Process Model Modularization by Subprocess Discovery

Sergio Angelastro
dept. of Computer Science
University of Bari
Bari, Italy
sergio.angelastro@uniba.it

Stefano Ferilli
dept. of Computer Science
University of Bari
Bari, Italy
stefano.ferilli@uniba.it

***Abstract*—Most companies exploit information systems to manage their business processes. Logs generated by such systems might be used to automatically learn models of such processes, e.g. for analysis and conformance checking purposes. Since logs are often not generated specifically for this purpose, the reported activities might be too fine-grained, leading to very complex and incomprehensible ('spaghetti') models. Modularization is a way to improve understandability and reusability of the models. This work proposes an approach to automatically discover modules, in the form of subprocesses, in unstructured processes, using the WoMan framework. Experimental results on synthetic data and models are promising.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

A *process* consists of *agents* carrying out *tasks* [1], [2]. An *activity* is the actual execution of a task by an agent. A process execution, called a *case*, can be described as a sequence of *events* (e.g., the start and end of the case, or of activities in the case [2]), associated with *steps* (time points) and collected in *traces* [3]. To manage their activities, most companies exploit information systems that, while executing cases of their business process, record the corresponding traces in *event logs*. However, often these systems are not driven by, or even aware of, the underlying intended process model. A *process model* (or *workflow*) formally specifies a process. In these environments, one might want to discover such models (e.g., in order to analyze and improve them).

Process Discovery aims at automatically learning process models from logs. The learned model should be *complete*[1], *irredundant*[2], and *minimal*[3]. While a model's *accuracy* is proportional to completeness and irredundancy, minimality pertains to effectiveness and efficiency. To help human understandability, process discovery techniques should provide compact models. However, compactness may be affected by the domain's complexity. In these cases, producing event logs or process models at an appropriate level of abstraction is of utmost importance. Unfortunately, since logs are often not generated specifically for process discovery, the reported activities might be too fine-grained. When used by most existing process discovery techniques (e.g., [3]–[5]), that produce 'flat' models (i.e., expressed in terms of the basic tasks found in the logs), this leads to very complex and incomprehensible ('spaghetti') models, which are of little value. In these cases, a possible solution is to raise the abstraction level by *modularization*, aimed at yielding more structured ('lasagna-like') models.

This paper focuses on modules in the form of subprocesses. A *subprocess* is an encapsulation of activities representing a coherent complex logical unit of work, having its own attributes and goals, that contribute to achieve the main goal. A single activity is its minimal expression [6]. Just like activities, subprocesses can be composed to obtain a higher-level process by means of the usual control-flow constructs [7]: **sequence**, if occurring one after the other; **choice** (XOR or OR), if arranged as options in a decision point; **parallelism**, if occurring simultaneously; **iteration**, if repeated many times. A subprocess, being itself a process, can be in turn decomposed into subprocesses, resulting in a hierarchy where higher levels are more abstract and leaves are atomic activities. More specifically, we investigate subprocess discovery in unstructured processes, where a subprocess is a frequent behavior which is common to 'sufficiently many' process executions (*behavioral pattern* [8]). Our proposal is based on the WoMan [9] (Workflow Management) Process Mining framework. We evaluate the proposed approach by using a Ground Truth, to assess whether the framework can provide valuable candidates subprocess.

Sections II and III report related works and the specific framework adopted. Then, Section IV presents our proposal for automated modularization. Section V reports the evaluation of performance and draws some conclusion. Finally, Section VI concludes the paper.

II. Background and Related Work

Process models are usually expressed as graphs, where vertexes represent tasks and edges causal connections among tasks. Albeit several specialized formalisms have been proposed in the literature, Petri Nets, and their restriction Workflow Nets (WF-nets, for short) [7], are classical representations used for process models. *Declarative* process mining approaches [10] learn models expressed as a set of constraints, rather than a monolithic model.

[1]Should account for all event sequences in the log.
[2]Should generate as few event sequences not in the log as possible.
[3]Should be as simple and compact as possible.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

While suitable for application to operational and well-structured processes, process discovery techniques have limitations when facing unstructured ones, often yielding 'spaghetti'-like or complex process models. An event log may contain different traces showing similar execution behaviors (*patterns*), i.e., manifestations of larger subflows of activities. The discovery of common patterns of activities in traces (beyond the immediate succession relation), i.e. *subprocesses*, can help in these cases. Only few approaches presented in the literature to deal with this problem, perform subprocess analysis. In the following we report all the works, to the best of our knowledge, aimed at dealing with process model complexity by finding subprocesses.

Trace cluster analysis has been exploited as a pre-processing task. Groups of low-level events are clustered into high-level events, or similar traces are clustered, in order to solve the problem of unstructured processes. E.g., [11]–[13] tackle the case in which the event log is not originated from a single process, and use this approach to prevent mixing different usage scenarios into one unstructured process. However, not all types of data exhibit a cluster tendency.

Contrarily to trace clustering, model clustering is almost new. To the best of our knowledge, only [14], [15] adopted this approach. [14] mines a set of frequent patterns from a collection of process models, suitably transformed into real-valued vectors in order to apply an agglomerative clustering algorithm. However, results are poor in trying to discover similar subprocesses. [15] adopts a similar approach, using a graph clustering technique. The main drawback of these approaches is that process models are not always available or already structured.

In [16], [17] authors proposed the discovery of hybrid process models (procedural for structured, and declarative for unstructured), meant as hierarchical ones. Each imprecise part (subprocess) in the hierarchy is specified in a declarative fashion. Experiments have shown that hybrid models can be more precise (no over fit). A technique for discovering a process-subprocess hierarchy from a log is presented in [18]. Given this hierarchy and the projected logs associated with each node in the hierarchy, the parent process and subprocess models are then discovered (using techniques for flat process discovery) and analyzed (for boundary markers). They validated the models with a synthetic log and two real life logs, showing that they are more precise and less complex than those derived with flat process discovery techniques. [19] proposed an approach to decompose process models by using abstraction criteria based on quantitative measures. Given a slider threshold, all the elements below the threshold are filtered out. The higher the threshold, the more compact the model. An obvious limitation of this approach is that, to provide effective threshold values, users should understand thoroughly the process model. [20] proposed some guidelines for selecting parts of process models for modularization. [21] proposed to group low level events into higher level activities, i.e., events abstraction, to be used to discover better (high quality) process models. They aim to discover local process models and, then, use these models to lift the event log to a higher level of abstraction. In fact, preliminary results on different real-life event logs have shown that fitness and precision scores are more balanced. [22] proposed Post Sequential Patterns Mining (PSPM) which takes as input patterns of sequences, discovered by any sequential pattern mining algorithm, and merges them in a single graph. Unfortunately, only sequence and choice, not parallelism, are considered as constructs. Manifestations of patterns ('commonly used model constructs') within a trace or across traces have been explored in [23]. These patterns are then used to pre-process (i.e., to abstract) the log. While for sequential patterns the abstracted sequences must be equal, for more complex constructs (parallelism or choice) they may be just approximately similar. Results in the health care domain are promising, reducing the spaghetti-ness of the mined models. The approach is suitable for hierarchical process mining (as in [24]), trace clustering and fault diagnosis.

[6] proposed a method, based on trace alignment as defined in [25], to segment aligned traces and form representative groups of subprocesses. The authors create and exploit the tree of building blocks, i.e., subprocesses, each represented by its matrix of trace alignment. The process is decomposed according to 4 constructs. This technique allows to detect interesting patterns, which can be framed in different contexts, providing a holistic view of the process. It also allows to detect subprocesses and their dependencies which compose the process. Authors in [26] proposed an extension to well-known process mining techniques to handle multi-instantiation (MI) of subprocesses, i.e., several instances of a subprocess being executed concurrently, for both discovery and conformance checking. It defined strategies to extract subprocess identifiers and make them explicit in a pre-processing step. This step allows to hierarchically discover process models with MI. First, based on subprocess IDs, log splitting is performed, obtaining a *high-level* (collapsed subprocess) and a *low-level* (with explicit subprocesses) process. Then, standard process discovery techniques can be separately applied for each trace. In the end, process models are merged by replacing the collapsed subprocesses in the high-level process with the corresponding expanded subprocesses. While producing more structured and accurate models than classical approaches, this approach requires expert knowledge to annotate subprocesses in the event log with identifiers. [8] takes into account event logs in which parallelism is explicitly expressed (as in [9], [27]) in order to represent traces in the form of instance graphs on which (occurrence-based) frequent subgraph mining techniques are applied to extract behavioral patterns. The dataset of instance graphs undergoes a repair procedure with respect to a process model (deleting superfluous relations, or inserting missing ones). Then, hierarchical cluster analysis is applied using SUBDUE (an inexact graph matching technique) to extract a lattice of frequent substructures. The main drawback of this approach is that it cannot provide subprocesses involving choices or loops.

Summing up, several techniques were developed to gain an insight of less structured processes (where traditional methods

fail), and to extract sub-processes from them. Unfortunately, some of them cannot deal with event logs where some frequent patterns are surrounded by random events. A common issue when dealing with complex sub-structures is that modules are just approximations of the original fragments. Other works reduce complexity without using subprocesses or requiring many process models to start the discovery. A fair comparison is actually infeasible, since the proposed methods either require a completely different type of input (a collection of graphs), or support only a part of the known constructs. Our contribution is aimed at mining sub-processes, although being approximations, involving all the constructs.

III. The WoMan Framework

WoMan [9] is a Declarative Process Mining framework that pervasively uses First-Order Logic (FOL) techniques and representations for enhanced expressiveness (e.g., allowing one to describe contextual information using relationships). Experiments proved that its techniques can handle efficiently and effectively very complex processes (involving a very large number of tasks, high concurrency, duplicate and hidden activities, and short or nested loops). It is *fully incremental*: not only can it refine an existing model according to new cases whenever they become available, it can even start learning from an empty model and a single case, while other approaches need a (large) number of cases to draw significant statistics before learning starts. This allows continuous adaptation of the learned model to the actual practice efficiently, effectively and transparently to users [9].

WoMan's input formalism allows traces to report both start and end time of activities. So, parallelism can be inferred by just evaluating the time span overlapping between activities. If activity start and end times are not available, activities are considered as instantaneous. It consists of 6-tuples $\langle T, E, W, P, A, O[, R]\rangle$ where T is the event timestamp, E is the type of the event (one of 'begin_process', 'end_process', 'begin_activity', or 'end_activity'), W is the name of the reference workflow, P is the case identifier, A is the name of the activity, and O is the progressive number of occurrence of A in case P, and R (optional) is the agent performing A.

Each input trace is transformed into an internal representation formalism based on two predicates:

`activity(S,A,R)` :
activity A, performed by agent R, occurred at step S;

`next(S',S'')` :
step S'' directly follows step S'.
Case representations are handled separately, so there is no need to report the case identifier in these predicates.

Let $\mathcal{C}$ be a set of training cases to learn a process model. WoMan's output representation formalism specifies process models mainly using two kinds of declarative constraints:

`task(`t`,`C_t`)` :
task t was observed in training cases $C_t \subseteq \mathcal{C}$;

`transition(`I`,`O`,`p`,`C_p`)` :
transition p (often denoted $p : I \Rightarrow O$), observed in training cases $C_p \subseteq \mathcal{C}$, involves two multisets of tasks[4], *input* tasks $I = [i_1, \ldots, i_n]$ and *output* tasks $O = [o_1, \ldots, o_m]$. It is enabled if all tasks in I are running. It occurs when, after stopping the execution of all tasks in I (in any order), the execution of all tasks in O is started (again, in any order). A transition is completed when the execution of all of its output tasks terminates. For clarity, a model might involve both $A \Rightarrow B$ and $B \Rightarrow A$, each one carrying its own relative frequency (which can be exploited for filtering strategies).

Transitions can be seen as 'consumers' of their input tasks, and 'producers' of their output tasks. In this perspective, the completion of an activity during a case can be seen as the production of a resource that is supposed to be consumed by some transition. WoMan tracks which transitions consume resources produced by which transitions. This relationship is expressed in the models using the following predicate:

`transition_provider(`$[p_1, \ldots, p_n]$`,`p`,`q`)` :
transition p, involving input tasks $I = [i_1, \ldots, i_n]$, is enabled provided that each task $i_k \in I$ was produced as an output of transition p_k; several combinations of transition providers can be allowed, numbered by progressive q. It partitions the input multiset of a transition according to the producers of the activities to be consumed.

Example. Consider a model including, among others, the following transitions:
$p_1 : [x, y, z] \Rightarrow [a, b]$
$p_2 : [x, y] \Rightarrow [a]$
$p_3 : [x] \Rightarrow [a, d]$
and the following constraints related to those transitions:
`transition_provider(`$[p', p', p'']$`,`p_1`,`$1`)` .
`transition_provider(`$[p', p'', p'']$`,`p_1`,`$2`)` .
`transition_provider(`$[p', p'', p']$`,`p_1`,`$3`)` .
`transition_provider(`$[p', p']$`,`p_2`,`$1`)` .
`transition_provider(`$[p']$`,`p_3`,`$1`)` .
Suppose that the current set of activities to be consumed is $\{x, y, z\}$, where x and z were produced by p', and y was produced by p''. So, p_1, p_2 and p_3 are enabled. If an activity a is started, any of the above transitions might in principle be the consumer. However, based on the consumer-producer constraints, transition p_2 is not a valid consumer, since it requires that both x and y are produced by transition p', while in our case they were produced by transitions p' and p'', respectively. Conversely, the third constraints for transition p_1 is compliant with the available producers, which makes it an eligible candidate. Also transition p_3 is enabled.

`transition_provider` constraints allow us to map each case to a graph where nodes represent transitions, and directed edges connect producer-consumer transitions.

WoMan performs all classical process mining tasks. Process model *discovery* is carried out by the learning module **WIND** [9] (Workflow INDucer). It may also learn from standard log formats for case representation, and *import/export* workflow models from/into standard formalisms (Petri nets). Module **WEST** [28] (Workflow Enactment Supervisor and

[4]I.e., several instances of a task can be running at the same time.

Trainer) performs *conformance checking* of an event log with a process model. The learned models can be used for process *simulation* by module **SWAP** (Simulate Workflow Arbitrary Process), that runs a workflow and generates any of the possible cases from which it was learned (see [9] for more details).

IV. Process Model Modularization Framework

As said, WoMan assumes that parallelism among activities in a case is explicitly represented in the event log. WoMan's internal representation of a case (in terms of `activity/3` and `next/2` predicates) can be represented as a labeled *Directed acyclic graph* (DAG), where nodes represent time steps, labeled with activities occurring at those time steps, and edges represent the immediate succession relation between time steps. This representation is compliant with the concept of *instance graph* in the literature, used to represent a process case as a DAG [27]: edges outgoing from a node indicate an *and-split*, and edges incoming to a node an *and-join*. Instead of traditional instance graphs, we propose to exploit higher level DAGs for subprocess mining in WoMan, where nodes represent transitions and edges represent the *provider-consumer* relation between transitions, expressed by `transition_provider` constraints in Section III. The main advantage is that, since transitions naturally express *and-splits/joins* through their I and O sets, we don't need to explicitly represent them as edges. As explained in [29] (see Prototyped Process Discovery section), we want to clarify that each case trace can have just one possible DAG encoding (both traditional and higher level). Conversely, each DAG may refer to many different traces where only the execution order of the concurrent activities varies.

Using this representation, the problem of finding common patterns can be cast as a Frequent Subgraph Mining (FSM) task, aimed at extracting all the subgraphs whose *support* exceeds a given threshold in the training dataset. The extracted subgraphs can be then grouped by similarity, using cluster analysis techniques (similar subgraphs representing variants of the same subprocess). In the following we describe in more detail the processing steps for this approach.

A. Processing Pipeline

Given an event log $\mathcal{L}$, we first learn the corresponding process model M (including *and-split/join* constructs), to be modularized, by running WoMan's WIND module. Then, we apply the following pipeline of steps (i.e., each step in the pipeline takes the output of the previous step as input and provides its output to the next one):

1) **Transformation** of $\mathcal{L}$ into a database $\mathcal{D}$ of graph data objects. Each case trace t in the event log $\mathcal{L}$ is expressed as an instance graph i based on the `next/2` and `activity/3` predicates. Then, the set of `transition/4` atoms is extracted from i by applying a procedure inspired to Algorithm 2 in [9], and the `transition_provider/3` constraints for these atoms are computed. `transition/4` and `transition_provider/3` atoms represent, respectively, the nodes and arcs of the high level DAG for t. The declarative formalism is kept in each node. $\mathcal{D}$ is the set of such DAGs.
2) **Frequent Subgraph Mining (FSM)** applied on $\mathcal{D}$, using a given *support* threshold, in order to obtain the set $\mathcal{S}$ of subgraph patterns.
3) **Repairing** of each $s \in \mathcal{S}$ into a subgraph s' that has single source and sink nodes, as required by (sub-)processes. Indeed, the FSM step may return subgraphs with many source and/or sink nodes. If s does not have a single source node and a single sink node, the repairing procedure works by finding its 'edge boundary', ensuring that s' has single source and sink nodes. An edge boundary of a graph s is a set of edges (representing in turn a graph), having just one start and end point[5]. So the procedure is aimed at reconstructing s by adding missing edges needed to complete and lead it to have just one terminal node, i.e., entry and exit point. Formally, given the set of nodes S of s (graph to be repaired), and the set of nodes T of all process cases (expressed as graphs) from which s was mined, the aim is to extract all those edges (u, v) needed to complete the graph, such that $u \in S$ and $v \in T$. Let us call $\mathcal{S}'$ the resulting set of repaired patterns.
4) **Cluster Analysis** of $\mathcal{S}'$, to partition it into k groups, each of which should contain semantically similar subgraphs, i.e., given two subgraphs, the more different they are, the less likely they represent the same subprocess. So, subgraphs in a cluster are meant as variants of the same subprocess. Note that, currently, we cannot guarantee that clusters are non-overlapping (i.e., that they don't share transitions). Let us call $\mathcal{C}$ the resulting clustering.
5) **Modularization**, by transforming each $c \in \mathcal{C}$ into a module (i.e., a subprocess model). Indeed, each cluster contains a group of patterns that are variant executions of the same subprocess, and so they should be generalized into one sub-process model. We do this by extracting and merging transitions and tasks (obtained from transitions) from all subprocesses in c. Let us call $\mathcal{M}$ the set of extracted subprocess models. As discussed in [15], desirable features of a good clustering are: *few* and *big* clusters, i.e., *large coverage* and *good generality*, and *minimal/no overlap*, i.e., better defined concepts. To pursue these objectives, only a subset $\mathcal{M}' \subseteq \mathcal{M}$ of non-ovelapping modules is taken into account for modularization, choosing those that do not overlap with each other and that are optimal based on some criterion defined over M (e.g., a subprocess minimizing the Description Length of M).
6) **Abstraction** of M. Each $m \in \mathcal{M}'$ abstracts a (different) portion of M by just replacing the corresponding tasks and transitions in M with a non-atomic task.

[5]It means that there exist just two nodes, in the graph, respectively with in-degree and out-degree equals to 0.

B. Tools and Technologies

In our implementation of the above procedure, the Frequent Subgraph Mining task was performed by the *gSpan* algorithm [30], which has been proven to outperform other state-of-the-art algorithms [31]. It is an exact search algorithm, which means that a complete mining is performed, and it makes use of a depth first search (DFS) with a pattern-growth strategy. The aim of *gSpan* is to extract all the subgraphs in a database occurring a number of times greater than a support threshold δ. Since δ is domain-dependent, we will test different thresholds.

For the Cluster Analysis step, we adopted two kinds of algorithms:

- *HAC* (the *Hierarchical Agglomerative Clustering*) produces the dendrogram of the clusters. The criterion to choose the pair of clusters to merge at each step is based on *Ward's minimum variance* method [32] (objective function). The optimal number k of clusters of $\mathcal{S}$ is automatically determined by applying the *elbow* method [33], which cuts-off the dendrogram. It gives us a way to obtain an appropriate level of granularity for process decomposition.
- *DBSCAN* (the *Density-Based Spatial Clustering of Applications with noise* [34]) is a non-parametric algorithm, and one of the most common clustering algorithms. The optimal number k of clusters of $\mathcal{S}$ is automatically selected by choosing the one maximizing the *average silhouette* (ranging between -1 and 1) over a range of possible values of k.

For computing the pairwise distance between graphs, we adopted a symbolic approach based on *maximum common subgraph* [35] (MCS). [36] proved that the MCS distance measure yields the best performance among the symbolic graph measures proposed in the literature.

V. Experiments

In the following experiment, we evaluate the pipeline up to the *Modularization* task using a ground truth, in order to assess whether it can be effective for the discovery of subprocesses, i.e., suitable candidate modules. Then, applying the *Abstraction* task is quite trivial after selecting the best candidate modules.

A. Dataset and Ground Truth

We validated our approach by running it on a publicly available benchmark dataset[6] (as far as we know, the only publicly available dataset with a ground truth) concerning a large bank transfer process. Specifically, it describes a realistic transaction process within a banking context. The dataset provides both the original and the decomposed model, which was used in [37] for decomposed conformance checking (of course, this is applicable only if a modularized process model is already available). To the best of our knowledge, it was not used for subprocess discovery or process abstraction so far.

[6]https://doi.org/10.4121/uuid:c1d1fdbb-72df-470d-9315-d6f97e1d7c7c

TABLE I
Dataset and Model statistics.

Cases	Events		Act		Task		Trans	
	all	avg	all	avg	all	avg	all	avg
2000	262960	131.48	129480	64.74	113	0.06	108	0.05

The process involves all monetary checks, authority notifications, and logging mechanisms compliant with the new degree of responsibility and accountability that current economic environments demand. Fig. 1 shows the high-level overview of the complete process, composed by 8 subprocesses. The process is structured as follows: it starts with a request for a new transaction, opening a new instance in the system and registering all the components involved (**ORT** for short). The second step is checking the person (or entity) originating the monetary transaction (**CS**). Then, the actual payment is processed differently, depending on whether it is in cash (**PCaP**), cheque (**PChP**) or eletronic form (**PEP**). Later, the receiver is checked (**CR**) and the money is transferred (**TM**). Finally, the process ends by recording the transaction information, notifying it to the required actors and authorities, and emitting the corresponding receipt (**NCT**). The process is characterized by a peculiar mix of features, in addition to parallelism (involving up to 3 activities), such as some short loops and optional activities.

First of all, using WoMan's *import* functionalities, we translated the original event log (stored in the XES[7] standard format), including 2000 traces[8], into WoMan's input formalism, and the complete model (i.e., the low-level Petri Nets shown in Fig. 2[9]) into WoMan's process model formalism (expressed in terms of `transition/4` atoms).

Table I reports statistics on the experimental dataset: on both the event log (number of cases and number of events and activities) and the imported process model (number of tasks and transitions), also on average per case. The number of events is in the hundreds thousands, and more than twice the number of activities (since each activity is associated to two events —start and end—, plus the start and end of cases). Other sources of complexity are a large number of tasks, and a large number of activities per case (about 64). However, WoMan has already been able to deal with these kinds of complexity. The number of transitions is comparable to the number of tasks, denoting low variety of combinations. The average number of tasks and transitions per case is very low (also due to the large number of activities per case), so we expect little variability between cases.

B. Evaluation

We evaluated quantitatively the applicability of a part of the framework's pipeline on a flattened process model,

[7]http://www.xes-standard.org/

[8]All having the start and the completion time for each activity (to compute the temporal overlap).

[9]By Munoz-Gama, J. from https://data.4tu.nl/repository/uuid:c1d1fdbb-72df-470d-9315-d6f97e1d7c7c

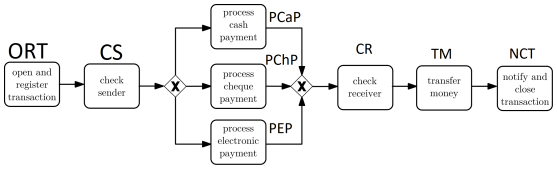

Fig. 1. High-level overview of the running example process, structured in subprocesses.

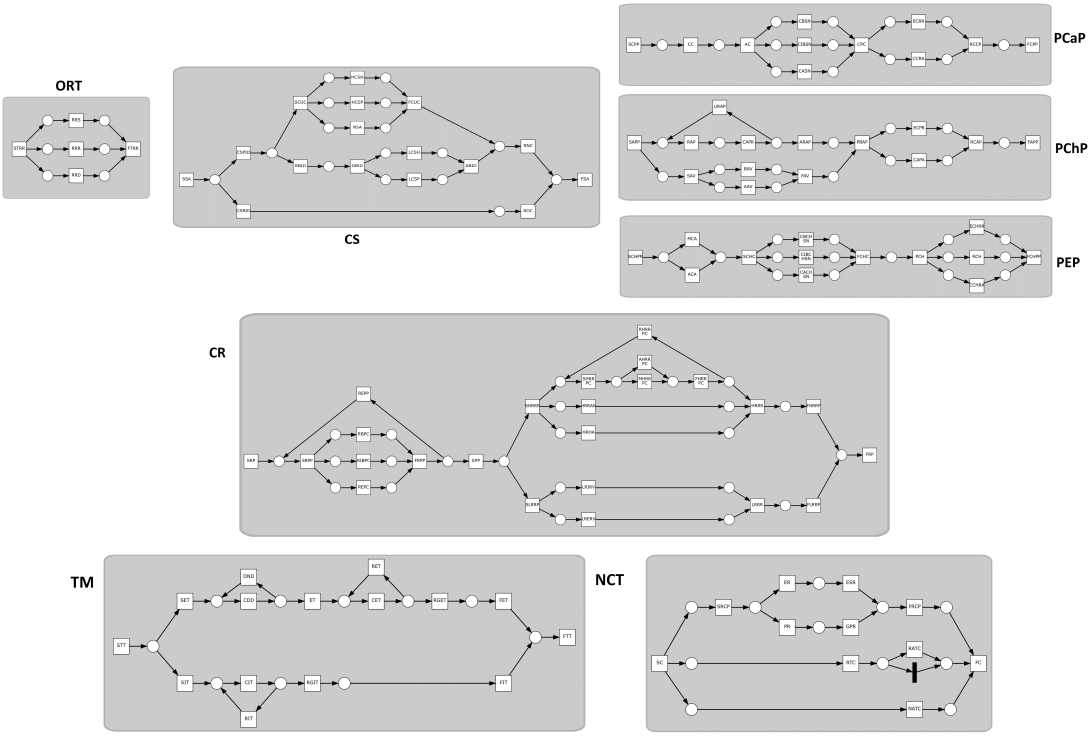

Fig. 2. Low-level overview of the running example process.

in order to obtain a subprocess decomposition. We experimented several support threshold values ({100, 300, 500, 700, 900}) for the *gSpan* algorithm. So, in this experimental setting we present performance for threshold values $\delta = \{100, 200, 300, 400, 500\}$. We purport to assess how the framework behaves as long as more frequent, but also less (in terms of number), patterns are required. Concerning cluster analysis algorithms, we adopted DBSCAN and HAC (denoted, respectively, by *DB* and *H* for the sake of compactness). Thence, the number of different setting combinations is 10 (5 thresholds for 2 algorithms).

Inspired by the overlap-based *most relevant match* presented in [38], we evaluated the goodness of our approach by comparing the structure of a gold standard subprocess with each mined (automatically retrieved) subprocess, and choosing as the 'most relevant' the one that reported the highest overlapping. Since the problem we tackled is one-shot, the evaluation does not involve any statistical inference. The overlap was measured by adopting the FOL similarity framework proposed in [39] to the graphs expressed using `transition/4` and `transition_provider/3` atoms. This ensures that not only the activities, but also the flow is the same in the compared graphs. In practical applications, where no golden standard will be available, the best candidate modules will have to be selected based on the state-of-the-art measures: completeness (or coverage), representativeness, frequency, and diversity.

A subprocess can be represented as the set of nodes it contains. While an automatically retrieved subprocess might not have some relevant nodes (w.r.t. the gold standard subprocess), one can consider to not reject it. For example, let us suppose that the node set $\{a, b, c, d, e\}$ is a target subprocess. If the set $\{a, b, c, d\}$ is mined, while not being a perfect match, it still provides useful information to the process analyst. Indeed, the subprocess can be manually examined and adjusted, instead of rejecting it and not having any information at all.

Table II reports the various settings (*Settings*) on the row headings, and corresponding performance and measures on the columns. Columns *Subprocesses* report the number of discovered clusters (column *#c*) with the relative silhouette score (*Sil*). In the following columns, for each gold standard subprocess, the overlap ratio between its most similar mined

```
transition([FIT]-[FTT],[1,2,...,1996,2000],10).
transition([FET]-[FTT],[3,...,1998,1999],70).
transition([FTT]-[SC],[1,2,3,...,1998,1999,2000],9).
transition([SC]-[NATC,RTC,SRCP],[1,2,3,...,1998,1999,2000],8).
transition([FRCP,NATC,RTC]-[FC],[1,2,3,...,1998,1999,2000],102).
transition([RTC]-[RATC],[1,2,3,4,5,6,7,8,9,10,...,1998,1999,2000],7).
transition([FRCP,NATC,RATC]-[FC],[1,2,3,...,1998,1999,2000],3).
transition([FC]-[FT],[1,2,3,4,5,6,7,8,9,10,,...,1997,1998,1999,2000],2).
transition([FT]-[stop],[1,2,7,9,...,1991,1992,1993,1994,2000],1).
transition([SRCP]-[ER],[1,2,3,...,1998,1999,2000],56).
transition([SRCP]-[PR],[1,2,3,...,1998,1999,2000],6).
transition([ER]-[ESR],[2,3,4,7,8,10,...,1997,1998,1999],55).
transition([PR]-[GPR],[1,5,6,9,...,1991,1994,1995,1996,2000],5).
transition([GPR]-[FRCP],[1,9,...,1991,1994,1996,2000],4).
transition([ESR]-[FRCP],[2,7,...,1992,1993],54).
```

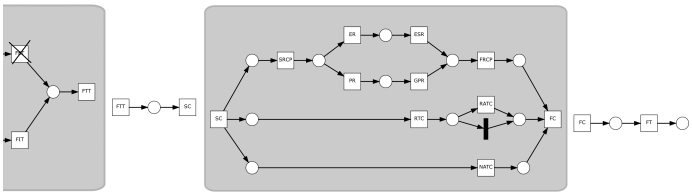

Fig. 3. Mined version of the NCT subprocess.

one (*sim*), and the average overlap ratio between discarded mined subprocesses, are reported. An empty field means that no discovered subprocess could be assigned to that gold standard (especially when *#c* is lower than 8). Merged cells mean that one discovered subprocess was big enough to overlap different gold standard subprocesses.

First note that the gold standard subprocess ORT (the first one) was never discovered with support $\delta < 500$. Basically, its small size (it is the smallest one) caused its incorporation within CS, which is its immediate successor. This issue might be solved by a further decomposition. As a consequence, no mined subprocess was able to suitably cover CS (the best similarity is 56.3%), because when discovered, it often involved its predecessor ORT too. The most challenging high-level construct in the gold standard process model is the *OR-split/join* between PCaP, PChP and PEP. Table II shows that it was discovered only by strategies with a low support threshold (100 for DB and H). In fact, the more frequent patterns required, the less likely that all OR construct alternatives are extracted. PCaP was the most frequent subprocess among OR-join alternatives, simply because almost all strategies produced it (except for support 900); its similarity values for the various strategies range between 62.5% and 75.2%, which are very good, considering the challenging task. Similarity values for PChP and PEP, when discovered, ranged between 62.1% and 70.5%. The last subprocesses sequence (CR, TM and NCT) were (almost) always discovered. Specifically, the *sim* values for CR and TM were, on average, above 60%. As regards NCT, its similarity value decreases from 80.2% to 70% as long as the support value δ was increased. It is noteworthy the framework was always able to find a consistent ending block for the mined subprocesses.

Looking at the silhouette values, clustering algorithms always produced a positive score. Strategy $(H, \delta = 100)$, in bold in Table II, yielded the highest silhouette score (0.40). It also covers the largest number of subprocesses (7 out of 8) compared to other settings, with *#c* = 16. Overall, we consider this strategy to be the best one, representing a good trade-off among all values and modules. Accordingly, Figure 3 shows an example[10] of a mined subprocess (**NCT**), in both WoMan and Petri formalism, that reported the highest *sim* value for $(H, \delta = 100)$[11]. It is outstanding that its structure is almost compliant (80.2%) with the gold standard one except for two activities (FIT and FTT). It is also interesting to note that *HAC*, with a suitable cut-off method (e.g., the elbow one), is able to provide a number of clusters reflecting more closely the actual number of gold standard subprocesses than the DBSCAN algorithm.

[10]Due to space limitations we can not show all subprocesses.

[11]This holds for almost all strategies as well.

VI. Conclusions and Future works

The activity logs generated by most information systems might be used to automatically learn models of business processes, to be used for analysis and conformance checking. The reported activities might be too fine-grained, leading to complex and incomprehensible ('spaghetti') models. In this work we proposed an approach to modularize process models by automatically discovering modules, in the form of subprocesses, in the context of unstructured processes. Modularization is a possible solution to improve understandability and reusability of the models. Experimental results on synthetic data and models confirmed that the approach is viable.

One future work direction concerns the use of other overlapping assessment strategies (e.g., normalized similarity/distance scores based on the Maximum-Common-Subgraph distance, used for sub-graph clustering). Also, we will investigate non-overlapping discovery for specific clusters of subgraphs, in order to better decompose the workflow and to accomplish the modularization step. Besides, a criterion to select the potential module, among the candidates, will be investigated.

References

[1] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining process models from workflow logs," in *Proceedings of the 6th International Conference on Extending Database Technology (EDBT)*, 1998.

[2] J. Cook and A. Wolf, "Discovering models of software processes from event-based data," Department of Computer Science, University of Colorado, Tech. Rep. CU-CS-819-96, 1996.

[3] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, pp. 1128–1142, 2004.

[4] A. Weijters and W. van der Aalst, "Rediscovering workflow models from event-based data," in *Proc. 11th Dutch-Belgian Conference of Machine Learning (Benelearn 2001)*, 2001, pp. 93–100.

[5] J. M. E. Van der Werf, B. F. van Dongen, C. A. Hurkens, and A. Serebrenik, "Process discovery using integer linear programming," in *International conference on applications and theory of petri nets*. Springer, 2008, pp. 368–387.

TABLE II
EVALUATION OF THE DISCOVERED SUBPROCESSES

Settings		Subprocesses		Gold Standard Subprocesses															
		#c	Sil	ORT		CS		PCaP		PChP		PEP		CR		TM		NCT	
				sim	avg	sim	avg	sim	avg	sim	avg	sim	avg	sim	avg	sim	avg	sim	avg
DB	100	23	0.37	-		-		75.2%	8%	64.3%	7%	67.1%	-	66.3%	8%	64.2%	11%	80.2%	11%
	300	16	0.32	-		56.3%	6%	73%	8%	-		69%	8%	66.3%	8%	61.3%	11%	80.2%	10%
	500	7	0.13	-		-		75.2%	8%	-		-		63.7%	8%	52.5%	13%	80.2%	10%
	700	8	0.29	16.3%	19%	51%	6%	75.2%	8%	-				68%	6%	52.5%	11%	70%	6%
	900	8	0.30	62.5%	18%	43%	8%	15% — 9%						65.2%	7%	52.5%	15%	70%	15%
H	100	**16**	**0.40**	-		**42.5%**	**10%**	**62.5%**	**9%**	**65.5%**	**7%**	**70.5%**	**9%**	**66.3%**	**5%**	**66.7%**	-	**80.2%**	**11%**
	300	16	0.36	-		55.5%	7%	73%	7%	-		70.5%	8%	66.3%	6%	61.9%	11%	80.2%	10%
	500	8	0.27	-		37%	12%	75.2%	6%	-		-		66.3%	5%	67.5%	12%	80.2%	10%
	700	4	0.22	33.5% — 5%				68%*l*	9%	-		-		66.3%	2%	-		70%	10%
	900	4	0.20	40% — 8%				-						65.2%	16.4%	54%	11%	70%	19%

[6] R. Yzquierdo-Herrera, R. Silverio-Castro, and M. Lazo-Cortés, "Subprocess discovery: Opportunities for process diagnostics," in *Enterprise Information Systems of the Future*. Springer, 2013, pp. 48–57.

[7] W. van der Aalst, "The application of petri nets to workflow management," *The Journal of Circuits, Systems and Computers*, vol. 8, pp. 21–66, 1998.

[8] C. Diamantini, L. Genga, and D. Potena, "Behavioral process mining for unstructured processes," *Journal of Intelligent Information Systems*, vol. 47, no. 1, pp. 5–32, 2016.

[9] S. Ferilli, "Woman: Logic-based workflow learning and management," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 6, pp. 744–756, 2014.

[10] M. Pesic and W. M. P. van der Aalst, "A declarative approach for flexible business processes management," in *Proceedings of the 2006 international conference on Business Process Management Workshops*, ser. BPM'06. Springer-Verlag, 2006, pp. 169–180.

[11] R. J. C. Bose and W. M. van der Aalst, "Trace clustering based on conserved patterns: Towards achieving better process models," in *International Conference on Business Process Management*. Springer, 2009, pp. 170–181.

[12] F. Folino, G. Greco, A. Guzzo, and L. Pontieri, "Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction," *Data & Knowledge Engineering*, vol. 70, no. 12, pp. 1005–1029, 2011.

[13] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca, "Discovering expressive process models by clustering log traces," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 8, pp. 1010–1027, 2006.

[14] J. L. L. Jung, Jae-Yoon; Bae, "[ieee 2008 ieee international conference on services computing (scc) - honolulu, hi, usa (2008.07.7-2008.07.11)] 2008 ieee international conference on services computing - hierarchical business process clustering," 2008.

[15] C. Diamantini, D. Potena, and E. Storti, "Mining usage patterns from a repository of scientific workflows," in *Proceedings of the 27th annual ACM symposium on applied computing*. ACM, 2012, pp. 152–157.

[16] D. M. Schunselaar, T. Slaats, F. M. Maggi, H. A. Reijers, and W. M. Van Der Aalst, "Mining hybrid business process models: a quest for better precision," in *International Conference on Business Information Systems*. Springer, 2018, pp. 190–205.

[17] F. M. Maggi, T. Slaats, and H. A. Reijers, "The automated discovery of hybrid processes," in *International Conference on Business Process Management*. Springer, 2014, pp. 392–399.

[18] R. Conforti, M. Dumas, L. García-Bañuelos, and M. La Rosa, "Bpmn miner: Automated discovery of bpmn process models with hierarchical structure," *Information Systems*, vol. 56, pp. 284–303, 2016.

[19] A. Polyvyanyy, S. Smirnov, and M. Weske, "Process model abstraction: A slider approach," in *12th International IEEE Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2008, pp. 325–331.

[20] J. Vanhatalo, H. Völzer, and F. Leymann, "Faster and more focused control-flow analysis for business process models through sese decomposition," in *International Conference on Service-Oriented Computing*. Springer, 2007, pp. 43–55.

[21] F. Mannhardt and N. Tax, "Unsupervised event abstraction using pattern abstraction and local process models," *arXiv preprint arXiv:1704.03520*, 2017.

[22] J. Lu, W. Chen, O. Adjei, and M. Keech, "Sequential patterns postprocessing for structural relation patterns mining," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 4, no. 3, pp. 71–89, 2008.

[23] R. J. C. Bose and W. M. Van der Aalst, "Abstractions in process mining: A taxonomy of patterns." in *BPM*, vol. 9. Springer, 2009, pp. 159–175.

[24] R. J. C. Bose, H. E. Verbeek, and W. M. van der Aalst, "Discovering hierarchical process models using prom." in *CAiSE Forum*, vol. 107. Springer, 2011, pp. 33–48.

[25] R. J. C. Bose and W. M. van der Aalst, "Process diagnostics using trace alignment: opportunities, issues, and challenges," *Information Systems*, vol. 37, no. 2, pp. 117–141, 2012.

[26] I. Weber, M. Farshchi, J. Mendling, and J.-G. Schneider, "Mining processes with multi-instantiation," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 1231–1237.

[27] B. F. Van Dongen and W. M. Van der Aalst, "Multi-phase process mining: Building instance graphs," in *International Conference on Conceptual Modeling*. Springer, 2004, pp. 362–376.

[28] S. Ferilli, F. Esposito, D. Redavid, and S. Angelastro, "Predicting process behavior in woman," in *AI*IA 2016: Advances in Artificial Intelligence - XVth International Conference of the Italian Association for Artificial Intelligence, Genova, Italy, November 29 - December 1, 2016, Proceedings*, 2016, pp. 308–320.

[29] S. Ferilli and S. Angelastro, "Efficient declarative-based process mining using an enhanced framework," in *Complex Pattern Mining*. Springer, 2020, pp. 121–136.

[30] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* IEEE, 2002, pp. 721–724.

[31] M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen, "A quantitative comparison of the subgraph miners mofa, gspan, ffsm, and gaston," in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2005, pp. 392–403.

[32] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *Journal of the American statistical association*, vol. 58, no. 301, pp. 236–244, 1963.

[33] A. E. Zambelli, "A data-driven approach to estimating the number of clusters in hierarchical clustering," *F1000Research*, vol. 5, 2016.

[34] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[35] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern recognition letters*, vol. 19, no. 3-4, pp. 255–259, 1998.

[36] S. A and B. H., *Graph-theoretic techniques for web content mining*. World Scientific, 2005, vol. 62.

[37] S. K. vanden Broucke, J. Munoz-Gama, J. Carmona, B. Baesens, and J. Vanthienen, "Event-based real-time decomposed conformance analysis," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2014, pp. 345–363.

[38] H. A. Reijers, J. Mendling, and R. M. Dijkman, "Human and automatic modularizations of process models to enhance their comprehension," *Information Systems*, vol. 36, no. 5, pp. 881–897, 2011.

[39] S. Ferilli, T. M. A. Basile, M. Biba, N. Di Mauro, and F. Esposito, "A general similarity framework for horn clause logic," *Fundamenta Informaticae*, vol. 90, no. 1-2, pp. 43–66, 2009.