Queensland University of Technology
Brisbane Australia

This is the author's version of a work that was submitted/accepted for publication in the following source:

Ekanayake, Chathura C., Dumas, Marlon, Garcia-Banuelos, Luciano, & La Rosa, Marcello (2013) Slice, mine and dice : complexity-aware automated discovery of business process models. In *11th Int. Conference on Business Process Management*. (In Press)

This file was downloaded from: http://eprints.qut.edu.au/58949/

**© Copyright 2013 (please consult the authors).**

**Notice**: *Changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published source:*

# Slice, Mine and Dice: Complexity-Aware Automated Discovery of Business Process Models

Chathura C. Ekanayake[1], Marlon Dumas[2], Luciano García-Bañuelos[2], and Marcello La Rosa[1]

[1] Queensland University of Technology, Australia
{c.ekanayake, m.larosa}@qut.edu.au
[2] University of Tartu, Estonia
{marlon.dumas, luciano.garcia}@ut.ee

**Abstract.** Automated process discovery techniques aim at extracting models from information system logs in order to shed light into the business processes supported by these systems. Existing techniques in this space are effective when applied to relatively small or regular logs, but otherwise generate large and spaghetti-like models. 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. The result is 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. 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 by means of subprocess extraction. The proposed technique allows users to set a desired bound for the complexity of the produced models. Experiments on real-life logs show that the technique produces collections of models that are up to 64% smaller than those extracted under the same complexity bounds by applying existing trace clustering techniques.

## 1 Introduction

Process mining is concerned with the extraction of knowledge about business processes from information system logs [16]. 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 and comprehensibility 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 by means of trace clustering [2,4,8,14]. The idea is to slice the log into separate clusters, each one grouping similar traces, and to discover (via standard mining 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 [10, 12].

While process discovery techniques based on trace clustering produce smaller individual models than single-model techniques, they do not seek to minimize the overall size of the discovered collection of 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 model mined from all traces.

In this setting, 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 and Dice) in reference to the steps performed at each level of the recursion.

SMD can be applied as a post-processing phase on top of any automated discovery technique based on (hierarchical) trace clustering. The paper reports on experiments using three real-life logs that put into evidence the improvements achieved by SMD on top of three existing trace clustering methods.

The rest of the paper is structured as follows. Section 2 provides an overview of related work on process mining and trace clustering, and introduces techniques for clone detection and process model merging, upon which SMD builds. Next, Section 3 presents and illustrates the algorithms behind SMD. Section 4 discusses the experimental setup and results, and Section 5 draws conclusions and spells out directions for future work.

## 2 Background and Related Work

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 in turn and discusses how they are used by 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 [16, 20]. For example, Weijters et al. [21] 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. [17] 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.

Process discovery techniques can be evaluated along four dimensions: fitness (recall), appropriateness (precision), generalization and complexity [16]. Fitness measures the extent to which the traces in a log can be parsed by the discovered model. Appropriateness is a measure of additional behavior allowed by a discovered model, that is not found in the log. 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. The complexity of a model can be measured using several metrics proposed in the literature [10]. 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. [10], 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 [20] 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 specifically use the Heuristics Miner as a basis for the evaluation of SMD .

### 2.2 Hierarchical trace clustering

Several approaches to trace clustering have been proposed [1, 2, 4, 8, 14, 15, 19]. Some of these techniques produce a flat collection of trace clusters, e.g. [19], though most produce hierarchical collections of trace clusters from which models can be mined. Specifically, hierarchical trace clustering methods construct a so-called *dendogram*. The

dendogram 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 also 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. [14, 15] and Bose et al. [1, 2] 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. [4, 8] 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 dendogram. Thus, the output is a flat collection of models of different levels of complexity. Accordingly, SMD does not take as input the collection of models produced by these techniques, but instead it takes the dendogram. The dendogram is traversed top-down to extract models at the required level of complexity.

To the best of our knowledge only two methods have been proposed aimed at mining hierarchies of process models. Bose et al. [3] presents a method that mines a single root process and a set of subprocesses that correspond to the factorization of motifs observed in the traces (a.k.a. conserved patterns). The method is hence intended to work on a single process model and not on a collection thereof. Meanwhile, Greco et al. [8] use trace clustering to mine 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 pairs of activities observed in models of descendant nodes. Thus the end result is a generalization-specialization hierarchy of models. In contrast, SMD aims at producing a collection of process models with (shared) sub-processes, thus the relation between lower-level and higher-level models is a part-of relation.

### 2.3 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 [6]. 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.

In addition to allowing us to identify exact clones, the RPSDAG provides a basis for approximate clone detection [7]. 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 [5]). Two clustering techniques for approximate clone detection based on this principle are presented in [7]. 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.

### 2.4 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

[1] Note 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.

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 [13] 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 [13] 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 The SMD Technique

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 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**: Discover process model collection

**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 [6] (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 [7] (line 8). For each fragment cluster, we merge all approximate clones in that cluster into a configurable fragment (line 11) using the technique in [13]. If this fragment satisfies the threshold, 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 [7], which is the ratio be-

**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
```

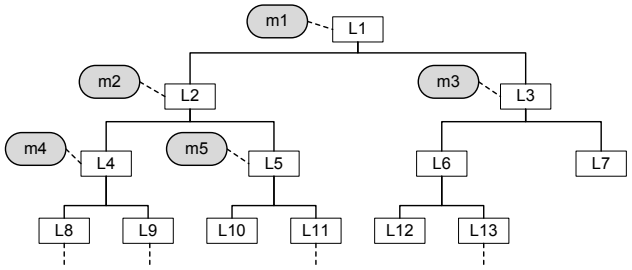


**Fig. 1.** A possible dendrogram generated by hierarchical trace clustering.

tween 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.

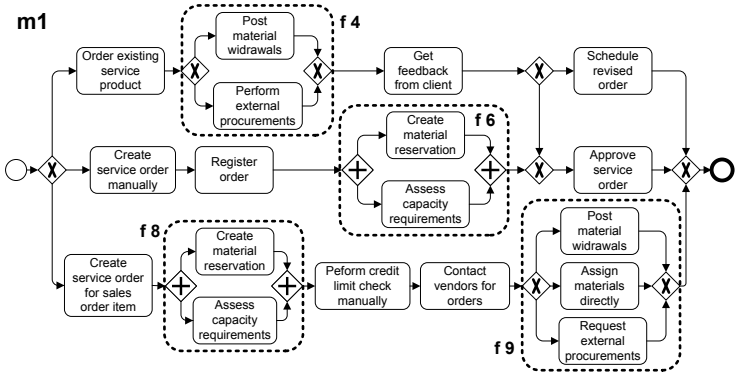


**Fig. 2.** Process model $m1$ with similar fragments

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 clone detection is determined by the insertion of fragments into the RPSDAG, which dominates the complexity of deleting fragments [6]. The complexity of approximate clone detection is dominated by that of computing the graph-edit distance between fragments [7]. Let $F$ be the set of all SESE fragments of the process models that can be discovered from all trace clusters

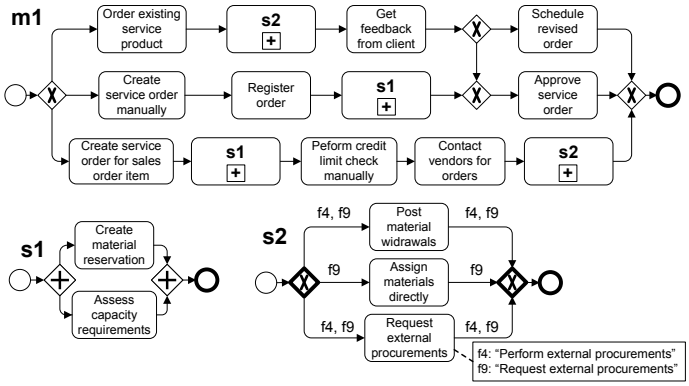


**Fig. 3.** Process model $m1$ and subprocess $s1$ after subprocess extraction

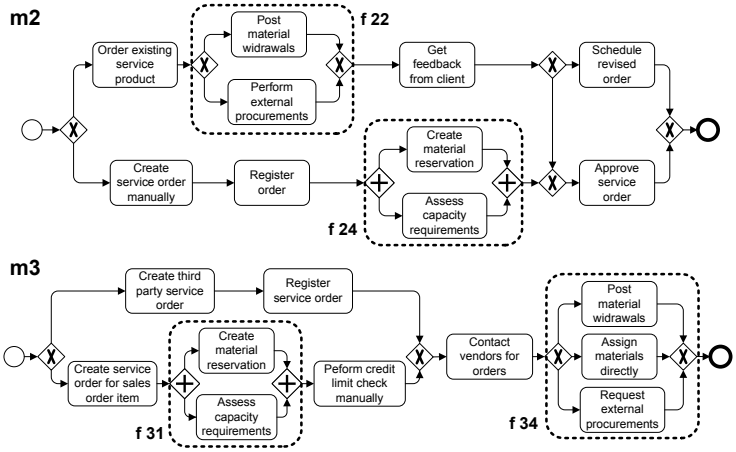


**Fig. 4.** Process models $m2$ and $m3$ mined from trace clusters $L2$ and $L3$

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 dendrogram, 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 [6]. Thus in practice $c_2$ is linear on $|F|$ [6]. $c_3$ is cubic on the size $n$ of

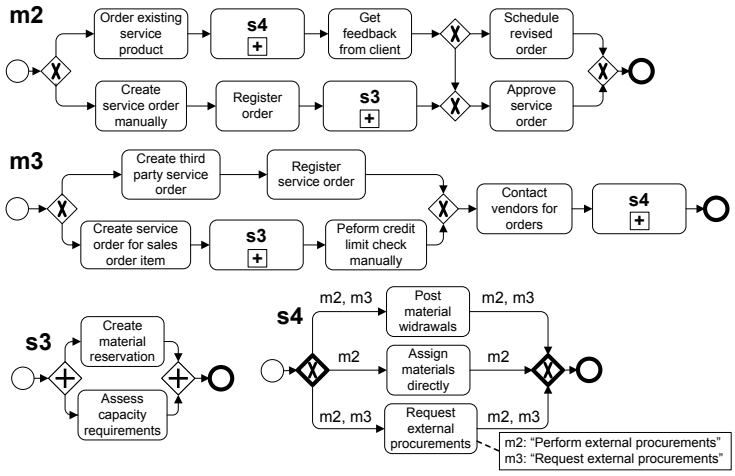


**Fig. 5.** Process models and subprocesses after subprocess extraction from $m2$ and $m3$

the largest fragment if using a greedy algorithm [5], as in the experiments reported in this paper. Finally, $c_4$ is $O(n\log(n))$.

## 4 Evaluation

We implemented the SMD technique on the Apromore [9] platform.[2] We then used this tool to evaluate the technique on two event logs extracted from a large insurance company and on the log of the BPI challenge 2012[3] (hereafter called BPI Log). The first log of the insurance company was taken from a motor insurance claims handling process for windscreen claims (called Motor Log). The second log was taken from a commercial insurance claims handling process (called Commercial Log). 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 the similar number of traces. In particular, there is a substantial difference in duplication ratio (i.e. the ratio between events and event classes).

Using these logs, we measured the reductions in overall size and number of models achieved by SMD on top of three hierarchical trace clustering techniques: Song et al. [14, 15], Bose et al. [1, 2] and the DWS technique by Medeiros et al. [4, 8]. These techniques were integrated in our tool. In particular, we used the DWS technique with K=2 and adapted it to split clusters until the process models mined from all trace clusters have complexity lower than or equal to the threshold, so that irrelevant clusters are not generated. For consistency, we used the Heuristics Miner [21] to discover process models from the clusters retrieved by all three techniques. For clone detection we

[2] The tool is available at `www.apromore.org/platform/tools`

[3] `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 | 5,312 | 91,949 | 36 | 2,554 |

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

used the implementation described in [6] while for approximate clone clustering, we used the implementation of the DBSCAN algorithm described in [7] with graph-edit distance threshold of 0.4. These implementations, as well as that of the technique for merging process models described in [13], were also integrated into our tool.

In this evaluation, we set the user-defined complexity threshold on the process model size, as it has been shown that size has the largest impact on perceived process model complexity [10]. There is an *implicit 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 model collection mined from the dendrogram of 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. This high number of models is due to the fact that the technique would explore the dendrogram as deep as possible. 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 measure the size of the largest process model we obtain. This would be inefficient. However, we empirically found out that a good approximation of this implicit limit, which can be computed in a matter of seconds, 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.[4] The results of the experiments are shown in Fig. 6 (Motor Log), Fig. 7 (Commercial Log) and Fig. 8 (BPI Log), where "S", "B" and "M" stand for the technique by Song et al., Bose et al. and Medeiros et al., respectively, while "$\text{SMD}_S$", "$\text{SML}_B$" and "$\text{SMD}_M$" indicate their respective SMD extensions.

As we can observe from the histograms, SMD consistently yields a significant reduction in the overall size across all three logs and all three trace clustering techniques used. This reduction ranges from 14.2% (with $\text{SMD}_M$ on the Motor log) to 63.9% (with $\text{SMD}_M$ on the BPI log), as evidenced by Tab. 2. In particular, we can observe that despite the technique of 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 compensates for the redundancies between clusters that may be introduced by a trace clustering technique as the number of clusters increases.

Similarly, we can observe significant reductions in the number of models, ranging from 22% (with $\text{SMD}_M$ on the Commercial log) to 65.8% (with $\text{SMD}_M$ on the BPI log) if considering root models only (see Tab. 2). Adding subprocesses to the count, the

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

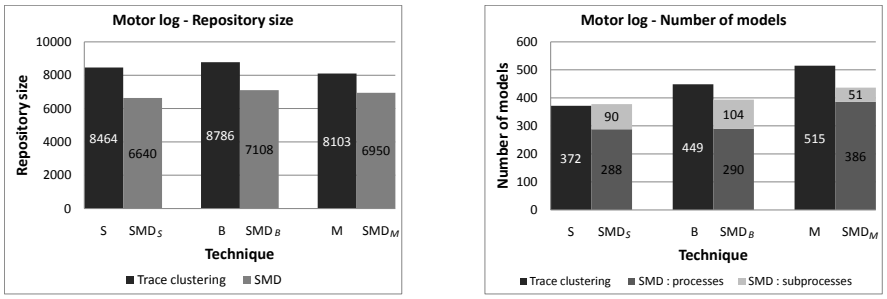


**Fig. 6.** Overall size and number of models obtained from the Motor log.

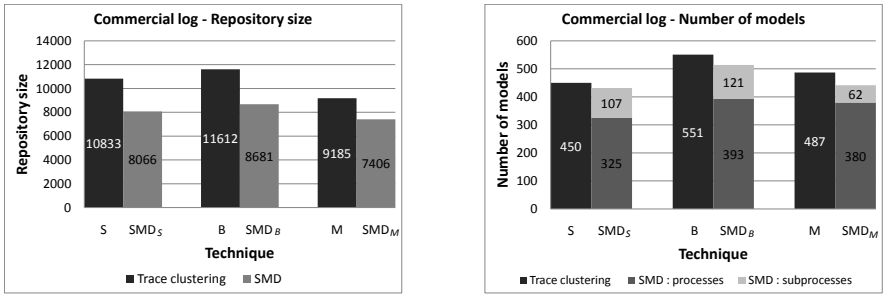


**Fig. 7.** Overall size and number of models obtained from the Commercial log.

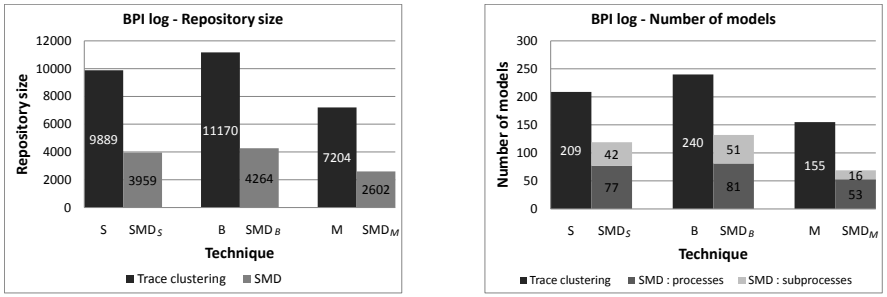


**Fig. 8.** Overall size and number of models obtained from the BPI log.

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.

From Tab. 2 we can also observe that the extent of the improvement, both for size and models number, increases with the increase of the log's duplication ratio (from the

Motor log to the BPI log – see Tab. 1). This is confirmed by the strong correlation between the duplication ratio and the percentage of size savings produced by SMD (0.99), and the strong correlation between the duplication ratio and the percentage of models number savings (0.95). Thus, we can conclude that the amount of improvement achieved by SMD depends on the amount of duplication in the log.

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 the average values for structural complexity measures remain 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, the average model size is reduced after applying SMD (up to 30.6% savings in the case of the BPI log).

| **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 |
| 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 |
| 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 |

**Table 2.** Savings in the overall size and number of models yielded by 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 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.

## 5 Conclusion

SMD advances the state-of-the-art in automated process discovery along two directions. First, it is to the best of our knowledge the first complexity-aware automated process discovery method, insofar as it seeks to produce models that meet user-specified complexity thresholds. Second, SMD provides significant reductions in overall size relative to existing process discovery techniques based on hierarchical trace clustering, while preserving the fitness, appropriateness and generalization of process models mined from trace clusters. The experimental evaluation based on three large real-life logs shows size

| Log | Method | Size | | | | 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 | 49 | | 8.34 | **2.12** | 1.14 | 0.12 |
| 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** | 0.1 |
| | 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 |
| 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 |

**Table 3.** Size and structural complexity metrics for model collections mined with SMD.

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.

While complexity metrics have been shown to be correlated with comprehensibility [12], 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 [11], the interplay between overall size of a collection of process models, size of individual models and their structural complexity is less understood. Building an 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.

**Acknowledgments** This work is funded by the Smart Services Cooperative Research Centre (CRC) under the Australian CRC Program and EU Regional Development Funds via the Estonian Centre of Excellence in Computer Science.

## References

1. R. P. J. C. Bose. *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2012.
2. R. P. J. C. Bose and W. M. P. van der Aalst. Trace clustering based on conserved patterns: Towards achieving better process models. In *BPM Workshops*, volume 43 of *LNBIP*, pages 170–181. Springer, 2010.

3. R. P. J. C. Bose, H. M. W. Verbeek, and W. M. P. van der Aalst. Discovering hierarchical process models using prom. In *CAiSE Forum (Selected Papers)*, volume 107 of *LNBIP*, pages 33–48. Springer, 2012.
4. A. K. A. de Medeiros, A. Guzzo, G. Greco, W. M. P. van der Aalst, A. J. M. M. Weijters, B. F. van Dongen, and D. Saccà. Process mining based on clustering: A quest for precision. In *BPM Workshops*, volume 4928 of *LNCS*, pages 17–29. Springer, 2008.
5. R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling. Similarity of business process models: Metrics and evaluation. *Inf. Syst.*, 36(2):498–516, 2011.
6. M. Dumas, L. García-Bañuelos, M. La Rosa, and R. Uba. Fast detection of exact clones in business process model repositories. *Inf. Syst.*, 38(4):619–633, 2012.
7. C. C. Ekanayake, M. Dumas, L. García-Bañuelos, M. La Rosa, and A. H. M. ter Hofstede. Approximate clone detection in repositories of business process models. In *BPM*, volume 7481 of *LNCS*, pages 302–318. Springer, 2012.
8. G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering expressive process models by clustering log traces. *IEEE Trans. Knowl. Data Eng.*, 18(8):1010–1027, 2006.
9. M. La Rosa, H.A. Reijers, W.M.P. van der Aalst, R.M. Dijkman, J. Mendling, M. Dumas, and L. García-Bañuelos. APROMORE: An Advanced Process Model Repository. *Expert Syst. Appl.*, 38(6), 2011.
10. J. Mendling, H.A. Reijers, and J. Cardoso. What Makes Process Models Understandable? In *BPM*, volume 4714 of *LNCS*, pages 48–63. Springer, 2007.
11. J. Mendling, L. Sánchez-González, F. García, and M. La Rosa. Thresholds for error probability measures of business process models. *J. Syst. Software*, 85(5):1188–1197, 2012.
12. H. A. Reijers and J. Mendling. A study into the factors that influence the understandability of business process models. *IEEE T. Syst. Man Cy. A*, 41(3):449–462, 2011.
13. M. La Rosa, M. Dumas, R. Uba, and R. Dijkman. Business process model merging: An approach to business process consolidation. *ACM T. Softw. Eng. Meth.*, 22(2), 2013.
14. M. Song, C. W. Günther, and W. M. P. van der Aalst. Improving process mining with trace clustering. *J. Korean Inst. of Industrial Engineers*, 34(4):460–469, 2008.
15. M. Song, C. W. Günther, and W. M. P. van der Aalst. Trace clustering in process mining. In *BPM Workshops*, volume 17 of *LNBIP*, pages 109–120. Springer, 2009.
16. W. M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
17. J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. *Fundam. Inform.*, 94(3-4):387–412, 2009.
18. J. Vanhatalo, H. Völzer, and J. Koehler. The Refined Process Structure Tree. *Data Knowl. Eng.*, 68(9):793–818, 2009.
19. G. M. Veiga and D. R. Ferreira. Understanding spaghetti models with sequence clustering for prom. In *BPM Workshops*, volume 43 of *LNBIP*, pages 92–103. Springer, 2009.
20. J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens. A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. *Inf. Syst.*, 37(7):654–676, 2012.
21. A. J. M. M. Weijters and J. T. S. Ribeiro. Flexible heuristics miner (fhm). In *CIDM*, pages 310–317. IEEE, 2011.