Methods and techniques for discovering taxonomies of behavioral process models

Francesco Folino,[1] Gianluigi Greco,[2]* Antonella Guzzo[2] and Luigi Pontieri[1]

Modeling behavioral aspects of business processes is a hard and costly task, which usually requires heavy intervention of business experts. This explains the increasing attention given to process mining techniques, which automatically extract behavioral process models from log data. In the case of complex processes, however, the models identified by classical process mining techniques are hardly useful to analyze business operations at a suitable abstraction level. In fact, the need of process abstraction emerged in several application scenarios, and abstraction methods are already supported in some business-management platforms, which allow users to manually define abstract views for the process at hand. Therefore, it comes with no surprise that process mining research recently considered the issue of mining processes at different abstraction levels, mainly in the form of a taxonomy of process models, as to overcome the drawbacks of traditional approaches. This paper presents a general framework for the discovery of such a taxonomy, and offers a survey of different kinds of basic techniques that can be exploited to this purpose: (1) workflow modeling and discovery techniques, (2) clustering techniques enabling the discovery of different behavioral process classes, and (3) activity abstraction techniques for associating a generalized process model with each higher level taxonomy node.

How to cite this article:
WIREs Data Mining Knowl Discov 2013, 3: 170–189 doi: 10.1002/widm.1081

INTRODUCTION

Workflow models are an effective way to specify the behavior of complex processes in terms of elementary activities and routing constructs (e.g., parallelism, loops, splits), and have been largely used in many Business Process Management (BPM) platforms. Unfortunately, modeling the behavioral aspects of a business process is a time-consuming task, usually requiring heavy intervention by business experts. This motivates the recent surge of interest toward process mining techniques,[1] which allow for automatically extracting a workflow model based on the execution logs available for a given process.

However, traditional process discovery approaches designed to eventually support process enactments, extract workflow models specifying all the operational details of the process. Conversely, business users often want to analyze business operations at higher abstraction levels, and several commercial business-management platforms (e.g., iBOM,[2] ARIS[3]) offer capabilities for manually defining abstract views over a process. Thus, the automated discovery of multiple process views, at different granularity levels, is a natural extension of process mining and of workflow analysis techniques.

In this work, we specifically consider the case where multilevel views are induced for describing the behavior of a process, and eventually organized in the form of a taxonomy, a valuable kind of knowledge representation tool, which has found application in a disparate fields. A *process taxonomy*, specifically, is essentially a tree of workflow models, where the root provides the most abstract view over the executions of a process, and any other node refines this abstract model to describe a subclass of executions.

*Correspondence to: ggreco@mat.unical.it
[1]ICAR-CNR, Rende, Italy
[2]University of Calabria, Rende, Italy
DOI: 10.1002/widm.1081

Clearly, such a structure, describing the main behavioral variants of a process in an articulated and modular way, allows for effectively consolidating, sharing, and reusing knowledge about its behavior. In fact, process taxonomies have been profitably used in the modeling and reengineering of business processes (see, e.g., the MIT's *Process Handbook* project).[4]

A first step toward such an automatic construction of process taxonomies, based on process mining, was presented in Ref 5, where different behavioral classes of a process are discovered with a clustering method, and equipped eventually with separate workflow models. Indeed, such a result can be used as a basis for obtaining a taxonomy of process models, by possibly exploiting diverse process abstraction techniques,[6–13] to provide high-level nodes with coarser grain process models.

In fact, the discovery of taxonomies, in the form of concept hierarchies, was widely studied in the past, especially in the context of ontology learning systems.[14] Various approaches have been proposed to extract concepts' taxonomies from different kinds of data sources. For instance, as to the case of structured input data, taxonomy learning methods have been defined that can take as input database schemas,[15] other existing ontologies,[16] knowledge bases,[17] and lexical semantic nets such as *WordNet*. Some learning systems (e.g., those in Refs 18–21) can also exploit semi-structured data (such as, e.g., dictionaries, HTML, XML, and DTS's documents) in the discovery of a concept taxonomy. In general, the most difficult source to deal with are unstructured data, such as sequences and text documents. In such a case, the typical approach (see, e.g., Refs 22–25) relies on using some clustering algorithm to automatically induce a hierarchy of classes (for words and/or documents), and regarding each of these classes as the evidence for a distinct concept. Despite this problem is logically similar to the one addressed in this work, a main difference lies in the fact that every node in a concept taxonomy has a 'static' nature, in that it does not encode dynamic behaviors, as it happens, instead, in the case of process models. Hence, these methods cannot be trivially reused when discovering a process taxonomy, where ad hoc process induction/abstraction mechanisms are needed to capture process dynamics and guarantee some sort of behavioral consistency between each model in the taxonomy and its parent.

This paper presents a survey of some major issues and solutions related to the discovery of process taxonomies. After introducing preliminary concepts (concerning workflow models, activity abstractions and behavioral consistency notions), a general approach to the discovery of process taxonomies is sketched in the third section, parametrically to three basic tasks: process discovery, trace clustering, and process abstraction. The following three sections discuss and compare some major approaches in the literature that can help solve each of these subproblems, whereas few concluding remarks are drawn in the last section.

PRELIMINARIES: WORKFLOWS, ABSTRACTIONS, AND BEHAVIORAL CONSISTENCY

Workflow models (precisely control-flow models) are a popular means for representing the behavior of a process, and hence constitute a special kind of model for it. However, as in this paper we are not considering any other kinds of process models (e.g., data-flow models, organizational models, etc.), the terms 'workflow model' and 'process model' will be used interchangeably hereinafter. In the rest of this section, some basic concepts on workflow models and on activity abstraction are introduced. The section then presents some notions of behavior inheritance/preservation for workflow models, which can help provide a semantical foundation to parent–child relationships in a taxonomy of process models.

Workflow Models (Schemas) and Logs

A workflow model (a.k.a. workflow schema) specifies all possible flows along the activities of a process, by way of a set of constraints defining 'legal' execution in terms of simple relationships of precedence and/or more elaborate constructs such as loops, parallelism, synchronization, and choice (just to cite a few). A significant amount of research has been done for the specification of process models (e.g., *EPCs*, *Petri Nets*).

For the sake of clarity, a simple modeling language for workflow models is used hereinafter, where precedence relationships are depicted as arrows between two nodes of a *workflow* graph, whereas further execution constraints are specified with special labels associated with the input/output of a task. Specifically, an *AND*-join node (i.e., a node with AND on its input) acts as synchronizer (i.e., it can be executed only after all its predecessors have been completed), whereas a *OR*-join node can start as soon as one of its predecessors completes. Once finished, an *AND* (respectively, *OR*, *XOR*)-split node activates all (respectively, some, one) of its output activities. Notice that most of the methods discussed in this paper are orthogonal to the language adopted to represent

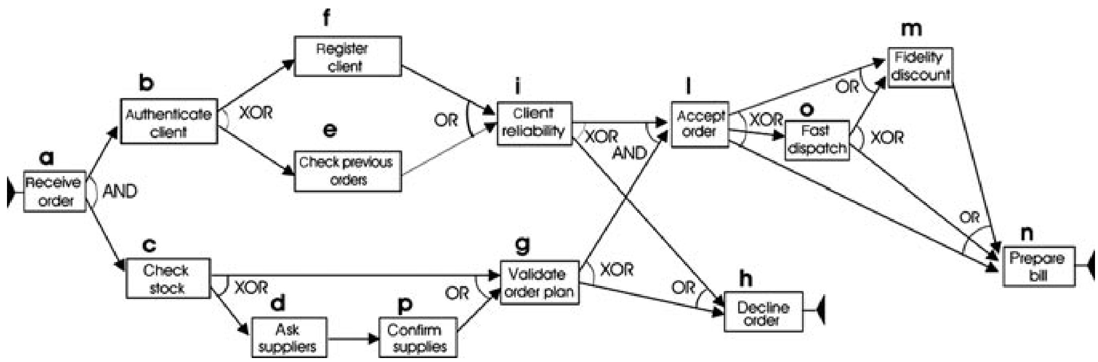

FIGURE 1 | Workflow model for the sample *HandleOrder* process.

s_1 : acbgfh	s_9 : abefcgil
s_2 : abfcgh	s_{10} : acgbefijl
s_3 : acgbfh	s_{11} : abcedfgil
s_4 : abcgfil	s_{12} : acdbefgil
s_5 : abfcgikl	s_{13} : abcfdgikl
s_6 : acbfgijl	s_{14} : acdbfgikl
s_7 : acbgfijkl	s_{15} : abcdgfikl
s_8 : abcegfijl	s_{16} : acbfdgil

FIGURE 2 | Sample log for the *HandleOrder* process.

process behavior, and they do not depend on the simplified notation introduced above.

Example Figure 1 shows a workflow model for a process concerning the handling of customers' orders in a business company. For example, task *l* is an AND-join activity, as it must be notified that both the client is reliable and the order can be supplied. Conversely, *b* is a XOR-split activity, whereas it can activate just one of its adjacent activities.

Each time a workflow model is enacted, its activities are executed according to the associated constraints, till some final configuration is reached. Many process-oriented systems store information on process instances in a log repository, keeping track of the events happened during each of them. Basically, a process log can be seen as a set of *traces*, which, in the most simplistic scenario, correspond to strings over activity identifiers, representing sequences of activities. A small log is shown in Figure 2, for the example process *HandleOrder*.

Essentially this is the type of historical data that process discovery algorithms[1] take in input to find a workflow model, even when the original one is unknown. The quality of a workflow model W can be evaluated relatively to a log L (the one actually used for inducing the model, or another log of the same process) by way of 'conformance' measures [usually ranging over $(0\ldots1)$], which can be distinguished into two main families: (1) *fitness* measures (a sort of completeness measures), which roughly tell how much the traces in L comply with the behavior encoded in W, by typically counting the violations that are needed to perform to replay all the traces through the model; and (2) *precision* measures, which try to quantify how much of the flexibility (ascribable to alternative/parallel constructs) of W is really necessary to reproduce L.

Activity Abstractions and Process Taxonomies

Many process abstraction approaches store and exploit activity abstraction relationships. To make thinks concrete, we next describe a basic form of activity ontology, as defined in Ref 6, which intuitively captures two different kinds of abstraction, corresponding to IS-A (a.k.a., 'hypernimy' or 'generalization') relations and PART-OF (a.k.a., 'partonomy' or 'meronimy') relations, respectively. Such relations were widely used for representing business activities in several application contexts, such as, for example, the MIT Process Handbook project,[4] where a catalogue of business processes models was defined, based on interviews with experts, which span several business domains and features about 5000 activities.

Hereinafter, we will name *activity ontology* a tuple $D = \langle A, IsA, PartOf\rangle$, where A still denotes a set of activities, whereas *IsA* and *PartOf* are binary relations over A. Intuitively, given two activities a and b, $(b,a)\in IsA$ indicates that b is a specialization of a, whereas $(b,a)\in PartOf$ indicates that b is a component of a. These basic properties can be extended in a transitive fashion, as follows. Given two activities a and x, *a abstracts x* if there is a path from a to x in the graphs induced by *IsA* and *PartOf*. In such a case, we also say that a is a *complex* activity; otherwise, a is a *basic* activity. In a sense, complex activities constitute high-level concepts defined by aggregating or generalizing the basics activities that actually occur in real process executions. This notion is the building block for defining a taxonomy of process models, where the knowledge about process behavior is structured into different abstraction levels.

On the basis of activity abstraction (respectively, refinement) relationships, an intuitive notion of process model generalization (respectively, specialization) can be stated: Given two workflow models W_1 and W_2, we say that W_1 *generalizes* W_2 (W_2 *specializes* W_1) with respect to an activity ontology D, if for each activity a_2 of W_2 (1) either a_2 appears in W_1 or there is an activity a_1 in W_1 such that a_1 *abstracts* a_2, and (2) there is no activity b_1 in W_1 such that a_2 *abstracts* b_1.

Definition 1 A *process taxonomy* is a tree of workflow models, where the leaves correspond to distinct classes of process executions, whereas any nonleaf schema provides a unified and summarized representation over multiple heterogeneous behavioral classes. Formally, given an activity ontology D, like that introduced in the previous section, a tree of workflow models G is said to be a *process taxonomy* with respect to D if, for any pair of models W and Wp such that W is a child of Wp in G, Wp *generalizes* W.

The basic abstraction relations described above could be specified in many formal knowledge representation languages, such as, for example, the family of Description Logics—in this case each activity label x can be regarded as a concept term, while assuming that a log trace t is an instance of that concept if t contains at least one occurrence of x. Even more directly, one can use object-oriented modeling frameworks (e.g., OntoDLV)[26] natively supporting the constructs.

Notice, moreover, that the definition of process taxonomy does not explicitly embed a precise notion of behavioral inheritance for the models appearing in it. Such a desirable property is indeed delegated to the underlying activity abstraction relationships, which could be practically defined in accordance with some kind of behavioral inheritance notion, as discussed in the next subsection. Notably, the choice of a reference notion of behavioral inheritance, determines the semantics of trace abstraction with respect to a given activity ontology like that defined before, as far as concerns the possible ways of replacing multiple occurrences of concrete activities with a higher level one that features as an ancestor of these activities in one of the abstraction hierarchies of the ontology.

Behavior Inheritance/Preservation

Diverse notions of specialization and inheritance were defined in several application contexts, for example, OO-Design/Programming and Process Modelling. The possibility of defining business process taxonomies was first considered in Ref 4, where a repository of process descriptions is envisaged supporting the design and sharing of process models. However, this pioneering work builds on a ‘static’ representation of the processes, which disregards the evolution of the process over time: each process P is modeled as a class featuring P’s activities as properties, which will be inherited by any P’s subclass—by the way, the framework also allows to remove inherited activities (nonmonotonic inheritance).

The problem of defining a specialization/inheritance notion, capable to account for dynamic behavior, has been studied against different process modeling languages, such as, for example, UML diagrams,[27] Petri nets,[28] and DataFlow diagrams.[29] Notably, the classical IS-A property relating the instances of a given class to its superclass is typically rephrased in these contexts by stating that all the execution instances of a model may also be regarded as instances of any model generalizing it, in connection with some suitable behavioral equivalence criterion (e.g., trace equivalence or branching bisimulation).

Two main kinds of specialization may be considered on a process model: *extension* (i.e., one or more activities, and their associated flow links, are added the model) and *refinement* (i.e., one or more activities in the model are refined by replacing each of them with some more specific activity or with a subprocess, composed of multiple finer grain activities).

In the first case, a key point for defining a proper notion of behavioral inheritance concerns how to abstract the execution of any activity that has been added to the subclass model. Quite a complete and deep theoretical framework for dealing with such a situation is presented in Ref 28, where two basic notions of behavior inheritance (and two derived notions based on them) are defined for workflow models represented as WF nets[30–32] (a special kind of Petri nets). There, it is stated that the external behaviors shown by a model and by any of its specializations must not be distinguishable whenever: (1) only common activities are performed, while blocking the additional ones (‘protocol inheritance’, conceptually similar to the notion of ‘invocation consistency’[27]); or (2) when one simply abstracts from activities that are not in the base model (‘projection inheritance’, analogous to ‘observation consistency’[27]). Moreover, four inheritance-preserving transformation rules are presented in the same paper, which allow to specialize a WF-net model by adding new elements as part of typical control flow constructs (choice, iteration, sequential composition, and parallel composition, respectively). Notably, when using these rules, the resulting model is ensured to be a subclass of the original one, without requiring any explicit verification of behavior equivalence (which is, in general, a costly task).

The concept of process specialization as refinement of activities has been largely adopted in the field of process abstraction[2,3,6–12]—closely related to the theme of this paper—where the aim is to simplify the description of a process model by providing the user with more abstract and readable process views. In fact, in this perspective, a more general (and succinct) view of a process can be formed by making the given model undergo some activity abstraction transformation, which essentially amounts to replace a group of activities (or an entire subprocess) with a single higher level activity. Clearly enough such a transformation is the inverse of refining the resulting abstract model. In actual fact, specialization via extension as well has a counterpart in a process abstraction setting, which obviously corresponds to the elimination of activities; however, we do not discuss such a type of transformation further in the rest of this paper for two reasons: (1) it has not found as a wide usage as activity abstraction in the literature, (2) one can still think of replacing one or more activities he/she wants to remove with some sort of 'phantom' high-level activities, which can be kept hidden (along with their associated links) in the abstracted view shown to the user.

As far as concerns the similarity of behaviors between a process model and its abstracted version, most process abstraction approaches do not fulfill a precise notion of inheritance like that in Ref 28. Anyway, some approaches have been defined, which try to satisfy some kind of behavioral consistence, ensuring that routing and causal constraints among the activities are somewhat preserved in the resulting model. The prevalent way of obtaining such a result is to decompose the structure of the input workflow model into a number of process fragments, possibly defined in a recursive way (as in the case of the *SPQR*-tree structure[33]), such that each fragment corresponds to a well-specified composition pattern (e.g., sequential composition, or split/join structures). In this way, an order-preserving notion of process generalization can be met, provided that each set of partonomical relationships stored in the activity ontology are created in accordance with these fragments.

An alternative solution consists in taking account for the ordering relationships between process activities that are implied by the control-flow model, when deciding which activities are to be aggregated together into a higher-level abstract activity. In particular, in Ref 9, the resulting model M' is ensured to be an 'order preserving' view of the original process model M, in that, for any pair of activities x and y in M', if x *precedes* (respectively, *follows*, *is-independent-of*) y, then any activity abstracted by x *precedes* (respectively, *follows*, *is-independent-of*) all the activities abstracted by y. In other words, the implied ordering constraints between concrete activities of the process, which are produced by the abstraction process, must coincide with the ordering constrains in the original model. For example, with regard to Figure 1, the model obtained by simply abstracting activities d and p together into a single complex activity, say x, would comply with this notion of behavior consistency; the converse would happen, instead, if we defined x as consisting of c and g.

GENERAL APPROACH TO PROCESS TAXONOMY DISCOVERY

The problem of discovering a *process taxonomy* (cf. Definition 1) can be approached via a two-phase strategy, consisting of two macro-steps: (S1) *Clustering-Based Hierarchy Discovery*, where a hierarchical clustering of the log is computed by looking at behavioral similarities between log traces, so that each cluster can be regarded as representative of a different behavioral subclass of the process, and equipped with a separate workflow model (with the help of a workflow discovery algorithm); and (S2) *Abstraction-Based Hierarchy Restructuring*, where the hierarchy of workflow models is restructured into a taxonomy, by using process abstraction mechanisms allowing to associate each nonleaf node v with a workflow model that generalizes all the models appearing in the subtree rooted in v.

Clearly, such an approach hinges on three different basic computation tasks: (1) *Workflow Induction*, amounting to extracting a workflow model out of a given (sub-)set of log traces; (2) *Trace Clustering*, aimed at partitioning a given set of log trace into a number of behaviorally homogeneous groups; (3) *Workflow Abstraction*, devoted to deriving a generalized coarser grain workflow model for a given set of workflow models. A range of methods are available in the literature, which can, in principle, help solve each of these core subproblems. Figure 3 offers a rough, and yet hopefully intuitive, picture of how such methods can be exploited to discover a process taxonomy, according to the two-phase approach mentioned above. For each task, the figure shows the respective inputs and outputs, as well as a list of works in the literature (namely, references

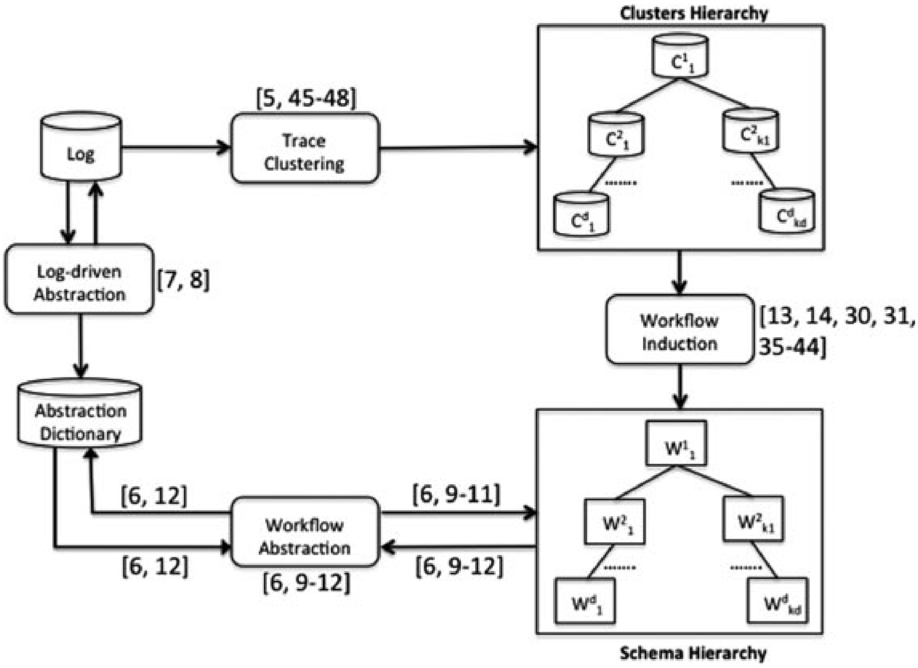

FIGURE 3 | A pictorial representation of the overall approach to the discovery of a process taxonomy: core tasks and related works in the literature (macro-tasks and information flows are annotated with references to the bibliography).

to items in the bibliography) that can be reused to implement it.

Notice that, in Figure 3, it is envisaged the integration of this main computation procedure with *Log-Driven Abstraction* techniques,[7,8] recently appeared in the literature, as an optional preprocessing step. Essentially, these techniques allow for identifying abstract activities as groups of correlated log events (based on clustering or pattern-mining algorithms). Each of such high-level activities can be stored in an activity ontology (along with its associated low-level activities), and can be used to produce and abstract view of the log—where, in each log trace, low-level activities are replaced with the corresponding high-level one. In fact, such a preprocessing step can be very effective in the case where the logs contains low-level events, which are not directly linked to semantically relevant process activities. Indeed, as the raw application of process discovery algorithms to such logs would results in 'spaghetti-like' models (with many task nodes and links between one another), it is convenient to bring the traces to a higher level of abstraction, before clustering and analyzing them.

The rest of this section is devoted to illustrate, in two separate subsections, two meta-algorithms encoding a computational scheme for the high-level computation steps (S1) and (S2) introduced above, respectively. Notice that these meta-algorithms are mainly meant to describe, in a more precise manner, how the core *Workflow Induction*, *Trace Clustering*, and *Workflow Abstraction* techniques are employed within the discovery of a process taxonomy. A deeper discussion on these three families of techniques will be provided later on, in the next three sections of the paper.

Clustering-Based Hierarchy Discovery

Hierarchical clustering methods (agglomerative or divisive) have been extensively used in several application contexts to construct taxonomies automatically (see, e.g., Refs 22,23). Traditionally, these schemes rely on suitable distance measures and linkage strategies, and produce a tree-like partitioning structure ('dendrogram'), which can serve as a basis to derive class hierarchy. However, many of these classical clustering methods risk being too time consuming on large logs. A possible solution consists in finding an initial set of, fine grain, clusters for the input log, and then grouping them into higher level clusters according to an agglomerative scheme, where the similarity among clusters is computed by only comparing the workflow models associated with them (with the help of workflow discovery techniques). Workflow-oriented graph edit distances[34] and behavioral similarity measures[35] could be exploited to this end.


```
Input:    a log L, two natural numbers maxSize and K, a real quality threshold γ
Output:   a workflow hierarchy H
Method: Perform the following steps:
A)  Initialize the hierarchy with one workflow model for the whole log
    W0 = mineWFSchema(L)        // mine a model W0 from all traces in L
    cluster(W0) = L              // associate W0 with the whole log L
    DW = { W0 }                  // DW is used to contain all the leaf schemas of H
B)  WHILE size(DW)≤ maxSize and quality(DW)< γ
      Extract the least accurate model W* from DW    // according to measure quality(·)
      {C1,...,Ck} = partition(traces(W*))
      For each cluster Cj ( j = 1..k ) extracted from traces(W*),
            Wj = mineWFSchema(L)                 // mine a (refined) model for the cluster
            put Wj in DW, and extend H by adding Wj as a child of W*
```

FIGURE 4 | Meta-algorithm *HierarchyDiscovery*.

As an alternative solution, in Figure 4, a top-down clustering scheme is illustrated, where whatever (more scalable) clustering method can be exploited (as in Ref 5). In this meta-algorithm, a given log is decomposed hierarchically into a number of sublogs, by iteratively splitting a cluster whose associated model is expected to mix different usage scenarios. The result is a tree-like model where each node corresponds to a set of executions (i.e., process instances) and its children to a partition of that set.

Initially a single workflow model W_0 is extracted that is a first attempt to model the whole log. Iteratively, one of the models not refined yet (i.e., corresponding to a leaf of the tree) is refined: the set of traces that are associated with it are split into clusters by using the meta-function *Partition*, which could be implemented by some of the different trace clustering approaches proposed in the literature (and discussed later on). A new workflow model is then mined out for each of these clusters, by using some workflow discovery technique (see next section for more details). At the end of the process, a hierarchy of workflow model is obtained, where the leaf nodes constitute a disjunctive model representing the execution logs more accurately than $W_{0.}$ Note that the method is also parametric with respect to the measure *quality*, which should evaluate how much adequately the current set of unrefined workflow models—that is, the ones on the frontier of the tree—capture the behavior of the process under analysis.

Such an evaluation could be made by resorting to log conformance measures like the ones mentioned in the previous section and/or to structural complexity measures.

Example (***contd.***) To provide insight on how the above meta-algorithm could work in a practical case, we randomly generated 100,000 traces from the workflow in Figure 1, under the additional constraint that task m cannot occur in any trace containing f (a fidelity discount is never applied to a new customer), and task o cannot appear in any trace containing d and p (fast dispatching cannot be performed whenever external supplies are asked for), hence simulating the presence of different process variants. We then applied the meta-algorithm *HierarchyDiscovery*, using the feature-based clustering in Ref 5 (see the section on log clustering methods for further details) to implement meta-function *partition*, without performing any quality check in the test of the main loop (i.e., function *quality* is implemented as to always return the maximal score). This peculiar choice bases on the observation that such an approach is expected to be effective enough in dealing with behavioral constraints like the ones used in our simulation, and in identifying behaviorally homogenous clusters. The resulting hierarchy is shown in Figure 5(a), where each node logically corresponds to a cluster of traces and to an associated workflow model (induced from the cluster). The model W_0 preliminary found for the whole log (and associated with node v_0) actually coincides with the one shown in Figure 1. Because it was not as precise as required by the user, the log was partitioned into two clusters ($k = 2$). The cluster associated with v_2 was not refined further, whereas that of v_1 was split again into two subclusters. In fact, models W_0 and W_1 (corresponding to v_0 and v_1, respectively) are just preliminary model for their associated log traces, which are indeed modeled in more precisely by the leaf models—shown in Figure 5(b)–(d).

Abstraction-Based Hierarchy Restructuring

We next study how a hierarchy of workflow models can be restructured into a taxonomy of models, describing the process at different levels of details. The key point is to equip each nonleaf node with an abstract model generalizing those associated with the children of the node. To this aim, some suitable

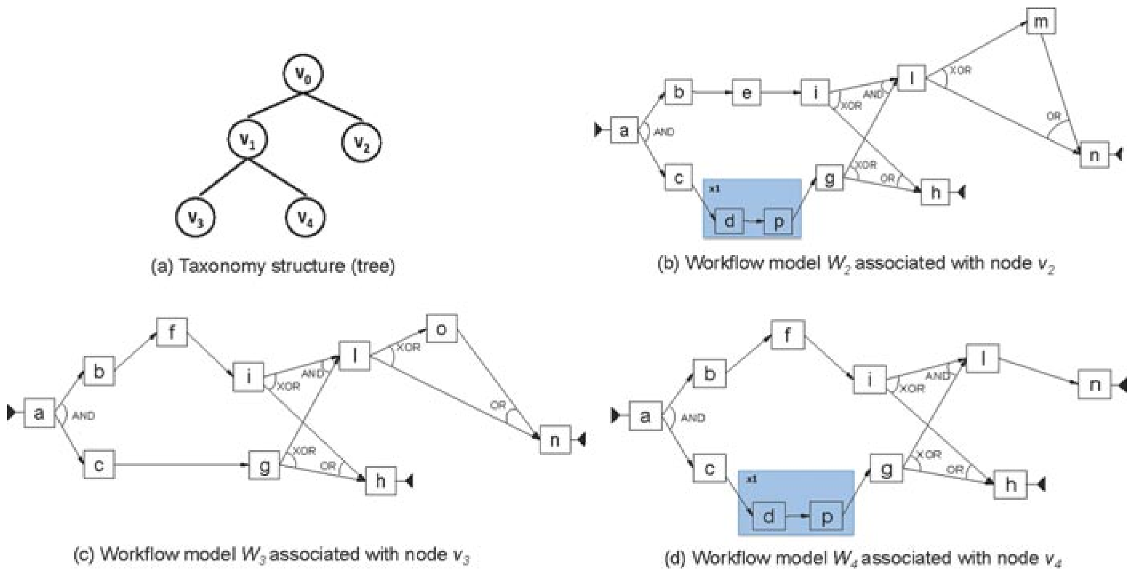

FIGURE 5 | Hierarchy found by *HierarchyDiscovery* on the running example (details for leaf models only).

activity abstraction method must be used to replace groups of (structurally correlated) activities with higher level activities.

The crucial steps are illustrated in Figure 6, via a meta-algorithm, named *BuildTaxonomy*, inspired to the approach in Ref 6. The algorithm transforms a given model hierarchy H into a taxonomy, possibly using an activity ontology D, storing basic activity abstraction relationships. In a bottom-up fashion, each nonleaf node v in the hierarchy is equipped with a new workflow model, by using meta-function *abstractSchema*. This latter is provided with the model v and with the indication of which activities are to be abstracted (namely, the tasks that appear only in a proper subset of v' children). Optionally, this task is carried out based on the contents of ontology D, which is then updated, as to store the links between abstracted activities and their corresponding complex ones in the novel (abstract) model of v. In such a case, D will be restructured eventually by removing 'superfluous' activities—that is, activities that does not appear in any model of H. The above meta-scheme is parametric to the initial contents of the activity ontology D (which can be empty or encoding existing domain knowledge), as well as to the actual algorithm implementing *abstractSchema*, which may disregard its third argument, which is just returned as it in output.

Example (***contd.***) We next consider the application of the meta-algorithm in Figure 6, where function *abstractSchema* is implemented as in Ref 6. Notably, any workflow model taken as input by this function is transformed by replacing 'specific' activities (i.e., activities that does not appear in all input models) with new 'virtual' ones abstracting them all in the *IsA* or *PartOf* relations based on the current contents of the reference activity ontology. Figure 7 illustrates the final outcomes of this restructuring process: (1) a tree representing the process taxonomy, replicating the structure of the input workflow hierarchy; (2) the contents of the activity ontology, mapping the abstract activities, created by the algorithm, to the corresponding concrete ones; (3) the two restructured workflow models produced for the nodes v'_0 and v'_1 (i.e., the only two nonleaf nodes in the taxonomy)—the three remaining (leaf) nodes in the taxonomy (namely v'_2, v'_3, and v'_4) are simply equipped with the same workflow models as their corresponding nodes (namely v_2, v_3, and v_4, respectively) in the original workflow hierarchy (cf. Figure 5)—that is, leaf models are left unchanged in the restructuring phase.

Let us now briefly describe how these results have been obtained. Provided with the hierarchy of Figure 5 and with an initially empty activity ontology—we assume that no background knowledge on activity abstraction is available—algorithm *BuildTaxonomy* first generalizes the leaf models W_3 and W_4 (associated with v_3 and v_4, respectively), which share all the activities but o, d, and p. In the resulting model W_1^*, shown in Figure 7(c), d and p have been aggregated into a new complex activity x_1—while putting the pairs (d, x_1) and (p, x_1) in the *PartOf* relationship. W_1^* is then merged with the model W_2, and a new abstract model W_0^*, shown in Figure 7(d), is build for the root. This model features three complex activities: x_1, aggregating d and p, as discussed before;


```
Input:   a workflow hierarchy H, and an activity ontology D (possibly empty)
Output:  a modified version of H (such that H is a process taxonomy w.r.t. D) and, possibly, of D
Method: Perform the following steps
Create a set S of models containing only the leaves of H
WHILE there is a model v in H, s.t. v ∉ S and its children are in S
   Let aS be the set of all tasks that appear in some of the children of v, but not in all of them
   (v,D) := abstractSchema(v,aS,D) // compute a new model where the tasks aS are abstracted
                                   // while possibly taking account for D and eventually updating it
   put v in S
Remove 'superfluous' complex activities from D (according to H's schemas)
Return H and D
```

FIGURE 6 | Algorithm *BuildTaxonomy*.

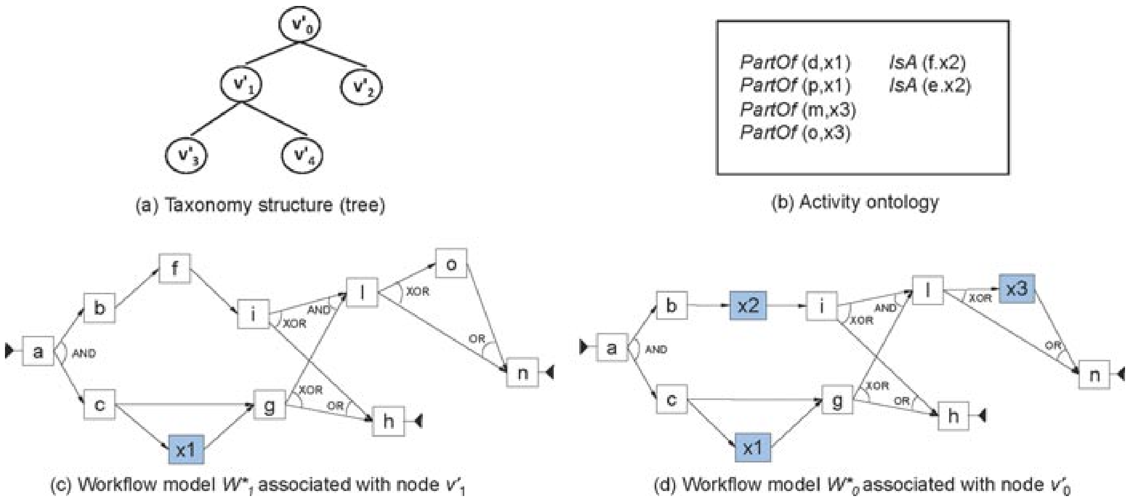

FIGURE 7 | Generalized workflow models in the taxonomy found for the example *HandleOrder* process.

x_2, aggregating e and f; and x_3 composed of m and o. The pairs (e, x_2), (f, x_2), (m, x_3), and (o, x_3) eventually appear in the *PartOf* relation of the activity ontology. Notice that a further complex activity x_0 was created during the creation of the abstract schema W_1^* to aggregate d and p; x_0 was then abstracted by x_1 through an IS-A link (in fact these two complex activities had the same set of subactivities and the same control flow links), and eventually removed from the activity ontology, for it does not feature in any of the workflow models in the resulting process taxonomy.

WORKFLOW DISCOVERY TECHNIQUES

Process mining techniques[1] try to extract knowledge on the behavior of a process from an execution log. The rest of this section illustrates a number of techniques specifically designed for the induction of a workflow models. In general, the proposals in the literature differ both in the specific induction algorithm and in the language for representing workflows—ranging from simple directed graphs,[13,36,37] expressing precedence relationships, possibly extended with simple split/join constraints,[5,38] to more expressive formalisms, sometimes enjoying deep behavioral semantics, like WF nets.[30–32]

The problem of discovering a workflow model was analyzed in Ref 31, where a class of Petri nets, named *structured workflow* (*SWF*) *net*, is identified. The algorithm proposed, named α, can rediscover such a model, under the hypothesis that the input log is 'complete'—that is, all pairs of tasks linked directly in the *SWF* appear consecutively in at least one log trace. Two extended versions of the algorithm,[32,39] were proposed subsequently to discover two specific kinds of control-flow constructs: short loops (loops involving one or two activities only) and nonfree-choice constructs (where the choice of which outgoing edges of a XOR-split node x is to be executed does not depends on x only), respectively.

Simple metrics concerning task dependency and task frequency are exploited in a heuristics approach,[38] capable of discovering a graph-based model, called 'dependency/frequency graph', which encodes both precedencies and split/join constraints. Notably, this approach can cope with noisy logs, based on user-given frequency thresholds.

The discovery of block-structured workflows possibly containing duplicate tasks was addressed in Refs 40 and 41, where a two-step solution is presented: first a stochastic activity graph (*SAG*) is induced from the log, and then the *SAG* is turned into a block-structured workflow by suitable transformation rules. The use of term rewriting systems was also proposed to discover a hierarchically structured workflow model, in the form of an expression tree, where the leaves represent tasks (operands), whereas any other node is associated with a control flow operator.[42]

An alternative solution[43] to the workflow discovery problem relies on a global search method, based on genetic algorithms. This allows for dealing with complex routing constructs (including nonfree-choice and hidden tasks, i.e., routing activities that do not appear in the traces) and with noisy data, but implies highest computational costs.

On the basis of the observation that extracting a single workflow model for very different cases may lead to overgeneralized process models, workflow discovery has been combined with the clustering of log traces.[5] This permits to improve the precision of basic workflow discovery algorithms by capturing constraints that are beyond the expressiveness of their associated modeling languages. To this end, the approach in Ref 5 essentially exploits a top-down clustering scheme very alike the one in Figure 4 (without any test on the quality of the current hierarchy), and returns a collection of workflow schemas, corresponding to the ones induced from the leaves of the discovered clusters' tree. A more detailed discussion on the clustering technique is given in the next subsection, devoted to trace clustering approaches.

A recent trend in the Process Mining community concerns the opportunity to exploit background knowledge to deal with incomplete logs, first pinpointed in Ref 44, where an ILP-based discovery method is described. Specifically, after extracting temporal constraints, capturing dependence, and parallelism relations between activities, negative events are generated artificially for each prefix of any log trace; using both log traces and artificial negative events as input, a logic program is induced with algorithm *TILDE*, which is eventually converted into a Petri net. Importantly, domain experts can directly provide an *a priori* set of temporal constraints, possibly stating that (1) two activities are parallel (respectively, not parallel), and (2) that one precedes/succeeds (respectively, does not precede/succeed). A constraint-based discovery framework was recently proposed in Ref 45, where the information gathered from the log and background knowledge are both expressed as precedence constraints, that is, constraints over the topology of the graphs. The search of a simple kind of process model, encoding only precedencies between tasks, is then rephrased into a constraints satisfaction problem (CSP), which is eventually solved by leveraging an existing CSP solver. Even though the resulting model does not capture typical control-flow constructs, the activity dependencies encoded by it can be given as input to other workflow induction algorithms,[30–32] to extract a fully expressive process model. Some major features of the approaches presented so far are summarized in Table 1.

As to the effectiveness of workflow discovery techniques in recognizing the actual structure of the analyzed process, various dimensions can be considered, which include the fitness and precision ones mentioned previously. Maximal fitness is actually achieved by almost all the algorithms above. However, this does not imply that the resulting model really captures the possible behavior of the unknown process, if the log does not satisfy the completeness notion underlying the induction algorithm. In particular, most algorithms based on heuristics-driven local search, assume that adjacent tasks appear consecutively in some traces. This discourse gets more varied when the process follows complex control-flow constructs (nonfree-choices, duplicate tasks, hidden tasks, etc.) and the logs are noisy. As a matter of fact, Table 1 also reports the behavior of some major process mining algorithms with respect to such issues.

On the contrary, the precision of process mining algorithms may rapidly fall when the analyzed process exhibits different execution scenarios, possibly combined with global behavioral constraints. In such a case, good results are achieved by approaches based on genetics algorithms[43] or on clustering.[5] Clearly, the first solution might be computationally unviable for large logs, whereas an excessive partitioning of log traces may lead to overfitting. In fact, more generally, the size of the log can impact severely on the real value of a discovered process model, especially when the analyzed process exhibits complex dynamics and a high level of concurrency. Indeed, in such a case, small samples of log traces hardly capture the different sequencing of activities that are admitted for the process, so that the model eventually discovered is likely to provide an undergeneralized ('overfitted') representation of the process' behavior. For example, it may happen that a precedency is incorrectly discovered between activities belonging to mutually

TABLE 1 | Summary of Process Discovery Techniques

Paper	Handled Issues: Noise	Duplic. Tasks	Hidden Tasks	Nonfree Choice	Loops	Learning Approach	Repr. Language	Backgr. Knowl.
35	–	–	–	–	–	Heuristics	Dep. graph	–
42	√	–	√	√	√	Genetic alg.	Petri Nets	–
5	√	–	–	√	–	Heuristics + clustering	Dep. graph	–
43	√	√	√	√	√	ILP	Petri Nets	√
39,40	√	√	√	–	–	Heuristics	Block structured	–
30	–	–	–	–	–	Heuristics	Petri Nets	–
14	–	–	–	–	–	Heuristics	EPC, Petri Nets	–
37	√	–	–	√	√	Heuristics	Dep. graph	–
31	–	–	–	√	–	Heuristics	Petri Nets	–
44	–	–	–	√	–	CSP solver	DAG	√
13	√	–	–	√	√	Heuristics	Dep. graph	–

The last two columns correspond to the formalism used for representing discovered process models and to the capability to take advantage of background knowledge about the structure of the process.

TABLE 2 | Summary of Trace Clustering Techniques

Paper	Structure Trace Representation	Nonstructural Properties	Clustering Bias	Approach
45	Sequences/string	√	Likelihood	Model based
5	Pattern-based vectors	√	Euclidian distance	K means
46	Sequences/string	–	Edit distance	AHC
47	Pattern-based vectors	√	Euclidian distance	AHC
48	Bag of activities/transitions	√	Euclidian/Hamming/Jaccard distance	K-means/AHC/SOM/QTC

The third and forth columns indicate the capability of accounting for properties going beyond the list of executed activities (e.g., data parameters, executors) and the basic similarity/dissimilarity criterion guiding the clustering, respectively.

parallel branches of the process, only because, in the given (incomplete) log, these activities always appear in the same order. A possible way to somewhat prevent the generation of overfitted models, in the case of clustering-based methods, is to simply set an upper bound to the number of clusters (as done in the algorithm of Figure 4).

Anyway, using abstraction mechanisms as a preprocessing or postprocessing tool can help alleviate this problem. In fact, as discussed in more details later on (in the section illustrating abstraction algorithm) a process discovery approach leveraging embedded abstraction capabilities was recently proposed in Ref 13. This method provides the analyst with a simplified dependency graph, where only significant enough activities and edges are depicted, while omitting (or aggregating) minor structural elements of the process structure.

TRACE CLUSTERING TECHNIQUES

Clustering techniques can help recognize different behavioral classes of process instances automatically, by exploiting the information captured in log data. In this section, we overview a series of recent methods for the clustering of workflow traces, which could be employed, within a recursive partitioning scheme, to induce a hierarchy of process execution classes, as discussed previously. Some major features of these methods are summarized in Table 2.

A first kind of approach to trace clustering relies on sequence-oriented techniques,[46,47] operating on the whole event trace 'as-is' based on string distance metrics. For instance, a context-aware approach based on the generic edit distance was proposed in Ref 47. The edit distance between two sequences is defined as the cost of the optimal combination of edit operations (insertion, deletion, or substitution) that allow to transform one sequence into another. The cost of edit operations is tailored to the peculiarities (primarily, concurrence nature) of workflow processes by devising ad hoc algorithms for automatically deriving an optimal setting of such costs. Moreover, an agglomerative clustering scheme is adopted, and the minimum variance criterion (trying to locally minimize intracluster variances) is used to select how clusters are to be grouped into higher level ones. Conversely, a model-based (probabilistic) approach is used in Ref 46, where, still regarding log traces as sequences, a mixture of first-order Markov models is found, via the *Expectation-Maximization* algorithm, which approximates their distribution at best.

The main drawback of string-oriented techniques[46,47] is the typically higher computational cost, which may make them unpractical when massive logs are to be analyzed. Notice that this problem cannot be circumvented, in general, by way of sampling techniques, as a huge number of distinct log traces can be actually necessary to rediscover a workflow with many parallel branches—which may yield many distinct log traces that only differ from each other in the ordering of the parallel (and hence mutually independent) tasks. And yet, heuristics-based correction mechanisms, for taking account of the concurrent nature of workflow processes, might by ineffective against highly concurrent processes.

In principle, higher scalability is achieved with feature-based approaches,[5,48,49] owing to the possibility to exploit consolidated efficient methods for clustering vectorial data and efficient algorithms for deriving the features from the given log. On the contrary, the quality of results depends on the capability of the considered structural patterns to capture and discriminate the main execution variants of the process. Hence, a trade-off between the expressiveness of the patterns used as features and the cost of extracting them must be suitably selected, according to the specific application context. Different methods have been proposed to map log traces into such a feature space, most of which focus on the frequency of activities in the log. A prevalent approach to clustering traces consists in transforming them into vectors where each dimension corresponds to an activity.[5] Clearly such a *bag-of-activities* representation, suffers, as a major drawback, from the loss of temporal information, as it disregard the ordering of activities.

One way to alleviate this problem is to regard any trace as a sequence of activities and to extract a number of k-grams (i.e., subsequences of length k) from it, as features for the clustering. In particular, in Ref 49, the vector space model is used with multiple feature types, corresponding to different trace *profiles*, that is, sets of related items describing traces from a specific perspective (activities, transitions, data, performance, etc.). Each item is associated with a measure assigning a numeric value to any trace. Therefore, by transforming each log trace into a vector containing all these measures, any distance-based clustering method can be exploited to partition the log. In particular, three distinct distance measures are considered to calculate the similarity between cases: Euclidean distance, Hamming distance, and *Jaccard* distance. Using these similarity measures, four clustering schemes are exploited applied to partition log traces: K-means, Quality Threshold Clustering (QTC), Agglomerative Hierarchical Clustering (AHC), and Self-Organizing Map (SOM).

This vector space model was combined with new context-aware features,[48] by expanding the core idea of considering activity subsequences that are conserved across multiple traces. Unlike the k-gram approach, subsequences of variable length are detected, which frequently occur in the log, and are assumed to correspond to some hidden functionalities of the process. Using these conserved subsequences as features, the clustering is expected to put together traces that are mutually similar from a functional viewpoint. In more details, the following kinds of conserved subsequences (inspired to sequence mining approaches) are used: Maximal Repeats, Super Maximal Repeats, and Near Super Maximal Repeats. Such subsequences are eventually as the dimensions of the vector space, while adopting Euclidean distance and the minimum variance criterion for the clustering.

The hierarchical clustering approach in Ref 5 exploits a special kind of sequential features, named *discriminant rules*, devised for capturing behavioral patterns that are not properly modeled by a given workflow model. Precisely, a *discriminant rule* has the from $[a_1 \ldots a_h]$ −/−> a such that (1) $[a_1 \ldots a_h]$ and $[a_h a]$ are both 'highly' frequent (i.e., the frequency is above a given threshold σ), and (2) $[a_1 \ldots a_h a]$ is 'lowly' frequent (its frequency is below another threshold γ). As an instance, the rule [fil]−/−>m for the example process *HandleOrder*, captures the fact that a fidelity discount is never applied when a (new) client is registered—this constraint is not captured by the worfklow model in Figure 1. Such rules can be straightforwardly derived from frequent sequential patterns, discovered efficiently via a level-wise search strategy.

As a final remark, we observe that the clustering of log traces might well take advantage of the good results achieved in the field of coclustering, one of the hottest topics in Data Mining community in recent years, where multiple data types are to be partitioned simultaneously based on their mutual correlations. Indeed, coclustering methods have shown to work well even when the real goal is to cluster one data type with a sparse and high-dimensional space of attributes (like, e.g., text documents and associated terms). A pioneering effort along such a direction was done in Ref 50 (in an outlier detection setting), where the mining of structural patterns is combined with a coclustering scheme focusing on the associations between such patterns and the given traces. In our opinion, such an approach can achieve good quality results when used for clustering the log of a process featuring a large number of structural patterns, without incurring in the notorious 'curse of dimensionality' problem.

Moreover, it could be beneficial to further investigate on the exploitation of model-based clustering schemes, which exhibited very good effectiveness and scalability performances in diverse data mining applications. Clearly this requires them to be suitably extended to effectively cope with the peculiar nature of workflow executions, and, in particular, with the presence of concurrent execution branches. In fact, this issue is not considered adequately in Ref 46, which mainly reuses a classical method conceived for purely sequential data.

PROCESS ABSTRACTION TECHNIQUES

A large body of work has been done to (semi-) automatically derive abstract views from a workflow model, to simplify the representation of the process. In principle, one could think of exploiting some of the inheritance-preserving transformation rules defined in Ref 28 to this end. However, to the best of our knowledge, no automated abstraction approach exists in the literature following that theoretical framework. By contrast, such a kind capability is featured by more recent process abstraction approaches,[6–13] typically based on less precise modeling languages and looser behavioral consistency notions. As a matter of fact, we believe that such an approximated modeling of process dynamics can be tolerated in a knowledge discovery setting, in exchange for a higher automation degree. Therefore, we next focus on these latter techniques, whose main features are reported in Table 3. In particular, the table reports, for each technique, which kinds of data it takes as input—by specifically telling whether it receives, or not, an execution log, a workflow model, an activity ontology, and which activities are to be abstracted (named here 'target tasks')—and which kinds of results it produces—that is, only an abstract workflow model, or a set of activity abstractions (regarded here as a sort of activity ontology), or both. Moreover, for each technique it is reported the underlying abstraction mode (i.e., aggregation or elimination of activities/edges) as well as whether the returned workflow model (if any) fulfils some notion of order preservation with respect to the input one (if any).

Most of these works only resort to the aggregation of process activities.[9–12] In particular, in Ref 9, an abstract view of a workflow model is obtained automatically by replacing multiple real activities with 'virtual' ones, based on ad hoc aggregation rules, ensuring that all original ordering relationships among the activities are preserved. In more detail, three rules must be followed to ensure the ordering property: (1) activity membership, (2) activity atomicity, and (3) order preservation. The first rule allows either base or previously defined virtual activities to be members of other virtual activities. The atomicity rule serves to describe the operational semantic property of the abstracted model. Finally, the order preservation principle provides a syntactical constraint ensuring that the abstracted processes also follow the atomicity property. In practice, this is meant to ensure that implied ordering relations in an abstract model comply with those in the base process. An algorithm is then illustrated, which can compute such an order-preserving abstracted version for a given process model (represented as a dependency graph with AND/XOR logics and single-entry single-exit loops), based on a reference subset of activities (named 'essential' activities) specified by the user, as mandatory abstraction targets (presumably corresponding to irrelevant or private tasks). The algorithm iteratively aggregates essential activities and adjacent ones into legal virtual activities (i.e., groups of base activities that does not violate any order-preservation constraint) until a fix point is reached.

In Ref 10, an abstraction approach is described, which relies on the partonomical decomposition obtained by building an *SPQR*-tree[33] for the given workflow model. As mentioned above, in such a tree, each leaf node coincides with a single (atomic) process task, whereas any other node corresponds to a 'fragment' of the workflow model. The approach relies on a manual control by the user, who is incharge of specifying which process task (or collection of tasks) in the original workflow model is to be abstracted. On the basis of a series of abstraction rules (specifically defined for each kind of composition pattern) the approach automatically replaces each task t, explicitly indicated by the user, with the finest grain workflow fragment encompassing t (i.e., the closest

TABLE 3 | Summary of Process Abstraction Approaches

	Input				Output		Method		
Paper	Log	Model	Abs. Dict.	Target Tasks	Abstr. Model	Abs. dict.	Abstr. Modes	Technical Aspects	Order Pres.
6	–	√	√	√	√	√	Aggreg.	Ontology-based matching scores and a fuzzy order-preservation score	–
7	√	–	–	–	–	√	Aggreg.	Hierarchical clustering of tasks (mapped to vectors of log-driven features)	–
8	√	–	–	–	–	√	Aggreg.	Extracts sequence-oriented frequent patterns from log traces	–
9	–	√	–	Many	√	–	Aggreg.	Graph reduction rules	√
10	–	√	–	One	√	–	Aggreg.	Replaces the target task with the smallest SPQR's block enclosing it	√
11	–	√	–	Many	√	–	Aggreg. + elimination	Finds, and remove or aggregate, minimal SESE fragments that enclose target tasks	√
12	–	√	√	–	–	√	Aggreg.	Finds task aggregations that best match nodes in the given partonomy	–
13	√	–	–	–	√	–	Aggreg. + elimination	Uses significance and correlation scores to decide which tasks/edges are to be abstracted	–

Note that in the case of Ref 6, we just consider the method implementing function *abstractSchema* (see Figure 6).

ancestor of t in the *SPQR*-tree). Clearly, the process can be iterated to produce coarser representations of the workflow.

A similar approach is proposed in Ref 11, aiming at providing personalized process visualization to the user based on her specific needs. Two basic abstraction operations are defined to this purpose: (1) aggregation and (2) reduction (i.e., elimination). Aggregation allows for replacing original activities with some abstracted (more general) elements. Reduction makes it possible to remove process activities when relevant information or confidential process details must be hidden to a particular user group. Both these operations rely on the so-called *SESE* (single entry single exit) fragments, that is, subprocesses having exactly one incoming edge and one outgoing edge connected with it. In more details, a reduction operation substitutes a SESE with a new edge between its predecessor and successor activity. Aggregation, instead, introduces a new, more general activity abstracting a SESE block. Aggregation and reduction operations are performed while trying to preserve, at best, the ordering relationships between activities, and other control-flow constraints.

The aggregation-based approach in Ref 12, still taking a workflow model as input, can yet take advantage of a partonomy relation over the activities, as a form of semantics-oriented background knowledge guiding the abstraction process. The approach is semi-automated in that it only suggests a list of possible activity aggregations (in the table, this fact is summarized with the sole return of an activity ontology as output), without computing an abstracted process view. However, the user can exploit each of these aggregations to eventually obtain such a view, by simply replacing it with an abstract activity. Essentially, the approach selects a list of activity groups such that, for each group G, (1) all the activities in G are topologically close enough in the process model (namely, their mutual distance in the workflow graph is lower than a given threshold) and (2) the activities in G achieve an optimal compliance score with respect to the input partonomy. This latter score is computed by way of a coverage measure, which, basically, evaluates what a percentage of the descendants of a partonomy node match some of the activities in G. Notably, the activity names in the input partonomy are allowed not to range over the same vocabulary as the activity labels in the model being abstracted, and a thesaurus-based similarity measure is exploited to meaningfully match process activities to partonomy terms.

As mentioned above, the approach in Ref 6 generally attempts to generalize multiple workflow schemas, describing different variants of a process, by producing an overall workflow schema where the structural elements connected with all shared activities are kept unchanged, whereas groups of 'specific' activities (i.e., activities not appearing in all the models) are replaced with new 'higher-level' activities, abstracting them all via *IS-A* or *PART-OF* relationships. Precisely, such abstraction is performed by way of a heuristics algorithm (whose major features are summarized in Table 3), which iteratively selects a pair (x,y) of 'specific' activities to be abstracted together into a single higher level activity. Such a pair is chosen in a greedy fashion, trying to minimize the number of spurious flow links that their merging introduces between the remaining activities, and considering their mutual similarity with respect to the contents of the activity ontology D. This is done by resorting to a series of affinity measures assessing how much any two 'specific' activities are suitable to be merged according the abstraction relationships already stored in D: (1) a 'topological' affinity measure $sim^E(x,y)$, measuring how similar the neighbourhoods of x and y are with respect to the flow graph; and (2) two 'semantical' affinity measures, $sim^P_D(x, y)$ and $sim^G_D(x, y)$, expressing how similar x and y are with respect to the relationships of *IS-A* and *PART-OF*, respectively, stored in D. All these measures are combined into an overall ranking function *score* as follows: $score(x,y) = 0$, if (x,y) is not a 'merge-safe' pair of activities; and $score(x,y) = \max\{sim^E(x,y), sim^P_D(x, y), sim^P_C(x, y)\}$, otherwise. By the way, a pair (x,y) of activities is said merge-safe (with respect to a given an set E of precedence relationships), if one of the following conditions holds: (1) there exist no path in E connecting x and y; (2) x and y are directly linked by some edges in E and after removing these edges no other path exists between them.

An emerging trend of research in the Process Mining community concerns the derivation of activity abstractions directly out of execution logs.[7,8] In general, such approaches tries to aggregate activities based on how they appears to be mutually correlated in past process traces. As we pointed out previously, such a kind of tools can be very helpful in the discovery of process taxonomies, especially for gaining a suitable level of abstraction over overly detailed logs. Moreover, by iterating the application of such techniques to abstracted logs, it is possible to discover a multilayer activity ontology. In particular, in the sequence-based approach of Ref 7, multiple abstraction levels are discovered for log events by using a hierarchical agglomerative clustering method, based on the proximity of events within log traces. Log traces can be then transformed into sequences of abstract activities by choosing a cut of the discovered hierarchy of event clusters, and by replacing each event with the ancestor lying on that cut. The second proposal[8] exploits instead repetition patterns (tandem repeats) and sequence patterns borrowed from bioinformatics, to capture loops and groups of correlated activities. Specifically, the approach works in two phases: first, it extracts repetition patterns by looking at log traces individually, and then discovers common groups of activities by logically regarding the whole log as a sequence. More complex constructs, such as choice and intraloop parallelism, are resolved by applying the preprocessing method on log traces iteratively. Efficient (suffix-tree based) structures are used to curb computation time. Moreover, to make the approach robust to the presence of both parallelism and choice constructs, a single abstract activity is created for patterns whose associated activity sets either contain each other or share many elements.

Leveraging the idea of using a process model as a map showing relevant aspects of process behavior, an abstraction-enhanced process discovery approach has been proposed in Ref 13, which is meant to describe effectively lowly structured processes. Essentially, the approach associate activities and edges with significance and correlation scores, and eventually shows a compact dependency graph where only edges and activities whose scores are above a user-given threshold, whereas less significant activities/edges are either grouped into abstract nodes, named (activity) clusters, if they are correlated enough among each other, or removed at all from the model, otherwise. As mentioned previously, this method, mixing features from both workflow induction techniques and log-driven abstraction ones, is a process discovery technique empowered with flexible abstraction capabilities.

A key point, in this setting, is the capability of the abstraction algorithm to take account for the fact that the output model is to generalize multiple process variants (i.e., execution classes). To this regard, a concept of common abstraction of multiple workflow schemas was also defined in Ref 28 (according to the behavioral inheritance notions given in the same work), but without providing any automated technique for its computation. In fact, the only method that was explicitly conceived to perform abstraction on (the nonleaf nodes of) a hierarchy of process models is the one in Ref 6. This is done by exploiting a particular implementation of meta-function *abstractSchema*, which takes as input a workflow schema (modeling a node *n* in the hierarchy), an activity ontology, and a set of tasks to be abstracted—which coincide with all the tasks associated with some of the children of *n*, but not with all of them. In fact, all of the other (semi-)automated model-based abstraction approaches[9–12] just consider a pair of workflows per time (i.e., a 'sub-class' workflow and a 'super-class' workflow), and furnish mechanisms for deriving one from the other. However, in principle, all of these methods can be exploited as well to carry out such a task, as they still allow the user to indicate which tasks need to be abstracted (named 'target tasks' in Table 3). Therefore, it is possible to reuse these approaches to abstract all the tasks in a process model that are not shared by all of its submodels in the taxonomy tree. Clearly, as the method in Ref 10 only takes a target task, such an operation would require multiple iterations of it. Moreover, all of these methods can be exploited by the analyst (possibly interactively) to further increase the level of abstraction of some models (e.g., the root one) in the discovered taxonomy.

Moreover, only a few methods[9,10,11] enjoy some kind of behavior consistency notion. More specifically, all of them are guaranteed to produce an abstracted model that preserves all ordering relationships in the original one—actually, in the case of Ref 11, little violations to this constraint are tolerated in exchange for achieving a more compact process view. Such an order preservation property is not fulfilled instead by any of the remaining approaches. However, it is possible to make the method in Ref 6 overcome this limitation, by simply ensuring that all the *PartOf* relation created in the abstraction, conforms to such an order-preservation notion. This can be done, for example, either (1) by enforcing ordering-oriented constraints over the activities, like in Ref 9, or (2) by requiring that activities can be abstracted by only using partonomical relationships implied by some given structural decomposition scheme (like the fragment-oriented ones used in Refs 10 and 11).

It is worth noticing that only the abstraction algorithms in Refs 6 and 12 can take advantage of existing abstraction relations (e.g., partonomies, hypernymies, or both) over process activities—even though the latter does not compute an abstract process model, but only gives the analyst a set of possible activity aggregations. Notably, this feature allows for possibly reusing available domain knowledge or the results of other process abstraction techniques, producing such a kind of relations,[6,7,8,12] or of classical approaches to the discovery of concept ontologies/hierarchies[14,22–25] from text documents (provided, in our case, with some textual description of process activities).

Finally, we observe that the computation times of all schema-oriented abstraction methods[6,9–11] are practically equivalent, as all of them are (low-degree) polynomials in the size of the input workflow models, which usually consist of few hundred of tasks at most (and of a sparse network of dependency links).

CONCLUSIONS

Several mining and abstraction methods have been presented, which can help discover a taxonomical process model for a given process, representing it at different abstraction levels. In particular, after automatically discovering a hierarchy of behavioral classes by way of suitable clustering methods, a process taxonomy can be derived by applying process abstraction methods to nonleaf nodes, eventually equipping them with higher-level models. Choosing an optimal combination of these basic tools is not easy in general, and it may well depend on the specific application domain at hand. However, we hope that our study can give some basic guidelines for analysts/designers who want to take advantage of such (semi-)automated techniques in their attempt to build such an expressive representation for a given business process and for its execution variants. A number of challenging issues are still open and deserve further investigation. For example, the recognition of abstract activities can benefit from available background knowledge on the activities' semantics, possibly extracted from a given thesaurus or a process ontology. Moreover, discovered process taxonomy can be exploited profitably to analyze relevant measures, such as usage statistics and performance metrics, along the different usage scenarios of the process at hand. Specifically, by using a taxonomy as an aggregation hierarchy for multidimensional OLAP analysis, it is possible to enable the user to interactively evaluate such measures over different groups of process instances. Notably, such an extension would be a valuable feature of interactive process mining systems, where the user is assisted in evaluating the discovered process models and in the tuning of parameters. Finally, the discovered taxonomies can serve as a basis for further knowledge discovery tasks, such as the mining of generalized association rules between, for example, the users or resources involved in the workflow process under analysis.

APPENDIX: EXPERIMENTAL RESULTS

To give some evidence for the benefits that can descend from the discovery of process models hierarchies and taxonomies, we next report an excerpt of the experimental analysis illustrated in Ref 6. As a matter of facts, in that work a specific instantiation of the two general meta-algorithms in Figures 4 and 6 are considered. Specifically, as concerns the discovery of a schema hierarchy (Figure 4), the induction of each workflow model (i.e., function *mineWFSchema*) is performed by using the algorithm in Ref 38, whereas the k-means-based clustering method introduced in Ref 5 is exploited to split a (sub-)log into clusters (i.e., function *partition*). Moreover, the taxonomy restructuring process hinges on the abstraction procedure introduced in the same work.[6]

Benefits of Log Clustering

A series of tests were performed on 10 synthesized benchmark log files, available in ProM, which reproduce different kinds of behavior, ranging from basic constructs like sequences, choices, parallel forks, and loop, to complex ones, like nonfree choice and invisible tasks. Three conformance measures,[51] all ranging over [0,1], are used here to evaluate the quality of discovered models. can roughly be defined as follows: (1) *Fitness* evaluates the percentage of mismatches occurring along a nonblocking replay of log traces through the model: the more the mismatches the lower the measure; (2) *Simple behavioral appropriateness* (*SB-Precision* for short) estimates the amount of the 'extra behavior' allowed by the model, quantified according to the average number of transitions that are enabled during a replay of the log; (3) *Advanced Behavioral Appropriateness* (*AB-Precision* for short), which expresses the amount of model flexibility (i.e., alternative or parallel behavior) that was not needed to replay the log. Conformance measures were only computed on leaf schemas only, which actually represent the concrete process variants found via the clustering, and averaged by assigning each schema a weight equal to the fraction of log traces fallen in its associated cluster.

The results computed in this way are compared, in Table A1, with those obtained by simply discovering a single workflow schema for the whole log, still using the base learning algorithm in Ref 38.

TABLE A1 | Quality Improvement, on Different Benchmark Logs, Achieved by the Clustering-Based Workflow Induction Scheme of Figure 4, (With Respect to a Single Overall Workflow Induced from the Whole Log)

Dataset	Fitness	SB-Precision	AB-Precision
a6nfc	1.2%	3.8%	54%
Example Log	2.0%	3.9%	93%
a7	3.9%	9.6%	39%
a100Skip	4.2%	1.8%	–
al1	4.2%	1.0%	–
DriversLicence	–	3.9%	29%
herbstFig6p36	–	2.0%	20%
al2	–	1.2%	–
CHOICE	4.8%	2.7%	–
a12	–	1.2%	–

More precisely, the table reports the increase (in percent) in the value of the three conformance measures that is achieved when passing from the base workflow induction algorithm[38] to the clustering-enhanced workflow discovery approach. Notably, this latter seems to overcome the difficulty of the base learner to deal with complex routing constructs and nonlocal task dependencies (cf., log files *a6nfc*, *herbstFig6p36*, and *DriversLicence*). This proves that more complete and precise process models can often be discovered by taking advantage of a clustering scheme, capable to separate different process variants.

Benefits of Process Abstraction

To show the benefit of abstraction mechanisms, we next report some results of an extensive experimentation (presented in Ref 6), which was conducted on the logs of an Italian maritime container terminal. Roughly speaking, the operational system supports and registers several logistic tasks for each container which come to the port, and underwent various kinds of moves over the 'yard'—that is, the main storage area used in the harbor, consisting of bidimensional slots, organized in blocks (nearly 100). A sample 5336 of such data was selected, corresponding to history of containers handled along the first two months of year 2006, and exchanged with other ports of the Mediterranean Sea. These data were converted in a process-oriented form, by encoding the sequence of yard blocks occupied by a container into a distinct log trace.

By applying the approach in Figure 4 (instantiated with the techniques in Refs 38 and 5), followed by the restructuring scheme of Figure 6 (instantiated with the abstraction method of Ref 6), a taxonomy of five workflow schemas was found, structured into three abstraction levels: the root T, two nodes T_0 and T_1, as children T, and two children of T_0, denoted as T_{0_0} and T_{0_1}. These schemes (omitted here for lack of space) differ neatly in complexity and readability. In particular, the leaf schema T_{0_0} consists of about 90 nodes, whereas the other two contain less than one half. A more compact process view was obtained for the higher-level views: 37 nodes in the schema T_0, and 32 nodes in the root schema.

Interestingly, the effect of using abstraction mechanisms was not merely syntactical. Indeed, many of the activities that were abstracted together share some semantical affinity. For example, some pairs of activities (i.e., yard blocks in this case) that were merged together via IS-A or Part-Of relationships are reported below: (1) (004D,04D), subsequently reckoned as two different names for the same block, due to a misspelling error; (2) (REF01,REF5), which are two blocks equipped with refrigerating systems; (3) (TR5,TRF5), (TR7,TRF7), which are both pairs of codes for multitrailer vehicles, logically viewed as (mobile) yard areas.

In conclusion, because of the large number of activity labels, any classical workflow discovery technique, in its own, would yield a rather unreadable and imprecise process model. By contrast, the process taxonomies computed with the approach described so far were reckoned by experts as really helpful for an explorative analysis of log data, thanks to both the recognition of distinct execution variants and the compactness of higher level schemas.

REFERENCES

1. van der Aalst WMP, van Dongen BF, Herbst J, Maruster L, Schimm G, Weijters AJMM. Workflow mining: a survey of issues and approaches. *Data Knowledge Eng* 2003, 47(2):237–267.
2. Casati F, Castellanos M, Dayal U, Shan MC. iBOM: a platform for intelligent business operation management. *Proceedings of 21st International Conference on Data Engineering (ICDE'05)*. IEEE Computer Society, Tokio; 2005, 1084–1095.
3. Davis R. *ARIS Design Platform: Advanced Process Modelling and Administration*. London: Springer; 2008, 426.
4. Malone TW, Crowston K, Lee J, Pentland B, Dellarocas C, Wyner G, Quimby J, Osborn CS, Bernstein A, Herman G et al. Tools for inventing organizations: toward a handbook of organizational processes. *Manage Sci* 1999, 45(3):425–443.
5. Greco G, Guzzo A, Pontieri L, Saccà D. Discovering expressive process models by clustering log traces. *IEEE Trans Knowledge Data Eng* 2006, 18(8):1010–1027.
6. Greco G, Guzzo A, Pontieri L. Mining taxonomies of process models. *Data Knowledge Eng* 2008, 67(1):74–102.
7. Günther CW, Rozinat A, van der Aalst WPM. Activity mining by global trace segmentation. *Business Process Management Workshops* 2009; 2009, 129–139.
8. Jagadeesh Chandra Bose RP, van der Aalst WPM. Abstractions in process mining: a taxonomy of patterns. *Proceedings of 7th International Conference on Business Process Management (BPM'09)*. Springer-Verlag, Ulm; 2009, 159–175.
9. Liu DR, Shen M. Workflow modeling for virtual processes: an order-preserving process-view approach. *Inform Syst* 2003, 28:505–532.
10. Polyvyanyy A, Smirnov V, Weske M. The triconnected abstraction of process models. *Proceedings of 7th International Conference on Business Process Management (BPM'09)*. Springer-Verlag, Ulm; 2009, 229–244.
11. Bobrik R, Reichert M, Bauer T. View-based process visualization. *Proceedings of 5th International Conference on Business Process Management (BPM'07)*. Springer-Verlag, Brisbane; 2007, 88–95.
12. Smirnov S, Dijkman R, Mendling J, Weske M. Meronymy-based aggregation of activities in business process models. *Proceedings of 29th International Conference on Conceptual Modeling (ER'10)*. Springer, Vancouver; 2010, 1–14.
13. Günther CW, van der Aalst WPM. Fuzzy mining: adaptive process simplification based on multi-perspective metrics. *Proceedings of 5th International Conference on Business Process Management (BPM'07)*. Springer-Verlag, Brisbane; 2007, 328–343.
14. Zhou L. Ontology learning: state of the art and open issues. *Inform Technol Manage* 2007, 8(3), 241–252.
15. Kashyap V. Design and creation of ontologies for environmental information retrieval. *Proceedings of 12th Workshop on Knowledge Acquisition, Modeling and Management* (EKAW'00). Springer-Verlag, Juan-les-Pins; 1999.
16. Williams AB, Tsatsoulis C. An instance-based approach for identifying candidate ontology relations within a multi-agent system. *Proceedings of ECAI*

2000 Workshop on Ontology Learning (OL'2000). IOS Press, Berlin; 2000.

17. Suryanto H, Compton P. Learning classification taxonomies from a classification knowledge based system. *Proceedings of ECAI 2000 Workshop on Ontology Learning (OL'2000)*. IOS Press, Berlin; 2000.
18. Pernelle N, Rousset MC, Ventos V. Automatic construction and refinement of a class hierarchy over semistructured data. *Proceedings of IJCAI 2001 Workshop on Ontology Learning (OL'2001)*. CEUR-WS.org, Seattle; 2001.
19. Kavalec M, Svatek V. Information extraction and ontology learning guided by web directory. *Proceedings of the ECAI 2002 Workshop on Machine Learning and Natural Language Processing for Ontology Engineering (OLT'2002)*. IOS Press, Berlin; 2002.
20. Maedche A, Staab S. Ontology learning for the Semantic Web. *IEEE J Intell Syst* 2001, 16(2):72–79.
21. Craven M, DiPasquo D, Freitag D, McCallum A, Mitchell T, Nigam K, Slattery S. Learning to construct knowledge bases from the World Wide Web. *Artific Intell* 2000, 118:69–113.
22. Chuang SL, Chien LF. A practical web-based approach to generating topic hierarchy for text segments. *Proceedings of 13th ACM International Conference on Information and Knowledge Management (CIKM'04)*. ACM, Washington, D.C.; 2004, 127–136.
23. Kozareva Z, Hovy R. A semi-supervised method to learn and construct taxonomies using the web. *Proceedings of International Conference on Empirical Methods in Natural Language Processing (EMNLP'10)*. ACL, MIT, MA; 2010, 1110–1118.
24. Li T, Zhu S. Hierarchical document classification using automatically generated hierarchy. *J Intell Inform Syst* 2007, 29(2):211–230.
25. Navigli R, Velardi P, Faralli S. A graph-based algorithm for inducing lexical taxonomies from scratch. *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'11)*. IJCAI/AAAI, Barcelona; 2011, 1872–1877.
26. Ricca F, Gallucci L, Schindlauer R, Dell'Armi T, Grasso G, Leone N. OntoDLV: an ASP-based system for enterprise ontologies. *J Log Comput* 2009, 19(4):643–670.
27. Stumptner M, Schrefl M. Behavior consistent inheritance in UML. *Proceedings 19th International Conference on Conceptual Modeling (ER'00)*. Springer, Salt Lake City, UT; 2000, 527–542.
28. Basten T, van der Aalst WMP. Inheritance of behavior. *J Log Algebraic Program*2001, 47(2):47–145.
29. Lee J, Wyner GM. Defining specialization for dataflow diagrams. *Inform Syst* 2003, 28(6):651–671.
30. van Dongen BF, van der Aalst WMP. Multi-phase process mining: aggregating instance graphs into EPCs and Petri Nets. *Proceedings of International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management (PNCWB'05)*. Florida International University, Miami, FL; 2005, 35–58.
31. van der Aalst WMP, Weijters AJMM, Maruster L. Workflow mining: discovering process models from event logs. *IEEE Trans Knowledge Data Eng* 2004, 16(9):1128–1142.
32. Wen L, van der Aalst WMP, Wang J, Sun JG. Mining process models with non-free-choice constructs. *Data Min Knowledge Discov* 2007, 15, 145–180.
33. Battista GD, Tamassia R. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica* 1996, 15(4):302–318.
34. Dijkman RM, Dumas M, Garcia-Bañuelos L, Käärik R. Aligning business process models. *Proceedings of 13th International Enterprise Distributed Object Computing (EDOC'09)*. IEEE Computer Society, Auckland; 2009, 45–53.
35. van Dongen BF, Dijkman RM, Mendling J. Measuring similarity between business process models. *Proceedings of 24th International Conference on Advanced Information Systems Engineering (CAiSE'08)*. Springer, Montpellier; 2008, 450–464.
36. Agrawal R, Gunopulos D, Leymann F. Mining process models from workflow logs. *Proceedings of 6th International Conference on Extending Database Technology (EDBT'98)*. Springer, Valencia; 1998, 469–483.
37. Chen CWK, Yun DYY. Discovering process models from execution history by graph matching. *Proceedings 4th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'03)*. Springer, Hong Kong; 2003, 887–892.
38. Weijters AJMM, van der Aalst WMP. Rediscovering workflow models from event-based data using Little Thumb. *Integr Comput Aided Eng* 2003, 10(2):151–162.
39. de Medeiros AKA, van Dongen BF, van der Aalst WMP, Weijters AJMM. Process mining: extending the α-algorithm to mine short loops. *Technical Report WP* 2004, 113.
40. Herbst J, Karagiannis D. Integrating machine learning and workflow management to support acquisition and adaptation of workflow models. *J Intell Syst Account, Finance Manage* 2000, 9:67–92.
41. Herbst J, Karagiannis D. Workflow mining with InWoLvE. *Comput Ind* 2003, 53(3):245–264.
42. Schimm G. Mining most specific workflow models from event-based data. *Proceedings of International Conference on Business Process Management (BPM'03)*. Eindhoven, The Netherlands: Springer; 2003, 25–40.
43. de Medeiros AKA, Weijters AJMM, van der Aalst WMP. Genetic process mining: an experimental evaluation. *Data Min Knowledge Discov* 2007, 14(2):245–304.

44. Goedertier S, Martens D, Vanthienen J, Baesens B. Robust process discovery with artificial negative events. *J Mach Learn Res* 2009, 10:1305–1340.

45. Greco G, Guzzo A, Pontieri L. Process discovery via precedence constraints. *Proceedings of 20th European Conference on Artificial Intelligence (ECAI'12)*. IOS Press, Montpellier; 2012.

46. Ferreira DR, Zacarias M, Malheiros M, Ferreira P. Approaching process mining with sequence clustering: Experiments and findings. *Proceedings of 5th International Conference on Business Process Management (BPM'07)*. Springer, Brisbane; 2007, 360–374.

47. Jagadeesh Chandra Bose RP, van der Aalst WPM. Context aware trace clustering: towards improving process mining results. *Proceedings of SIAM International Conference on Data Mining (SDM'09)*. SIAM, Sparks, NV; 2009, 401–412.

48. Jagadeesh Chandra Bose RP, van der Aalst WPM. Trace clustering based on conserved patterns towards achieving better process models. *Proceedings of 5th International Workshop on Business Process Intelligence (BPI'09)*. Springer, Ulm; 2009, 170–181.

49. Song M, Günther CW, van der Aalst WPM. Trace clustering in process mining. *Business Process Management Workshops (BPM'08)*. Springer, Milan; 2008, 109–120.

50. Ghionna L, Greco G, Guzzo A, Pontieri L. Outlier detection techniques for process mining applications. *Proceedings of 17th International Symposium on Foundations of Intelligent Systems (ISMIS'08)*. Springer, Toronto; 2008, 150–159.

51. Rozinat A, van der Aalst WMP. Conformance checking of processes based on monitoring real behaviour. *Inform Syst* 2008, 33(1):64–95.

