Controlled Automated Discovery of Collections of Business Process Models

Luciano García-Bañuelos[a], Marlon Dumas[a], Marcello La Rosa[b,c], Jochen De Weerdt[b], Chathura C. Ekanayake[b]

[a] *University of Tartu, Estonia*
[b] *Queensland University of Technology, Australia*
[c] *NICTA Queensland Lab, Australia*

Abstract

Automated process discovery techniques aim at extracting process models from information system logs. Existing techniques in this space are effective when applied to relatively small or regular logs, but generate spaghetti-like and sometimes inaccurate models when confronted to logs with high variability. In previous work, trace clustering has been applied in an attempt to reduce the size and complexity of automatically discovered process models. The idea is to split the log into clusters and to discover one model per cluster. This leads to a collection of process models – each one representing a variant of the business process – as opposed to an all-encompassing model. Still, models produced in this way may exhibit unacceptably high complexity and low fitness. In this setting, this paper presents a two-way divide-and-conquer process discovery technique, wherein the discovered process models are split on the one hand by variants and on the other hand hierarchically using subprocess extraction. Splitting is performed in a controlled manner in order to achieve user-defined complexity or fitness thresholds. Experiments on real-life logs show that the technique produces collections of models substantially smaller than those extracted by applying existing trace clustering techniques, while allowing the user to control the fitness of the resulting models.

Key words: Process Mining, Process Discovery, Trace Clustering, Clone Detection

Email addresses: luciano.garcia@ut.ee (Luciano García-Bañuelos), marlon.dumas@ut.ee (Marlon Dumas), m.larosa@qut.edu.au (Marcello La Rosa), jochen.deweerdt@qut.edu.au (Jochen De Weerdt), chathura.ekanayake@student.qut.edu.au (Chathura C. Ekanayake)

1. Introduction

Process mining is concerned with the extraction of knowledge about business processes from information system logs [1]. Process mining encompasses a vast array of techniques, including techniques for automated discovery of business process models. Numerous algorithms for automated process discovery have been developed, which strike various tradeoffs between accuracy, generalization and simplicity of the discovered models.

One key limitation of the bulk of techniques for automated process discovery is that they fail to scale to processes with high levels of variance, i.e. high number of distinct traces. This is mainly because traditional process discovery techniques aim at producing a single model covering all traces in the log, leading to large and spaghetti-like models as the variance increases.

A common divide-and-conquer approach to address this issue is *trace clustering* [2, 3, 4, 5]. The idea is to slice the log into separate clusters, each one grouping similar traces, and to discover (via standard process discovery techniques) one process model per cluster. Accordingly, the output is a collection of process models, each covering a subset of the traces, as opposed to a single model encompassing all traces. The underlying assumption is that each model in this collection has lower complexity than a single all-encompassing model mined from all traces. In this context, complexity can be measured in terms of size (number of nodes or edges) or in terms of structural complexity metrics such as control-flow complexity or average connector degree, which have been shown to be correlated with model comprehensibility [6, 7].

While process discovery techniques based on trace clustering produce smaller individual models than single-model techniques, they do not seek to minimize the cumulative size of the discovered collection of process models. On the contrary, these techniques generally yield models that share duplicate fragments. This duplication entails that collectively, a set of models produced via trace clustering can be much larger and not necessarily easier to comprehend as a whole than a single process model discovered from all traces. A second drawback of trace clustering techniques is that they produce

models with low accuracy, specifically low fitness, where fitness is a measure of how well the process model can parse the traces in the event log. De Weerdt et al. [17] have shown that existing trace clustering techniques produce models that fail to parse between 30% to 50% of the traces on average, according to a certain notion of fitness.

To address the first drawback, this paper presents a two-way divide-and-conquer process discovery technique, wherein discovered process models are split on the one hand by variants via trace clustering (an operation we term "slicing"), but also hierarchically via shared subprocess extraction and merging ("dicing"). Slicing enables high-complexity mined models to be split into lower-complexity ones at the expense of duplication. Dicing, on the other hand, reduces duplication by refactoring shared fragments. By slicing, mining and dicing recursively, the technique attempts in a best-effort way to produce a collection of models each with size or structural complexity below a user-specified threshold, while minimizing the overall size of the discovered collection of models and without affecting accuracy. The technique is termed SMD (Slice, Mine, Dice) in reference to the steps performed at each level of the recursion.

To address the second drawback (low fitness), we combine the principles of SMD with an existing algorithm for fitness-aware trace clustering, namely ActiTraC [17]. This latter algorithm attempts to group traces in such a way as to achieve a user-specified level of fitness for each output process model. However, the algorithm produces a flat collection of models without seeking to control their individual size or complexity. Accordingly, this paper puts forward an approach to combine SMD with ActiTraC in such a way as to produce models that fulfill both fitness and complexity requirements. More generally, the paper describes how SMD can be applied on top of both hierarchical and flat trace clustering techniques, so as to achieve multiple quality tradeoffs on the discovered process models.

The paper reports on experiments using three real-life logs that put into evidence the improvements achieved by SMD relative to three existing trace clustering methods, both in terms of reduction in the overall number of output process models and their cumulative size. In addition, the experiments show that the combination of SMD and ActiTraC allow us to also control the fitness of the output process models (in addition to their complexity) without major impact on the total number of output process models

nor their cumulative size relative to the case where fitness is left uncontrolled.

The rest of the paper is structured as follows. Section 2 provides an overview of process discovery, trace clustering, clone detection and process model merging techniques upon which SMD builds. Next, Section 3 presents and illustrates the SMD algorithms for complexity-aware process model discovery. Section 4 describes how SMD can be used to also control the fitness of the discovered process models in addition to controlling their complexity. Section 5 presents the experimental setup and results. Finally, Section 6 discusses related work while Section 7 draws conclusions and spells out directions for future work.

2. Background

SMD builds upon techniques for: (i) automated process discovery; (ii) hierarchical trace clustering; (iii) clone detection in process models; and (iv) process model merging. This section introduces these techniques and discusses their use in SMD.

2.1. Automated process discovery techniques

Numerous techniques for discovering a single (flat) process model from a process execution log have been proposed in the literature [1, 8]. For example, Weijters et al. [9] propose the *Heuristics Miner*, which is based on an analysis of the frequency of dependencies between events in a log. In essence, frequency data is extracted from the log and used to construct a graph of events, where edges are added based on different heuristics. Types of splits and joins in the resulting event graph can be determined by analyzing the frequency of events associated to those splits and joins. This information can be used to convert the output of the Heuristics Miner into a Petri net. The Heuristics Miner is robust to noise in the event logs due to the use of frequency-based thresholds, which makes it suitable for use with real-life event logs. Meantime, van der Werf et al. [10] proposed a discovery method where relations observed in the logs are translated to an Integer Linear Programming (ILP) problem. The ILP miner is independent of the number of events in the log, making it applicable in practical scenarios.

Automated process discovery techniques can be evaluated along four dimensions: fitness (recall), appropriateness (precision), generalization and complexity [1]. Fitness

measures the extent to which the traces in a log can be parsed by the discovered model. Several measures of fitness have been proposed in the literature, most notably *token fitness* [14], which measures the number of missing and remaining tokens after replaying the original traces against the discovered process model represented as a Petri net; *continuous parsing measure*, which measures the number of missing and remaining activations while replaying a heuristics net; and *alignment-based fitness* [15], which measures the alignment of events from a trace with activities in an execution of the process model. Another measure of fitness that trades off correctness for performance is the *improved continuous semantics* (ICS) fitness [16], which can be seen as an optimization of the continuos parsing measure.

Appropriateness on the other hand is a measure of additional behavior allowed by a discovered model, that is not found in the log. Appropriateness in essence corresponds to the number of traces that can be generated by the discovered model, but not presented in the traces. A model with low appropriateness is one that can parse a proportionally large number of traces that are not in the log from which the model is discovered.

Generalization captures how well the discovered model generalizes the behavior found in a log. For example, if a model can be discovered using 90% of the traces of the log and this model can parse the remaining 10% of traces in the logs, it can be said the model generalizes well the log.

Finally, complexity of a model can be measured using several metrics proposed in the literature [6]. A simple complexity metric is the size the model, measured by the total number of nodes in the model (or alternatively number of edges). Empirical studies, e.g. [6], have shown that process model size is strongly correlated with model comprehensibility and error probability. Other (structural) complexity metrics correlated with comprehensibility include:

- CFC (Control-Flow Complexity): sum of all connectors weighted by their potential combinations of states after a split.
- ACD (Average Connector Degree): average number of nodes a connector is connected to.
- CNC (Coefficient of Network Connectivity): ratio between arcs and nodes.

- Density: ratio between the number of arcs and the maximum possible number of arcs for the same number of nodes.

An extensive empirical evaluation [8] of automated process discovery techniques has shown that Heuristics Miner offers a good tradeoff between precision and recall with satisfactory performance. The ILP miner achieves high recall – at the expense of some penalty on precision – but it does not scale to larger logs due to memory requirements. The SMD technique presented in this paper abstracts from the mining algorithm used to extract a model from a collection of traces. However, due to its scalability, we use the Heuristics Miner as a basis for the evaluation of SMD.

2.2. Trace clustering

Several approaches to trace clustering have been proposed [2, 3, 11, 4, 12, 13, 5, 17]. Some of these techniques produce a flat collection of trace clusters, e.g. [12, 17], though most produce hierarchical collections of trace clusters from which models can be mined. Specifically, hierarchical trace clustering methods construct a so-called *dendrogram*. The dendrogram is a tree wherein the root corresponds to the entire log. The root is decomposed into N (typically 2) disjoint trace clusters of smaller size, each of which is split again into N clusters and so on recursively.

A trace cluster is a set of "similar" traces. The notion of trace similarity varies between approaches and is generally defined with respect to a feature space. For instance, if traces are seen as strings on the alphabet consisting of the set of activity labels, the feature space corresponds to the set of all possible permutations of activity labels. With such a feature space, similarity of traces can be assessed by means of standard string similarity functions, such as Hamming distance or Levenshtein edit distance. However, mappings to other feature spaces have been used in the literature, such as the count of occurrences of activities, the count of motifs over such activities (e.g. n-grams), etc.

In addition to differing by the choice of similarity notion, trace clustering techniques differ in terms of the underlying clustering technique. Hierarchical clustering techniques can be divided in two families: agglomerative and divisive clustering. In agglomerative clustering, pairs of clusters are aggregated according to their proximity

following a bottom-up approach. In divisive clustering, a top-level cluster is divided into a number of sub-clusters and so on recursively until a stop condition is fulfilled.

The techniques of Song et al. [11, 4] and Bose et al. [5, 13] both use agglomerative hierarchical clustering. Song et al. also consider other clustering techniques, such as k-means and self-organizing maps. The main difference between the approaches of Song et al. and Bose et al. lie in the underlying feature space. Song et al. map traces into a set of features such as count of occurrences of individual activities, or count of occurrences of pairs of activities in immediate succession. On the other hand, Bose et al. evaluate the occurrence of more complex motifs such as repeats (i.e., n-grams observed at different points in the trace). Meanwhile, the DWS method of Medeiros et al. [2, 3] adopts divisive hierarchical clustering with k-means for implementing each division step. They use a similarity measure based on the count of occurrences of n-grams.

The above techniques produce a collection of models by applying single-model process mining techniques (e.g. Heuristics Miner) to each cluster at the lowest level of the dendrogram. Thus, the output is a collection of models that are in general simpler (lower complexity) than a single model discovered from the entire set of traces. However, the above techniques do not seek to control the complexity of the resulting models. Also, they do not attempt to reduce the amount of duplication across the produced process models. SMD addresses these limitations by taking as input the dendrogram produced by the above techniques and traversing it top-down in search for models of a certain level of complexity, while attempting to reduce duplication along the traversal.

2.3. Fitness-Aware Trace Clustering

The trace clustering techniques discussed above generally do not take into account the accuracy of the resulting models, where accuracy refers to the fitness and appropriateness dimensions discussed above. An exception is the technique of de Medeiros et al. [3] that proposes to stop the hierarchical decomposition of trace clusters when process models of a certain level of appropriateness are found, but without actively seeking to construct clusters that lead to process models with the required appropriate-

ness. Also, this technique as well as the others mentioned above do not seek to control the fitness of the resulting models.

In an empirical study, De Weerdt et al. [17] showed that the trace clustering algorithms discussed above produce models with low fitness (in the order of 0.5 to 0.7). This observation inspired the authors to design a fitness-aware algorithm for trace clustering, namely Active Trace Clustering (ActiTraC). ActiTraC produces a flat collection of clusters whereby each model discovered from a cluster meets a *target fitness* (a threshold on the fitness value, e.g. 1), and each cluster has a *minimum cluster size* in terms of overall log size (e.g. a cluster should at least be 30% of the log size). The algorithm is based on three phases: selection, look ahead and residual trace resolution. In the selection phase, traces are selected to create clusters either solely based on their frequency in the log (*frequency-based selective sampling*), or also based on their Euclidean distance to the traces already in a cluster (*distance-based selective sampling*). A trace is added to the current cluster if the model discovered from the cluster including that trace meets the target fitness. If so, the selection phase continues by choosing a new trace; otherwise the trace is discarded and a new one is selected, until the minimum cluster size is reached. This brings the algorithm to the second phase, wherein the cluster is enlarged by adding those traces that fully fit in the model discovered from the cluster, and that have not been considered in the selection phase. Observe that in this phase the model is not rediscovered, but a trace is added based on its individual fitness with the existing model. In the third phase, the *noisy* traces, i.e. those that the technique was unable to cluster, can either be distributed over the created clusters according to the individual trace fitness for the different process models discovered (i.e. a trace is assigned to the cluster whose model the trace fits the most), or form a separate *noisy cluster*. The algorithm terminates once a predefined *maximum number of clusters* is reached.

ActiTraC relies on the Heuristics miner to discover process models and on the ICS fitness for computing both model fitness and individual trace fitness, while the Euclidean distance is computed using the definition in [5].

ActiTraC is a best-effort approach as it cannot guarantee that the target fitness is always met. In fact, if noise redistribution is chosen, this causes a drop in the fitness of

the models whose clusters are enlarged with noise, since noisy traces do not fit a model perfectly, while if a noisy cluster is created, its fitness will typically not meet the target fitness.

2.4. Clone detection in process models

SMD relies on techniques for detecting duplicate fragments (a.k.a. clones) in process models. The idea is that these clones will be refactored into shared subprocess models in order to reduce the overall size and possibly also the complexity of discovered process models. Given that subprocess models must have a clear start point and a clear end point[1] we are interested in extracting single-entry, single-exit (SESE) fragments. Accordingly, SMD makes use of a clone detection technique based on a decomposition of process models into a tree representing all SESE fragments in the model, namely the Refined Process Structure Tree (RPST) [18]. Each node in an RPST corresponds to a SESE fragment in the underlying process model. The root node corresponds to the entire process model. The child nodes of a node N correspond to the SESE fragments that are contained directly under N. In other words, the parent-child relation in the RPST corresponds to the containment relation between SESE fragments. A key characteristic of the RPST is that it can be constructed for any model captured in a graph-oriented process modeling notation (e.g. BPMN or EPC).

For the purpose of exact clone detection, we make use of the RPSDAG index structure [19]. Conceptually, an RPSDAG of a collection of models is the union of the RPSTs of the models in the collection. Hence, a node in the RPSDAG corresponds to a SESE fragment whereas edges encode the containment relation between SESE fragments. Importantly, each fragment appears only once in the RPSDAG. If a SESE fragment appears multiple times in the collection of process models (i.e. it is a clone), it will have multiple parent fragments in the RPSDAG. This feature allows us to efficiently identify duplicate clones: a duplicate clone is simply a fragment with multiple parents.

[1] Observe that top-level process models may have multiple start and end events, but subprocess models must have a single start and end event in order to comply with the call-and-return semantics of subprocess invocation.

In addition to allowing us to identify exact clones, the RPSDAG provides a basis for approximate clone detection [20]. Approximate clone detection is achieved by applying clustering techniques on the collection of SESE fragments of an RPSDAG, using one minus the graph-edit distance as the similarity measure (as defined in [21]). Two clustering techniques for approximate clone detection based on this principle are presented in [20]. The first is an adaptation of the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm, the second is an adaptation of the Hierarchical Agglomerative Clustering (HAC) algorithm. Both of these techniques take as input a collection of process models and return a set of approximate clone clusters – each cluster representing a set of SESE fragments that are similar within a certain similarity threshold. To evaluate SMD, we adopted the DBSCAN approach to approximate clone clustering due to it being more scalable.

2.5. Process model merging

Approximate clone detection allows us to identify clusters of similar SESE fragments in a collection of process models. Having done so, we can replace each of the identified approximate clones with references to a single subprocess model representing the union of these similar fragments, so as to reduce the overall size of the collection of process models. It can be argued that this single subprocess should represent the collective behavior of all the SESE fragments in a cluster, otherwise some behavior would be lost when replacing the approximate clones with the single shared subprocess.

The technique for process model merging presented in [22] allows us to achieve this property. This technique takes as input a collection of process models (or SESE fragments) and returns a single merged process model, such that the set of traces of the merged model is the union of the traces of the input models. Thus, applying this technique on fragments of automatically discovered process models does not affect the fitness, appropriateness or generalization of the particular discovery technique used. An experimental evaluation reported in [22] shows that, if the input process models (or fragments) are similar, the size of the merged process model is significantly lower than the sum of the sizes of the input models. Also, the more similar the merged models are, the more significant is the size reduction achieved during merging.

This merging technique is applicable to any graph-oriented process modeling language that includes the three connectors XOR, AND and OR (e.g EPCs and BPMN).

3. Complexity-Aware SMD

The idea of SMD is to traverse the dendrogram produced by hierarchical trace clustering in a top-down manner (breadth-first), attempting at each level of the traversal to produce models of complexity below a certain user-defined threshold. This threshold can be placed on the size of a model or on its structural complexity measured in terms of CFC, density or other complexity metrics. For example, the user can specify an upper-bound of 50 for the number of nodes in a model or a maximum control-flow complexity of 20 per model. At each level of the traversal, the algorithm applies subprocess extraction based on clones detection, and merging in order to reduce duplication. The traversal stops at a given cluster d in the dendrogram – meaning that its child clusters are not visited – if a single model can be mined from d that after subprocess extraction meets the complexity threshold, or if d is a leaf of the dendrogram, in which case the model mined from d is returned.

The detailed description of SMD is given in Algorithm 1. Hereafter we illustrate this algorithm by means of the example dendrogram shown in Fig. 1 and we use size 12 as the complexity threshold. Observe that the root cluster $L1$ of the dendrogram is the log used as input to generate the dendrogram. As we traverse the dendrogram D, we mark the current position of the dendrogram with the clusters from which process models need to be mined. At the beginning, the root cluster is the only marked cluster (line 2). While there are marked trace clusters, we perform the following operations (lines 3–16). First, we mine a set of process models from marked trace clusters in D (line 4). As only $L1$ is marked at the beginning, a single process model $m1$ is mined. Let us assume that the model mined from $L1$ is that shown in Fig. 2. If we reach a leaf trace cluster of D at any stage, we cannot simplify the process model mined from that trace cluster anymore by traversing D. Thus, when a leaf of D is reached, we add the process model mined from that leaf to the set of leaf level process models M_l (line 5). As $L1$ is not a leaf, we do not update M_l at this stage. We then unmark all the clusters in M_l to avoid mining a process model again from these clusters, in next iterations of the

Algorithm 1: SMD/hierarchical

Input: Dendrogram D, complexity threshold k
Output: Set of root process models M_s, set of subprocesses S

```
1  Initialize M_l with ∅
2  Mark the root trace cluster of D
3  while there are marked trace clusters in D do
4      Mine a set of process models M from all marked trace clusters in D
5      Add to M_l the set of models from M mined from marked leaves of D
6      Unmark all trace clusters used to mine models in M_l
7      Invoke Algorithm 2 to extract subprocesses from M ∪ M_l and obtain a
       simplified set of root process models M_s and a set of subprocesses S
8      Let M_c be the process models in M_s that do not satisfy k
9      Let S_c be the subprocesses in S that do not satisfy k
10     Let P be the process models of M_s containing subprocesses in S_c
11     Add all models in P to M_c
12     Remove M_l from M_c
13     if M_c is empty then Unmark all trace clusters in D
14     foreach model m_c in M_c do
15         Get the trace cluster d used to mine m_c
16         Mark child trace clusters of d in D and unmark d
17 return M_s and S
```

while cycle (line 6). Then we extract subprocesses using Algorithm 2 (line 7) from the union of all mined models so far and all models mined from leaves M_l. In our example, we extract subprocesses only from $m1$, as M_l is empty.

In Algorithm 2, we first construct the RPSDAG from the set of process models in input (line 3). Then we identify sets of exact clones using the technique in [19] (line 4). For each set of exact clones, we create a single subprocess and replace the occurrence of these clones in their process models with a subprocess activity pointing to the subprocess just created (lines 6-7). Once exact clones have been factored out, we identify clusters of approximate clones using the technique in [20] (line 8). For each fragment cluster, we merge all approximate clones in that cluster into a configurable fragment (line 11) using the technique in [22].[2] If this fragment satisfies the threshold,

[2]Note that this merging technique is designed to preserve the behavior of the input process models. In other words, the merged process model contains exactly the union of the behaviors of the input models, thus the technique does not affect behavioral accuracy (fitness or appropriateness).

Algorithm 2: Extract subprocesses

Input: Set of process models M, complexity threshold k
Output: Set of root process models M_s, set of subprocesses S

```
1  Initialize M_s with M
2  Initialize S with ∅
3  Let F_s be the set of SESE fragments of M_s
4  Let F_e in F_s be the set of exact clones
5  Add F_e to S
6  foreach fragment f in F_e do
7      Replace all occurrences of f in models of M_s ∪ S with a subprocess activity
       pointing to f
8  Apply approximate clone detection on F_s \ F_e to identify fragment clusters C
9  while C is not empty do
10     Retrieve the cluster c with highest BCR from C
11     Merge fragments in c to obtain a merged fragment f_m
12     Remove c from C
13     if f_m satisfies k then
14         Add f_m to S
15         foreach fragment f in c do
16             Replace all occurrences of f in models of M_s with a subprocess
               activity pointing to f_m
17             Remove all ascendant and descendant fragments of f from all
               clusters in C
18             Remove all clusters that are left with less than 2 fragments from C
19 return M_s and S
```

we embed it into a subprocess (line 14) and replace all occurrences of the corresponding approximate clones with a subprocess activity pointing to this subprocess (lines 15–16).

A cluster of approximate clones may contain the parent or the child of a fragment contained in another cluster. As a fragment that has been used to extract a subprocess does no longer exist, we need to also remove its parent and child fragments occurring in other clusters (lines 17–18). We use the RPSDAG to identify these containment relationships efficiently. One or more fragment clusters may be affected by this operation. Thus, we have to order the processing of the approximate clones clusters based on some *benefit-cost-ratio* (BCR), so as to prioritize those clusters that maximize the number of process models satisfying the threshold after approximate clones extraction (line 10). If we set our threshold on size, we can use the BCR defined in [20], which is the

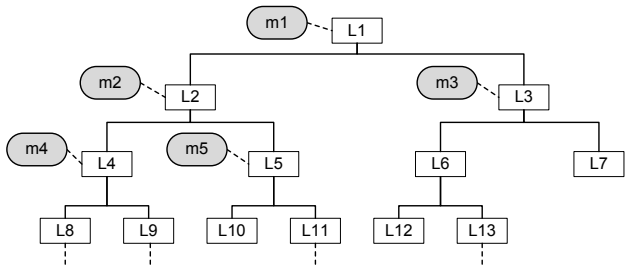

Figure 1: A possible dendrogram generated by hierarchical trace clustering.

ratio between overall size reduction (benefit) and distance between approximate clones within a cluster (cost). Similar BCRs can be defined on other complexity metrics.

Coming back to our example, we can see there are two exact clones ($f6$ and $f8$) and two approximate clones ($f4$ and $f9$) in $m1$, as highlighted in Fig. 2. After applying Algorithm 2 we obtain the process model collection in Fig. 3, where we have two subprocesses ($s1$ and $s2$) with $s2$ being a configurable model. In particular, we can observe that $s2$ has two configurable gateways – the XOR-split and the XOR-join represented with a thicker border – so that the selection of outgoing edges of the XOR-split (incoming edges of the XOR-join) is constrained by the annotated fragment identifiers. In addition, $s2$ has an annotated activity to keep track of the original labels for that activity in $f4$ and $f9$. For example, if we want to replay the behavior of $f4$, only the top and bottom branches of this merged model will be available with the bottom branch bearing activity "Perform external procurements".

Once subprocesses have been extracted, we add all models that have to be further simplified to set M_c (lines 8–12 of Algorithm 1). M_c contains all non-leaf models not satisfying the threshold and all non-leaf models containing subprocesses not satisfying the threshold. Algorithm 1 terminates if M_c is empty (line 13). Otherwise, for each model in M_c, we mark the respective cluster (lines 14–16) and reiterate the while loop.

In our example, the size of $m1$ after subprocess extraction is 19, which does not satisfy the threshold 12. Thus, we discard $m1$ and mine two process models $m2$ and $m3$ from $L2$ and $L3$, which are shown in Fig. 4. $m2$ and $m3$ contain two exact clones ($f24$ and $f31$) and two approximate clones ($f22$ and $f34$). Now we apply Algorithm 2

on $m2$ and $m3$ and obtain the process model collection shown in Fig. 5. The sizes of $m2$ and $m3$ after subprocess extraction are 14 and 11 respectively. Thus, $m3$ satisfies our threshold while $m2$ has to be further simplified. We then discard $m2$ and mine two fresh models $m4$ and $m5$ from $L4$ and $L5$ and so on.

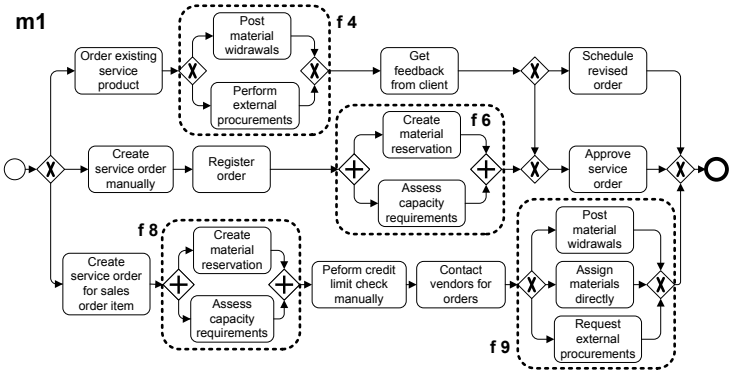

Figure 2: Process model $m1$ with similar fragments.

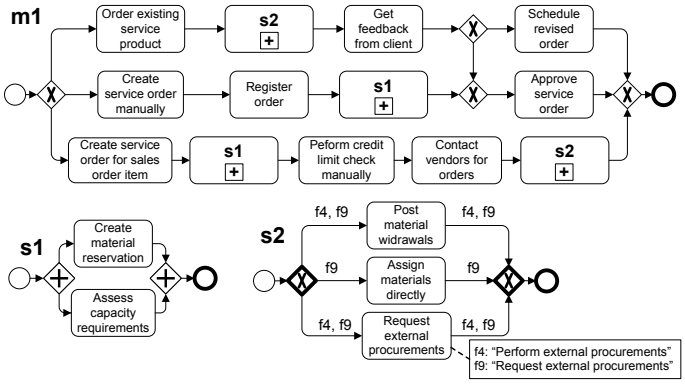

Figure 3: Process model $m1$ and subprocess $s1$ after subprocess extraction.

The complexity of Algorithm 1 depends on four external algorithms which are used to i) discover process models from the clusters of the dendrogram (line 4), ii) detect exact clones (line 4 of Algorithm 2), iii) detect approximate clones (line 8 of Algorithm 2) and iv) merge approximate clones (line 11 of Algorithm 2). Let c_1, c_2, c_3 and c_4 be the respective costs of these algorithms. The complexity of exact

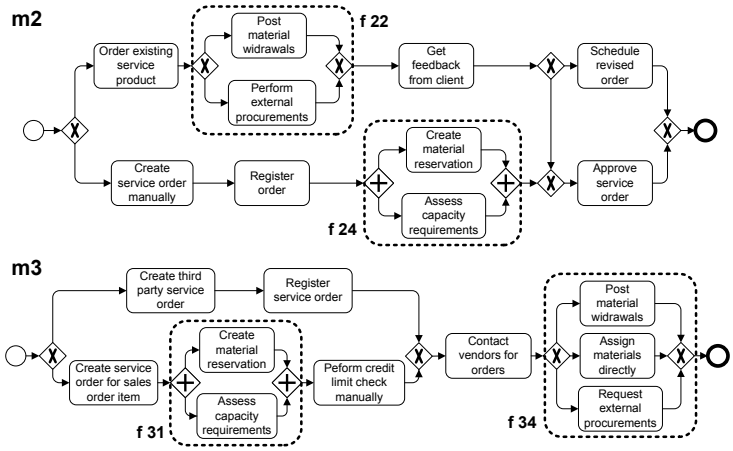

Figure 4: Process models $m2$ and $m3$ mined from trace clusters $L2$ and $L3$.

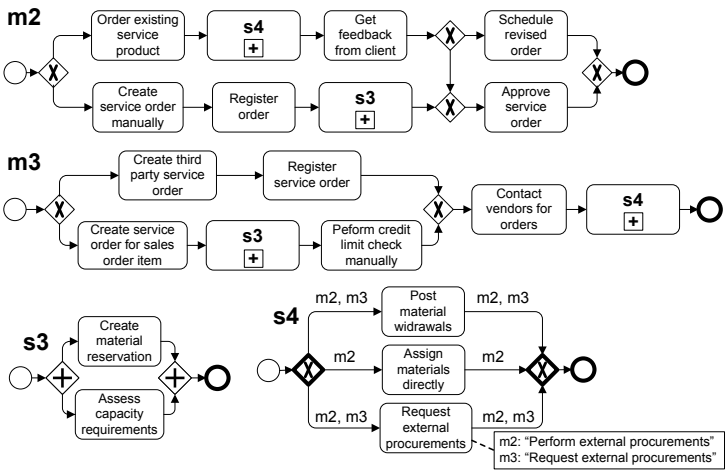

Figure 5: Process models and subprocesses after subprocess extraction from $m2$ and $m3$.

clone detection is determined by the insertion of fragments into the RPSDAG, which dominates the complexity of deleting fragments [19]. The complexity of approximate clone detection is dominated by that of computing the graph-edit distance between fragments [20]. Let F be the set of all SESE fragments of the process models that can be discovered from all trace clusters of dendrogram D, i.e. F is the union of all F_s. In the worst case, we need to discover a process model from each cluster of the den-

drogram, which is $O(|D|c_1)$; insert all fragments in the RPSDAG, which is $O(|F|c_2)$; compute the graph-edit distance of all pairs of fragments, which is $O(|F|^2c_3)$; and merge $|F|/2$ fragments, which is $O(|F|c_4)$. Thus, the worst-case complexity of Algorithm 1 is $O(|D|c_1 + |F|(c_2 + c_4) + |F|^2c_3)$. c_1 depends on the specific discovery technique used. For example, the Heuristic Miner is quadratic on the number of event classes in the log. Theoretically, c_2 is factorial in the number of nodes with the same label inside a single SESE fragment, though in practice this number is often very small or equal to zero thanks to various optimizations of exact clone detection [19]. Thus in practice c_2 is linear on $|F|$ [19]. c_3 is cubic on the size n of the largest fragment if using a greedy algorithm [21], as in the experiments reported in this paper. Finally, c_4 is $O(n\log(n))$.

SMD relies on a breadth-first exploration of the dendrogram. In this respect, one could wonder if a possible optimization would be to employ depth-first search instead. This modification however is not straightforward, since at each level of the dendrogram traversal, SMD needs to extract sub-processes shared across all process models at the current level, in order to determine which process models require further slicing to reach the complexity threshold. In other words, the alternation of slicing and dicing steps is a fundamental characteristic of SMD, thus precluding a depth-first exploration of the dendrogram based purely on a series of slicing steps.

As with any process mining technique, the outcome of SMD is affected by the quality of the logs used as input. In other words, any noise (e.g. due to logging errors) in the input log may have an impact on the usefulness of the discovered process models. While SMD itself does not address issues arising from imperfect logs, SMD has the advantage that it can be combined with various trace clustering or process discovery techniques that address such issues. In this respect, SMD preserves the behavioral accuracy (fitness and appropriateness) of the base techniques upon which it builds. What SMD adds on top of the base techniques are the operations for refactoring exact and approximate clones, and these operations do not affect the behavioral accuracy of the discovered process models.

4. Fitness-Aware SMD

In the previous section, we showed how SMD can be combined with hierarchical trace clustering techniques to make them complexity-aware while factoring out duplication across the discovered models. In this section we show that SMD can also be combined with a fitness-aware trace clustering technique to control both complexity and accuracy of the discovered models. Specifically, we use ActiTraC (cf. Section 2.3) as the baseline clustering technique since it builds fitness-aware clusters.

ActiTraC is a flat trace clustering technique, i.e. it produces a flat collection of process models rather than a hierarchy. Using the technique "as-is" with SMD as a stop condition when building the flat collection of clusters would produce a suboptimal solution where the number of process models and the overall collection's size would not necessarily be minimized. A more promising approach is to build a top-down variant of ActiTraC that incrementally constructs a dendrogram, and use SMD to point out which branches to further explore at each iteration of ActiTraC. To realize this approach, we embedded the original ActiTraC algorithm as the subroutine in a divisive clustering algorithm, where via SMD the complexity of each model underlying a cluster determines whether the cluster should be subdivided (the model is too complex) or not (the model is within the complexity threshold).

The detailed description of SMD on top of a flat trace clustering technique is given in Algorithm 3. This is a variant of Algorithm 1 which takes as input the root log r rather than a prebuilt dendrogram D, and populates a set T with the trace clusters that need to be analyzed at each iteration of the procedure.

At each iteration of the while loop in Algorithm 3, the ActiTraC subroutine is applied to each trace cluster whose underlying process model M_c is too complex. In order to keep the dendrogram balanced, we set the number of clusters to be generated by ActiTraC to two and enable redistribution of noisy traces, as shown in Figure 6. This leads to three possible cases:

- Clusters $c1$ and $c2$ are non-empty (typical case). After noise redistribution, if any, the two clusters will be added to the set of clusters T upon which to apply SMD.

Algorithm 3: SMD/flat

Input: Root log r, complexity threshold k
Output: Set of root process models M_s, set of subprocesses S

```
1  Initialize M_l with ∅ and add r to set T
2  while T is non-empty do
3      Mine a set of process models M from all trace clusters in T
4      Add to M_l the set of models from M mined from singleton clusters in T
5      Remove from T all trace clusters used to mine models in M_l
6      Invoke Algorithm 2 to extract subprocesses from M ∪ M_l and obtain a
       simplified set of root process models M_s and a set of subprocesses S
7      Let M_c be the process models in M_s that do not satisfy k
8      Let S_c be the subprocesses in S that do not satisfy k
9      Let P be the process models of M_s containing subprocesses in S_c
10     Add all models in P to M_c
11     Remove M_l from M_c
12     if M_c is empty then Remove all trace clusters from T
13     foreach model m_c in M_c do
14         Get the trace cluster d used to mine m_c
15         Invoke ActiTraC on d to obtain two clusters c1 and c2
16         Add c1 and c2 to T and remove d from T
17 return M_s and S
```

- $c2$ is empty, $c1$ and n are not. If the fitness threshold is too strict (e.g. 1), it may be impossible to even find a single trace in the set of remaining traces from which to discover a perfectly fitting model. This depends on the underlying mining algorithm chosen. For example, Heuristics Miner cannot guarantee to build a perfectly fitting process model out of any single sequence of events. In this scenario, the noise is not redistributed but considered as the second cluster.

- $c2$ and n are empty, or $c1$ and $c2$ are both empty. Again, this is due to the fitness threshold being too strict. In this case, ActiTraC will fail to create two clusters. To cope with this situation, we compute the Euclidean distance matrix between the traces in the log using the *bag-of-activities* profile in [5]. Based on this, we then apply hierarchical agglomerative clustering to the log used as input to form two clusters.

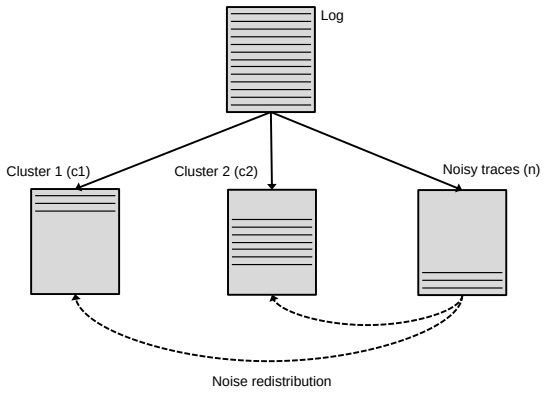

Figure 6: ActiTraC generating two clusters with noise redistribution.

5. Evaluation

We implemented the SMD technique on top of the Apromore [23] platform.[3] Using this implementation, we evaluated the technique on two event logs extracted from a large insurance company and on the log of the BPI challenge 2012[4] (herein called BPI Log). The first log of the insurance company (herein Motor Log) was taken from a motor insurance claims handling process for windscreen claims. The second log (herein Commercial Log) was taken from a commercial insurance claims handling process. We extracted completed traces from the first two months of each log, leading to a total of 4,300 to 5,300 traces. As we can see from Tab. 1, the three logs exhibit different characteristics despite similar number of traces. In particular, there is a substantial difference in duplication ratio (i.e. the ratio between events and event classes).

As a baseline for the experiments, we used the three hierarchical trace clustering techniques by Song et al. [11, 4], Bose et al. [13, 5] and the DWS technique by Greco, de Medeiros et al. [2, 3]. Both algorithms proposed by Song and Bose generate a hierarchy of trace clusters where the leaves are singletons. We adapted these two techniques to traverse down the dendrogram until we find a set of trace clusters whose mined mod-

[3]The tool is available at `www.apromore.org/platform/tools`
[4]`http://www.win.tue.nl/bpi2012/doku.php?id=challenge`

Log	Traces	Events	Event classes	Duplication ratio
Motor	4,293	33,202	292	114
Commercial	4,852	54,134	81	668
BPI[5]	5,312	91,949	36	2,554

Table 1: Characteristics of event logs used in the experiments.

els have complexity lower than or equal to the threshold. The DWS technique uses the K-Means clustering algorithm for clustering traces. It performs hierarchical trace clustering by applying K-Means at each level to obtain the next level clusters. We adapted the DWS technique to split clusters until the process models mined from all trace clusters have complexity lower than or equal to the threshold. Further, we configured this technique to split each cluster into two child clusters at each level (K=2).

We also implemented a hierarchical (divisive) version of ActiTraC based on the algorithm in Section 4, but without the refactoring steps. In other words, the algorithm runs ArtiTraC once, and for each obtained cluster that does not produce a process model meeting the complexity threshold, it re-runs ArtiTraC recursively until it reaches clusters that produce models meeting the complexity threshold. This hierarchical version of ActiTraC is hereby called ActiTraC$_H$. Besides using the parameters of ActiTraC described in Section 4, we used distance-based selective sampling, we set the fitness threshold to its strictest value of 1 and the minimal clustering size to 50% (this means the algorithm tries to add to cluster $c1$ at least half the traces in the root log).

Finally, we included in the experiments the SMD versions of the above four algorithms. This means SMD on top of Song et al. [11, 4], Bose et al. [13, 5] and de Medeiros at al. using Algorithm 1 to post-process the dendrogram of clusters produced by these techniques, as well as SMD on top of ActiTraC$_H$ as defined in Algorithm 3.

For consistency, we used the Heuristics Miner [9] to discover process models from the clusters retrieved by all four techniques. For clone detection we used the implementation described in [19] while for approximate clone clustering, we used the implemen-

[5]The extract of the BPI log used in these experiments contains multiple end events, thus we had to add an artificial single end event to every trace. After this, the total number of traces in this log becomes 97,261 with 37 instead of 36 event classes. This log is available at `www.apromore.org/platform/tools`

tation of the DBSCAN algorithm described in [20] with graph-edit distance threshold of 0.4. These implementations, as well as that of the technique for merging process models described in [22], were also integrated into our tool.

We set the user-defined complexity threshold on the model size, as it has been shown that size has the largest impact on perceived process model complexity [6]. We targeted the lowest possible size threshold in the experiments, taking into account however that there is an *implicit lower limit* on the minimum size each mined process model can have. This limit, which is a lower-bound for the user-defined threshold, depends on the number and size of the clones we can identify in the process models discovered from the trace clusters. The risk of choosing a threshold lower than this limit is that we may end up with a proliferation of process models, many of which still with size above the threshold since fundamentally a process model discovered from a cluster cannot be smaller than the size of the largest trace in that cluster.[6]

To discover this implicit limit we would need to run SMD using a size threshold of 1, so as to fully explore the dendrogram, and then measure the size of the largest resulting process model. This would be inefficient. However, we empirically found out that a good approximation of this implicit limit is given by the size of the largest process model that can be mined from a single trace.

We set the size threshold to this approximate implicit limit, which is 37 for the Motor log, 34 for the Commercial log and 56 for the BPI log.[7] The results of the experiments are shown in Fig. 7 (Motor Log), Fig. 8 (Commercial Log) and Fig. 9 (BPI Log), where "S", "B", "M" and "A" stand for the technique of Song et al., Bose et al., Medeiros et al., and ArtiTraC$_H$ respectively, while "SMD$_S$", "SML$_B$", "SMD$_M$" and "SMD$_A$" refer to their respective SMD extensions.

As we can observe from the histograms, SMD$_B$, SMD$_M$, SMD$_S$ and SMD$_A$ consistently yield a significant reduction in the overall size across all three logs and all four trace clustering techniques used. This reduction ranges from 13.5% (with SMD$_A$ on the Commercial log) to 63.9% (with SMD$_M$ on the BPI log), as evidenced by Tab. 2.

[6] Except if there is repetition in the trace and if this repetition can be captured via a cycle in the process model, but even then there is a minimum possible size of the model once the repetition has been factored out

[7] It turns out that these values correspond to the actual implicit size limits of the three logs.

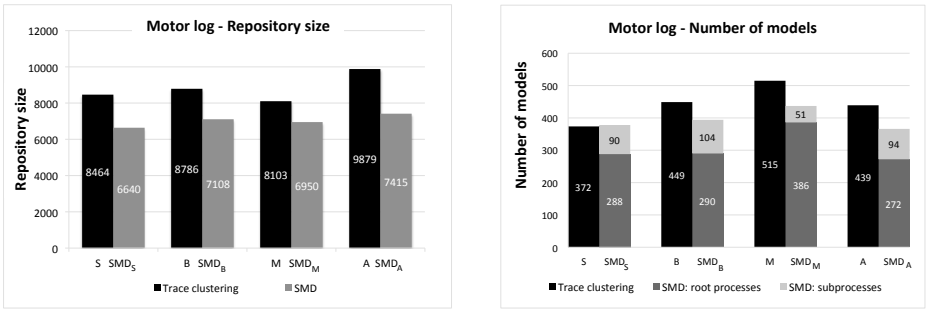

Figure 7: Overall size and number of models obtained from the Motor log.

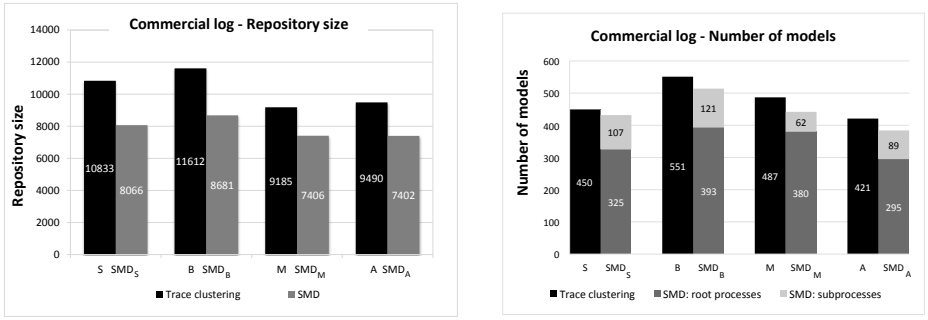

Figure 8: Overall size and number of models obtained from the Commercial log.

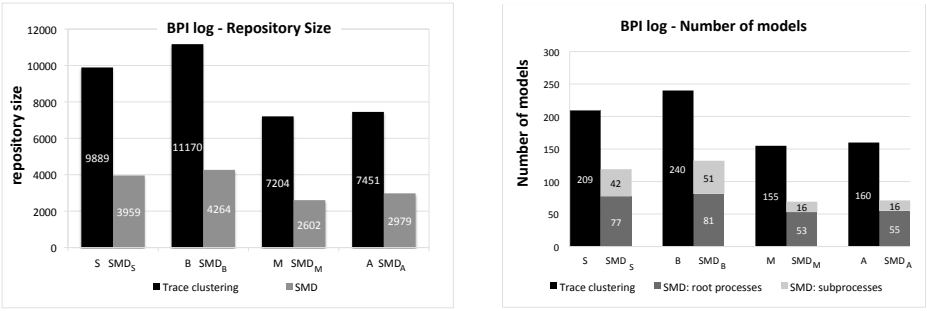

Figure 9: Overall size and number of models obtained from the BPI log.

In particular, we can observe that despite the technique of de Medeiros et al. always produces the lowest overall size while that of Bose et al. produces the highest one among the trace clustering techniques, these differences are thinned out by SMD. This is because SMD refactors redundancies across clusters that are typically introduced by trace clustering techniques.

We also observe significant reductions in the number of models, ranging from 22% (with SMD_M on the Commercial log) to 65.8% (with SMD_M on the BPI log) if con-

sidering root models only (see Tab. 2). Adding subprocesses to the count, the extent of this reduction is clearly diminished (there is even a slight increase of 1.6% in the total number of models in the case of SMD_S on the Motor log). These results should be interpreted as an indication that SMD can often achieve the complexity threshold with less process models (particularly less root process models) compared to the three baseline trace clustering techniques used in the experiments.

In the case of SMD_A we observe that the overall repository size and number of models are comparable to those achieved by SMD on top of the other three techniques. In the case of the Motor log, the repository size produced by $ArtiTraC_H$ and SMD_A is slightly larger than those produced by the other techniques. This could be explained by the fact that the motor claims process is very irregular (high variability), thus requiring slightly larger models in order to achieve a level of fitness close to 1. We note that the fitness achieved by $ArtiTraC_H$ (and hence SMD_A) on the three logs was between 0.95 and 0.98. As explained in Section 2.3, the reason why ArtiTraC does not achieve a fitness of 1 is because of redistribution of traces in the noise cluster.

Log	**Method**	**Size savings (%)**	**(Root) models number savings (%)**
Motor	SMD_S	21.6	(22.6) -1.6
	SMD_B	19.1	(35.4) 12.2
	SMD_M	14.2	(25.0) 15.1
	SMD_A	17.4	(38.0) 16.6
Commercial	SMD_S	25.5	(27.8) 4.0
	SMD_B	25.2	(28.7) 6.7
	SMD_M	19.4	(22.0) 9.2
	SMD_A	22.0	(29.9) 8.8
BPI	SMD_S	60.0	(63.2) 43.1
	SMD_B	61.8	(66.3) 45.0
	SMD_M	63.9	(65.8) 55.5
	SMD_A	60.0	(65.6) 55.6

Table 2: Savings in the overall size and number of models obtained with SMD.

From Tab. 2 we can also observe that the extent of model count and overall size reduction is more significant when the log's duplication ratio is higher (see Tab. 1). Indeed, there is a strong correlation between duplication ratio and overall size reduction produced by SMD (correlation of 0.99), and between duplication ratio and model count

reduction (correlation of 0.95). Thus, we conclude that the amount of improvement achieved by SMD depends on the amount of duplication in the log. This observation was expected, as the size reduction achieved by clone refactoring is proportional to the similarity between the process models [20].

Further, the average size and structural complexity of individual models reported in Tab. 3, indicate that SMD achieves the size threshold on individual models without affecting structural complexity. The table shows that on average, structural complexity remains largely unchanged after applying SMD (the increase in density is due to the inverse correlation of density and size). It is also worth noting that in most cases, average model size is reduced after applying SMD.

Log	**Method**	**Size**[8]				**CFC**	**ACD**	**CNC**	**Density**
		avg	**min**	**max**	**savings (%)**	**avg**	**avg**	**avg**	**avg**
Motor	S	22.75	4	37	22.8	12.07	2.71	1.26	**0.07**
	SMD_S	**17.57**	4	37		**10.07**	**2.34**	**1.21**	0.11
	B	20.01	4	37	9.8	**9.97**	2.51	1.2	**0.08**
	SMD_B	**18.04**	4	37		10.05	**2.38**	1.2	0.11
	M	**15.73**	**3**	49	-1.1	**7.36**	2.14	**1.12**	**0.11**
	SMD_M	15.9	4	**37**		8.34	**2.12**	1.14	0.12
	A	**21.50**	6	37	5.6	**10.31**	2.60	**1.21**	**0.07**
	SMD_A	20.30	6	37		10.34	**2.56**	1.27	0.10
Commercial	S	24.07	6	34	22.4	13.65	2.96	1.32	**0.06**
	SMD_S	**18.67**	2	34		**11.34**	**2.49**	**1.24**	0.10
	B	21.11	2	34	20.3	11.04	2.65	1.23	**0.07**
	SMD_B	**16.82**	2	34		**9.73**	**2.29**	**1.18**	0.12
	M	18.86	2	40	11.1	10.18	2.47	1.22	**0.09**
	SMD_M	**16.76**	2	**34**		**9.71**	**2.38**	**1.21**	0.11
	A	22.54	7	34	14.5	**11.69**	2.84	**1.27**	**0.06**
	SMD_A	**19.28**	**2**	34		11.89	**2.76**	1.31	0.09
BPI	S	47.32	15	56	29.7	20.77	**2.34**	**1.24**	**0.03**
	SMD_S	**33.27**	**4**	56		**20.18**	2.41	1.28	0.07
	B	46.54	13	56	30.6	20.48	2.35	**1.23**	**0.03**
	SMD_B	**32.3**	**4**	56		**19.29**	**2.33**	1.27	0.07
	M	46.48	21	61	18.9	**21.16**	**2.34**	**1.24**	**0.03**
	SMD_M	**37.71**	**7**	**56**		25.29	2.38	1.3	0.04
	A	46.57	27	56	9.9	**19.81**	4.10	**1.12**	0.08
	SMD_A	**41.96**	**10**	56		20.75	**4.10**	1.14	0.08

Table 3: Size and structural complexity metrics for model collections obtained with SMD.

In most of the experiments, SMD took more time than the corresponding baseline trace clustering technique. This is attributable to the reliance on graph-edit distance for

[8]The size values do not include artificial start and end events that were added when creating subprocesses.

process model comparison. In the worst case, SMD took double the time required by the baseline (e.g., 58 mins instead 28 mins of Medeiros et al. on the Commercial log). However, in other cases, SMD took less time than the baseline (e.g., 17 mins instead of 22 mins of Bose et al. on the BPI log). This is because if SMD mines less models relative to its baseline trace clustering technique, the time saved by the mining steps can compensate for the time taken to compute graph-edit distances.

In any case, the execution times of SMD are in the same order of magnitude as other trace clustering techniques, entailing in particular that execution times in the order of hours can be expected for more complex event logs. This weakness in terms of scalability can be addressed by means of parallelization techniques, which are orthogonal to the contribution of this paper and left as future work.

6. Related work

To the best of our knowledge two methods have previously been put forward to discover hierarchies of process models: one by Bose et al. [24] and another by Greco et al. [25, 2].

Bose et al. [24] present a method that discovers a single process model decomposed into sub-processes, each subprocess corresponding to a recurrent motif observed in the log traces. Given that this method is aimed at discovering process models with subprocesses, it is related to the dicing phase of the SMD technique. The difference is that the dicing phase in SMD discovers subprocesses that are shared by multiple parent processes, with the aim of reducing duplication across the parent processes. This choice is driven by the fact that the dicing phase of SMD is aimed at reducing some of the duplication introduced by the slicing phase. In contrast, Bose et al. [24] extract sub-processes out of one single parent process model. This having been said, the technique in Bose et al. [24] could be combined with the SMD technique. Specifically, for any trace cluster in the hierarchy produced by SMD, one could apply the technique of Bose et al. to identify motifs that could be factored out as non-shared subprocesses, in addition to the shared subprocesses identified by SMD. In this respect, SMD and the method of Bose et al. are orthogonal and complementary.

Meanwhile, Greco et al. [25, 2] use trace clustering to discover hierarchies of process models. In these hierarchies, the models associated to leaf nodes correspond to "concrete" models. In contrast, the models associated to inner nodes correspond to generalizations, resulting from the abstraction of one or several activities of models of descendant nodes. Thus the end result is a generalization-specialization hierarchy of process models. In a similar vein, SMD constructs a generalization-specialization hierarchy (like other trace clustering methods) but additionally, it extracts sub-processes shared by multiple leaf nodes in the hierarchy. Thus the resulting models are linked both by process-subprocess relations and generalization-specialization relations. Another key difference between SMD and the technique in Greco et al. is that in SMD the expansion of the dendrogram is controlled by the complexity of the discovered models.

As other trace clustering techniques, SMD relies on the idea of partitioning an event log into sub logs. Recently, approaches to partition an event log for the purpose of parallelizing the automated discovery of a process model have been studied [26, 27]. The idea of these techniques is to split the task of discovering one process model into several smaller sub-tasks that can be processed in parallel, such that the outputs of the sub-tasks can be recombined to produce a single process model. While these proposals also consider the problem of splitting a log into sub-logs, the purpose is different. Trace clustering techniques aim at producing multiple process models – each representing a variant of the process – whereas parallelization techniques [26, 27] aim at producing process model fragments that can later be recombined into a single model.

The terms slicing and dicing in the context of process mining are also used in [28]. However, in this latter work, the terms slicing and dicing are used in the sense of operations on data cubes. Specifically, data in an event log is represented as a data cube consisting of three dimensions: a temporal dimension, a "case" dimension (one value for each case) and an event type dimension. Slicing refers to projecting the cube over two of its three dimensions (i.e. removing a dimension), for example restricting the log to events occurring in a given month and focusing on the cases and event types occurring only during the month in question. The dice operation refers to computing a sub-cube by selecting values across multiple dimensions, for example restricting the log only to events occurring during a certain time window (e.g. 2-3 most recent months)

and only cases that satisfy a given condition (e.g. cases where customers lodged a complaint during the execution of the case). Meanwhile, SMD uses the term slicing in the sense of spitting a process into variants according to similarity of traces, while dicing refers to splitting a process into sub-processes.

This article is an extended version of a conference paper [29]. With respect to the conference version, the main extension relates to the ability to take into account fitness during the discovery of collections of process models in addition to complexity. This is achieved by integrating the principles of the SMD technique with the ActiTraC discovery technique. The experimental evaluation has also been extended to demonstrate that it is possible to control both fitness and complexity of discovered models without impacting on model count or overall size of the generated model collection.

7. Conclusion

SMD advances the state-of-the-art in automated process discovery along two directions. First, it provides a framework for designing automated process discovery techniques that produce collections of process models taking into account user-defined complexity thresholds. Second, SMD provides significant reductions in overall size of output process models relative to existing process discovery techniques based on trace clustering. The experimental evaluation shows overall size reductions of up to 64%, with little impact on structural complexity metrics of individual process models – barring an increase in density attributable to the dependency of this complexity metric on size (lower size generally entailing higher density).

A key feature of SMD is that it does not substitute itself to existing trace clustering techniques, but rather complements them. In other words, SMD can be seen as an enhancer of other trace clustering techniques rather than a completely new technique. In this paper, we have shown how SMD can be applied on top of three existing techniques for hierarchical trace clustering in order to make these techniques complexity-aware while producing collections of models with less duplication and thus smaller cumulative size. Next we showed how SMD can be combined with a fitness-aware automated process discovery technique that produces flat collections of process models, thus leading to a technique that is both fitness-aware and complexity-aware. The versatility of

SMD opens up manifold possibilities for combining SMD with other trace clustering techniques (hierarchical or flat) that address various optimization objectives. In essence, what SMD brings on top of other trace clustering techniques is the ability to control the complexity of each output process model, while reducing duplication in the resulting collection of process models and preserving accuracy relative to the base process discovery technique employed.

While complexity metrics have been shown to be correlated with comprehensibility [7], it is unclear how exactly to tune the thresholds used by SMD so as to produce models that users would best comprehend. While methods for determining complexity thresholds on individual models have been put forward [30], the interplay between overall size of a collection of process models, size of individual models and their structural complexity is less understood. Building an empirical understanding on how to set complexity thresholds for automated process discovery is a direction for future work.

Another direction for future work is to optimize SMD in order to reduce its execution time. Given that SMD uses a top-down (breadth-first search) approach to traverse or unfold a dendrogram, there is an opportunity to parallelize the processing of sibling nodes at each level of the breadth-first search. Secondly, there is an option to combine SMD with parallel (map-reduce) techniques for automated process discovery of individual process models [26]. Finally and perhaps more significant would be to parallelize the computation of GED matrices required by the approximate clone detection technique, which is the most complex step from a computational complexity viewpoint.

Acknowledgments This work is funded by the ERDF via the Estonian Centre of Excellence in Computer Science and by the ARC Linkage Project Facilitating Business Process Standardisation and Reuse (LP110100252). 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.

References

[1] W. M. P. van der Aalst, Process Mining - Discovery, Conformance and Enhancement of Business Processes, Springer, 2011.

[2] G. Greco, A. Guzzo, L. Pontieri, D. Saccà, Discovering expressive process models by clustering log traces, IEEE Trans. Knowl. Data Eng. 18 (8) (2006) 1010–1027.

[3] A. K. A. de Medeiros, A. Guzzo, G. Greco, W. M. P. van der Aalst, A. J. M. M. Weijters, B. F. van Dongen, D. Saccà, Process mining based on clustering: A quest for precision, in: Proc. of BPM Workshops, Vol. 4928 of LNCS, Springer, 2008, pp. 17–29.

[4] M. Song, C. W. Günther, W. M. P. van der Aalst, Improving process mining with trace clustering, J. Korean Inst. of Industrial Engineers 34 (4) (2008) 460–469.

[5] R. P. J. C. Bose, W. M. P. van der Aalst, Trace clustering based on conserved patterns: Towards achieving better process models, in: Proc. of BPM Workshops, Vol. 43 of LNBIP, Springer, 2010, pp. 170–181.

[6] J. Mendling, H. Reijers, J. Cardoso, What Makes Process Models Understandable?, in: Proc. of BPM, Vol. 4714 of LNCS, Springer, 2007, pp. 48–63.

[7] H. A. Reijers, J. Mendling, A study into the factors that influence the understandability of business process models, IEEE T. Syst. Man Cy. A 41 (3) (2011) 449–462.

[8] J. D. Weerdt, M. D. Backer, J. Vanthienen, B. Baesens, A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs, Inf. Syst. 37 (7) (2012) 654–676.

[9] A. J. M. M. Weijters, J. T. S. Ribeiro, Flexible heuristics miner (fhm), in: Proc. of CIDM, IEEE, 2011, pp. 310–317.

[10] J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, A. Serebrenik, Process discovery using integer linear programming, Fundam. Inform. 94 (3-4) (2009) 387–412.

[11] M. Song, C. W. Günther, W. M. P. van der Aalst, Trace clustering in process mining, in: Proc. of BPM Workshops, Vol. 17 of LNBIP, Springer, 2009, pp. 109–120.

[12] G. M. Veiga, D. R. Ferreira, Understanding spaghetti models with sequence clustering for prom, in: Proc. of BPM Workshops, Vol. 43 of LNBIP, Springer, 2009, pp. 92–103.

[13] R. P. J. C. Bose, Process mining in the large: Preprocessing, discovery, and diagnostics, Ph.D. thesis, Eindhoven University of Technology, Eindhoven (2012).

[14] A. Rozinat, W. van der Aalst, Conformance checking of processes based on monitoring real behavior, Information Systems 33 (1) (2008) 64–95.

[15] A. Adriansyah, B. van Dongen, W. van der Aalst, Conformance checking using cost-based fitness analysis, in: Proc. of EDOC, IEEE Computer Society, 2011, pp. 55–64.

[16] A. A. de Medeiros, Genetic process mining, Ph.D. thesis, Eindhoven University of Technology (2006).

[17] J. De Weerdt, S. vanden Broucke, J. Vanthienen, B. Baesens, Active trace clustering for improved process discovery, Knowledge and Data Engineering, IEEE Transactions on 25 (12) (2013) 2708–2720. doi:10.1109/TKDE.2013.64.

[18] J. Vanhatalo, H. Völzer, J. Koehler, The Refined Process Structure Tree, Data Knowl. Eng. 68 (9) (2009) 793–818.

[19] M. Dumas, L. García-Bañuelos, M. L. Rosa, R. Uba, Fast detection of exact clones in business process model repositories, Inf. Syst. 38 (4) (2012) 619–633. doi:10.1016/j.is.2012.07.002.
URL http://www.sciencedirect.com/science/article/pii/S0306437912000993

[20] 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: Proc. of BPM, Vol. 7481 of LNCS, Springer, 2012, pp. 302–318.

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

[22] M. L. Rosa, M. Dumas, R. Uba, R. Dijkman, Business process model merging: An approach to business process consolidation, ACM T. Softw. Eng. Meth. 22 (2).

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

[24] R. P. J. C. Bose, H. M. W. Verbeek, W. M. P. van der Aalst, Discovering hierarchical process models using prom, in: CAiSE Forum (Selected Papers), Vol. 107 of LNBIP, Springer, 2012, pp. 33–48.

[25] G. Greco, A. Guzzo, L. Pontieri, Mining taxonomies of process models, Data Knowl. Eng. 67 (1) (2008) 74–102.

[26] W. M. P. van der Aalst, Decomposing Petri nets for process mining: A generic approach, Distributed and Parallel Databases 31 (4) (2013) 471–507.

[27] W. M. van der Aalst, Decomposing process mining problems using passages, in: Proc. of ICATPN, Vol. 7347 of LNCS, Springer, 2012, p. 72?91.

[28] W. M. van der Aalst, Process cubes: Slicing, dicing, rolling up and drilling down event data for process mining, in: Proc. of APBPM, Vol. 159 of LNBIP, 2013, pp. 1–22.

[29] C. C. Ekanayake, M. Dumas, L. García-Bañuelos, M. L. Rosa, Slice, mine and dice: Complexity-aware automated discovery of business process models, in: Proceedings of BPM, Vol. 8094 of LNCS, Springer, 2013, pp. 49–64.

[30] J. Mendling, L. Sánchez-González, F. García, M. L. Rosa, Thresholds for error probability measures of business process models, J. Syst. Software 85 (5) (2012) 1188–1197.