Finding Suitable Activity Clusters for Decomposed Process Discovery

B.F.A. Hompes, H.M.W. Verbeek, and W.M.P. van der Aalst

Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, The Netherlands
{b.f.a.hompes,h.m.w.verbeek,w.m.p.v.d.aalst}@tue.nl

Abstract. Event data can be found in any information system and provide the starting point for a range of process mining techniques. The widespread availability of large amounts of event data also creates new challenges. Existing process mining techniques are often unable to handle "big event data" adequately. *Decomposed process mining* aims to solve this problem by decomposing the process mining problem into many smaller problems which can be solved in less time, using less resources, or even in parallel. Many decomposed process mining techniques have been proposed in literature. Analysis shows that even though the decomposition step takes a relatively small amount of time, it is of key importance in finding a high-quality process model and for the computation time required to discover the individual parts. Currently there is no way to assess the quality of a decomposition beforehand. We define three quality notions that can be used to assess a decomposition, before using it to discover a model or check conformance with. We then propose a decomposition approach that uses these notions and is able to find a high-quality decomposition in little time.

Keywords: decomposed process mining, decomposed process discovery, distributed computing, event log

1 Introduction

Process mining aims to discover, monitor and improve real processes by extracting knowledge from event logs readily available in today's information systems [1]. In recent years, (business) processes have seen an explosive rise in supporting infrastructure, information systems and recorded information, as illustrated by the term Big Data. As a result, event logs generated by these information systems grow bigger and bigger as more event (meta-)data is being recorded and processes grow in complexity. This poses both opportunities and challenges for the process mining field, as more knowledge can be extracted from the recorded data, increasing the practical relevance and potential economic value of process mining. Traditional process mining approaches however have difficulties coping with this sheer amount of data (i.e. the number of events), as most interesting algorithms are linear in the size of the event log and exponential in the number of different activities [2].

In order to provide a solution to this problem, techniques for *decomposed process mining* [2–4] have been proposed. Decomposed process mining aims to decompose the process mining problem at hand into smaller problems that can be handled by existing process discovery and conformance checking techniques. The results for these individual sub-problems can then be combined into solutions for the original problems. Also, these smaller problems can be solved concurrently with the use of parallel computing. Even sequentially solving many smaller problems can be faster than solving one big problem, due to the exponential nature of many process mining algorithms [2]. Several decomposed process mining techniques have been developed in recent years [2–4, 9, 10, 12, 16]. Though existing approaches have their merits, they lack in generality. In [3], a generic approach to decomposed process mining is proposed. The proposed approach provides a framework which can be combined with different existing process discovery and conformance checking techniques. Moreover, different decompositions can be used while still providing formal guarantees, e.g. the fraction of perfectly fitting traces is not influenced by the decomposition.

When decomposing an event log for (decomposed) process mining, several problems arise. In terms of decomposed process discovery, these problems lie in the step where the overall event log is decomposed into sublogs, where submodels are discovered from these sublogs, and/or where submodels are merged to form the final model. Even though creating a decomposition is computationally undemanding, it is of key importance for the remainder of the decomposed process discovery process in terms of the overall required processing time and the quality of the resulting process model.

The problem is that there is currently no clear way of determining the quality of a given decomposition of the events in an event log, before using that decomposition to either discover a process model or check conformance with.

The current decomposition approaches do not use any quality notions to create a decomposition. Thus, potential improvements lie in finding such quality notions and a decomposition approach that uses those notions to create a decomposition with.

The remainder of this paper is organized as follows. In Section 2 related work is discussed. Section 3 introduces necessary preliminary definitions for decomposed process mining and the generic decomposition approach. Section 4 introduces decomposition quality notions to grade a decomposition upon, and two approaches that create a high quality decomposition according to those notions. Section 5 shows a (small) use case. The paper is concluded with views on future work in Section 6.

2 Related Work

Process discovery aims at discovering a process model from an event log while conformance checking aims at diagnosing the differences between observed and modeled behavior (resp. the event log and the model). Various discovery algorithms and many different modeling formalisms have been proposed in literature.

As very different approaches are used, it is impossible to provide a complete overview of all techniques here. We refer to [1] for an introduction to process mining and an overview of existing techniques. For an overview of best practices and challenges, we refer to the Process Mining Manifesto [5]. The goal of this paper is to improve decomposed process discovery, where challenging discovery problems are split in many smaller problems which can be solved by existing discovery techniques.

In the fields of data mining and machine learning many efforts have been made to improve the scalability of existing techniques. Most of these techniques can be distributed [8, 17], e.g. distributed clustering, distributed classification, and distributed association rule mining. To support this, several distributed data processing platforms have been created and are widely used [14, 20]. Some examples are Apache Hadoop [23], Spark [26], Flink [21], and Tez [19]. Specific data mining and machine learning libraries are available for most of these platforms. However, these approaches often partition the input data and therefore cannot be used for the discovery of process models. Decomposed process mining aims to provide a solution to this problem.

Little work has been done on the decomposition and distribution of process mining problems [2–4]. In [18] MapReduce is used to scale event correlation as a preprocessing step for process mining. More related are graph-partitioning based approaches. By partitioning a causal dependency graph into partially overlapping subgraphs, events are clustered into groups of events that are causally related. In [11] it is shown that region-based synthesis can be done at the level of synchronized State Machine Components (SMCs). Also a heuristic is given to partition the causal dependency graph into overlapping sets of events that are used to construct sets of SMCs. Other region-based decomposition techniques are proposed in [9, 10]. However, these techniques are limited to discovering Petri Nets from event logs. In [16] the notions of Single-Entry Single-Exit (SESE) and Refined Process Structure Trees (RPSTs) are used to hierarchically partition a process model and/or event log for decomposed process discovery and conformance checking. In [6], passages are used to decompose process mining problems.

In [3] a different (more local) partitioning of the problem is given which, unlike [11], decouples the decomposition approach from the actual conformance checking and process discovery approaches. It is indicated that a partitioning of the activities in the event log can be made based on a causal graph of activities. It can therefore be used together with any of the existing process discovery and conformance checking techniques. The approach presented in this paper is an extension of the approach presented in [3], though we focus on discovery. Where [3] splits the process mining problem at hand into subproblems using a *maximal* decomposition of a causal dependency graph, our approach first aims to recombine the many created activity clusters into better and fewer clusters, and only then splits the process mining problem into subproblems. As a result, fewer subproblems remain to be solved.

The techniques used to recombine clusters are inspired by existing, well-known software quality metrics and the business process metrics listed in [22],

i.e. *cohesion* and *coupling*. More information on the use of software engineering metrics in a process mining context is described in [22]. However, in this instance, these metrics are used to measure the quality of the decomposition itself rather than the process to be discovered. As such, the quality of the decomposition can be assessed before it is used to distribute the process mining problem.

3 Preliminaries

This section introduces the notations needed to define a better decomposition approach. A basic understanding of process mining is assumed [1].

3.1 Multisets, Functions, and Sequences

Definition 1 (Multisets).
Multisets are defined as sets where elements may appear multiple times. $\mathcal{B}(A)$ *is the set of all multisets over some set* A*. For some multiset* $b \in \mathcal{B}(A)$*, and element* $a \in A$*,* $b(a)$ *denotes the number of times* a *appears in* b*.*

For example, take $A = \{a, b, c, d\}$: $b_1 = []$ denotes the empty multiset, $b_2 = [a, b]$ denotes the multiset over A where $b_2(c) = b_2(d) = 0$ and $b_2(a) = b_2(b) = 1$, $b_3 = [a, b, c, d]$ denotes the multiset over A where $b_3(a) = b_3(b) = b_3(c) = b_3(d) = 1$, $b_4 = [a, b, b, d, a, c]$ denotes the multiset over A where $b_4(a) = b_4(b) = 2$ and $b_4(c) = b_4(d) = 1$, and $b_5 = [a^2, b^2, c, d] = b_4$. The standard set operators can be extended to multisets, e.g. $a \in b_2$, $b_5 \setminus b_2 = b_3$, $b_2 \uplus b_3 = b_4 = b_5$, $|b_5| = 6$

Definition 2 (Function Projection).
Let $f \in X \not\rightarrow Y$ *be a (partial) function and* $Q \subseteq X$*.* $f{\restriction_Q}$ *denotes the projection of* f *on* Q*:* $dom(f{\restriction_Q}) = dom(f) \cap Q$ *and* $f{\restriction_Q}(x) = f(x)$ *for* $x \in dom(f{\restriction_Q})$*.*

The projection can be used for multisets. For example, $b_5{\restriction_{\{a,b\}}} = [a^2, b^2]$.

Definition 3 (Sequences).
A sequence is defined as an ordering of elements of some set. Sequences are used to represent paths in a graph and traces in an event log. $\mathcal{S}(A)$ *is the set of all sequences over some set* A*.* $s = \langle a_1, a_2, \ldots, a_n \rangle \in \mathcal{S}(A)$ *denotes a sequence* s *over* A *of length* n*. Furthermore:* $s_1 = \langle\ \rangle$ *is the empty sequence and* $s_1 \cdot s_2$ *is the concatenation of two sequences.*

For example, take $A = \{a, b, c, d\}$: $s_1 = \langle a, b, b \rangle$, $s_2 = \langle b, b, c, d \rangle$, $s_1 \cdot s_2 = \langle a, b, b, b, b, c, d \rangle$

Definition 4 (Sequence Projection).
Let A *be a set and* $Q \subseteq A$ *a subset.* ${\restriction_Q} \in \mathcal{S}(A) \rightarrow \mathcal{S}(Q)$ *is a projection function and is defined recursively: (1)* $\langle\ \rangle{\restriction_Q} = \langle\ \rangle$ *and (2) for* $s \in \mathcal{S}(A)$ *and* $a \in A$*:*

$$(\langle a \rangle \cdot s){\restriction_Q} = \begin{cases} s{\restriction_Q} & \text{if } a \notin Q \\ \langle a \rangle \cdot s{\restriction_Q} & \text{if } a \in Q \end{cases}$$

So $\langle a, a, b, b, c, d, d \rangle{\restriction_{\{a,b\}}} = \langle a, a, b, b \rangle$.

3.2 Event Logs

Event logs are the starting point for process mining. They contain information recorded by the information systems and resources supporting a process. Typically, the executed *activities* of multiple *cases* of a *process* are recorded. Note that only *example behavior* is recorded, i.e. event logs only contain information that has been seen. An event log often contains only a fraction of the possible behavior [1]. A trace describes one specific instance (i.e. one case) of the process at hand, in terms of the executed activities. An event log is a multiset of traces, since there can be multiple cases having the same trace. For the remainder of this paper, we let $\mathcal{U}_A$ be some universe of activities.

Definition 5 (Event log).
Let $A \subseteq \mathcal{U}_A$ be a set of activities. A trace $s \in \mathcal{S}(A)$ is a sequence of activities. Let $L \in \mathcal{B}(\mathcal{S}(A))$ be a multiset of traces over A. L is an event log over A.

An example event log is $L_1 = [\langle a, b, c, d\rangle^5, \langle a, b, b, c, d\rangle^2, \langle a, c, d\rangle^3]$. There are three unique traces in L_1, and it contains information about a total of 10 cases. There are $4 \cdot 5 + 5 \cdot 2 + 3 \cdot 3 = 39$ events in total. The projection can be used for event logs as well. That is, for some log $L \in \mathcal{B}(\mathcal{S}(A))$ and set $Q \subseteq A : L{\restriction_Q} = [s{\restriction_Q} | s \in L]$. For example $L_1{\restriction_{\{a,b,c\}}} = [\langle a, b, c\rangle^5, \langle a, b, b, c\rangle^2, \langle a, c\rangle^3]$. We will refer to these projected event logs as *sublogs*.

3.3 Activity Matrices, Graphs, and Clusters

In [3] different steps for a generic decomposed process mining approach have been outlined. We *decompose* the overall event log based on a *causal graph* of activities. This section describes the necessary definitions for this decomposition method.

Definition 6 (Causal Activity Matrix).
Let $A \subseteq \mathcal{U}_A$ be a set of activities. $\mathcal{M}(A) = (A \times A) \to [-1.0, 1.0]$ denotes the set of causal activity matrices over A. For $a_1, a_2 \in A$ and $M \in \mathcal{M}(A)$, $M(a_1, a_2)$ denotes the "causal relation strength" from a_1 to a_2.

A value close to 1.0 signifies that we are quite confident there exists a causal relation (e.g. directly follows relation) between two activities while a value close to -1.0 signifies that we are quite sure there is no relation. A value close to 0.0 indicates uncertainty, i.e., there may be a relation, but there is no strong evidence for it.

For example, Table 1 shows an example causal activity matrix for the event log L_1. It shows that we are confident that casual relations exists from a to b, from a to c, from b to c, and from c to d, that we are uncertain about a causal relation from b to b, and that we are confident that other causal relations do not exist.

Table 1: Example causal activity matrix M_1 for event log L_1.

From\To	a	b	c	d
a	-0.46	0.88	0.75	-1.00
b	-1.00	0.00	0.88	-1.00
c	-1.00	-1.00	-0.90	1.00
d	-1.00	-1.00	-1.00	-0.67

Definition 7 (Causal Activity Graph).
Let $A \subseteq \mathcal{U}_A$ be a set of activities. $\mathcal{G}(A)$ denotes the set of causal activity graphs over A. A causal activity graph $G \in \mathcal{G}(A)$ is a 3-tuple $G = (V, E, w)$ where $V \subseteq A$ is the set of nodes, $E \subseteq (V \times V)$ is the set of edges, and $w \in E \rightarrow (0.0, 1.0]$ is a weight function that maps every edge onto a positive weight. $G = (V, E, w) \in \mathcal{G}(A)$ is the causal activity graph based on $M \in \mathcal{M}(A)$ and a specific causality threshold $\tau \in [-1.0, 1.0)$ iff

- $E = \{(a_1, a_2) \in A \times A \mid M(a_1, a_2) > \tau\}$,
- $V = \bigcup_{(a_1,a_2)\in E} \{a_1, a_2\}$, *and*
- $w((a_1, a_2)) = \frac{M(a_1,a_2)-\tau}{1-\tau}$ *for* $(a_1, a_2) \in E$.

That is, for every pair of activities $(a_1, a_2) \in A$, there's an edge with a positive weight from a_1 to a_2 in G iff the value for a_1 to a_2 in the causal activity matrix M exceeds some threshold τ. Note that $V \subseteq A$ since some activities in A might not be represented in G.

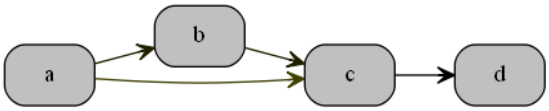

Fig. 1: Example causal activity graph G_1 for causal activity matrix M_1.

For example, Figure 1 shows the causal activity graph that was obtained from the causal activity matrix M_1 using $\tau = 0$.

Definition 8 (Activity Clustering).
Let $A \subseteq \mathcal{U}_A$ be a set of activities. $\mathcal{C}(A)$ denotes the set of activity clusters over A. An activity cluster $C \in \mathcal{C}(A)$ is a subset of A, that is, $C \subseteq A$. $\widehat{\mathcal{C}}(A)$ denotes the set of activity clusterings over A. An activity clustering $\widehat{C} \in \widehat{\mathcal{C}}(A)$ is a set of activity clusters, that is, $\widehat{C} \subseteq \mathcal{P}(A)$ [1]. A k-clustering $\widehat{C} \in \widehat{\mathcal{C}}(A)$ is a

[1] $\mathcal{P}(A)$ denotes the powerset over A

clustering with size k, i.e. $|\widehat{C}| = k$*. The number of activities in* $\widehat{C}$ *is denoted by* $||\widehat{C}|| = |\bigcup_{C \in \widehat{C}} C|$*, i.e.* $||\widehat{C}||$ *signifies the number of unique activities in* $\widehat{C}$*.*

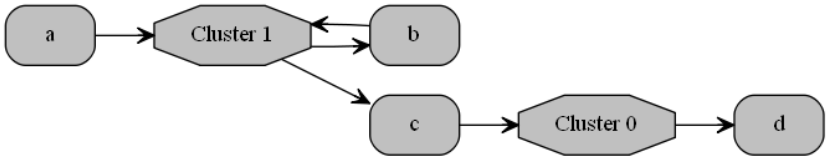

Fig. 2: Example activity clustering $\widehat{C}_1$ for causal activity graph G_1.

For example, Figure 2 shows the activity clustering $\widehat{C}_1$ with size 2 for the causal activity graph G_1. Cluster 0 contains the activities c (as input) and d (as output), whereas Cluster 1 contains a (as input), b (as input and as output), and c (as output). Note that the inputs and output are not part of the definition, but they are included here to better illustrate the fabric of the clusters. Also note that $||\widehat{C}_1|| = |\{a, b, c\} \cup \{c, d\}| = |\{a, b, c, d\}| = 4$.

3.4 Process Models and Process Discovery

Process discovery aims at discovering a model from an event log while conformance checking aims at diagnosing the differences between observed and modeled behavior (resp. the event log and the model). Various discovery algorithms have been proposed in literature. Literature suggests many different notations for models. We abstract from any specific model notation, but will define the set of algorithms that *discover* a model from an event log. These discovery algorithms are often called *mining algorithms*, or *miners* in short.

Definition 9 (Process Model).
Let $A \subseteq \mathcal{U}_A$ *be a set of activities.* $\mathcal{N}(A)$ *denotes the set of process models over A, irrespective of the specific notation (Petri nets, transition systems, BPMN, UML ASDs, etc.) used.*

Definition 10 (Discovery Algorithm).
Let $A \subseteq \mathcal{U}_A$ *be a set of activities.* $\mathcal{D}(A) = \mathcal{B}(\mathcal{S}(A)) \rightarrow \mathcal{N}(A)$ *denotes the set of discovery algorithms over A. A discovery algorithm* $D \in \mathcal{D}(A)$ *discovers a process model over A from an event log over A.*

For example, Figure 3 shows the Petri net N_1 which was discovered from the event log L_1 using the "ILP-Based Process Discovery" algorithm[2].

[2] This algorithm is available in ProM 6.5, see subsection 4.3.

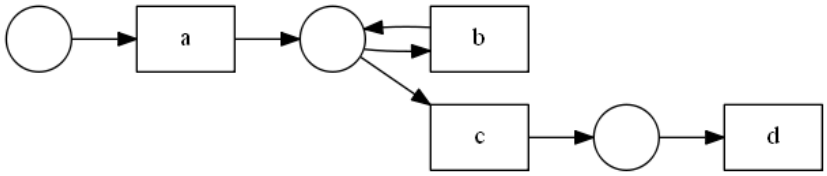

Fig. 3: Example Petri net N_1 discovered from the event log L_1.

3.5 Decomposed Process Discovery

As discussed, in [3], a general approach to decomposed process mining is proposed. In terms of decomposed process discovery, this approach can be explained as follows: Let $A \subseteq \mathcal{U}_A$ be a set of activities, and let $L \in \mathcal{B}(\mathcal{S}(A))$ be an event log over A. In order to decompose the activities in L, first a causal activity matrix $M \in \mathcal{M}(A)$ is *discovered* (cf. Table 1). Any causal activity matrix discovery algorithm $D_{CA} \in \mathcal{B}(\mathcal{S}(A)) \rightarrow \mathcal{M}(A)$ can be used. From M a causal activity graph $G \in \mathcal{G}(A)$ is *filtered* (using a specific causality threshold, cf. Figure 1). By choosing the value of the causality threshold carefully, we can filter out uncommon causal relations between activities or relations of which we are unsure, for example those relations introduced by noise in the event log.

Once the causal activity graph G has been constructed, an activity clustering $\widehat{C} \in \widehat{\mathcal{C}}(A)$ is created (cf. Figure 2). Any activity clustering algorithm $AC \in \mathcal{G}(A) \rightarrow \widehat{\mathcal{C}}(A)$ can be used to create the clusters. Effectively, an activity clustering algorithm partitions the causal activity graph into partially overlapping clusters of activities. Many graph partitioning algorithms have been proposed in literature [7]. Most algorithms however partition the graph into non-overlapping subgraphs, while in our case some overlap is required in order to merge submodels later on in the process. In [3], the so-called *maximal decomposition* is used, where the causal activity graph is cut across its vertices and each edge ends up in precisely one submodel, according to the method proposed in [15]. This leads to the smallest possible submodels.

Next, for every cluster in the clustering, L is *filtered* to a corresponding sublog by projecting the cluster to L, i.e., for all $C \in \widehat{C}$ a sublog $L{\restriction}_C$ is created. For example, based on the activity cluster array $\widehat{C}_1$ (cf. Figure 2), the event log L_1 would be split into sublogs $L_1{\restriction}_{\{a,b,c\}}$ and $L_1{\restriction}_{\{c,d\}}$, referring to clusters 1 and 0 respectively. A process model is *discovered* for each sublog. These are the submodels. Any discovery algorithm $D \in \mathcal{D}(A)$ can be used to discover the submodels. For example, Figure 4 shows the submodels that may be discovered from the sublogs of L_1 as mentioned above.

Finally, the submodels are merged into an overall model (cf. Figure 3). Any merging algorithm in $\mathcal{B}(\mathcal{N}(A)) \rightarrow \mathcal{N}(A)$ can be used for this step. Currently, submodels are merged based on activity labels. Note that we have $|\widehat{C}|$ clusters, sublogs and submodels, and $||\widehat{C}||$ activities in the final, merged model. Figure 5 shows the general decomposed process discovery workflow.

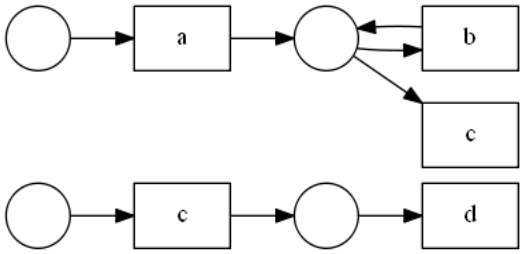

Fig. 4: Submodels discovered for the sublogs of L_1.

This workflow clarifies the generality of the approach. The necessary steps in the approach are defined by their input and output, but any applicable algorithm can be used to perform each step. This also shows the strength of a generic decomposed process mining approach. It might be of interest to only discover a small fragment of a particular process, e.g. only show the part of a medical process where repeated treatment is necessary, or only show the activities (events) performed by some selected resource (provided resource-data is stored in the log). The same holds for decomposed conformance checking. Clustering the activities such that the activities belonging to the interesting part of a process are clustered together will lead to a "filtered" sublog. Applying a fast discovery algorithm to non-interesting parts of the process or only discovering or checking conformance of the submodel of interest can greatly reduce calculation time.

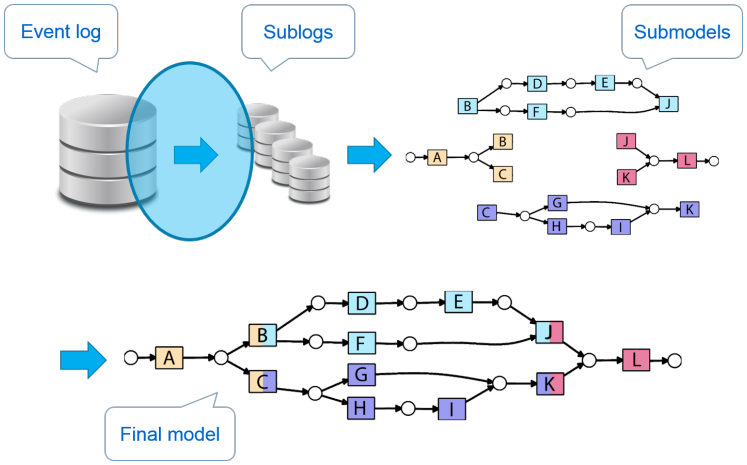

Fig. 5: The general decomposed process discovery workflow, using a more complex event log L_2 as an example. Finding a suitable activity clustering is of key importance.

4 A Better Decomposition

It is apparent that the manner in which activities are clustered has a substantial effect on required processing time, and it is possible for similarly sized clusterings (in the average cluster size) to lead to very different total processing times. As a result of the vertex-cut (*maximal*) decomposition approach [3], most activities will be in two (or more) activity clusters, leading to double (or more) work, as the clusters have a lot of overlap and causal relations between them, which might not be desirable. From the analysis results in [13] we can see that this introduces a lot of unwanted overhead, and generally reduces model quality. Also, sequences or sets of activities with high causal relations are generally easily (and thus quickly) discovered by process discovery algorithms, yet the approach will often split up these activities over different clusters.

Model quality can potentially suffer from a decomposition that is too fine-grained. It might be that the sublogs created by the approach contain too little information for the process discovery algorithm to discover a good, high quality submodel from, or that a process is split up where it shouldn't be. Merging these low-quality submodels introduces additional problems.

In order to achieve a high quality process model in little time, we have to find a decomposition that produces high quality submodels with as little overlap as possible (shared activities) and causal relations between them. Also, the submodels should preferably be of comparable size, because of the exponential nature of most process discovery algorithms [2]. Hence, a *good* decomposition should (1) *maximize* the causal relations between the activities *within* each cluster in the activity clustering, (2) *minimize* the causal relations and overlap *across* the clusters and (3) have approximately *equally sized* clusters. The challenge lies in finding a good balance between these three clustering properties. In subsection 4.1, we formally define these properties and provide metrics in order to be able to asses the quality of a given decomposition before using it to discover a model or check conformance with.

A clustering where one cluster is a subset of another cluster is not *valid* as it would lead to double work, and would thus result in an increase in required processing time without increasing (or even decreasing) model quality. Note that this definition of a valid clustering allows for disconnected clusters, and that some activities might not be in any cluster. This is acceptable as processes might consist of disconnected parts and event logs may contain noise. However, if activities are left out some special processing might be required.

Definition 11 (Valid Clustering).
Let $A \subseteq \mathcal{U}_A$ be a set of activities. Let $\widehat{C} \in \widehat{\mathcal{C}}(A)$ be a clustering over A. $\widehat{C}$ is a valid clustering iff: $\widehat{C} \neq \emptyset \wedge \forall_{C_1, C_2 \in \widehat{C} \wedge C_1 \neq C_2} C_1 \not\subseteq C_2$. $\widehat{\mathcal{C}}_{\mathcal{V}}(A)$ denotes the set of valid clusterings over A.

For example, the clustering shown in Figure 2 is valid, as $\{a, b, c\} \not\subseteq \{c, d\}$ and $\{c, d\} \not\subseteq \{a, b, c\}$.

4.1 Clustering Properties

We define decomposition quality notions in terms of clustering properties.

The first clustering property we define is *cohesion*. The cohesion of an activity clustering is defined as the average cohesion of each activity cluster in that clustering. A clustering with good cohesion (cohesion ≈ 1) signifies that causal relations between activities in the same cluster are optimized, whereas bad cohesion (cohesion ≈ 0) signifies that activities with few causal relations are clustered together.

Definition 12 (Cohesion).
Let $A \subseteq \mathcal{U}_A$ be a set of activities. Let $G = (V, E, w) \in \mathcal{G}(A)$ be a causal activity graph over A, and let $\widehat{C} \in \widehat{\mathcal{C}}_{\mathcal{V}}(A)$ be a valid clustering over A. The cohesion of clustering $\widehat{C}$ in graph G, denoted $Cohesion(\widehat{C}, G)$, is defined as follows:

$$Cohesion(\widehat{C}, G) = \frac{\sum_{C \in \widehat{C}} Cohesion(C, G)}{|\widehat{C}|}$$

$$Cohesion(C, G) = \frac{\sum_{(a_1,a_2) \in E \cap (C \times C)} w((a_1, a_2))}{|C \times C|}$$

For example:

- $Cohesion(\{a, b, c\}, G_1) = (0.88 + 0.75 + 0.88)/9 = 0.28$,
- $Cohesion(\{c, d\}, G_1) = 1.00/4 = 0.25$, and
- $Cohesion(\widehat{C}_1, G_1) = (0.28 + 0.25)/2 = 0.26$.

The second clustering property is called *coupling*, and is also represented by a number between 0 and 1. Good coupling (coupling ≈ 1) signifies that causal relations between activities across clusters are minimized. Bad coupling (coupling ≈ 0) signifies that there are a lot of causal relations between activities in different clusters.

Definition 13 (Coupling).
Let $A \subseteq \mathcal{U}_A$ be a set of activities. Let $G = (V, E, w) \in \mathcal{G}(A)$ be a causal activity graph over A, and let $\widehat{C} \in \widehat{\mathcal{C}}_{\mathcal{V}}(A)$ be a valid clustering over A. The coupling of clustering $\widehat{C}$ in graph G, denoted $Coupling(\widehat{C}, G)$, is defined as follows:

$$Coupling(\widehat{C}, G) = \begin{cases} 1 & \text{if } |\widehat{C}| \leq 1 \\ 1 - \frac{\sum_{C_1, C_2 \in \widehat{C} \wedge C_1 \neq C_2} Coupling(C_1, C_2, G)}{|\widehat{C}| \cdot (|\widehat{C}| - 1)} & \text{if } |\widehat{C}| > 1 \end{cases}$$

$$Coupling(C_1, C_2, G) = \frac{\sum_{(a_1,a_2) \in E \cap ((C_1 \times C_2) \cup (C_2 \times C_1))} w((a_1, a_2))}{2 \cdot |C_1 \times C_2|}$$

For example:

- $Coupling(\{a, b, c\}, \{c, d\}, G_1) = (0.75 + 0.88 + 1.00)/12 = 0.22$,
- $Coupling(\{c, d\}, \{a, b, c\}, G_1) = (0.75 + 0.88 + 1.00)/12 = 0.22$, and
- $Coupling(\widehat{C}_1, G_1) = 1 - (0.22 + 0.22)/2 = 0.78$.

Note that the weights of the causal relations are used in the calculation of cohesion and coupling. Relations of which we are not completely sure of (or that are weak) therefore have less effect on these properties than stronger ones.

The *balance* of an activity clustering is the third property. A clustering with good balance has clusters of (about) the same size. Like cohesion and coupling, balance is also represented by a number between 0 and 1, where a good balance (balance ≈ 1) signifies that all clusters are about the same size and a bad balance (balance ≈ 0) signifies that the cluster sizes differ quite a lot. Decomposing the activities into clusters with low balance (e.g. a k-clustering with one big cluster holding almost all of the activities and $(k-1)$ clusters with only a few activities) will not speed up discovery or conformance checking, rendering the whole decomposition approach useless. At the same time finding a clustering with perfect balance (all clusters have the same size) will most likely split up the process / log in places that "shouldn't be split up", as processes generally consist out of different-sized natural parts.

This balance formula utilizes the standard deviation of the sizes of the clusters in a clustering to include the magnitude of the differences in cluster sizes. A variation of this formula using squared errors or deviations could also be used as a clustering balance measure.

Definition 14 (Balance).
Let $A \subseteq \mathcal{U}_A$ be a set of activities. Let $\widehat{C} \in \widehat{\mathcal{C}}_{\mathcal{V}}(A)$ be a valid clustering over A. The balance of clustering $\widehat{C}$ denoted $Balance(\widehat{C})$ is defined as follows:

$$Balance(\widehat{C}) = 1 - \frac{2 \cdot \sigma(\widehat{C})}{||\widehat{C}||}$$

Where $\sigma(\widehat{C})$ signifies the standard deviation of the sizes of the clusters in the clustering $\widehat{C}$.

For example, $Balance(\widehat{C}_1) = 1 - (2 \cdot 0.5)/2 = 0.5$.

In order to assess a certain decomposition based on the clustering properties, we use a weighted scoring function, which grades an activity clustering with a score between 0 (bad clustering) and 1 (good clustering). A weight can be set for each clustering property, depending on their relative importance. A clustering with clustering score 1 therefore has perfect cohesion, coupling and balance scores, on the set weighing of properties.

Definition 15 (Clustering Score).
Let $A \subseteq \mathcal{U}_A$ be a set of activities. Let $G \in \mathcal{G}(A)$ be a causal activity graph over A, and let $\widehat{C} \in \widehat{\mathcal{C}}_{\mathcal{V}}(A)$ be a valid clustering over A. The clustering score (score) of clustering $\widehat{C}$ in graph G, denoted $Score(\widehat{C}, G)$, is defined as follows:

$$\begin{aligned} Score(\widehat{C}, G) = {} & Cohesion(\widehat{C}, G) \cdot \left(\frac{Coh_W}{Coh_W + Cou_W + Bal_W} \right) \\ & + Coupling(\widehat{C}, G) \cdot \left(\frac{Cou_W}{Coh_W + Cou_W + Bal_W} \right) \\ & + Balance(\widehat{C}) \cdot \left(\frac{Bal_W}{Coh_W + Cou_W + Bal_W} \right) \end{aligned}$$

where Coh_W, Cou_W, and Bal_W are the weights for Cohesion, Coupling, and Balance.

For example, if we take all weights to be 10, that is, $Coh_W = Cou_W = Bal_W = 10$, then $Score(\widehat{C}_1, G_1) = 0.26 \cdot (10/30) + 0.78 \cdot (10/30) + 0.5 \cdot (10/30) = 0.51$.

4.2 Recomposition of Activity Clusters

As described in subsection 3.5, creating a good activity clustering is essentially a graph partitioning problem. The causal activity graph needs to be partitioned in parts that have (1) good cohesion, (2) good coupling and (3) good balance. The existing *maximal* decomposition approach [3] often leads to a decomposition that is too decomposed, i.e. too fine-grained. Cohesion and balance of clusterings found by this approach are usually quite good, since all clusters consist of only a few related activities. However, coupling is inherently bad, since there's a lot of overlap in the activity clusters and there are many causal relations across clusters. This decomposition approach leads to unnecessary and unwanted overhead and potential decreased model quality. We thus want to find a possibly *non-maximal* decomposition which optimizes the three clustering properties.

Instead of applying or creating a different graph partitioning algorithm, we *recompose* the activity clusters obtained by the vertex-cut decomposition, since it is maximal (no smaller valid clustering exists [3]). The idea is that it is possible to create a clustering that has fewer, larger clusters, requiring less processing time to discover the final model, because overhead as well as cluster overlap are reduced. Additionally, model quality is likely to increase because of the higher number of activities in each cluster and the lower coupling between clusters.

For example, if we put all activities from event log L_1 in a single cluster $\{a, b, c, d\}$, yielding clustering $\widehat{C}'_1$, and use the same weights, then we would get the following scores:

- $Cohesion(\widehat{C}'_1, G_1) = (0.88 + 0.75 + 0.88 + 1.00)/16 = 0.22$,
- $Coupling(\widehat{C}'_1, G_1) = 1$,
- $Balance(\widehat{C}'_1) = 1$, and
- $Score(\widehat{C}'_1, G_1) = 0.22 \cdot (10/30) + 1 \cdot (10/30) + 1 \cdot (10/30) = 0.74$.

Clearly, as $0.74 > 0.51$, the chosen weights would lead to the situation where a single cluster $\{a, b, c, d\}$ would be preferred over two clusters $\{a, b, c\}$ and $\{c, d\}$.

There are often many ways in which a clustering can be recomposed to the desired amount of clusters, as shown in Figure 6. We are interested in the highest quality clustering of the desired size k, i.e. the k-clustering that has the best cohesion, coupling and balance properties. A k-clustering that has a high clustering score will very likely lead to such a decomposition.

In order to find a good decomposition in the form of a high-scoring clustering quickly, we propose two agglomerative hierarchical recomposition approaches, which iteratively merge clusters, reducing the size of the clustering by one each iteration. As the amount of k-clusterings for a given causal activity graph is finite, it is possible to exhaustively find the best k-clustering. However, for even moderately-sized event logs (in the number of activities) this is too resource- and time-consuming, as shown in [13]. Also, a semi-exhaustive "random" recomposition approach was implemented that randomly recomposes the clustering to k clusters a given amount of times and returns the highest-scoring k-clustering found. This method is used as a benchmark for the hierarchical recomposition approaches.

Proximity-based approach We propose an hierarchical recomposition approach based on proximity between activity clusters, where cluster coupling is used as the proximity measure (algorithm 1). The starting point is the clustering as created by the vertex-cut approach. We repeatedly merge the clusters closest to one another (i.e. the pair of clusters with the highest coupling) until we end up with the desired amount of clusters (k). After the k-clustering is found, it is made valid by removing any clusters that are a subcluster of another cluster, if such clusters exist. It is therefore possible that the algorithm returns a clustering with size smaller than k.

By merging clusters we are likely to lower the overall cohesion of the clustering. This drawback is minimized, as coupling is used as the distance measure. Coupling is also minimized. The proximity-based hierarchical recomposition approach however is less favored towards the balance property, as it is possible that -because of high coupling between clusters- two of the larger clusters are merged. In most processes however, coupling between two "original" clusters will be higher than coupling between "merged" clusters. If not, the two clusters correspond to parts of the process which are more difficult to split up (e.g. a loop, a subprocess with many interactions and/or possible paths between activities, etc.). Model quality is therefore also likely to increase by merging these clusters, as process discovery algorithms don't have to deal with missing activities, or incorrect causal relations introduced in the corresponding sublogs. A possible downside is that as the clustering might be less balanced, processing time can be slightly higher in comparison with a perfectly-balanced decomposition.

For example, Figure 7 shows the 2-clustering that is obtained by this approach when we start from the 4-clustering for event log L_2 shown earlier. Cluster 0 is merged with Cluster 3, and Cluster 1 with Cluster 2.

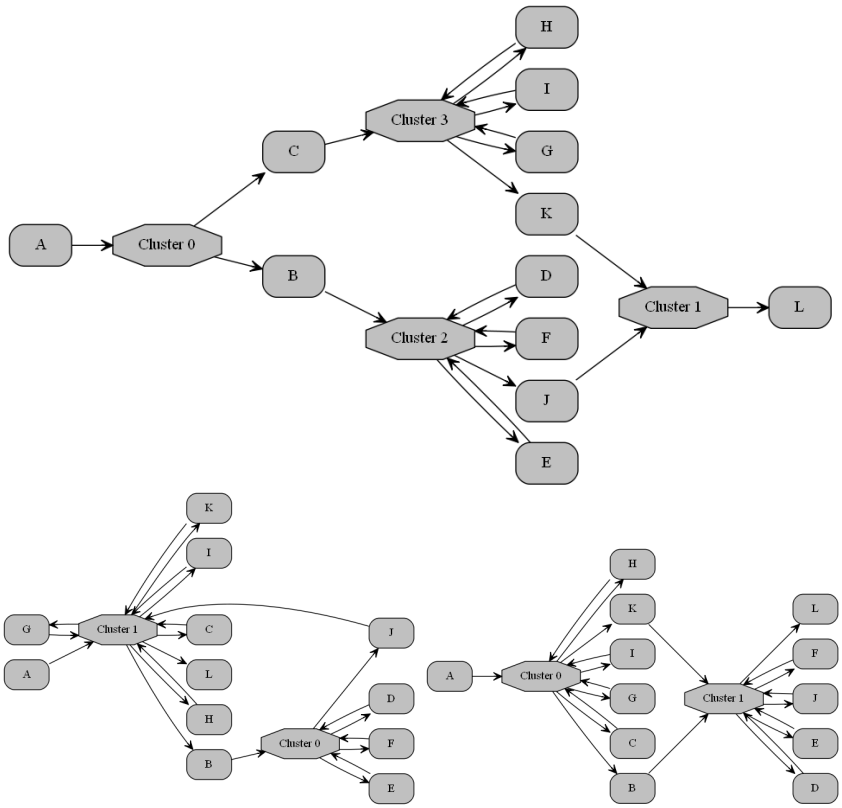

Fig. 6: 2 possible recompositions for event log L_2 (cf. Figure 5) from 4 (top) to 2 clusters (bottom left and right). Finding a good recomposition is key to creating a coarser clustering which could potentially decrease processing time and increase model quality.

Score-based approach We propose a second hierarchical recomposition algorithm that uses the scoring function in a look-ahead fashion (algorithm 2). In essence, this algorithm, like the proximity-based variant, iteratively merges two clusters into one. For each combination of clusters, the score of the clustering that results from merging those clusters is calculated. The clustering with the highest score is used for the next step. The algorithm is finished when a k-clustering is reached. Like in the proximity-based approach, after the k-clustering is found, it is made valid by removing any clusters that are a subcluster of another cluster, if such clusters exist.

The advantage of this approach is that specific (combinations of) clustering properties can be given priority, by setting their scoring weight(s) accordingly. For example, it is possible to distribute the activities over the clusters near perfectly, by choosing a high relative weight for balance. This would likely lead to a lower overall processing time. However, it might lead to natural parts of the

Algorithm 1: Proximity-based Agglomerative Hierarchical Recomposition

Input: $\widehat{C} \in \widehat{\mathcal{C}}_{\mathcal{V}}(A)$, $G \in \mathcal{G}(A)$, $k \in [1, |\widehat{C}|]$
Output: $\widehat{C}' \in \widehat{\mathcal{C}}_{\mathcal{V}}(A)$, $|\widehat{C}'| \leq k$
Result: The clustering $\widehat{C}'$ recomposed from $\widehat{C}$, into maximal k clusters.

```
begin
    Ĉ' ∈ Ĉ(A) ⟵ Ĉ
    while |Ĉ'| > k do
        highestcoupling ⟵ 0
        C_A ∈ Ĉ(A) ⟵ ∅
        C_B ∈ Ĉ(A) ⟵ ∅
        foreach C_1 ∈ C' do
            foreach C_2 ∈ C' \ {C_1} do
                if Coupling(C_1, C_2, G) > highestcoupling then
                    highestcoupling ⟵ Coupling(C_1, C_2, G)
                    C_A ⟵ C_1
                    C_B ⟵ C_2
        Ĉ' ⟵ Ĉ' \ {C_A, C_B} ⋃ {C_A ∪ C_B}
        Ĉ'' ∈ Ĉ(A) ⟵ Ĉ'
        foreach C ∈ Ĉ'' \ {C_A ∪ C_B} do
            if C ⊆ C_A ∪ C_B then
                Ĉ' ⟵ Ĉ' \ {C}
    return Ĉ'
```

process being split over multiple clusters, which could negatively affect model quality. A downside of this algorithm is that, as the algorithm only looks ahead one step, it is possible that a choice is made that ultimately leads to a lower clustering score, as that choice cannot be undone in following steps.

For example, Figure 8 shows the 2-clustering that is obtained by this approach when we start from the 4-clustering for event log L_2 shown earlier where all weights have been set to 10. Cluster 0 is now merged with Cluster 1, and Cluster 2 with Cluster 3.

4.3 Implementation

All concepts and algorithms introduced in this paper are implemented in release 6.5 of the process mining toolkit *ProM*[3] [24], developed at the Eindhoven University of Technology. All work can be found in the *ActivityClusterArrayCreator* package, which is part of the *DivideAndConquer* package suite. This suite is installed by default in ProM 6.5.

[3] ProM 6.5 can be downloaded from http://www.promtools.org

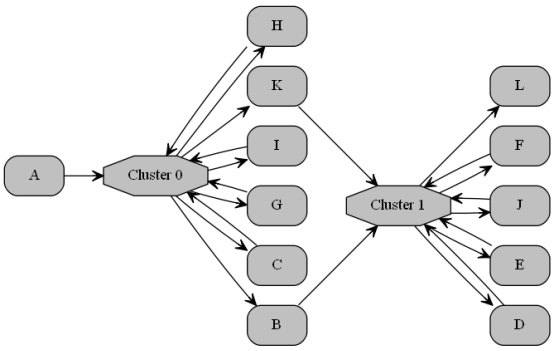

Fig. 7: Best 2-clustering found when starting from the 4-clustering shown at the top of Figure 6 using the Proximity-based approach.

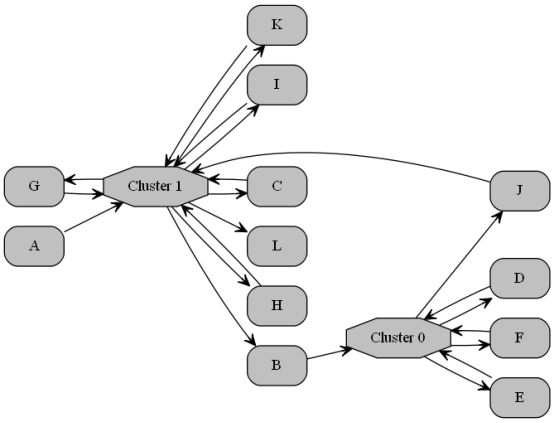

Fig. 8: Best 2-clustering found when starting from the 4-clustering shown at the top of Figure 6 using the Score-based approach with all weights set to 10.

The *plug-in* to use in ProM is *Modify Clusters*, which takes an activity cluster array (clustering) and a causal activity graph as input. The user can decide to either set the parameters of the action using a dialog, or to use the default parameter values. Figure 9 shows both options, where the top one will use the dialog and the bottom one the default parameter values. Figure 10 shows the dialog. At the top, the user can select which approach to use:

Brute Force:
Use a brute force approach to find the optimal k-clustering.

Incremental using Best Coupling:
Use the proximity-based approach to find a k-clustering.

Algorithm 2: Score-based Agglomerative Hierarchical Recomposition

Input: $\widehat{C} \in \widehat{\mathcal{C}}_{\mathcal{V}}(A)$, $G \in \mathcal{G}(A)$, $k \in [1, |\widehat{C}|]$
Output: $\widehat{C}' \in \widehat{\mathcal{C}}_{\mathcal{V}}(A)$, $|\widehat{C}'| \leq k$
Result: The clustering $\widehat{C}'$ recomposed from $\widehat{C}$, into maximal k clusters.

```
begin
    Ĉ' ∈ Ĉ(A) ⟵ Ĉ
    while |Ĉ'| > k do
        highestscore ⟵ 0
        foreach C1 ∈ Ĉ' do
            foreach C2 ∈ Ĉ' \ {C1} do
                Ĉ'' ⟵ Ĉ' \ {C1, C2} ⋃ {C1 ∪ C2}
                if Score(Ĉ'', G) > highestscore then
                    highestscore ⟵ Score(Ĉ'', G)
                    Ĉ''' ∈ Ĉ(A) ⟵ Ĉ''
        Ĉ' ⟵ Ĉ'''
    Ĉ' ⟵ Ĉ' \ {CA, CB} ⋃ {CA ∪ CB}
    foreach C1 ∈ Ĉ'' do
        foreach C2 ∈ Ĉ'' \ {C1} do
            if C1 ⊆ C2 then
                Ĉ' ⟵ Ĉ' \ {C1}
    return Ĉ'
```

Incremental using Best Coupling (only Overlapping Clusters):
Use the proximity-based approach to find a k-clustering, but allow only to merge clusters that actually overlap (have an activity in common). This is the default approach.

Incremental using Best Score:
Use the score-based approach to find a k-clustering.

Incremental using Best Score (only Overlapping Clusters):
Use the score-based approach to find a k-clustering, but allow only to merge clusters that actually overlap (have an activity in common).

Random:
Randomly merge clusters until a k-clustering is reached.

Below, the user can set the value of k, which is set to 50% of the number of existing clusters by default. At the bottom, the user can set the respective weights to a value from 0 to 100. By default, all weights are set to 100.

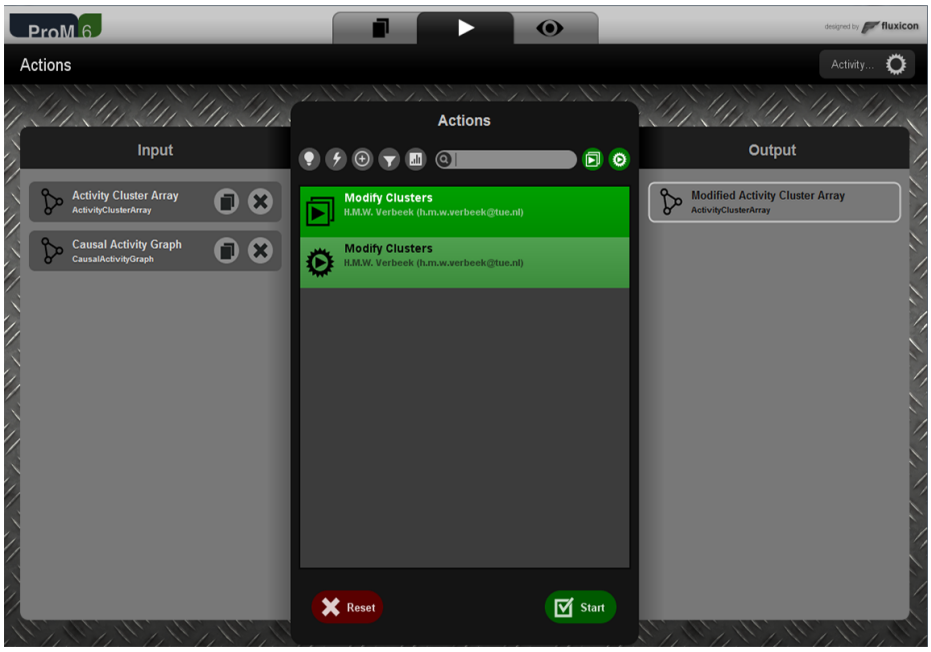

Fig. 9: The *Modify Clusters* actions in ProM 6.5.

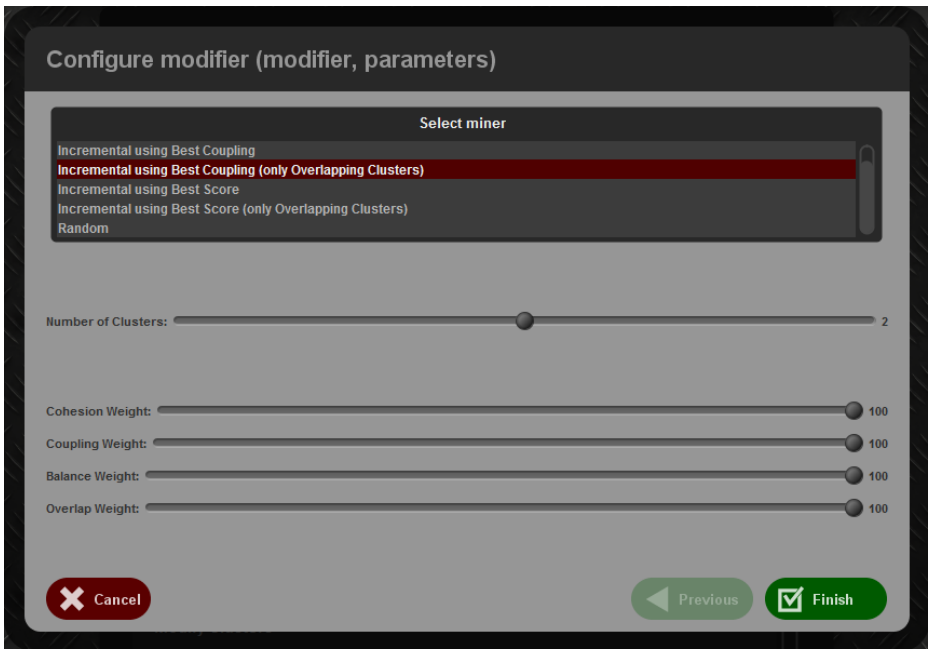

Fig. 10: The dialog that allows the user to set parameter values.

5 Use Case

The proposed recomposition techniques are tested using event logs of different sizes and properties. Results for an event log consisting of 33 unique activities, and 1000 traces are shown in this section. For this test the ILP Miner process discovery algorithm was used [25]. Discovering a model directly for this log will lead to a high quality model, but takes ∼25 minutes on a modern quad-core system [13]. The vertex-cut decomposed process mining approach is able to discover a model in roughly 90 seconds, however the resulting model suffers from disconnected activities (i.e. a partitioned model). The goal is thus to find a balance between processing times and model quality.

We are interested in the clustering scores of each algorithm when recomposing the clustering created by the vertex-cut approach to a smaller size. Exhaustively finding the best possible clustering proved to be too time- and resource-consuming. Therefore, besides the two approaches listed here, a random recomposition approach was used which recomposes clusters randomly one million times. The highest found score is shown as to give an idea of what the best possible clustering might be. Equal weights were used for the three clustering properties in order to compute the clustering scores. All clustering scores are shown in Figure 11. As can be seen, the vertex-cut approach creates 22 clusters. We can see that all algorithms perform very similarly in terms of clustering score. Only for very small clustering sizes the proximity-based approach performs worse than the other approaches, due to its tendency to create unbalanced clusters.

Besides clustering scores, we are even more interested in how each decomposition method performs in terms of required processing time and quality of the resulting process model. In Figure 12 we can see that decomposing the event log drastically reduces processing times. For an event log this size, the decomposition steps relatively takes up negligible time (see base of bars in figure), as most time is spent discovering the submodels (light blue bars). Processing times are reduced exponentially (as expected), until a certain optimum decomposition (in terms of speed) is reached, after which overhead starts to increase time linearly again.

We have included two process models (Petri Nets) discovered from the event log. Figure 14 shows the model discovered when using the vertex-cut decomposition. Figure 15 shows the model discovered when using the clustering recomposed to 11 clusters with the Proximity-based agglomerative hierarchical approach. We can see that in Figure 14, activity “10” is disconnected (marked blue). In Figure 15, this activity is connected, and a structure (self-loop) is discovered. We can also see that activities “9” and “12” now are connected to more activities. This shows that the vertex-cut decomposition sometimes splits up related activities, which leads to a lower quality model. Indeed, Figure 13 shows that the activities “9”, “10”, and “12” were split over two clusters in the vertex-cut decomposition (top), and were regrouped (bottom) by our recomposition. By recomposing the clusters we rediscover these relations, leading to a higher quality model. Processing times for these two models are comparable, as can be seen in Figure 12.

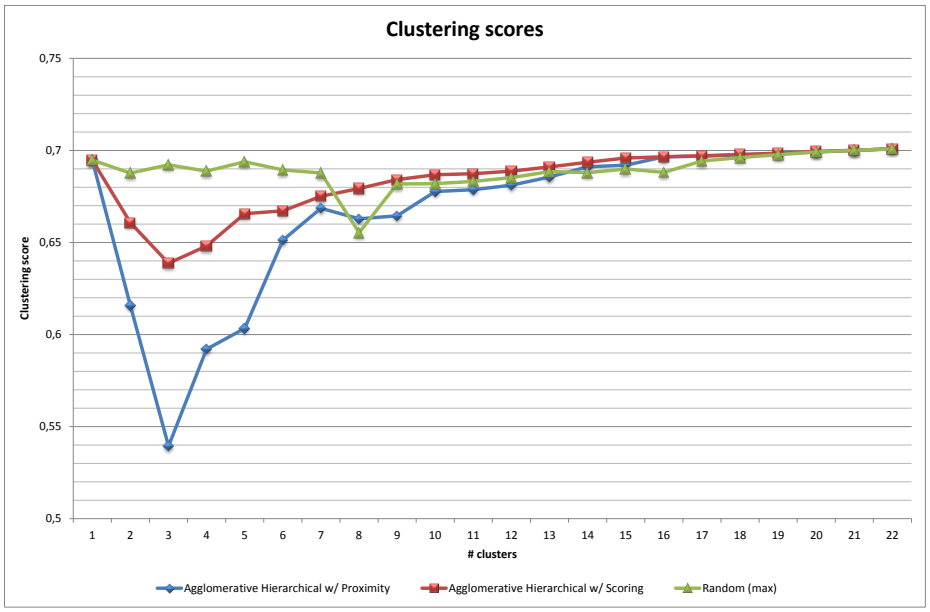

Fig. 11: Clustering score per recomposition algorithm.

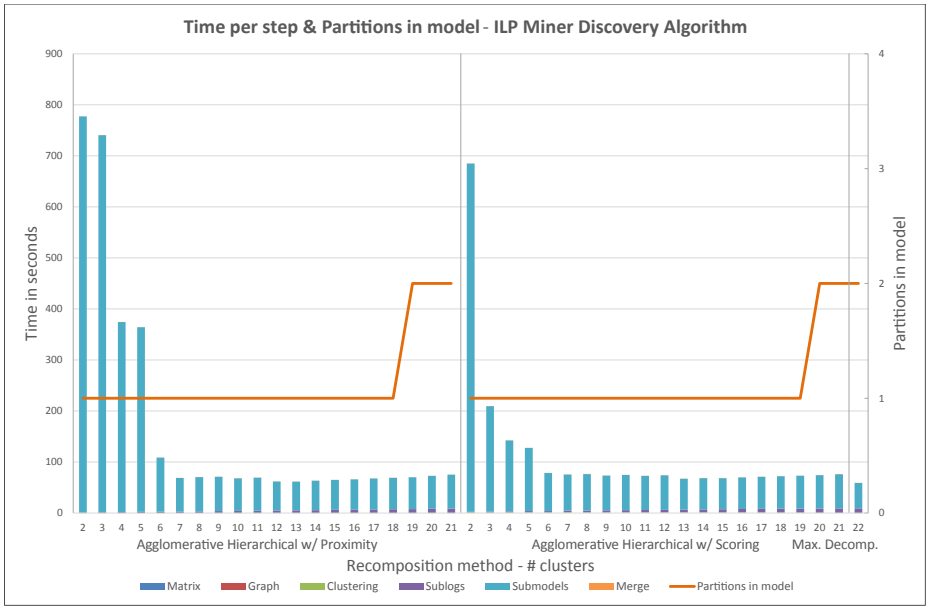

Fig. 12: Time per step & partitions in model using the Agglomerative Hierarchical recomposition approaches and the ILP Miner process discovery algorithm.

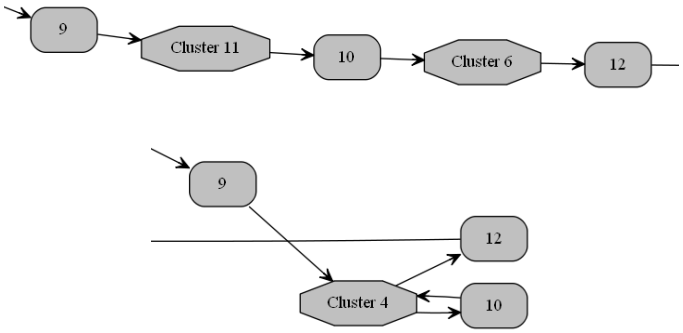

Fig. 13: Activities that were split over multiple clusters by the vertex-cut decomposition (top), that were regrouped by our recomposition (bottom).

The proposed agglomerative hierarchical recomposition algorithms are able to create activity clusterings that have good *cohesion*, *coupling* and *balance* properties in very little time. Often times the scores of these clusterings are almost as good as the scores of clusterings created by the much slower exhaustive approaches [13]. The results show that by creating such good clusterings, indeed a better decomposition can be made. Because cluster overlap and the amount of clusters are reduced, overhead can be minimized, reducing required processing time. This results in a comparable total required processing time from event log to process model, even though an extra step is necessary. The larger the event log, the higher the time gain. By recomposing the clusters, processes are split up less in places where they shouldn't be, leading to better model quality. Because the discovered submodels are of higher quality and have less overlap and coupling, the merging step introduces less or no unnecessary (implicit) or double paths in a model, which leads to improvements in precision, generalization and simplicity of the final model.

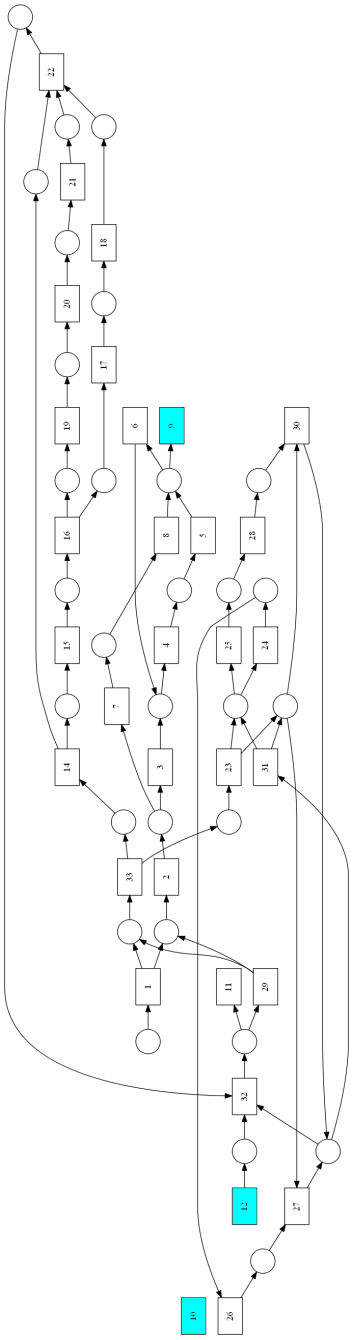

Fig. 14: Process model discovered using the vertex-cut decomposition. Some activities are disconnected in the final model.

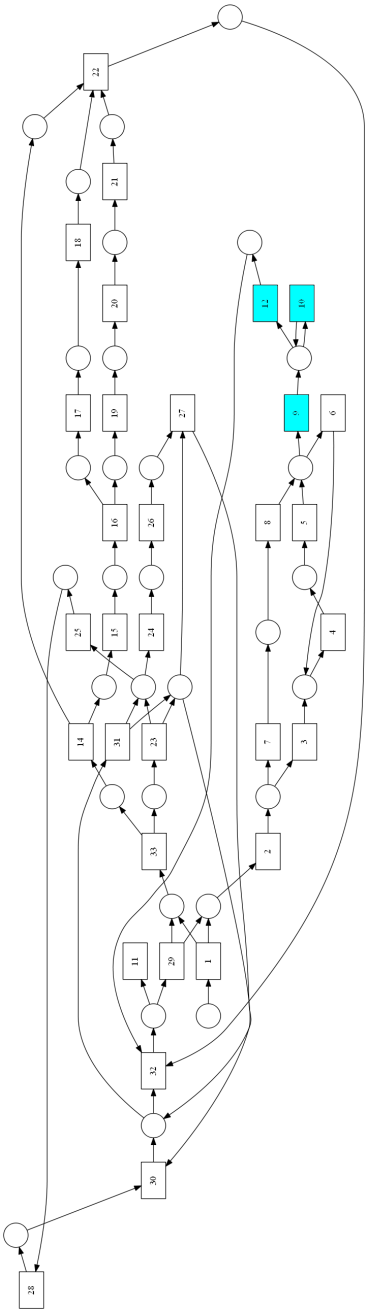

Fig. 15: Process model discovered using the vertex-cut clustering recomposed to 11 clusters. Previously disconnected activities are connected again, improving model quality.

6 Conclusions and Future Work

In decomposed process discovery, large event logs are decomposed by somehow clustering their events (activities), and there are many ways these activity clusterings can be made. Hence, good quality notions are necessary to be able to assess the quality of a decomposition before starting the time-consuming actual discovery algorithm. Being able to find a high-quality decomposition plays a key role in the success of decomposed process mining, even though the decomposition step takes relatively very little time.

By using a better decomposition, less problems arise when discovering submodels for sublogs and when merging submodels into the overal process model. We introduced three quality notions in the form of clustering properties: *cohesion*, *coupling* and *balance*. It was shown that finding a *non-maximal* decomposition can potentially lead to a decrease in required processing time while maintaining or even improving model quality, compared to the existing vertex-cut *maximal* decomposition approach. We have proposed two variants of an agglomerative hierarchical recomposition technique, which are able to create a high-quality decomposition for any given size, in very little time.

Even though the scope was limited to decomposed process discovery, the introduced quality notions and decomposition approaches can be applied to decomposed conformance checking as well. However, more work is needed to incorporate them in a conformance checking environment.

Besides finding a better decomposition, we believe improvements can be gained in finding a better, more elaborate algorithm to merge submodels into the overal process model. By simply merging submodels based on activity labels it is likely that implicit paths are introduced. Model quality in terms of fitness, simplicity, generality or precision could suffer. An additional post-processing step (potentially using causal relations) could also solve this issue.

Even though most interesting process discovery algorithms are exponential in the number of different activities, adding an infrequent or almost unrelated activity to a cluster might not increase computation time for that cluster as much as adding a frequent or highly related one. Therefore, besides weighing causal relations between activities in the causal activity matrix, activities themselves might be weighted as well. Frequency and connectedness are some of the many possible properties that can be used as weights. It might be possible that one part of a process can be discovered easily by a simple algorithm whereas another, more complex part of the process needs a more involved discovery algorithm to be modeled correctly. Further improvements in terms of processing time can be gained by somehow detecting the complexity of a single submodel in a sublog, and choosing an adequate discovery algorithm.

Finally, as discussed, the proposed recomposition algorithms expect the desired amount of clusters to be given. Even though the algorithms were shown to provide good results for any chosen number, the approach would benefit from some method that determines a fitting clustering size for a given event log. This would also mean one less potentially uncertain step for the end-user.

References

[1] van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin (2011) 1, 3, 4, 5

[2] van der Aalst, W.M.P.: Distributed Process Discovery and Conformance Checking. In: de Lara, J., Zisman, A. (eds.) FASE. Lecture Notes in Computer Science, vol. 7212, pp. 1–25. Springer (2012) 1, 2, 3, 10

[3] van der Aalst, W.M.P.: A general divide and conquer approach for process mining. In: Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on. pp. 1–10. IEEE (2013) 2, 3, 5, 8, 10, 13

[4] van der Aalst, W.M.P.: Decomposing Petri nets for process mining: A generic approach. Distributed and Parallel Databases 31(4), 471–507 (2013) 2, 3

[5] van der Aalst, W.M.P., Adriansyah, A., de Medeiros, A.K.A., Arcieri, F., Baier, T., Blickle, T., Bose, R.P.J.C., van den Brand, P., Brandtjen, R., Buijs, J.C.A.M., et al.: Process mining manifesto. In: Business process management workshops. pp. 169–194. Springer (2012) 3

[6] van der Aalst, W.M.P., Verbeek, H.M.W.: Process discovery and conformance checking using passages. Fundamenta Informaticae 131(1), 103–138 (2014) 3

[7] Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., Schulz, C.: Recent advances in graph partitioning. CoRR abs/1311.3144 (2013), http://arxiv.org/abs/1311.3144 8

[8] Cannataro, M., Congiusta, A., Pugliese, A., Talia, D., Trunfio, P.: Distributed data mining on grids: services, tools, and applications. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 34(6), 2451–2465 (2004) 3

[9] Carmona, J.: Projection approaches to process mining using region-based techniques. Data Min. Knowl. Discov. 24(1), 218–246 (2012), http://dblp.uni-trier.de/db/journals/datamine/datamine24.html 2, 3

[10] Carmona, J., Cortadella, J., Kishinevsky, M.: A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In: Business Process Management (BPM2008). pp. 358–373. Springer (2008) 2, 3

[11] Carmona, J., Cortadella, J., Kishinevsky, M.: Divide-and-conquer strategies for process mining. In: Business Process Management, pp. 327–343. Springer (2009) 3

[12] Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust Process Discovery with Artificial Negative Events. Journal of Machine Learning Research 10, 1305–1340 (2009) 2

[13] Hompes, B.F.A.: On Decomposed Process Mining: How to Solve a Jigsaw Puzzle with Friends. Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (2014), http://repository.tue.nl/776743 10, 14, 20, 22

[14] Kambatla, K., Kollias, G., Kumar, V., Grama, A.: Trends in big data analytics. Journal of Parallel and Distributed Computing 74(7), 2561–2573 (2014) 3

[15] Kim, M., Candan, K.: SBV-Cut: Vertex-cut based graph partitioning using structural balance vertices. Data & Knowledge Engineering 72, 285–303 (2012) 8
[16] Munoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Single-Entry Single-Exit decomposed conformance checking. Information Systems 46, 102–122 (2014), http://dx.doi.org/10.1016/j.is.2014.04.003 2, 3
[17] Park, B.H., Kargupta, H.: Distributed data mining: Algorithms, systems, and applications (2002) 3
[18] Reguieg, H., Toumani, F., Motahari-Nezhad, H.R., Benatallah, B.: Using mapreduce to scale events correlation discovery for business processes mining. In: Business Process Management, pp. 279–284. Springer (2012) 3
[19] Saha, B., Shah, H., Seth, S., Vijayaraghavan, G., Murthy, A., Curino, C.: Apache tez: A unifying framework for modeling and building data processing applications. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. pp. 1357–1369. ACM (2015) 3
[20] Shukla, R.K., Pandey, P., Kumar, V.: Big data frameworks: At a glance (2015) 3
[21] The Apache Software Foundation: Apache Flink: Scalable Batch and Stream Data Processing. http://flink.apache.org/ (7 2015), (website) 3
[22] Vanderfeesten, I.T.P.: Product-based design and support of workflow processes (2009) 3, 4
[23] Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al.: Apache hadoop yarn: Yet another resource negotiator. In: Proceedings of the 4th annual Symposium on Cloud Computing. p. 5. ACM (2013) 3
[24] Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: ProM 6: The process mining toolkit. In: Proc. of BPM Demonstration Track 2010. vol. 615, pp. 34–39. CEUR-WS.org (2010), http://ceur-ws.org/Vol-615/paper13.pdf 16
[25] van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: Applications and Theory of Petri Nets, pp. 368–387. Springer (2008) 20
[26] Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. vol. 10, p. 10 (2010) 3