataset (with 126 clusters below 2 for DBSCAN and 257 for HAC, and only 8 clusters above 7 for DBSCAN and 4 for HAC). That said, in the SAP dataset we obtain higher BCRs with HAC whilst for the Insurance dataset with DBSCAN. This suggests that depending on the type of the repository, one of the two techniques is more appropriate than the other. In fact, HAC is a stricter algorithm than
Figure 4: Number of clusters vs clusters size for both techniques. The clusters sizes in the X axis are grouped by intervals. For example, the interval (2,4] contains all the clusters of size 3 and 4, which are 133 for DBSCAN and 93 for HAC in the SAP dataset.

DBSCAN since it is based on cluster diameter rather than radius. For this reason it performs better on process model collections that are more homogeneous, such as the SAP dataset. On the contrary, DBSCAN is a more permissive algorithm, better suited for heterogenous collections such as the Insurance dataset.

5.2. Accuracy

Next, we evaluated the accuracy of the two techniques in retrieving clusters of clones that have emanated from a single original fragment, by means of copy-pasting followed by independent changes to the duplicated fragments. We did so by simulating a situation where new fragments are created by copying a master fragment across various models of the repository, and then applying minor changes, as often happens in reality. We randomly selected 50 such master fragments from the two industry-size datasets used in the occurrence analysis, such
Figure 5: Number of clusters vs BCR for both techniques. The values of the BCR in the X axis are grouped by intervals. For example, the interval [1,2) contains all the clusters of BCR ranging from 1 to 1.99, which are 124 for DBSCAN and 102 for HAC.

that they were sufficiently different from each other (pairwise graph-edit distance above 70%).

To test the accuracy of the DBSCAN algorithm, we used these 50 fragments as “seeds” to generate 50 artificial clusters by producing from 2 to 10 variants for each seed, and grouping each seed with its variants in a cluster. We obtained a total of 311 fragments in 50 clusters. Seed variants were obtained automatically, by randomly applying simple change operations (edge/node removal or insertion and label substitution) such that the graph-edit distance between a variant and its seed was no more than 40% — the same threshold that we used in Section 5.1.\(^8\) In particular, for label substitution we replaced an existing node label with a random label extracted from the set of labels used in the related dataset. The clusters’ size

---

\(^8\)The idea of using random mutation of seed fragments to generate synthetic data for evaluating clone detection methods is also used in [22] in the context of source code clone detection.
ranged from 3 to 10 fragments (average 6.35). We then generated 300 process models from the two existing datasets, such that none of these models contained any of the seed fragments, and we randomly inserted the 311 fragments into these models such that a model would contain from 0 to 2 fragments. We then extracted the RPSDAG from this dataset and clustered the retrieved fragments using our DBSCAN. The algorithm retrieved 328 clusters. We matched each artificial cluster with the retrieved cluster that yielded the maximum \( F-Score \) [23]. F-Score is the harmonic mean of the recall and precision of a retrieved cluster with respect to (w.r.t.) an artificial cluster. Precisely, given an artificial cluster \( l \) and a retrieved cluster \( s \), the F-Score of \( s \) w.r.t. \( l \) is \( F(s, l) = \frac{2 \cdot R(s, l) \cdot P(s, l)}{R(s, l) + P(s, l)} \) where \( R(s, l) \) and \( P(s, l) \) are the recall and precision of \( s \) w.r.t. \( l \).

In order to measure the overall quality of the algorithm, we then computed the weighted average F-Score (\( F_{wa} \)) [23]. \( F_{wa} \) is the maximum F-Score of each artificial cluster weighted against the combined size of all artificial clusters. Let \( L \) be the set of artificial clusters and \( S \) the set of retrieved clusters. Then \( F_{wa} = \sum_{l=1}^{L} \frac{|l|}{|L|} F(l) \), where \( F(l) = \max_{s \in S} F(s, l) \).

We repeated the same experiment for the HAC algorithm. In order to ensure that all fragments in an artificial cluster have pairwise graph-edit distance within the 40% threshold, we used a random walk approach.

<table>
<thead>
<tr>
<th></th>
<th>DBSCAN</th>
<th>HAC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recall</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>0.17</td>
<td>0.1</td>
</tr>
<tr>
<td>max</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>avg</td>
<td>0.71</td>
<td>0.82</td>
</tr>
<tr>
<td>std</td>
<td>0.37</td>
<td>0.25</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>0.2</td>
<td>0.17</td>
</tr>
<tr>
<td>max</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>avg</td>
<td>0.89</td>
<td>0.84</td>
</tr>
<tr>
<td>std</td>
<td>0.24</td>
<td>0.33</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_{wa} )</td>
<td>0.73</td>
<td>0.77</td>
</tr>
</tbody>
</table>

From each seed we generated a variant with graph-edit distance of at most 0.4. We chose one of these two fragments and generated another variant such that its distance to both fragments was at most 0.4, and so on until we generated from 2 to 10 variants for each cluster. This process was carried out automatically, and led to a total of 289 fragments in 50 clusters, with sizes ranging from 3 to 10 fragments (average 5.8). We added these fragments to the collection of 300 process models that was generated in the previous step, and then clustered the fragments retrieved
from the RPSDAG of this collection using HAC. This led to 295 clusters.

The results for both algorithms are reported in Table 4. Besides $F_{wa}$, this table reports the minimum, maximum, average and std. deviation of recall and precision for the best-matched retrieved cluster for each artificial cluster. The accuracy of the two algorithms is partly affected by the presence of approximate clones that exist in the generated process model collections, besides those that have been generated artificially. Despite this, the results show high $F_{wa}$ (0.73 for DBSCAN and 0.77 for HAC), as well as high average precision and recall for both algorithms, demonstrating the accuracy of the algorithms. None of the algorithms clearly outperforms the other.

Finally, we used the above data to evaluate the ranking accuracy of the BCR. For each algorithm, we plotted a ROC curve by ordering the retrieved clusters from the highest to the lowest BCR. In these curves, we considered a retrieved cluster as a true positive if it had a recall of 1, and as a true negative otherwise. The curves, provided in Fig. 6, show that the clusters with highest BCR are indeed those that most closely match the synthetically generated clusters. This result is confirmed by the Area Under the Curve which is 0.89 for DBSCAN and 0.72 for HAC (both with asymptotic significance less than 0.05).

![ROC Curve - DBSCAN](image1)

![ROC Curve - HAC](image2)

Figure 6: ROC curves for both algorithms.

It is not surprising that DBSCAN has a higher AUC than HAC. This algorithm clusters based on a most representative fragment, which mimics well the practice of creating approximate clones via independent evolutions of a single original fragment following copy-pasting. However, the clusters retrieved contain some
false positives, as shown in Table 4 (recall of 0.71). HAC on the other hand strikes a tradeoff between mimicking this practice and not adding too many fragments in each cluster (higher recall—0.82). This result is reflective of the difference between these two algorithms (DBSCAN being more permissive than HACs) as evidenced in Section 5.1.

6. User Evaluation

For the user evaluation of the two techniques as implemented in Apromore, we designed two between-groups experiments in which users of the two techniques were presented with process fragments identified through either the DBSCAN or the HAC technique, and were asked to provide answers to a set of standardization tasks and questions.

The first experiment concerned the evaluation of clusters of clones produced by the two techniques in terms of their perceived standardizability. The second experiment concerned the evaluation of the perceived correctness of the clustering performed by the two techniques in comparison to manual clustering.

The instruments used for this evaluation are available as supplementary material attached to this paper.

6.1. Perceived standardizability

The purpose of the first experiment was to shed light on the differences between the two techniques from a standardizability point of view, and on the usefulness of the BCR measure to inform an evaluation of the potential effort of standardization of clusters. Specifically, we examined how users judge the standardizability of the clusters produced by the two techniques in relation to varying BCR levels. We also measured which standardization strategy users deem most effective on clusters produced by the two techniques.

Design and Measures

We designed our experiment as follows. First, participants were asked to provide demographic information such as role and experience with process modeling. Each participant had to answer five questions about fundamental process modeling questions as a measure of process modeling knowledge [24]. We also asked them about their familiarity with EPC process models, which we used in our experiment, using the three item 7-point Likert scale developed in [25] (see Table 5).

Next we randomly distributed the participants into two groups, with each group being provided with five sets (i.e. clusters) of fragments extracted from the
SAP dataset, either identified by means of DBSCAN or HAC, which provided the treatment condition for our experiment. For both techniques, each of the five produced sets contained between three to five fragments, which were selected from the SAP dataset in order to vary in terms of BCR (from ‘0-2’, ‘2-4’, ‘4-6’, ‘6-8’ and ‘above 8’). Thus, we could evaluate the perceived standardizability of clusters produced by the two techniques in dependence on BCR levels of the clusters.

For each set, participants were asked to rate the perceived standardizability of that set on a five-item scale measuring similarity, complexity, suitability, readiness and ease of standardizability of the set of fragments. The measurement scales used are shown in Table 5. Scores for each item in the scale were aggregated to an average total factor score for the analysis. Additionally, for each set of fragments participants were asked to select a most suitable standardization strategy from a set of three options, under the assumption that the given set needed to be standardized (the question was: “Imagine that you want to standardize the business procedures described by the different process fragments in this set. Remember, standardization means replacing all fragments in the set with a single fragment.”). The three options were:

a) Replacement of all fragments within the cluster with one most suitable fragment from the cluster, which had to be identified; or

b) Replacement of all fragments within the cluster with any fragment from the cluster; or

c) Insertion of a new fragment as either a

   a. consolidation of all the fragments within the cluster, or
   b. new definition.

For options a) and b), participants were also asked to estimate the likely percentage of information loss that would be incurred by standardization through the selected strategy.

Participants

To gather useful data on our experiment, we required subjects that met three requirements: (a) having some but varying levels of expertise in process management and related technologies, (b) not having vast levels of experience or knowledge in the domain of the model collections we use (e.g. insurance), and (c) being
Table 5: Multi-item measurements used in the experiment

<table>
<thead>
<tr>
<th>Scale</th>
<th>ID</th>
<th>Measurement Items</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Familiarity with EPCs</strong></td>
<td>FAM1</td>
<td>Overall, I am very familiar with the EPC process modeling language.</td>
</tr>
<tr>
<td></td>
<td>FAM2</td>
<td>I feel very confident in my understanding of the EPC process modeling language.</td>
</tr>
<tr>
<td></td>
<td>FAM3</td>
<td>I feel very competent in using the EPC process modeling language.</td>
</tr>
<tr>
<td><strong>Cluster standardizability</strong></td>
<td>GS1</td>
<td>The process fragments in this cluster are similar to each other.</td>
</tr>
<tr>
<td></td>
<td>GS2</td>
<td>The process fragments in this cluster are all equally complex.</td>
</tr>
<tr>
<td></td>
<td>GS3</td>
<td>This cluster of process fragments is an ideal candidate for standardization.</td>
</tr>
<tr>
<td></td>
<td>GS4</td>
<td>This cluster of process fragments cannot readily be standardized.</td>
</tr>
<tr>
<td></td>
<td>GS5</td>
<td>It is very easy to identify an ideal candidate process fragment for standardization in this cluster.</td>
</tr>
</tbody>
</table>

comparable to the test subjects used in other process management, process modeling and technology validation studies in order to compare our results to those of others.

Similar to most other studies in this field [26, 27, 28, 29, 24], we recruited participants from universities we had access to. Overall, 73 users participated in this experiment. The majority of participants were post-graduate students that learned about process modeling, process model repositories and clone detection as part of their tertiary education (90%), followed by academic staff teaching these concepts and methods (8%), and process professionals (2%) with knowledge of the subject matter.

Participants, on average, had about 1.7 years of experience with process modeling and had read and/or created on average 28.3 process models over the last 12 months. Participants’ experience with the process modeling language used in the experiment, EPCs, ranged from 1 month to 5 years, with an average of 4.5 months. The self-reported familiarity with process models created with EPCs was significantly higher ($t = 5.17, p = 0.00$) than neutral with an average score of 4.8 on the 7-point scale, with ‘4’ representing the neutral value, indicating sufficient perceived experience in reading EPC diagrams. Overall, the demographics characterize our participants largely as proxies for novice BPM professionals, with one of our participants being representative of an expert practitioner (more than 5 years experience, more than 250 models created or read). Overall, our study population is roughly comparable to the reported demographic distribution of participants in related studies [26, 27, 28].
Analysis and Results

On the basis of the experimental data obtained, we performed a number of evaluations. First, we examined the perceived standardizability of clusters produced by the two techniques in terms of overall rating (the mean standardizability score in Table 6) and consistency of rating (the standard deviation of the standardizability score in Table 6), in relation to the i) technique used and ii) the BCR of the identified cluster. Table 6 provides relevant statistics and Fig. 7 visualizes the results in a scatter plot. Specifically, it shows that based on participants’ perceived standardizability ratings, the produced sets of clusters fall into two distinct groups. One group (D97, D287, H55, H106) of clusters were consistently rated as highly standardizable while the remaining fragments were not only rated lower in standardizability but also rated less consistently. When examining the clusters based on the data in Table 6, we see that the consistent and highly rated group of clusters is characterized by relative high BCRs (‘4-6’ and ‘above 8’).

Several findings emerge. The results indicate that users do not find clusters with low BCR standardizable, while clusters with high BCR levels can apparently be used for standardization. We further note that the results obtained are largely consistent for both DBSCAN and HAC across different levels of BCR (see Table 7). While overall, standardizability was rated slightly higher (0.05 points) for DBSCAN, we note that scorings for DBSCAN and HAC were equal (4.78) when BCR is high. These results highlight the importance of using the BCR to present clusters to business analysts in a decreasing order, from high to low BCR, in order to effectively aid the standardization effort.

Second, we examined participants’ preference for the three provided standardization strategies in dependence to the cluster of fragments received. Table 8 provides information about the preferred standardization ratings per cluster, as re-
Figure 7: Standardizability rating average and standard deviation for clusters

Table 7: Average cluster standardizability rating by BCR and technique

<table>
<thead>
<tr>
<th>BCR</th>
<th>Technique</th>
<th>N</th>
<th>Standardizability</th>
<th>Standardizability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>mean</td>
<td>Standard deviation</td>
</tr>
<tr>
<td>Low (below 6)</td>
<td>DBSCAN</td>
<td>3</td>
<td>4.32</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td>HAC</td>
<td>3</td>
<td>4.23</td>
<td>0.44</td>
</tr>
<tr>
<td>High (above 6)</td>
<td>DBSCAN</td>
<td>2</td>
<td>4.78</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>HAC</td>
<td>2</td>
<td>4.78</td>
<td>0.5</td>
</tr>
<tr>
<td>Total</td>
<td>DBSCAN</td>
<td>5</td>
<td>4.5</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>HAC</td>
<td>5</td>
<td>4.45</td>
<td>0.5</td>
</tr>
</tbody>
</table>

ported by overall 32 from the total of 73 participants, an effective response rate of 43.8% (answering was optional).

Overall, participants indicated a clear preference for standardizing fragments based on a most representative fragment per cluster. Average preference for strategy (a) was 57.6%, with strategy (c) (26.9%) and strategy (b) (15.5%) following in order. The preference for strategy (a) is also indicated by the estimated information loss incurred through the strategy, with the reported average information loss for strategy (a) (mean = 14.30%, st. dev. = 6.80%) being smaller than that estimated for strategy (b) (mean = 15.19%, st. dev. = 9.58%). This is the case for those participants who assessed DBSCAN clusters as well as those who assessed
Table 8: Reported standardization strategy by cluster

<table>
<thead>
<tr>
<th>ClusterID</th>
<th>Technique</th>
<th>BCR</th>
<th>Preference for strategy a)</th>
<th>Estimated information loss for strategy a)</th>
<th>Preference for strategy b)</th>
<th>Estimated information loss for strategy b)</th>
<th>Preference for strategy c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D151</td>
<td>DBSCAN</td>
<td>0-2</td>
<td>15</td>
<td>21.82</td>
<td>3</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>D56</td>
<td>DBSCAN</td>
<td>2-4</td>
<td>12</td>
<td>13.33</td>
<td>5</td>
<td>25</td>
<td>12</td>
</tr>
<tr>
<td>D97</td>
<td>DBSCAN</td>
<td>4-6</td>
<td>16</td>
<td>3.93</td>
<td>9</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>D364</td>
<td>DBSCAN</td>
<td>6-8</td>
<td>18</td>
<td>11.31</td>
<td>5</td>
<td>11.67</td>
<td>6</td>
</tr>
<tr>
<td>D287</td>
<td>DBSCAN</td>
<td>Above 8</td>
<td>25</td>
<td>4.58</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>H260</td>
<td>HAC</td>
<td>0-2</td>
<td>13</td>
<td>24.58</td>
<td>6</td>
<td>22</td>
<td>15</td>
</tr>
<tr>
<td>H177</td>
<td>HAC</td>
<td>2-4</td>
<td>18</td>
<td>20.29</td>
<td>4</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>H106</td>
<td>HAC</td>
<td>4-6</td>
<td>28</td>
<td>16.3</td>
<td>1</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>H83</td>
<td>HAC</td>
<td>6-8</td>
<td>13</td>
<td>12.69</td>
<td>6</td>
<td>33</td>
<td>13</td>
</tr>
<tr>
<td>H55</td>
<td>HAC</td>
<td>Above 8</td>
<td>26</td>
<td>14.2</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

HAC clusters, thus regardless of the type of clusters they were confronted with. Indeed, differences in preference for strategies (a) to (c) between DBSCAN and HAC clusters were all insignificant (with p-values ranging from 0.48 to 0.66). It is worth noting that strategy (a) is implemented by DBSCAN, which constructs clusters based on the vicinity of fragments to a common point, the cluster’s medoid. Thus, we may conclude that DBSCAN better implements the perceived preference for standardizing process model fragments by humans.

Participants were also asked to indicate for strategy (a) which one fragment is most suitable to replace all fragments. The data for clusters produced by DBSCAN shows that these participants did not identify this fragment with the medoid provided by the technique. In total, out of 86 responses that provided a preferred reference fragment for the 5 DBSCAN clusters, only 17 responses designated the medoid as the reference fragment. Instead, in all cases, the majority of participants designated the largest fragment of a cluster as the most representative (except when all fragments in the cluster had an equal size). This suggests that subjects tend to look for the reference fragment that “covers” as much as possible all fragments in the cluster (thus larger) rather than a reference fragment with minimum distance to all other fragments.

6.2. Perceived correctness

The purpose of the second experiment was to evaluate the correctness of the clustering techniques by means of comparison with manual clustering of process model fragments. Specifically, we measured how well the techniques place fragments that are perceived as being similar in a cluster and exclude fragments that
are perceived as noise, compared to the results obtained by users.

**Design and Measures**

In the follow-up experiment we again started by collecting demographic data about the participants, using measures identical to the first experiment (see above). That is, we collected data on process modeling experience, process modeling knowledge [24] and familiarity with EPC diagrams (using the same 7-point Likert scale as before [25]).

In the second part, participants had to perform two identical tasks in sequence. In each task we provided participants a collection of process fragments extracted from the SAP dataset, which they had to cluster. The first fragment collection contained 8 fragments previously classified into 3 clusters with varying BCR by the DBSCAN algorithm along with 9 fragments classified as noise (i.e. 17 in total). The second collection contained 9 fragments previously classified into 3 clusters with varying BCR by the HAC algorithm along with 9 fragments classified as noise (i.e. 18 in total). In both cases, a distance threshold of 40% was used. The 9 noise fragments of each collection were added by selecting 3 noise fragments per cluster where the distance between the medoid of the cluster and each noise fragment was more than 40%.

In each of the two clustering tasks, participants were asked to standardize the collections of fragments by grouping relevant fragments together into clusters. Participants did not know which fragments had been assigned to clusters by the two algorithms, how many fragments could or should be assigned to any one cluster, or how many clusters they could or should identify. This experimental design allowed us to compare the differences between manual clustering of fragments versus the clustering produced by the techniques in terms of two measures:

a) the placement of fragments into a number of clusters, and

b) the identification of noise.

The rationale for choosing two distinct collections of fragments for the two clustering tasks is as follows. We wanted the two techniques (DBSCAN and HAC) to produce results that could also be obtained by “manual” clustering, and in the context of a short experiment. For example, it would be very hard to manually create a cluster of 20 fragments where the fragments have average size of 50 nodes. Thus, we designed the experiment so that it could be completed within 1 hour (though we did not set any time limit). This requires imposing certain “simplicity” criteria on the fragments collection: small fragments (max 25 nodes),
small collection (no more than 20), small number of clusters (3), and small clusters size (2-3 fragments). However, we could not find a collection of fragments among those obtained by the two techniques that would respond to these criteria (due to the difference in the underlying clustering algorithms), and thus had to opt for two distinct collections. We refer to Section 7 for a discussion on how we mitigated the potential bias introduced by this choice.

Participants

Given the objective of this experiment, we were interested in participants with substantial process modeling experience. Participants were thus invited from the cohort of PhD students and academic staff working at the University of Tartu in Estonia and Queensland University of Technology in Australia. In both research groups, participants were actively researching topics on process modeling and related technologies, making them suitable participants.

A total of 16 users participated: nine doctoral students plus seven academic staff out of which three also had professional industry experience in process modeling and process model repositories. Participants, on average, had about 4.1 years of experience with process modeling and had read and/or created on average 71.6 process models over the last twelve months. The participants’ experience with the EPC process modeling language use ranged from 1 month to 5 years, with an average of 23.8 months. The average self-reported familiarity with process models created with EPCs was 4.40 out of 7. These characteristics describe a pool of participants as indeed considerably more experienced in process modeling than the participants in the first experiment.

Analysis and Results

The data collected in this second experiment allows us to examine the performance of the clustering techniques in comparison to manual clustering by end users. Essentially, we wanted to know whether the two techniques produce clusters of process fragments that are similar or very different compared to those produced by end users.

A suitable measure to answer this question is the adjusted Rand index [30]. This measure, which ranges from -1 to 1, captures the similarity between sets of clusters, and is commonly used to measure clustering accuracy. For each technique, we computed the Rand index between clusters identified by participants, and between clusters identified by each technique. Finally, we compared the results: i) participants’ clustering with DBSCAN versus participants’ clustering; ii) participants’ clustering with HAC versus participants’ clustering; iii) DBSCAN
clustering versus HAC clustering. Table 9 summarizes the results. For both experimental groups, the algorithmic clustering provided increased accuracy when compared to manual clustering, with the difference being significant for the HAC technique \((p = 0.02)\) but not for DBSCAN \((p = 0.30)\). In the comparison of the similarity of clusterings produced by the two techniques, we found that the Rand index is significantly higher \((p = 0.04)\) for the HAC technique. When interpreting the differences between DBSCAN and HAC directly, however, we note that the techniques perused two different model collections, which biases the Rand index comparison.

Table 9: Rand indices for DBSCAN and HAC in comparison to participants’ clusterings

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Comparison</th>
<th>Rand index (mean)</th>
<th>Rand index (st. dev.)</th>
<th>T-statistic (significance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBSCAN</td>
<td>Participants’ clustering</td>
<td>0.672</td>
<td>0.19</td>
<td>1.07 ((p = 0.30))</td>
</tr>
<tr>
<td></td>
<td>DBSCAN clustering</td>
<td>0.713</td>
<td>0.136</td>
<td></td>
</tr>
<tr>
<td>HAC</td>
<td>Participants’ clustering</td>
<td>0.715</td>
<td>0.211</td>
<td>2.47 ((p = 0.02))</td>
</tr>
<tr>
<td></td>
<td>HAC clustering</td>
<td>0.836</td>
<td>0.18</td>
<td></td>
</tr>
<tr>
<td>Both</td>
<td>DBSCAN clustering</td>
<td>0.713</td>
<td>0.136</td>
<td>2.19 ((p = 0.04))</td>
</tr>
<tr>
<td></td>
<td>HAC clustering</td>
<td>0.836</td>
<td>0.18</td>
<td></td>
</tr>
</tbody>
</table>

The experiment also allows us to examine how well users can identify process fragments that, as per the technique, should or should not be clustered, and which personal factors determine the correct identification of cluster fragments and noise, respectively. To that end, we estimated regression models that examined i) the Rand index between an individual’s clustering in comparison to the DBSCAN (or HAC) technique, and ii) the percentage of correctly identified noise, i.e., fragments that should not be clustered. Table 10 shows descriptive statistics of the distribution of the dependent variables.

Table 10: Descriptive statistics of correct Noise and Correct Clustering indices

<table>
<thead>
<tr>
<th>Metric</th>
<th>Correct noise (percentage)</th>
<th>Correct clustering (Rand)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DBSCAN</td>
<td>HAC</td>
</tr>
<tr>
<td>Mean</td>
<td>0.854</td>
<td>0.861</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>0.067</td>
<td>0.187</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.667</td>
<td>0.444</td>
</tr>
<tr>
<td>Maximum</td>
<td>0.889</td>
<td>1.000</td>
</tr>
</tbody>
</table>

In estimating the regression models, we considered the following variables as independent factors:
• the total score of process modeling competency (from 0-5) as per [24],
• the average total factor score for EPC familiarity [25],
• the process modeling experience in years,
• the number of days of training with EPC models within the last year, and
• the number of EPC models created or read within the last year.

The estimated linear regression models with the dependent variable Rand (technique versus participant’s clustering) showed that none of these factors was a significant determinant of the cluster similarity measure. In the interest of brevity we omit the detailed description of the coefficient weights and loadings. The overall regression models showed insignificant fit to the data for both DBSCAN (F = 1.84, p = 0.20) and HAC (F = 1.16, p = 0.40), indicating that manually producing clusters similar to the two techniques is not dependent on expertise or experience with process modeling.

In terms of correctly identifying noise, however, we found the two regression models to show significant determinants. Table 11 summarizes the results. The results show that correct noise identification was explained for 49% through modeling expertise and 22% through modeling experience factors. Notably, the overall process modeling experience was a significant positive contributor to the correct identification of clustering noise (p = 0.03 and 0.04), while EPCs training was a significant negative contributor (in that participants with more training days performed worse in terms of noise identification). Knowledge of modeling concepts appears to be a positive factor, with one out two beta weights being significant (p = 0.01 and p = 0.20). These results can be interpreted as suggesting that noise identification, at least in part, is a function of expertise and experience, and thus that algorithmic support is particularly beneficial in situations where such expertise or experience cannot be provided by end users.

7. Threats to validity

As any other evaluation, our evaluations of the two techniques are susceptible to threats to validity. In discussing these, we focus on those threats of [31] (p. 67) that are most relevant to the types of evaluations we chose.

At the forefront, the type of technology validations performed in this paper were focused on evaluating capability of the algorithms, leaving aside context-based concerns, such as the applicability across different model repositories from
Table 11: Results from regression models for correct noise identification

<table>
<thead>
<tr>
<th>Independent factor</th>
<th>Correct noise identification (DBSCAN)</th>
<th>Correct noise identification (HAC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>St. Beta</td>
<td>T (Sig.)</td>
</tr>
<tr>
<td>Proc. modeling know. score</td>
<td>0.76</td>
<td>3.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPCs familiarity</td>
<td>-0.33</td>
<td>-0.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Experience in years</td>
<td>0.65</td>
<td>2.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPC models created or read</td>
<td>-0.08</td>
<td>-0.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPCs training days</td>
<td>-0.5</td>
<td>-2.52</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPC Experience (months)</td>
<td>-0.18</td>
<td>-0.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>0.69</td>
<td></td>
</tr>
<tr>
<td>Adjusted R2</td>
<td>0.49</td>
<td></td>
</tr>
</tbody>
</table>

different industries, the usability and usefulness to different user groups, or the feasibility of subsequent actions such as model merging or process standardization. In turn, we deliberately designed our evaluations as tests of internal validity (do the techniques work in principle?) and conclusion validity (how much confidence do we have in observed performance impacts?) over tests of ecological or external validity. Our reasoning was that for our novel techniques we first needed to demonstrate support for the design theory (i.e. artifact produces desired effects) rather than the implied value theory (i.e. effects of artifacts provide value to stakeholders) [32]. We do note, however, in accordance to technology validation guidelines [32], that future testing will still be required and that such tests should focus on “street credibility” to support analogy, generalizability and wider performance claims.

Conclusion validity is concerned with the relationship between treatment and outcome, and the conclusions drawn from it. Two aspects have to be considered: the appropriateness of tests and the reported effect sizes. As reported above, for all our tests we relied on standard tests and screened our data for conformance with the assumptions of the tests we used (e.g., the regression model). We also perused, where possible, well-established measures such as F-Score, ROCs and the Rand index. Concerning effect sizes, we examined and reported F-statistics to examine the fit of the statistical model to the data where this was appropriate (see Section 6).

Internal validity demands that the treatment causes the effect that reported result differences originate from the different clustering techniques and not from personal differences. To mitigate potential threats to internal validity, first we explicitly examined algorithm clustering versus participants clustering as part of the evaluation (see Table 9). Second, we also collected established measures for process modeling expertise [24], familiarity [25] and experience [26, 27, 28] and
examined the relevance of these demographic factors to the results specifically in a regression model as reported in Section 6. Finally, in the user evaluations we included several clusters produced by each technique varied by different benefit-cost-ratios, to explore whether variations could also be caused by “better” or “worse” clusters (see Table 8).

Construct validity can be related to potential interactions between the measures used. From the outset, this threat has been mitigated by the choice of multiple tests of performance, accuracy and user evaluations. In each of the tests conducted we also reported on relevant descriptive statistics to allow readers to inspect measure correlations. Specifically, in the second user evaluation we did find some unexpected correlations as well as some non-existing effects that we expected, from the results of the regression analysis (see Table 11). For instance, we initially expected that familiarity with EPC diagrams would be a significant predictor for correct noise identification but it was insignificant, whereas experience in years was. In hindsight, this result is actually comparable to other studies that showed that self-reported familiarity actually is not a good measure of experience and tends not to correlate with performance metrics well (see e.g. [33, 29]. The second unexpected interaction was between EPC training and noise identification, which was negative. We do not have data available to inspect this result in more detail nor to elucidate potential root causes, therefore we classify this result as requiring future research attention.

Last, one specific threat to construct validity in the second user experiment (see Section 6.2) was that two types of bias may stem from the use of two collections of fragments (by DBSCAN and HAC, respectively):

1. Participants may perform better in the second task because of a learning effect. To examine this bias we checked (a) task completion times for both tasks, and (b) whether the noise identification scores differed significantly. Paired sample t-tests showed that there was a learning effect in task completion time but not in task performance. Participants completed the 2nd task, on average, 3.6 minutes faster ($t = 2.33$, $p = 0.03$) but not better (the difference in noise identification was 0.07%, $t = 0.19$, $p = 0.85$). Thus, the bias is unlikely to influence the performance results.

2. The collection characteristics may be significantly different, in turn changing the results. To mitigate this bias, we imposed several identical requirements to fragment selection in the design of the study (e.g. similar fragments size, overall number of fragments and clusters size). While we cannot guar-
antee that we eliminated all bias, the results shown in Table 10 show relatively similar user performance across all metrics (Mean, Std. Dev., Min. and Max.).

Finally, external validity concerns the transferability of the findings beyond the boundaries of the tests performed. In our case, the use of multiple tests with multiple measures, objective data from model collections as well as user evaluations strengthens the conclusions drawn about the utility of the two developed techniques. External validity was also increased by the use of fragments of real-world models from industry model collections rather than artificial models. The user evaluation as a specific test is susceptible to participant bias because of the involvement of students. We note that some of the students possessed prior practical experience with process modeling. Also, prior research found that students tend to have higher theoretical knowledge, in turn making them more appropriate proxies for modeling experts [34]. Of course, this may also be seen as a limitation of this research, as the population in our study is potentially more knowledgeable of formal aspects of process modeling theory than the wider population. Further arguments for the use of students are: i) the ability to relate findings to other experimental studies in process modeling, most of which rely on student samples (e.g., [24, 29, 28]), ii) the forthcoming employment of process modeling students as business and systems analysts of the future, which makes our results predictive of the future of process analysis, and iii) the absence of bias stemming from knowledge or experience levels that are too high or too close to the experimental material (e.g. the model fragments from our industry model collections). Thus, the student sample allowed us to evaluate the techniques in the absence of domain knowledge. However, we do caution the reader that these results do not necessarily imply that similar results emerge for users with high levels of domain or modeling knowledge. As we state above, this element of external validity was compromised in our study due to the desire of reaching internal and conclusion validity and we will welcome further studies on process standardization in dependence of domain knowledge, semantic similarity or indeed different supporting technologies and techniques.

8. Related Work

We analyzed work in the following related areas: i) code clone detection, ii) model clone detection, iii) process model refactoring, iv) configurable process modeling and variability modeling, and v) frequent subgraph mining.
8.1. Code Clone Detection

Clone detection in software repositories is an active field of research [1, 2, 5]. According to [35, 2, 5], approaches in this field can be classified into: text comparison, token comparison, metrics-based comparison, Abstract Syntax Tree (AST) comparison (or more generally tree-based comparison), and Program Dependence Graphs (PDG) comparison (or more generally graph-based comparison). Approaches for approximate clone detection can be found across all these categories. For example, approximate clone detection based on text comparison is supported by the NICAD [36], while CCFinder [37] adopts a more token-based comparison for approximate clone detection.

Naturally, the methods closer to the scope of this article are the tree-based and graph-based ones. Baxter et al. [38] describe a method for clone detection based on ASTs. The method applies a hash function to subtrees of the AST in order to distribute subtrees across buckets. Subtrees in the same bucket are compared by testing for tree isomorphism. Another representative technique for tree-based comparison is Deckard [39], which addresses scalability issues by extracting a characteristic vector that approximates the AST in an Euclidian space and then applies locality-based hashing to build clusters of similar vectors. The scope of these latter techniques differs from ours in that RPSTs are not perfect trees. Instead, RPSTs contain rigid components that are irreducible and need to be treated as subgraphs—thus for example tree isomorphism is not directly applicable.

An extension of Deckard to deal with PDGs has been proposed in [40]. The idea here is to first extract a set of significant subgraphs that are likely to hold (approximate) clone candidates. From each such subgraph, a forest of ASTs is then generated. The Deckard approach is then applied in order to identify groups of clones that are then filtered to remove redundant output. The technique relies on the specific semantics of PDGs, in particular the notion of slicing which allows to essentially extract sub-computations from a procedure. It is thus not directly applicable to process models. Arguably, an adaptation could potentially be made to process models, however, this would not necessarily lead to the identification of fragments that modelers would perceive to be “similar”. In contrast, our proposed techniques rely on a notion of similarity that has empirically been validated as reflecting perceived process similarity by process modelers [3]. On the other hand, the technique in [40] is designed to be highly scalable, and thus could be adapted in settings where tens or hundreds of thousands of models were involved. Another representative technique for detecting (exact) clones in PDGs is presented in [41]. This approach relies on a heuristic to approximate the set of maximal isomorphic graphs of a graph and is geared specifically to maximal exact clone detection.
A more sophisticated technique that detects approximate clones in PDGs using approximate subgraph isomorphism detection is GPLAG [42].

8.2. Model Clone Detection

In the field of model-driven engineering, approximate clone detection has been investigated in [43], [6], [44] and [45]. In [43] the authors present CloneDetective, a method for detecting clones in large repositories of Simulink/TargetLink models from the automotive industry. Models are partitioned into connected components which are compared pairwise using a heuristic subgraph matching algorithm. These pairs are then clustered based on the sets of their node labels. According to [6], CloneDetective suffers from low accuracy and low degree of completeness in detection, mainly due to the fact that small clones are absorbed by larger clone pairs. In other words, the algorithm tends to find as large clones as possible, whereas in our approach we allow related fragments to belong to different clusters, so that users can choose the abstraction level at which to standardize. Moreover, this method is not very sensitive to approximate clones having small differences. These cases commonly result from copy-pasting and as such they should not be discarded. Moreover, they yield low standardization costs making them easy to standardize. The work in [6] overcomes these problems by proposing two methods for exact and approximate matching of clones. In particular, the second method, namely aScan, represents graphs by a set of vectors built from graph features: e.g. path lengths and node in/out degrees. An empirical study shows that this feature-based approximate matching improves pre-processing and running times with respect to a brute-force approach. Despite these advantages, the method proposed in [6] does not fulfill our requirements: The resulting clones may be non-SESE fragments and the identified clusters do not satisfy any of the properties in Definition 8. The work in [44] detects clones in UML models, such as class or activity diagrams. In this work, each object, its properties and child objects (all called elements) form a fragment. The similarity between two fragments is computed by summing up the pairwise similarities of their respective elements. This method is not suitable for our purposes as it does not consider structural similarity, fragments are fixed to specific structures, and no clustering technique is proposed.

Another approach to detect approximate clones, specifically in Simulink models is proposed in [45]. The idea of this latter technique is to transform the graph-based models to normalized text form, and to then apply the NICAD text-based technique for near-clone detection. This technique could be transposed to process models subject to designing a suitable normalized text representation of process
models that would somehow preserve approximate clones. Simulink already provides a textual representation that turns out to be suitable for this purpose.

8.3. Process Model Refactoring

Refactoring business process models has been investigated in [3, 46]. In [3], pairs of similar process fragments are identified and given as input to the user. In contrast to our work, fragment similarity is exclusively based on label similarity rather than a combination of label and structural similarity. Also, fragments are considered pairwise (no clustering is performed). In [46], 11 process model refactoring techniques are identified and evaluated. Refactoring process fragments as subprocesses is one of the techniques discussed, but no tool support to identify refactoring opportunities is provided.

In this article, we leave aside the issue of how to standardize a cluster of fragments into a single one and the subsequent refactoring step (i.e. extracting the standardized fragment as a separate subprocess). This question touches upon issues of business process harmonization and consolidation, discussed for example in [4, 47, 12], which are outside the scope of this paper.

8.4. Configurable Process Modeling and Variability

Another body of related work is that on configurable process models [48, 49], which allow modelers to represent multiple variants of a given process in a single model. When standardizing a cluster of clones as a single fragment, it would be an option to represent the standardized fragment using a configurable process modeling notation, in such a way as to keep track of variations across the original clones. This having been said, the choice of representation of standardized fragments is orthogonal to the contribution of this article and outside its scope.

In [50] an approach is described to synthesize a representative process model from a collection of variants. This work does not seek to detect approximate clones. Instead it assumes that a set of similar models or fragments is given as input. The approach of [50] could be used after clone clustering in order to synthesize the centroid of a cluster (as opposed to a medoid as in DBSCAN). Whereas the medoid of a cluster of fragments is one of the fragments in the cluster, the centroid can be (and often is) a fragment that does not exist in the cluster. In our experiments, we opted to only show fragments that exist in the input collection of process models, as the insertion of artificially created fragments could potentially create confusion among the subjects in the experiment.
8.5. Frequent Subgraph Mining

Another vein of work that can be seen close to our setting is that of Frequent Subgraph Mining [51]. The goal of FSM is to find all subgraphs that appear frequently in a collection of graphs, according to a given frequency threshold. Generally speaking, FSM consists of two steps: generating non-duplicate candidate subgraphs and computing the frequency of each candidate. Candidate subgraphs are incrementally generated, starting with one vertex (or edge), and adding one vertex (or edge) in each iteration. The elimination of duplicate candidate subgraphs requires checking graph isomorphism. Similarly, the computation of frequency relies on subgraph isomorphism checking. Traditionally, a large body of work in the FSM field has focused on mining “exact” subgraphs. Recently though, some studies have addressed the problem of approximate FSM [52, 53, 54, 55]. In [52], the authors describe a method, namely gApprox, which defines a measure of similarity that tolerates differences on the labelling of vertices and edges. This notion is used when computing the frequency of a candidate subgraph. The notion of similarity used by gApprox is not based on that of graph-edit distance, but one based only in label similarity of the matched vertices/edges. Additionally, the algorithm is designed to work on a single graph and not with a collection of graphs. The applicability of gApprox is compromised by the cost of computing the similarity score between each candidate subgraph and every possible isomorphism. Accordingly, the authors of [54] propose heuristics to reduce the number of isomorphisms to be considered in the computation of subgraph frequency and provide hints on how to apply their technique to collections of graphs. In order to improve the overall performance, subgraph frequency is computed using random sampling and not by exhaustive search. In the same vein, [53] presents another randomized algorithm. This algorithm tolerates discrepancies on the presence of edges, however, it requires vertex/edge labels to match. In [55], an algorithm is introduced, called AGraMi, that discovers exact frequent subgraphs. AGraMi is referred to as an approximate mining algorithm because it tolerates misses (some frequent subgraphs are not returned) as a trade-off to gain in performance.

There exist significative differences in the goals behind FSM and those behind the clone detection techniques presented in this article. First, FSM aims at discovering connected subgraphs regardless of their topology. In our setting, however, process model clones are required to be SESE fragments. And although at some points in the computation, candidate subgraphs generated by FSM algorithms might be SESE fragments, an FSM algorithm would incur overhead in computing intermediate results that would not be useful in our setting. Second, FSM techniques focus on frequency as the selection parameter, whereas the clus-
ters we look for are not always frequent subgraphs. Indeed a cluster with a handful of large clones can be interesting from the perspective of standardization and have a higher BCR than a cluster consisting of many much smaller clones. And reducing the frequency threshold to catch such “small clusters of large clones” has a negative impact on the performance of FSM algorithms.

9. Conclusion

This article presented two techniques for retrieving clusters of approximate clones for possible standardization and refactoring into shared subprocesses: one based on the DBSCAN algorithm, the other based on the HAC algorithm. The techniques follow alternative approaches for identifying the reference fragment towards which other fragments will be standardized. The DBSCAN technique identifies the reference fragment with the cluster medoid; the technique based on HAC proposes any fragment from the cluster as the reference fragment. Accordingly, DBSCAN is more permissive than HAC in determining cluster membership, since this is based on the distance between the medoid and the other fragments (cluster radius). HAC is stricter as it relies on cluster diameter.

Additionally, the article put forward a measure of cluster quality (benefit-to-cost ratio) intended to capture the potential standardizability of the cluster. An experimental evaluation showed that both techniques, coupled with the proposed cluster quality measure, accurately retrieve clusters resulting from copy-pasting activities followed by independent modifications to the copied fragments. Other experiments showed that the proposed techniques produce clusters similar to those produced by human subjects. Finally, it was shown that the proposed techniques produce clusters that human subjects perceive to be amenable for standardization. Hence, it can be concluded that the proposed techniques provide a basis for identifying clusters of approximate clones that are amenable to standardization.

Specifically, the results of the evaluation show that the technique based on DBSCAN is better suited for handling process model collections that are heterogeneous, e.g. those being manipulated by different teams at different points in time and for different purposes, with little synchronization between the teams. Further, this technique can better detect approximate clones that originate from a master fragment which has undergone independent evolutions after copy-pasting. However, this is done at the price of a lower accuracy compared to HAC. The latter technique is less tolerant to inconsistencies in the process model collection and is thus more suited for detecting approximate clones from homogeneous process model collections, e.g. those exhibiting consistent labeling style and graph
structures across the various models. These differences between the two techniques are confirmed by the experiments with human subjects, which show that DBSCAN better matches perceived standardizability than HAC, while HAC has a slight advantage over DBSBAN in terms of noise identification.

This article also contributed a measure of cluster quality, the BCR, for the sake of standardization. This measure is intended to be used for presenting the retrieved clusters in decreasing quality order, given that one would typically be confronted with a large number of clusters to assess. The tests conducted over synthetically generated clusters show a very high ranking accuracy of the BCR, while the experiments with human subjects confirm the usefulness of this measure when assessing the potential effort of clusters standardization. These results highlight the importance of using the BCR to present clusters to business analysts in a decreasing order, from high to low BCR, in order to effectively aid the standardization effort. That said, the BCR only takes into account model characteristics (i.e. fragment size and number of required changes) neglecting other factors that may also contribute to standardization (e.g. the standardization context). Thus, the ranking produced by the BCR should be used to guide the standardization, without assuming that clusters with low BCR should be discarded a priori.

A question that remains open is how a clone cluster should be standardized into a single reference fragment in such a way that the stakeholders involved in the management and execution of the process are satisfied with the standardized process. We started with the hypothesis that the medoid of the cluster could serve as a reference fragment towards which all other fragments could be standardized. This hypothesis was not backed by the experimental results, where subjects almost always designated the largest fragment in the cluster as the reference fragment, thus favoring what could be called “standardization by union”. The question thus opened is whether in practice approximate clones would be standardized towards their union or towards a fragment that is smaller than their union. Further empirical studies are required to shed light on this question. In this respect, we envision empirical studies that would investigate the relation between non-standardized process models and their final versions after undergoing standardization.

Another direction for future work is to improve the scalability of the techniques, for example by further optimizing the computation of the distance matrix. One way to achieve this is by efficiently computing better lower-bounds of the distance between pairs of fragments, so that the need to calculate exact distances is reduced to cases of pairs of fragments that are relatively close to one another.

A further development is to extend the range of approximate clones that can be detected, by taking into account multi-entry multi-exit (MEME) fragments. For
example, a MEME fragment could be used to capture a sub-process for handling insurance claims where the various entry points represent the different submission channels (fax, phone, internet or branch), and the various exit points represent the different outcomes (approved, rejected). In this respect, we could reuse the RPST algorithm in [56], which handles a MEME fragment by converting it into a SESE equivalent via the introduction of an artificial split node before all entry nodes and an artificial join node after all exist nodes.

Finally, we reported on a varied set of analyses and user studies, which we designed and executed with the view of building both lab credibility and street credibility [32]. Together, our analyses provide a significant contribution by allowing a multi-faceted interpretation of the usefulness and usability of the proposed techniques in practice. Thus, we hope that our approach to empirical validation can be used as a reference for technology validation in general.

Acknowledgments NICTA is funded by the Australian Government (Department of Broadband, Communications & the Digital Economy) and the Australian Research Council through via ICT Centre of Excellence program. This work is partly funded by EU Regional Development Funds via the Estonian Centre of Excellence in Computer Science and by Australian Research Council grant LP110100252.

References


