Je-LKS
Journal of e-Learning and Knowledge Society
The Italian e-Learning Association Journal
PEER REVIEWED PAPERS
NEW TRENDS, CHALLENGES AND PERSPECTIVES ON HEALTHCARE COGNITIVE COMPUTING: FROM INFORMATION EXTRACTION TO HEALTHCARE ANALYTICS
Vol. 14, n.1, 2018
ISSN: 1826-6223 | eISSN: 1971-8829

A CONTEXT-AWARE MINER FOR MEDICAL PROCESSES

Luca Canensi[1]
Giorgio Leonardi[2]
Stefania Montani[2]
Paolo Terenziani[2]

[1] Department of Computer Science, University of Torino, Italy
[2] DISIT, Computer Science Institute, University of Piemonte Orientale, Alessandria, Italy
canensi@di.unito.it, giorgio.leonardi,stefania.montani,paolo.terenziani@uniupo.it

Keywords: Process Mining, Medical Processes, Context

Medical process mining is gaining much attention in recent years, but the available mining algorithms can hardly cope with medical application peculiarities, that require to properly *contextualize* process patterns. Indeed, most approaches lose the connection between a mined pattern and the relevant portion of the input event log, and can have a limited precision, i.e., they can mine incorrect paths, never appearing in the input log traces. These issues can be very harmful in medical applications, where it is vital that mining results are reliable as much as possible, and properly reference the contextual information, in order to facilitate the work of physicians and hospital managers in guaranteeing the highest quality of service to patients. In this paper, we propose a novel approach to medical process mining that operates in a context-aware fashion. We show on a set of critical examples how our algorithm is able to cope with all the issues sketched above. In the future, we plan to test the approach on a real-world medical dataset, and to

extend the framework in order to support efficient and flexible trace querying as well.

1 Introduction

Process mining describes a family of a-posteriori analysis techniques that exploit the so-called *event log*, which records information about the sequences (*traces* henceforth) of events (i.e., activities) executed at a given organization. The most relevant and widely used process mining technique is discovery; process discovery takes as an input the event log and produces a process model, without using any a-priori information. The result is typically expressed in terms of a Petri Net, or some other process notation, often shown as a graph, in which nodes represent activities, and arcs provide control flow information. Medical process mining is a research field which is gaining attention in recent years (see, e.g., Mans *et al.*, 2009; 2008, Perimal-Lewis, 2012). This application domain indeed presents particular challenges and issues (Mans *et al.*, 2013), mostly related to the fact that different types of patients exhibit different characteristics, that the patient's state dynamically evolves (and is influenced by the medical activities executed on her/him), and that different hospital settings may have different resource constraints. All these peculiarities lead to the need to properly contextualize medical processes and/or process patterns. The currently available process mining solutions, ranging from commercial tools (offered, e.g., by Fujitsu Ltd and Celonis), to the open-source framework ProM (van Dongen *et al.*, 2005) (developed at the Eindhoven University of Technology), which represents the state of the art in process mining research, are not tailored to medical applications, and do not take into account contextualization needs. Specifically, despite some differences, many current algorithms, including *heuristic miner* (Weijters *et al.*, 2006) one of the most popular and widely used algorithms available in ProM, show important similarities (see Section 2.1), and have common limitations:

- they learn "context-free" patterns of processes;
- they can mine paths that do not correspond to any input trace in the log (i.e., they can have a limited *precision* (Buijs *et al.*, 2012);
- they do not explicitly relate the mined patterns to the log (in the sense that there is no explicit correspondence between mined patterns, and the traces in the log "supporting" them).

Such limitations are quite relevant in general, and very relevant in the medical domain. Concerning the first limitation, it is well known that, e.g., the same (set of) activities may produce different effects on patients, depending on the context (e.g., on the activities previously performed on the patients themselves). The impact of the second limitation is obvious and dramatic: if the miner precision is limited, in the sense that it may also learn a path that never appears in any input trace, this can be very harmful in medical applications, where it is vital that mining results are reliable as much as possible, in order to facilitate the work of physicians and hospital managers in guaranteeing the highest quality of service to patients. Indeed, the mined model is the input for quality assessment procedures, such as verification of conformance with respect to clinical guidelines, or performance measurement and bottleneck detection. All these procedures will provide an unreliable output, if played on an unreliable input. However, surprisingly, limited precision is a common limitation of many current miners (see Section 2.1.) The third limitation is less critical, but still significant. Indeed, maintaining an explicit link between mined patterns and the input traces matching such patterns, can be important not only to characterize contexts, but also to provide physicians with an evidence of the learned output, and also to provide support for retrieving traces corresponding to a given pattern.

In this paper, we propose an innovative approach that supports **context-aware** process mining, and overcomes all the above limitations.

2 A critical analysis of current approaches

As discussed in the Introduction, several different miners have been developed in the literature. However, many existing miners, including *alpha miner* (Van der Aalst & van Dongen, 2002) *fuzzy miner* (Gunther & Van der Aalst, 2007), *heuristic miner* (Weijters *et al.*, 2006) and the very recent inductive *tree miner* (Leemans *et al.*, 2013) show interesting commonalities. In this section, we will first illustrate common assumptions and methodological choices of the available mining approaches. Then, we will move to the discussion of the impact of these common issues on a set of examples. For the sake of clarity and brevity, we will refer to just one miner to illustrate the common characteristics of these literature approaches. Specifically, we will concentrate on heuristic miner. Indeed, heuristic miner can abstract from exceptional behavior and noise and, therefore, is suitable for many real-life logs, including medical ones. Moreover, it can mine the presence of cycles and of short distance and long distance dependencies, by means of dedicated procedures (see Section 2.1). Another advantage of this algorithm is that its default output graph can be easily converted to other types of process representation formalisms, including Petri Nets. These features make heuristic miner one of the most popular and widely used algorithms available in ProM, and a good reference for our comparisons.

2.1 Mining algorithms commonalities

Although different representation formalisms are adopted to model the mined processes (for instance, heuristic miner itself supports three different possible representation formalisms for the output), most mining algorithms are based on a common assumption:

Uniqueness assumption. Each event is unique in the output, so that each event appears at most once in the output of the process miner.

As a second commonality, the mining methodology is typically based on two steps (corresponding to steps 1 and 3 in Algorithm 1 below, which abstractly characterizes the behavior of heuristic miner on a set T of traces):

(i) a **de-structuring step**, in which immediate precedence between pairs of events are detected in the input traces, and

(ii) a **re-structuring step**, in which patterns of events are reconstructed, by combining the immediate precedence relation mined in the de-structuring step.

Algorithm 1: HM pseudocode

```
1 function HM(T) ;
2 (1) for each pair of events $A$ and $B$ do
3     search T to detect immediate precedence $A \rightarrow B$ and $B \rightarrow A$
4 end
5 (2) Look for cycles of length one (same event repeated) or two (pair of events
  repeated) ;
6 (3) for each triplet of events $A$,$B$, and $C$, such that $A \rightarrow B$ and $A \rightarrow C$ do
7     use formulae [6] to discriminate whether B and C are in AND (both of them
      must be executed after A, in any order) or in XOR (exactly one of them
      must be executed after A), and construct the output
8 end
9 (4) Look for "long-distance dependencies" between events, and add them to the
  output.
```

The main critical issues in the algorithm are the following:

- C1 Given the uniqueness assumption, the de-structuring step (step 1 in Algorithm 1) evaluates the immediate precedence relations between events A and B looking at sequences "AB" and "BA" in the input traces, regardless of their position in the traces, and, thus, regardless of the context.
- C2 Given the fact that the de-structuring step ignores the context, and that the re-structuring step (step 3 in Algorithm 1) does not take into account the traces in the log, also the re-structuring step does not consider the context at all.
- C3 The uniqueness assumption imposes severe constraints on the re-structuring step.

In the following subsection, the critical impact of issues C1, C2 and C3 will be discussed on some easy examples. For the sake of concreteness, the examples will be processed using heuristic miner, and provided as Petri Nets, which are commonly assumed as an incontestable formalism to represent (the semantics of) processes.

2.2 Critical examples

To simplify the presentation, with no loss of generality, we suppose that each trace is prefixed with a distinguished starting symbol (*) and postfixed with another distinguished ending symbol (#).

***Ex. 1** Log content: 1000 equal traces: *ABCA#*

Although all the traces are equal, the approaches discussed in Section 2.1 cannot learn the (unique!) pattern. In this example, this is mainly due to the fact that the uniqueness assumption causes problems in the restructuring phase (critical issue C3): the two occurrences of the event A in the traces must correspond to a unique transition in the output Petri Net. Thus, a cycle (returning from C to A) must be introduced in the output. Notably, the output Petri Net also admits patterns like *A# or *ABCABCA#, which are not present in any of the input traces (see Figure 1(a)).

The pattern shown in this example represents one of the typical care processes for patients suffering from acute diseases. For instance, consider the diagnostic process of patients suffering from cerebral ischemia. After the symptom onset, the patient arrives at the emergency ward of the hospital. According to the latest medical guidelines for the treatment of stroke (Carlucci & Inzitari, 1996), it is recommended that the patient undergoes as soon as possible a computed tomography (CT), for: (1) the differential diagnosis between ischemic and hemorrhagic stroke and other cerebrovascular diseases, and (2) the identification of possible early signs of ischemic brain suffering. This action is indicated in Ex. 1 as action A = CT Execution. Subsequently, the patient should be evaluated by a trained neurologist, generating action B = Neurological Evaluation. Among the various additional investigations, which may be required depending on the type of patient, the guidelines suggest to always perform an electrocardiogram (ECG). In Ex. 1, this is referred to as action C = ECG Execution. Before transferring the patient to the Stroke Unit for further treatment, the guidelines recommend the repetition of the CT (within 48 hours), in order to obtain a better diagnostic and prognostic evaluation. The action A = CT Execution is therefore performed both as the first action of this process, and as the last one, thus generating the pattern of actions of this example.

In a similar manner, each of the following examples introduces situations which can be concretely instantiated as real clinical processes, or parts of them.

Ex.2 *Log content: 1000 equal traces: *ABCBA#*

This example highlights the limitations of “context-free”” approaches. Consider, in particular, critical issue C1 above: since the precedence relations are searched without considering the context (and on the basis of the uniqueness assumption), there is no way to decide whether A precedes B (100% of traces, considering the first part of the traces) or B precedes A (100% of traces, but considering the last part of the traces), and analogously for the relation between B and C. As a consequence, two separate Petri Nets are learned: the first contains only the pattern *A#, while the second contains a loop composed by C and B (see Figure 1(b)). By using heuristic miner without the “all event connected option”, a different model is learned (see Figure 1(c)), where it is possible to replay the input trace, but many other never observed behaviors can be generated as well.

Ex.3 *Log content: 500 traces *AD# and 500 traces *DB#*

Once again, the combination of a “context-free” analysis with the uniqueness assumption causes undesired effects. In this example, two different patterns should be mined. However, during the de-structuring phase, no distinction is made between D in the first type of traces (i.e., D in the context of being preceded by *A) and D in the second type of traces (i.e., D in the context of being preceded by *). Thus, the re-structuring phase (see critical issue C2) generates a “merging” between the two different patterns (due to the uniqueness of D), providing the Petri Net in Figure 1(d) as output. Notably, besides the correct patterns *AD# and *DB#, the output Petri Net also models the patterns *D# and ADB#, which do not correspond to any trace in the input log.

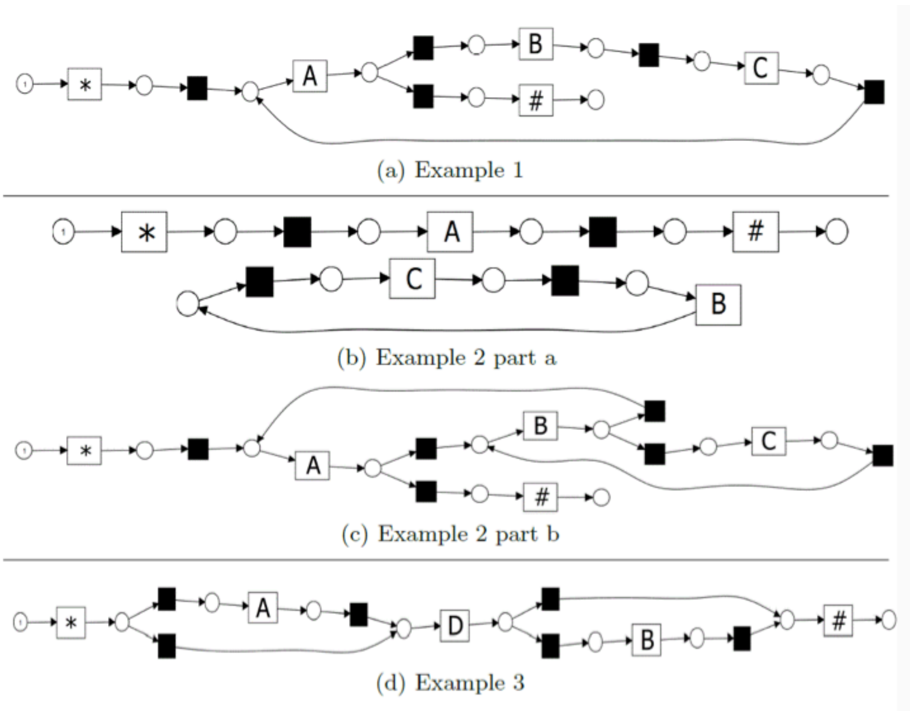


Fig. 1 - Models mined by heuristic miner referring to the examples of Section 2.2

3 Context-aware process mining

In order to overcome the limitations illustrated in the previous section, we propose an innovative approach to (medical) process mining, which is based on quite a different philosophy: in our methodology, process mining heavily takes into account contextual information, both in the data structure (the output graph), and in the mining algorithm. Our approach is based on three main ideas:

- in the data structure, we **remove the uniqueness assumption**: the same event may appear more than once in the output graph, to model the fact that it may occur in different contexts;
- in the data structure, and in the mining algorithm, we explicitly maintain the context, i.e., the set of log traces that *support* a given path in the mined graph;
- in the mining algorithm, we take advantage of the ordering of events in the log traces (which represents, indeed, the context in which each single event occurs) in order to directly mine the output graph, **without distinguishing between a de-structuring and a re-structuring phase**.

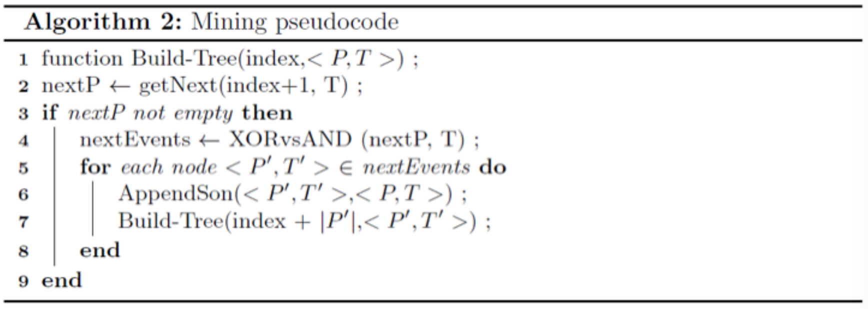
In the following, we present our approach and its application to the previously discussed critical examples.

3.1 Data structure and algorithm

Our mining algorithm takes in input a log (set of traces). The log is represented by a matrix, in which each row is a trace. It outputs the mined process as an acyclic directed graph, called a "process graph", in which nodes represent events (i.e., activities), and arcs represent a precedence relation between them. More precisely, in our model, each node is represented as a pair $< P; T >$:

- *P* denotes a (possibly unary) set of events; events in the same node are in *AND* relation, or, more properly, may occur in any order (with respect to each other). Note that, in such a way, each path from the starting node of the graph to a given node N denotes a set of possible process patterns (called *support patterns* of N henceforth), obtained by following the order represented by the arcs in the path to visit the process graph, and ordering in each possible way the events in each node (for instance, the path *{A,B}->{C}* represents the support patterns "ABC" and "BAC").
- *T* represents the context, i.e., a set of references to all and only those traces in the log which exactly match one of the patterns in *P* (called *support traces* henceforth).

Our mining algorithm operates in two steps. The first step (see Algorithm 2 below) is the core of our approach, and builds a tree of nodes described as above.



Algorithm 2: Mining pseudocode

```
function Build-Tree(index,$< P, T >$) ;
nextP ← getNext(index+1, T) ;
if nextP not empty then
    nextEvents ← XORvsAND (nextP, T) ;
    for each node $< P', T' > \in$ nextEvents do
        AppendSon($< P', T' >$,$< P, T >$) ;
        Build-Tree(index + $|P'|$,$< P', T' >$) ;
    end
end
```

Build-Tree in Algorithm 2 takes in input a variable *index*, representing a given position in the traces (i.e., a column in the input matrix), and a node. Initially, it is called on the first position, and on the root of the tree (which is a "dummy" node, corresponding to the * event; thus, initially, index=0, P=*

and T is the set of all the traces). The function *getNext* simply inspects the traces in *T* to find all possible next events (in the context *T*). On the basis of the current context (*support traces*) *T*, the function *XORvsAND* applies the formulae described in appendix A to identify which events are in AND and which are in XOR relation. The output of such a function is a set of nodes < *P', T'* >, one for each maximal set of events to be AND-ed. Note that, for each one of such sets *P'*, the corresponding set *T'* of support traces is also computed, on the basis of the current context *T*. Finally, each new node is appended in the output tree (function *AppendSon*), and *Build-Tree* is recursively applied to each node (with the parameter *index* properly set). The second step is a simple transformation of the tree into an acyclic directed graph, obtained by merging, starting from the leaves nodes, those paths in the tree whose *support pattern* postfixes are identical. Notably, our mining algorithm explicitly manages the context, focusing at each step on the proper *support traces*, and always taking into account the global ordering of events in the traces to build the graph. As a consequence, differently from the heuristic miner algorithm shown in Algorithm 1, we do not need any additional step to cope with long-distance dependencies (we have them "for free"). Analogously, we "naturally" cope with cycles by simply unfolding them. Thus, our algorithm already directly copes with *cycles* of any *length* (and no additional ad-hoc procedure for cycles is needed, differently from the heuristic miner algorithm).

3.2 Examples

In Figure 2, we show the output of our miner, applied to the examples Ex.1-Ex.3 above. Support traces are not reported in the figure, but are part of the output itself. As it can be observed, the critical situations discussed in Section 2.2 are all correctly managed by our approach. It is also worth noting that our process graphs could be easily converted into Petri Nets.

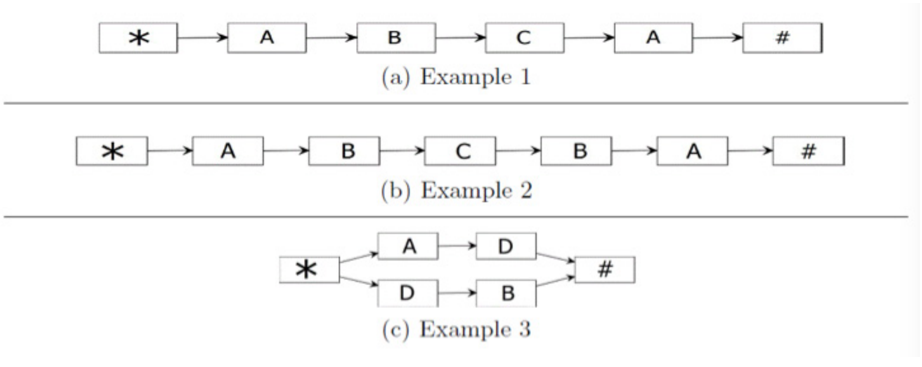


Fig. 2 - Models mined by our miner referring to the examples of Section 2.2

Conclusions

In this paper, we have introduced a novel process mining algorithm, able to overcome the limitations of many current approaches available in the literature. Specifically, our algorithm:

- learns "context-aware" patterns of processes;
- has a high precision, since it provides patterns that always correspond to input traces in the log;
- explicitly relates the mined patterns to the traces in the log "supporting" them.

Our approach properly deals with critical situations that may occur in practical application domains, as illustrated by the examples in section 2.2. These characteristics make the algorithm particularly well-suited for medical applications, where it is vital that mining results are reliable as much as possible, in order to facilitate the work of physicians and hospital managers in guaranteeing the highest quality of service to patients. In the future, we plan to test the approach on a real-world medical dataset, taken from the stroke patient management domain.

From the methodological viewpoint, we also aim at extending the framework, in order to support efficient and flexible trace querying. Indeed, the model we mine maintains an explicit link between mined patterns and the input traces supporting them, and can thus be seen as an indexing structure, well suited to quickly retrieve traces corresponding to the pattern at hand.

REFERENCES

Mans R., Schonenberg H., Song M., Van der Aalst W., Bakker P. (2009), *Application of process mining in healthcare - a case study in a Dutch hospital*, in: Fred A., Filipe J., Gamboa H. (eds), Proc. BEST, Communications in Computer and Information Science (25), 425-438, Springer.

Mans R., Schonenberg H., Leonardi G., Panzarasa S., Cavallini A., Quaglini S., Van der Aalst W (2008), *Process mining techniques: an application to stroke care*, in: Andersen S., Klein G., Schulz S., Aarts J. (eds), Proc. MIE, Studies in Health Technology and Informatics (136), 573-578. IOS Press.

Perimal-Lewis L., Qin S., Thompson C., Hakendorf P. (2012), *Gaining insight from patient journey data using a process-oriented analysis approach*, in: Butler-Henderson K., Gray K. (eds), Proc. Workshop HIKM, CRPIT (129), 5966, Australian Computer Society.

Mans R., Van der Aalst W., Vanwersch R., Moleman A. (2013), *Process mining in healthcare: data challenges when answering frequently posed questions*, in: Lenz

R., Miksch S., Peleg M., Reichert M., Riano D., ten Teije A. (eds),, ProHealth/KR4HC, LNCS (7738) 140-153. Springer.

van Dongen B., Alves De Medeiros A., Verbeek V., Weijters A., Van der Aalst W. (2005), *The proM framework: a new era in process mining tool support*, in: Ciardo G., Darondeau P. (eds), Knowledge Management and its Integrative Elements, 444-454, Springer.

Weijters .A., Van der Aalst W., Alves de Medeiros A. (2006), *Process Mining with the Heuristic Miner Algorithm*, WP 166, Eindhoven University of Technology.

Buijs J., van Dongen B., Van der Aalst W. (2012), *On the role of fitness, precision, generalization and simplicity in process discovery*, in: On the Move to Meaningful Internet Systems, OTM, 305-322. Springer.

Van der Aalst W., van Dongen B. (2002), *Discovering workflow performance models from timed logs*, in: Han Y., Tai S., Wikarski D. (eds), Proc. EDCIS 2002, LNCS (2480), 45-63, Springer.

Gunther G, Van der Aalst W. (2007), *Fuzzy mining - adaptive process simplification based on multi-perspective metrics*, in: Alonso G., Dadam P., Rosemann M. (eds), Proc. BPM LNCS (4714) 328-343, Springer.

Leemans S., Fahland D., Van der Aalst W. (2013), *Discovering block-structured process models from event logs containing infrequent behaviour*, in: Lohmann N., Song M., Wohed P. (eds), Proc. BPM Workshops, LNBIP (171) 66-78, Springer.

Carmona J., Cortadella J., Kishinevsky M. (2008), *A Region-Based Algorithm for Discovering Petri Nets from Event Logs*, Proc. BPM, 358-373, Springer.

Carlucci G., Inzitari D., (1996), *Italian stroke guidelines (spread): evidence and clinical practice*. Neurological Sciences (27) S225-S227.

APPENDIX: XOR VS AND FORMULAE

In Algorithm 2, after finding the set of successors *nextP* of the considered set of events P, we focus on the discovery of the relation between them. In order to do this, we calculate the *dependency frequency* between every event pairs < A;B >

in *nextP* _ *nextP* :

$$A \rightarrow B = \frac{1}{2}\left(\frac{|A > B|}{\sum_{X \in traces} |A > X|} + \frac{|A > B|}{\sum_{Y \in traces} |Y > B|}\right) \qquad (1)$$

where $|A > B|$ is the number of traces in which A is immediately followed by B, and $|A > X|$ is the number of traces in which A is immediately followed by some event X, $|Y > B|$ is the number of traces in which B is immediately preceded by some event Y. After evaluating the dependency frequency value *A -> B* and *B -> A*, we can have the following possible situations:

- if both the values are below a given threshold, this means that A and B rarely appear in the same trace, therefore they are in XOR relation;

- if *A -> B* is above the threshold and *B -> A* is below, then A precedes B, and, viceversa;
- if both the values are above the threshold, then A and B are in AND (any-order) relation.