Detecting Approximate Clones in Business Process Models

Marcello La Rosa[*,a,b], Marlon Dumas[c], Chathura C. Ekanayake[a], Luciano García-Bañuelos[c], Jan Recker[a], Arthur H.M. ter Hofstede[a]

[a] *Queensland University of Technology, Australia*
[b] *NICTA Queensland Lab, Australia*
[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, standardization.

[*]Corresponding author
Email addresses: m.larosa@qut.edu.au (Marcello La Rosa), marlon.dumas@ut.ee (Marlon Dumas), chathura.ekanayake@student.qut.edu.au (Chathura C. Ekanayake), luciano.garcia@ut.ee (Luciano García-Bañuelos), j.recker@qut.edu.au (Jan Recker), a.terhofstede@qut.edu.au (Arthur H.M. ter Hofstede)

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, i.e. replaced by a single clone instance, 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. Moreover, standardized processes are associated with increased process performance [4, 5, 6].

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, 7]. 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:

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

- Type-2: Syntactically identical fragments except for 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* [8] 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.
- Type-4: Behaviorally equivalent fragments with syntactic differences (e.g. fragments with different combinations of gateways but same set of traces).

In previous work, we proposed a technique for identifying Type-1 (exact) clones in process models [9]. 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.[1]

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.

The proposed techniques and standardizability measure are evaluated in a three-pronged manner. First, we report on runtime performance and characteristics of the clusters obtained by applying the two clustering techniques on two datasets from practice. Second, using a synthetic dataset, we evaluate the accuracy of the two techniques given the task of retrieving groups of clones that emanate from a single original fragment. Third, we report two experiments 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.

[1] Type-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 paper is organized as follows. Section 2 defines and justifies the notion of approximate clone adopted in this paper 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, while Section 5 describes their implementation in the Apromore process model repository system [10]. Section 6 presents the results of the evaluation. Finally, Section 7 frames the contributions in relation to the literature while Section 8 concludes and discusses limitations and possible extensions of the research.

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 and that has been shown to be correlated with perceived similarity [11]. This measure is defined over an abstract representation of process models based on labelled graphs, as follows.

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

Note that this notion of process graph is able to capture not only control-flow elements of process models (e.g. tasks, events and gateways), but also data objects attached to tasks or events. It is also possible to represent resource pools (roles) as nodes that are attached to tasks via edges. The adopted representation does not restrict the types of nodes that appear in the process model.

The adopted similarity measure is based on the well-known graph-edit distance [12]. 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: vertex substitution, vertex insertion/deletion and edge insertion/deletion. A vertex substitution refers to the fact that a vertex in one of the graphs is mapped to a vertex in the other graph. To define a valid vertex substitution, we require a notion of vertex similarity. In this respect, we consider that vertices are matched according to their label similarity measured in terms of string-edit distance, denoted as $Sim_{led}(label_1, label_2)$.[2] A vertex substitution is only allowed if the similarity between their labels is above a user-defined threshold (e.g. 0.4). Whenever a vertex in a graph is not matched to any vertex in the other graph, it is considered as either inserted in one graph or deleted in the other one. Similarly, an edge insertion (or deletion) operation is required for each edge that cannot be mapped to an edge in the other graph. This intuition is formalized as follows.

Definition 2 (Normalized process graph edit distance [11]) *Let $H_1 = (V_1, E_1, \lambda_1)$ and $H_2 = (V_2, E_2, \lambda_2)$ be two process graphs. Let $M : V_1 \nrightarrow V_2$ be a partial injective mapping that maps vertices of H_1 to vertices of H_2. Moreover, let $subv$ be the set of substituted vertices, i.e., $\forall v \in subv : v \in dom(M) \cup cod(M)$, $skipv$ the set of skipped vertices, i.e., $\forall v \in skipv : v \notin dom(M) \cup cod(M)$, and $skipe$ the set of skipped edges, i.e., $\forall (v, w) \in skipe : v \notin dom(M) \cup cod(M) \vee w \notin dom(M) \cup cod(M)$. The* normalized graph edit distance *induced by mapping M is:*

$$Dist^{M}_{GED}(H_1, H_2) = average(fskipv, fskipe, fsubn) \quad (1)$$

where $fskipv$ is the fraction of skipped vertices, $fskipe$ the fraction of skipped edges, and $fsubv$ the average distance between substituted vertices, i.e. $fskipv = \frac{|skipv|}{|V_1|+|V_2|}$, $fskipe = \frac{|skipe|}{|E_1|+|E_2|}m$ and $fsubv = \frac{2 \cdot \sum_{(v,w) \in M} 1 - Sim_{led}(\lambda_1(v), \lambda_2(w))}{|E_1|+|E_2|}$, where $Dist_{led}$ is the string-edit distance between vertex labels.[3]

Finally, the normalized graph-edit distance between H_1 and H_2, written $Dist_{GED}(H_1, H_2)$, is the smallest $Dist^{M}_{GED}(H_1, H2)$ across all mappings M.

A $Dist_{GED}$ of 0 means that the process graphs are identical, while a $Dist_{GED}$ 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

[2]Other measures of label similarity (e.g. semantic ones) can be used as discussed in [11].

[3]The rationale for the factor of 2 in the definition of $fsubv$ is so that replacement of two nodes with completely different labels (i.e. $Dist_{led}(\lambda_1(v), \lambda_2(w)) = 0$) is equivalent to deletion of v and insertion of w.

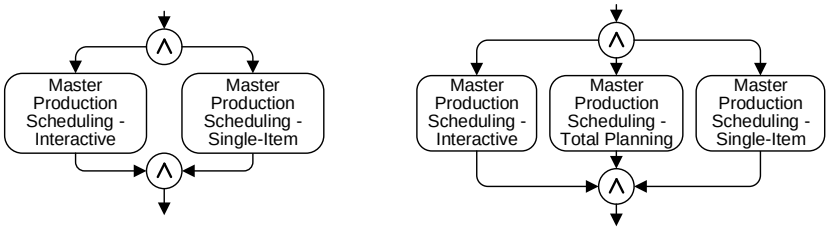

Figure 1: Sample pair of fragments.

in one fragment and not in the other, therefore $skipv = 1$. Two edges have been added thus $skipe = 2$. There is no replacement, thus $subn = fsubn = 0$. Hence, $Dist_{GED} = avg(1/9, 2/6, 0) \sim 0.15$.

The problem of computing the graph-edit distance is NP-Complete [12]. In this paper, we adopt a greedy heuristic described in [11]. 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.

2.2. Notion of Approximate Clone

Given the measure of similarity 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 [8]. 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 3 (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 4 (Approximate Clone Pair) *Given a distance metric $Dist$ and a distance threshold τ, two non-trivial, SESE process fragments g_1 and g_2 are approximate clones – written $Approx(g_1, g_2)$ – iff $g_1 \not\subset g_2$, $g_2 \not\subset g_1$ and $Dist(g_1, g_2) \leq \tau$.*

Armed with this latter 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 5 (Approximate Clone Cluster) *A set of SESE process model fragments C is a* cluster of approximate clones *iff one of the following properties holds:*

[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.

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

2. $\forall g, g' \in C : Approx(g, g')$.

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. 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 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 [6].

Accordingly, we use the absolute GED ($Dist_{AGED}(H_1, H_2)$) defined in the same way as $Dist_{GED}(H_1, H_2)$ in Definition 2 but replacing $fskipv$ and $fskipe$ with $|skipv|$, $|skipe|$ respectively, and removing the denominator in the definition of $fsubv$. In other words, we count actual number of edit operations as opposed to fraction of edit operations relative to total size. We do not used the normalized GED in this context ($Dist_{GED}$), because this normalized version is not reflective of the number of operations required to standardize the fragments. Instead, $Dist_{GED}$ 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. $\sum_{f \in C} Dist_{AGED}(f, m)$. 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, 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 is defined as $BCR(C) = \frac{\sum_{f \in C} |f| - |m| - |C|}{\sum_{f \in C} Dist_{AGED}(f,m)}$. 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.

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 Refine Process Structure Tree (RPST) [13] 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 to disjoint sets of edges. As the process graph is partitioned into 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 [13].

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. For example fragment B1 is a bond. This bond 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.

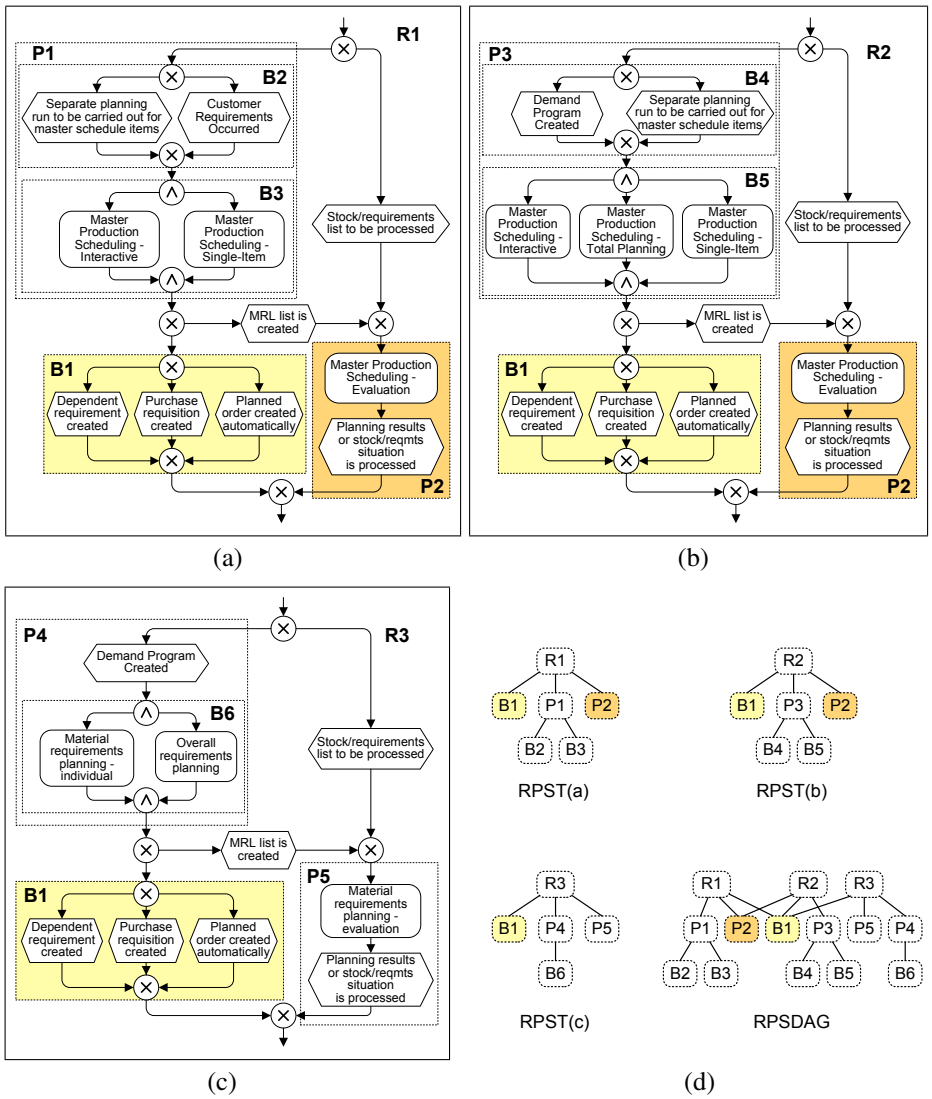

Figure 2: Sample process model fragments, their decomposition into RPSTs and corresponding RPSDAG. SESE fragments are delimited by dashed rectangles and labelled "R" for rigid, "B" for bond and "P" for polygon followed by a number. Highlighted boxes represent exact clones.

Figures 2(a)–(c) present sample process fragments extracted from models in the SAP Reference Model [14].[5] 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.

3.2. RPSDAG

The RPSDAG [9] 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.

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 [9] 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 ver-

[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.

sions 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)* [15] for the standardization approach A1, and the *Hierarchical Agglomerate Clustering (HAC)* [15] 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 NxN such that $Dist_{GED}(s, p)$ is the graph-edit distance between fragment s and p. As an optimization and given the cost of calculating $Dist_{GED}$ for two process graphs (cf. Section 2.1), the matrix only stores the distance $Dist_{GED}$ of Definition 2 for a pair of fragments if this is within the approximate clone threshold τ of Definition 4, and if the two fragments do not contain one another. For all other fragment pairs, it stores ∞.

As a further optimization, we first calculate a lower-bound of the GED. When this lower-bound is above threshold τ (cf. Definition 4), we do not need to compute $Dist_{GED}$ but instead store ∞. 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_1 is larger than H_2 (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, $fskipv = \left|\frac{|V_1|-|V_2|}{|V_1|+|V_2|}\right|$, $fskipe = \left|\frac{|E_1|-|E_2|}{|E_1|+|E_2|}\right|$ and $fsubv = 0$. 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 $Dist_{GED}(H_1, H_2)$ is greater than the one obtained by feeding the above lower-bound values of $fskipv$, $fskipe$ and $fsubv$ into the equation for $Dist^M_{GED}(H_1, H2)$ in Definition 2. Note that if the graphs have equal size, the obtained lower-bound is zero – which is not useful.

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 ϵ 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 used-defined distance threshold τ, 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 $Dist_{GED}$ 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 τ (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.

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 $N_f \setminus \{f\}$.
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 $p \in C \cup N_m$ $Dist_{GED}(s, p) \leq \tau$
11 Remove N_m from G and $Noise$ and add N_m to C and $N_m \setminus \{m\}$ to M_C.
12 Else, mark m as "unprocessed" and add it to G.
13 If $M_C \neq \varnothing$ go to 7.
14 If $G \neq \varnothing$ go to 2.

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. 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 its transitive closure on the RPS-DAG, 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.

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 $O(|G|^2)$ [15]. That said, the search space is greatly reduced by the cutoff conditions used when computing the distance of clusters, i.e. the distance threshold τ 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 pair of fragments in a cluster has a distance below the threshold τ. This goal can be straightforwardly mapped to the strategy followed by the basic hierarchical agglomerative clustering method [15]. This clustering method starts with singleton clusters and iteratively combine the pair of clusters that is found to be the closest among all other 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: taking the smallest distance between fragments in one of the clusters to the fragments in the other one, known as *single link*; taking the farthest distance, referred to as *complete link*; among others. It can be easily see that the complete link strategy suits well to 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 τ. Note that the identification of such situation can be accomplished ahead of time. The intuition is captured in the following definition.

Definition 6 (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 τ be the similarity threshold among fragments of C_i and fragments of C_j. Moreover, let $\mathcal{F}(C)$ be a function that returns the set of fragments associated to 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 $\mathcal{F}(C) = \{f\}$; (STEP)*

if C is an intermediate node then $\mathcal{F}(C) = \cup_{c \in C} \mathcal{F}(c)$. The distance of clusters C_i and C_j, denoted as $Dist(C_i, C_j)$, can defined as follows.

$$\begin{cases} \infty & \textit{if } \exists f \in \mathcal{F}(C_i), g \in \mathcal{F}(C_j) : g \subseteq f \vee f \subseteq g \\ \infty & \textit{if } max_{f \in \mathcal{F}(C_i), g \in \mathcal{F}(C_j)} \, Dist_{GED}(f, g) > \tau \\ max_{f \in \mathcal{F}(C_i), g \in \mathcal{F}(C_j)} \, Dist_{GED}(f, g) & \textit{otherwise} \end{cases}$$

We note that the distance of two clusters is set to ∞ when there exist one fragment in the first cluster which is in containment relationship with another fragment in the second cluster. Moreover, when farthest distance between fragments of both clusters is above the threshold τ, the distance is set to ∞. In the two previous cases, we are meeting the constraints described in Definitions 4 and 5. Finally, the farthest distance between fragments of both clusters is reported as the distance of the clusters, only when the value is less or equal to the threshold τ. 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, viz. $TopClusters$.

1. For each $f \in G$ create a singleton cluster. Initialize $TopClusters$ to contain all singleton clusters.
2. Using the distance matrix between fragments, calculate the initial distance matrix between clusters in $TopClusters$, i.e. $\mathcal{D}[i, j] \leftarrow Dist(C_i, Cj)$, where $C_i, C_j \in TopClusters$.
3. In the distance matrix $\mathcal{D}$, select a pair of clusters $C_i, C_j \in TopClusters$ such that their distance is the minimum. Stop if no such pair exists, i.e. either all distances in $\mathcal{D}$ are ∞ or $|TopClusters| = 1$.
4. Combine clusters C_i and C_j to form a new cluster C_{ij}. Remove clusters C_i and C_j from $TopClusters$. Add cluster C_{ij} to $TopClusters$.
5. Update matrix $\mathcal{D}$ by adding the distance between cluster C_{ij} and all other clusters in $TopClusters$.
6. Go to 3.

Algorithm 2 can be divided into two parts. Step 1 and 2, initialize the set of singleton clusters, stores them in $TopClusters$ and initializes the distance matrix between clusters (according to Definition 6). The remaining steps correspond to

the main loop. In Step 3, a pair of clusters is selected such that their distance is found to be the smallest among all other possible pairs. If the distance of such pair is ∞ or there is only one cluster left then the algorithm stops. In Step 4, a new cluster is created to hold the union the previously selected pair. In Step 5, the distance matrix is updated (according to Definition 6), by removing the pair 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 $|TopClusters| = 1$ or before if the entire distance matrix $\mathcal{D}$ is filled with ∞.

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 $O(|G|^3)$ [15]. 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.

5. Implementation

In order to offer concrete support for process standardization initiatives, we developed a tool based on the proposed technique that allows analysts to identify, cluster, analyze and visualize approximate clones. The tool is a plugin of the Apromore advanced process model repository [10].[6] Apromore uses an internal process representation format named *canonical process format*, which captures common features of widely-used process modeling languages. The clone detection plugin operates on this canonical format, thus it can detect approximate clones in process models defined in different modeling languages such as BPMN and EPC for example.

The Web interface of the approximate clone detection plugin (shown in Fig. 3) provides features for creating, browsing and visualizing fragment clusters. Users can select one or more process models from the repository, specify the clustering parameters (such as the preferred clustering algorithm), and kick off the clustering. Once the fragments included in the selected process models have been clustered, users can apply different filtering criteria (i.e. on the size of the clusters, on the average size of fragments, and on the BCR) and browse the resulting clusters

[6] Available at `www.apromore.org`

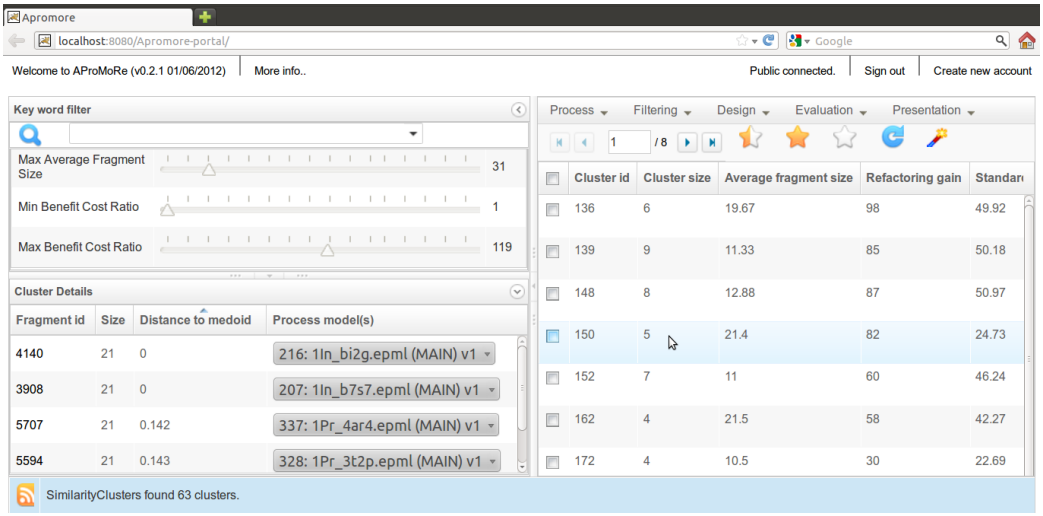

Figure 3: Web interface of the approximate clone detection plugin in Apromore

in a detailed list view. Another useful feature is the visualization of clusters in the 2D space. The visualization component (shown in Fig. 4) displays each fragment in a cluster as a point in the space and positions fragments within a cluster according to their distances to the medoid (distances being represented as edges between the points). It also positions the clusters in the space according to the GEDs among their medoids. One can also click on the point corresponding to a process fragment to visualize its corresponding model using any process modeling language supported by Apromore (e.g. EPCs, BPMN).

Under the hoods, the approximate clone detection plugin relies on three techniques that have also been integrated into Apromore: i) RPST, ii) RPSDAG and iii) graph-edit distance. In particular, the RPST implementation available in Apromore is that distributed with the jBPT library.[7]. This implementation can also detect multi-entry-multi-exit (MEME) fragments. This is achieved by adding a fictitious split node before all entry points and a fictitious join node after all exit points, to create a SESE fragment out of a MEME fragment. These nodes are removed once the fragment has been processed. An example of a MEME fragment identified as approximate clone is the one shown in Fig. 4 using the EPC language.

The approximate clone detection functionality is exposed via the Web ser-

[7] https://code.google.com/p/jbpt/

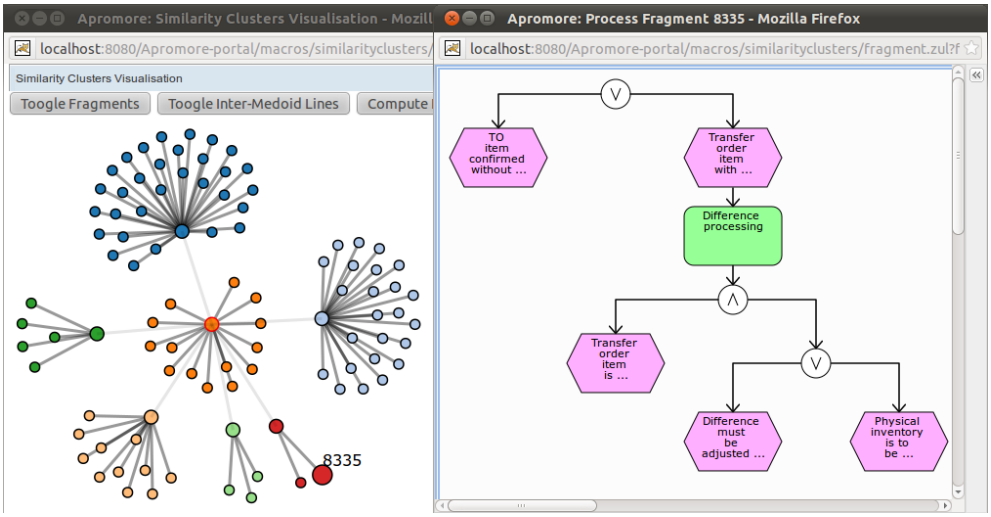

Figure 4: Cluster visualization component of the approximate clone detection plugin in Apromore

vice API of Apromore. Thus, remote applications can programmatically invoke approximate clone detection on process models available outside Apromore and filter and browse the identified clones, by using this API.

6. Evaluation

On the basis of the implementation of the two algorithms in Apromore, we now report on three evaluations of the techniques. First, we use 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 6.1). Second, we use 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 6.2). Finally, we conduct 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 (Section 6.3).

6.1. Runtime performance and clusters characteristics

We first used the two clustering techniques to examine the occurrence of approximate clones in practice and evaluate the runtime performance and the characteristics of the clusters retrieved. For this purpose we looked at two industry-size datasets. The first dataset is the SAP R/3 reference model [14]. 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 first computed the RPSDAG for both datasets and post-processed them by factoring out all exact clones using the technique presented in [9]. This yielded 2,238 non-trivial fragments with at least 4 nodes for the SAP dataset (11.47 average size) and 2,037 for the insurance dataset (16.58 average size). 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.

All 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 5 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.

Figure 6 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). In general, we observe that none of the

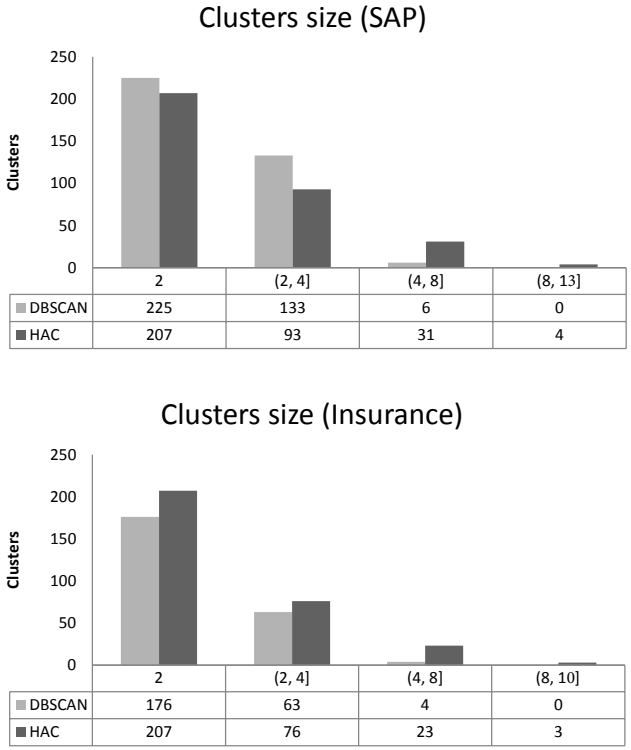

Figure 5: Number of clusters vs clusters size for both techniques.

techniques is better than the other, since for the SAP dataset we achieve higher BCRs for HAC than for DBSCAN, whilst for the insurance dataset it is the other way around. This suggests that depending on the type of the repository, one of the two techniques might be more appropriate than the other.

6.2. 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. We randomly selected 50 such master fragments from the two industry-size datasets used in the occurrence analysis, such 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

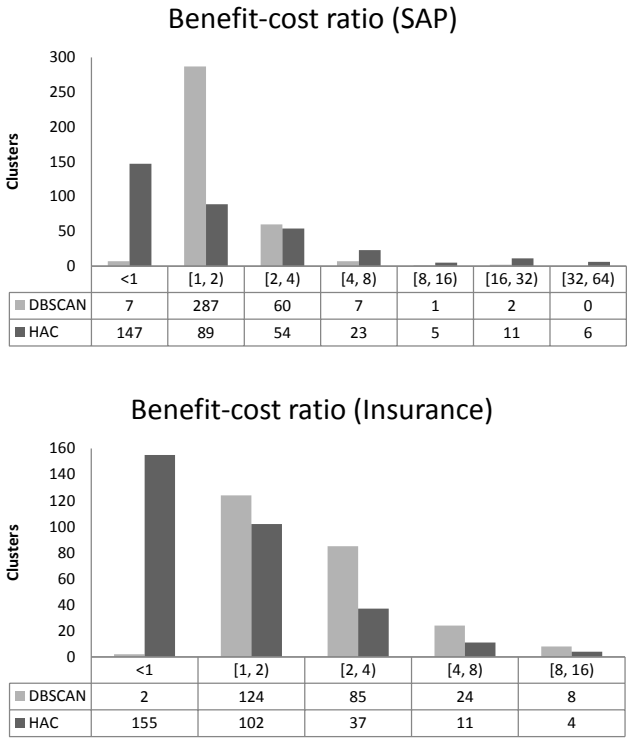

Figure 6: Number of clusters vs benefit/cost ratio for both techniques.

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) such that the graph-edit distance between a variant and its seed was no more than 40% – the same threshold that we used in the occurrence analysis.[8] The clusters' size 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 fragment that yielded the maximum *FScore* [17]. FScore is

[8]The idea of using random mutation of seed fragments to generate synthetic data for evaluating clone detection methods is also used in [16] in the context of source code clone detection.

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 FScore* (F_{wa}) [17]. F_{wa} is the maximum FScore 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 1: Quality metrics for both algorithms.

		DBSCAN	**HAC**
Recall	*min*	0.17	0.1
	max	1	1
	avg	0.71	0.82
	std	0.37	0.25
Precision	*min*	0.2	0.17
	max	1	1
	avg	0.89	0.84
	std	0.24	0.33
Fwa		0.73	0.77

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 1. 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.

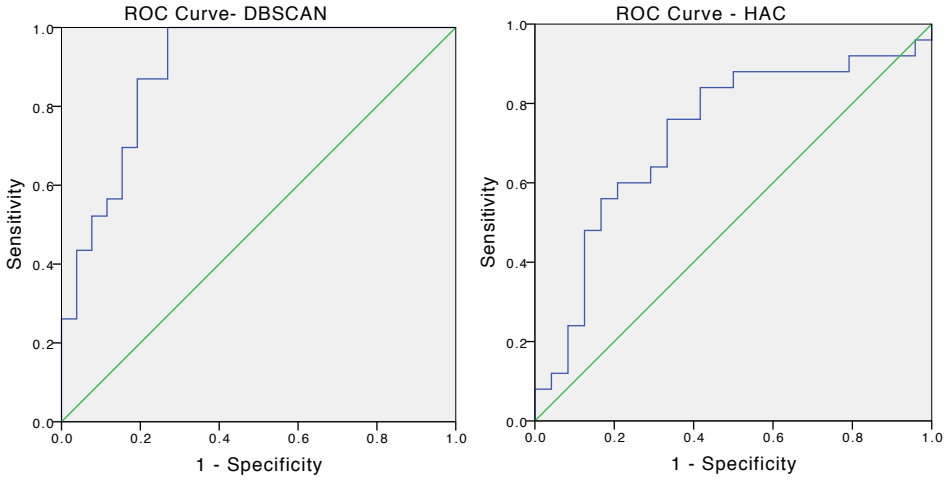

Figure 7: ROC curves for both algorithms.

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. 7, 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).

6.3. Perceived standardizability and correctness

Design and Measures

Finally, we designed two between-groups experiments in which users were presented with process fragments identified through either the DBSCAN or the HAC technique, and were asked to complete a set of standardization tasks and provide answers to a number of questions.

Experiment 1 concerned the evaluation of clusters produced by the two techniques in terms of their perceived standardizability. Experiment 2 concerned the evaluation of the perceived correctness of the clustering performed by the two techniques in comparison to manual clustering.

Experiment 1:

In the first experiment, overall 73 users participated. 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 a 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 [18, 19, 20].

In the experiment, participants were firstly asked to provide demographic information. Next they were randomly distributed into two groups, with each group being provided with 5 sets of fragments (each set containing 3 to 5 fragments), either identified by means of DBSCAN or HAC. For each technique, the 5 sets of fragments were varied in terms of BCR (from '0-2', '2-4', '4-6', '6-8' and 'above 8'). For each group, participants were asked to rate the standardizability of that group on a 5-item scale measuring similarity, complexity, suitability, readiness and ease of standardizability of the group of fragments. The measurement scales used are shown in Table 2. Scores for each item in the scale were aggregated to an average total factor score for the analysis. Additionally, for each group of fragments participants were asked to select a most suitable standardization strategy from a set of three options:

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.

Table 2: Multi-item measurements used in the experiment

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

Experiment 2:

In the second experiment, overall 16 users participated. Participants were invited from the cohort of PhD research 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. Nine doctoral students participated and 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 used 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. These characteristics describe the pool of participants for the second experiment as considerably more experienced in process modeling than the participants in Experiment 1.

In the experiment, each participant was given two collections of process fragments containing 17 and 18 fragments each. The first fragment collection contained 8 fragments classified into 3 clusters by the DBSCAN algorithm along with

9 fragments classified as noise. The second collection contained 9 fragments classified into 3 clusters by the HAC algorithm along with 9 fragments classified as noise. 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%. Participants were asked to standardize these collections of fragments by grouping relevant fragments together into clusters. Based on this experimental design, we can compare the differences between manual clustering of fragments versus the clusters produced by the algorithms in terms of two measures:

a) the placement of fragments into a cluster, and

b) the identification of noise.

Additionally, users were again asked to provide relevant demographic information in terms of modeling experience, familiarity with the EPC language [21] and their knowledge of important process modeling concepts such as concurrency and repetition [22], similar to Experiment 1.

Analysis and Results

Experiment 1:

On the basis of the experimental data obtained, we can perform a number of evaluations.

First, we examine the perceived standardizability of clusters produced by the two techniques in terms of overall rating and consistency of rating, in relation to the i) algorithm used and ii) the BCR of the identified cluster. Table 3 provides relevant statistics and Fig. 8 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 3, 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 confirm that it is hard to use clusters with low BCR for standardization, while clusters with high BCR can effectively be used for this purpose, as visualized in Fig. 8. We further note that these results are

Table 3: Cluster standardizability ratings

ClusterID	**Technique**	**BCR**	**Standardizability** *Mean*	**Standardizability** *Standard deviation*
D151	DBSCAN	0 - 2	4.05	1.48
D56	DBSCAN	2 - 4	3.97	1.46
D97	DBSCAN	4 - 6	4.95	1.28
D364	DBSCAN	6 - 8	4.48	1.55
D287	DBSCAN	Above 8	5.07	1.23
H260	HAC	0 - 2	3.72	1.43
H177	HAC	2 - 4	4.42	1.42
H106	HAC	4 - 6	4.54	1.25
H83	HAC	6 - 8	4.43	1.41
H55	HAC	Above 8	5.13	1.30

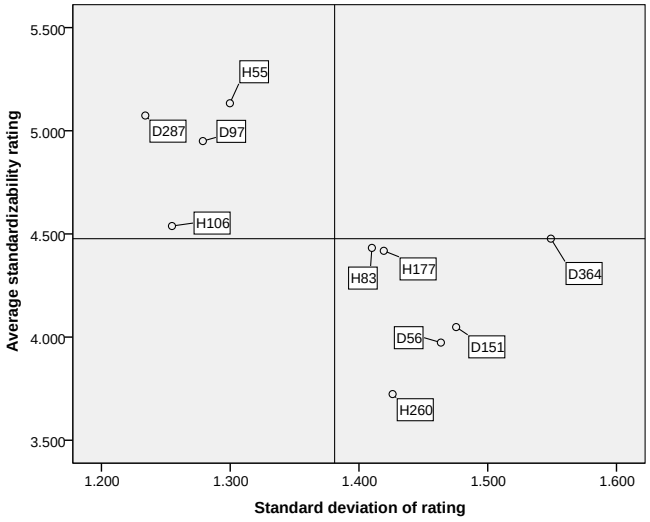

Figure 8: Standardizability rating average and standard deviation for clusters

consistent for both DBSCAN and HAC when BCR is high. These results highlight the importance of filtering out clusters with low BCR and only presenting clusters with high BCR to business analysts, in a decreasing order, from high to low BCR, in order to effectively aid the standardization effort. We can also observe there is no significant difference between the scorings of DBSCAN and HAC when BCR is high (see Table 4). If clusters with low BCR have to be standardized, however, we can observe differences between DBSCAN and HAC (see Table 4), suggesting that clusters generated with HAC might be more appropriate for this purpose due

to the lower standard deviation.

Table 4: Average cluster standardizability rating by BCR and technique

BCR	**Technique**	**N**	**Standardizability** *mean*	**Standardizability** *Standard deviation*
Low	DBSCAN	3	4.32	0.54
(below 6)	HAC	3	4.23	0.44
High	DBSCAN	2	4.78	0.42
(above 6)	HAC	2	4.78	0.5
Total	DBSCAN	5	4.5	0.5
	HAC	5	4.45	0.5

Second, we examined participants' preference for different standardization strategies in dependence to the cluster of fragments received. Table 5 provides information about the preferred standardization ratings per cluster, as reported by overall 32 from the total of 73 participants, an effective response rate of 43.8% (answering was optional).

Table 5: Reported standardization strategy by cluster

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

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

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 algorithm. 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 were looking for a 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.

Experiment 2:

The data collected in the follow-up experiment allows us to examine the performance of the clustering techniques in comparison to manual clustering performed by end users. The relevant question we ask is: "Does the clustering algorithm 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 [23]. 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 and those identified by the participants. Finally, we compared the results: i) participants' clustering with DBSCAN versus participants' clustering; ii) participants' clustering with HAC versus participants' clustering; iii) DBSCAN versus participants' clustering with HAC versus participants' clustering. Table 6 summarizes the results. For both experimental groups, the algorithmic clustering provided increased similarity to manual clustering when compared to similarity between manual clusterings, with the difference being significant for the HAC technique ($p = 0.02$). In the comparison of the similarity of clusterings produced by each of the two techniques and

manual clustering, the Rand index is significantly higher (p = 0.04) for the HAC technique.

Table 6: Rand indices for DBSCAN and HAC in comparison to participants' clusterings

Technique	Comparison	Rand index (mean)	Rand index (st. dev.)	T-statistic (significance)
DBSCAN	Participants' clustering	0.672	0.19	1.07(p = 0.30)
	DBSCAN versus participants	0.713	0.136	
HAC	Participants' clustering	0.715	0.211	2.47 (p = 0.02)
	HAC versus participants	0.836	0.18	
Both	DBSCAN versus participants	0.713	0.136	2.19 (p = 0.04)
	HAC versus participants	0.836	0.18	

The experiment also allows us to examine how well users can identify process fragments that, as per the algorithm, 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 7 shows descriptive statistics of the distribution of the dependent variables.

Table 7: Descriptive statistics of correct Noise and Correct Clustering indices

Metric	**Correct noise (percentage)**		**Correct clustering (Rand)**	
	DBSCAN	*HAC*	*DBSCAN*	*HAC*
Mean	0.854	0.861	0.713	0.836
Std. Dev.	0.067	0.187	0.136	0.18
Minimum	0.667	0.444	0.438	0.349
Maximum	0.889	1	0.826	1

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 [22],
- the average total factor score for EPC familiarity [21],
- 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 algorithms 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 8 summarizes the results. The results show that correct noise identification was explained for 49% through modeling expertise and 22% through 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.

Table 8: Results from regression models for correct noise identification

Independent factor	**Correct noise identification (DBSCAN)**			**Correct noise identification (HAC)**		
	St. Beta	*T*	*(Sig.)*	*St. Beta*	*T*	*(Sig.)*
Proc. modeling know. score	0.76	3.43	**0.01**	0.37	1.37	0.2
EPCs familiarity	-0.33	-0.75	0.48	0.22	0.41	0.69
Experience in years	0.65	2.56	**0.03**	0.77	2.47	**0.04**
EPC models created or read	-0.08	-0.37	0.72	-0.33	-1.26	0.24
EPCs training days	-0.5	-2.52	**0.03**	-0.61	-2.48	**0.04**
EPC Experience (months)	-0.18	-0.37	0.72	-0.45	-0.77	0.46
R2		0.69			0.53	
Adjusted R2		0.49			0.22	

7. Related Work

Clone detection in software repositories is an active field of research [1, 2, 7]. According to [24, 2, 7], approaches in this field can be classified into: text compar-

ison, 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 [25], while CCFinder [26] 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. [27] 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* [28], 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 [29]. 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 [29] 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 [30]. 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 [31].

In the field of model-driven engineering, approximate clone detection has been

investigated in [32], [8], [33] and [34]. In [32] 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 [8], CloneDetective suffers from low inaccuracy 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 [8] 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 vertex in/out degrees. An empirical study shows that this feature-based approximate matching improves pre-processing and running times, while keeping a high precision. Despite these advantages, the method proposed in [8] 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 5. The work in [33] detects clones in UML models, such as class or activity diagrams. In this work, each object, its properties and child objects (all called *model 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 [34]. 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.

Refactoring business process models has been investigated in [3, 35]. 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 [35], 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.

Clustering of process models has been dealt with in [36] and [37]. In both cases process models are clustered rather than process fragments leading to a small number of clusters. Using fragments instead of process models is more complex, but for the purposes of standardization and reuse it is more suitable as a fragment may be shared between process models, while the rest of these models may be quite different.

In [38] 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 [38] 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.

Another body of related work is that on configurable process models [39], 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.

This article is an extended version of our previous conference paper on the subject [40]. The main extensions with respect to the conference paper are the two empirical user evaluations of the proposed techniques (Section 6.3) and the implementation of the proposed technique on the Apromore platform.

8. Conclusion

This article presented two techniques for retrieving clusters of approximate clones for possible standardization and refactoring into shared subprocesses. 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 that are 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 – with the DBSCAN technique being able to better match perceived standardizability than the HAC one.

Hence, it can be concluded that the proposed techniques provide a basis for identifying clusters of approximate clones that are amenable to standardization. However, 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 proposed techniques, for example by 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.

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

References

[1] R. Koschke, Identifying and Removing Software Clones, in: Software Evolution, Springer, 2008.

[2] C. K. Roy, J. R. Cordy, R. Koschke, Comparison and evaluation of code clone detection techniques and tools: A qualitative approach, Sci. Comput. Program. 74 (7) (2009) 470–495.

[3] R. Dijkman, B. Gfeller, J. Küster, H. Völzer, Identifying refactoring opportunities in process model repositories, Information & Software Technology 53 (9) (2011) 937–948.

[4] T. H. Davenport, The coming commoditization of processes, Harvard Business Review 83 (6) (2005) 100–108.

[5] B. Muenstermann, A. Eckhardt, T. Weitzel, The performance impact of business process standardization: An empirical evaluation of the recruitment process, Business Process Management Journal 16 (1) (2010) 29–56.

[6] M. Schäfermeyer, C. Rosenkranz, R. Holten, The impact of business process complexity on business process standardization – an empirical study, Business and Information Systems Engineering 4 (5) (2012) 261–270.

[7] D. Rattan, R. K. Bhatia, M. Singh, Software clone detection: A systematic review, Information & Software Technology 55 (7) (2013) 1165–1199.

[8] N. Pham, H. Nguyen, T. Nguyen, J. Al-Kofahi, T. Nguyen, Complete and Accurate Clone Detection in Graph-based Models, in: Proceedings of ICSE, IEEE, 2009, pp. 276–286.

[9] M. Dumas, L. García-Bañuelos, M. L. Rosa, R. Uba, Fast detection of exact clones in business process model repositories, Information Systems 38 (4) (2013) 619–633.

[10] M. La Rosa, H. Reijers, W. Aalst, R. Dijkman, J. Mendling, M. Dumas, L. García-Bañuelos, APROMORE: An Advanced Process Model Repository, Expert Systems With Applications 38 (6).

[11] R. Dijkman, M. Dumas, B. van Dongen, R. Käärik, J. Mendling, Similarity of business process models: Metrics and evaluation, Information Systems 36 (2) (2011) 498–516.

[12] B. Messmer, Efficient Graph Matching Algorithms, Ph.D. thesis, University of Bern, Switzerland (1995).

[13] J. Vanhatalo, H. Völzer, J. Koehler, The Refined Process Structure Tree, Data and Knowledge Engineering 68 (9) (2009) 793–818.

[14] G. Keller, T. Teufel, SAP R/3 Process Oriented Implementation: Iterative Process Prototyping, Addison-Wesley, 1998.

[15] P.-N. Tan, M. Steinbach, V. Kumar, Introduction to Data Mining, Addison-Wesley, 2005.

[16] C. Roy, J. R. Cordy, A mutation/injection-based automatic framework for evaluating code clone detection tools, in: Proc. of ICST Workshops, IEEE, 2009, pp. 157–166.

[17] Y. Zhao, G. Karypis, Evaluation of hierarchical clustering algorithms for document datasets, in: Proceedings of CIKM, ACM, 2002, pp. 515–524.

[18] M. L. Rosa, A. H. M. ter Hofstede, P. Wohed, H. A. Reijers, J. Mendling, W. M. P. van der Aalst, Managing process model complexity via concrete syntax modifications, IEEE Trans. Industrial Informatics 7 (2) (2011) 255–265.

[19] M. L. Rosa, P. Wohed, J. Mendling, A. H. M. ter Hofstede, H. A. Reijers, W. M. P. van der Aalst, Managing process model complexity via abstract syntax modifications, IEEE Trans. Industrial Informatics 7 (4) (2011) 614–629.

[20] J. Recker, M. L. Rosa, Understanding user differences in open-source workflow management system usage intentions, Information Systems 37 (3) (2012) 200–212.

[21] J. Recker, Continued use of process modeling grammars: the impact of individual difference factors, European Journal of Information Systems 19 (1) (2010) 76–92.

[22] J. Mendling, M. Strembeck, J. Recker, Factors of process model comprehension - findings from a series of experiments, Decision Support Systems 53 (1) (2012) 195–206.

[23] W. M. Rand, Objective criteria for the evaluation of clustering methods, Journal of the American Statistical association 66 (336) (1971) 846–850.

[24] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, E. Merlo, Comparison and Evaluation of Clone Detection Tools, IEEE Trans. on Software Engineering 33 (9) (2007) 577–591.

[25] C. Roy, J. R. Cordy, Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization, in: Proc. of the ICPC, IEEE, 2008, pp. 172–181.

[26] T. Kamiya, S. Kusumoto, K. Inoue, Ccfinder: A multilinguistic token-based code clone detection system for large scale source code, IEEE Trans. Software Eng. 28 (7) (2002) 654–670.

[27] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, L. Bier, Clone Detection Using Abstract Syntax Trees, in: Proceedings of ICSM, IEEE, 1998, pp. 368–377.

[28] L. Jiang, G. Misherghi, Z. Su, S. Glondu, Deckard: Scalable and accurate tree-based detection of code clones, in: Proc. of ICSE, IEEE, 2007, pp. 96–105.

[29] M. Gabel, L. Jiang, Z. Su, Scalable detection of semantic clones, in: Proc. of ICSE, ACM, 2008, pp. 321–330.

[30] J. Krinke, Identifying Similar Code with Program Dependence Graphs, in: WCRE, 2001.

[31] C. Liu, C. Chen, J. Han, P. S. Yu, Gplag: detection of software plagiarism by program dependence graph analysis, in: Proc. of KDD, ACM, 2006, pp. 872–881.

[32] F. Deissenboeck, B. Hummel, E. Jürgens, B. Schätz, S. Wagner, J.-F. Girard, S. Teuchert, Clone Detection in Automotive Model-based Development, in: Proceedings of ICSE, 2008.

[33] H. Storrle, Towards clone detection in UML domain models, Software and Systems Modeling 12 (2) (2013) 307–329.

[34] M. H. Alalfi, J. R. Cordy, T. R. Dean, M. Stephan, A. Stevenson, Models are code too: Near-miss clone detection for simulink models, in: In Proc. of ICSM, IEEE, 2012, pp. 295–304.

[35] B. Weber, M. Reichert, J. Mendling, H. A. Reijers, Refactoring large process model repositories, Computers in Industry 62 (5) (2011) 467–486.

[36] J.-Y. Jung, J. Bae, Workflow clustering method based on process similarity, in: ICCSA, Vol. 3981 of LNCS, Springer, 2006.

[37] J. Melcher, D. Seese, Visualization and clustering of business process collections based on process metric values, in: SYNASC, IEEE, 2008.

[38] C. Li, M. Reichert, A. Wombacher, The Minadept clustering approach for discovering reference process models out of process variants, International Journal of Cooperative Information Systems 19 (3-4) (2010) 159–203.

[39] M. Rosemann, W. van der Aalst, A Configurable Reference Modelling Language, Information Systems 32 (1) (2007) 1–23.

[40] C. C. Ekanayake, M. Dumas, L. García-Bañuelos, M. L. Rosa, A. H. M. ter Hofstede, Approximate clone detection in repositories of business process models, in: Proceedings of BPM, Springer, 2012, pp. 302–318.