Data & Knowledge Engineering 70 (2011) 821–841

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Sequence partitioning for process mining with unlabeled event logs ☆

Michał Walicki [a], Diogo R. Ferreira [b,*]

[a] Institute of Informatics, University of Bergen, Norway
[b] IST, Technical University of Lisbon, Portugal

ARTICLE INFO

Article history:
Received 9 June 2010
Received in revised form 10 May 2011
Accepted 10 May 2011
Available online 20 May 2011

Keywords:
Process mining
Sequential pattern mining
Sequence partitioning
Combinatorics on words

ABSTRACT

Finding the case id in unlabeled event logs is arguably one of the hardest challenges in process mining research. While this problem has been addressed with greedy approaches, these usually converge to sub-optimal solutions. In this work, we describe an approach to perform complete search over the search space. We formulate the problem as a matter of finding the minimal set of patterns contained in a sequence, where patterns can be interleaved but do not have repeating symbols. This represents a new problem that has not been previously addressed in the literature, with NP-hard variants and conjectured NP-completeness. We solve it in a stepwise manner, by generating and verifying a list of candidate solutions. The techniques, introduced to address various subtasks, can be applied independently for solving more specific problems. The approach has been implemented and applied in a case study with real data from a business process supported in a software application.

1. Introduction

A business process is a structured set of activities which can be instantiated multiple times and whose execution is assumed to be recorded in an event log. The goal of process mining [1,2] is to rediscover the process model from the runtime behavior of process instances recorded in the event log. The event log contains a sequence of entries in the form (*case id*, *task id*) where *case id* identifies the process instance and *task id* specifies the task that has been performed. The sequence of tasks recorded during the execution of a process instance is called a *workflow trace* [3]. Since several process instances may be active simultaneously, the traces for different instances may overlap in time, as illustrated in Fig. 1.

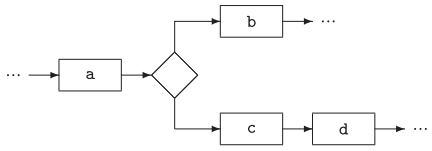
For the purpose of process mining, the overlapping of workflow traces is not a problem, since each event is associated with the *case id* which clearly identifies the process instance that the event belongs to. A wide range of process mining techniques [4] has been devised to discover process models from such event logs. However, while these event logs can be easily obtained from workflow and case-handling systems, in other applications that are not fully process-aware it may become difficult to retrieve event data in that form. If the case id attribute is missing, then the event log becomes an unlabeled sequence of events where it is unknown whether any two events belong to the same process instance or not.

The problem we address in this paper is how to recover the workflow traces from an unlabeled sequence of events. A sample sequence is illustrated in the top-right corner of Fig. 1. This sequence of symbols is the result of several workflow traces becoming interleaved in the event log. But since workflow traces essentially repeat the sequential patterns of the business process, in principle it should be possible to identify these repeating patterns in the unlabeled sequence. As in Fig. 1, and for reasons to be explained below, we consider patterns without repeating symbols. In addition, for a subsequence to qualify as a pattern, it should have at least 2 symbols and at least 2 occurrences in the sequence.

☆ This work was performed while the first author was visiting the Technical University of Lisbon.

* Corresponding author at: Instituto Superior Técnico, Campus do Taguspark, Avenida Prof. Dr. Cavaco Silva, 2744-016 Porto Salvo, Portugal. Tel.: +351 21 423 35 52.
E-mail address: diogo.ferreira@ist.utl.pt (D.R. Ferreira).

0169-023X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2011.05.003



case id	*task id*
1	a
2	a
2	c
1	b
3	a
…	…

```
   aacbabadaccddacd
1  a..b............
2  .ac....d........
3  ....ab..........
4  ......a..c..d...
5  ........a.cd....
…
```

Fig. 1. An excerpt of a generating process, its event log and the workflow traces.

Under these conditions, the sequence *aacbabadaccddacd* of Fig. 1 admits solutions with 2, 3 and 4 patterns. Table 1 lists some of these solutions. Each solution contains a set of patterns, and each pattern has its own multiplicity (number of repetitions). This is written as $\langle p_1^{n_1}, \ldots, p_k^{n_k}\rangle$, where each p_i is a pattern (sequence of symbols) and n_i is the number of occurrences of pattern p_i in the solution. The "true" solution, i.e. the one that corresponds to the generating patterns, can be found in the first column. Here it can be seen that the sequence admits an alternative solution with 2 patterns. There are many more solutions with 3 and 4 patterns, but in general we will be interested in solutions with a minimal set of patterns, as these can provide a more compact representation of the generating process.

1.1. Related work

While there is a host of problems and techniques in the area of sequential pattern mining [5–12], the present problem does not seem to have been investigated before in this general form. There are some variants completely unrelated to ours, for instance, of partitioning *numerical* sequences into substrings, minimizing some cost function, e.g. [13]. In closer variants, the discovery of sequential patterns is usually bound to constraints [14] such as time windows [15,16], regular expressions [17], or some domain-specific knowledge [18]. There are also approaches that focus on mining specific patterns or on counting their occurrences [19] but this represents just one of the subtasks in finding the minimal set of patterns that cover a given sequence.

Related problems have been formulated in other areas but they do not seem to match exactly the problem considered here. For instance, in the field of combinatorics on words there is a related problem of how many subsequences (and how long) are needed in order to determine a given sequence uniquely [20]. This has a different focus from our problem, since we want to find all possible sets of patterns in order to pick the minimal solution. An interesting result was reported in Ref. [21], allowing to decide if a given regular language can be obtained as the shuffle product of two other regular languages. Unfortunately, as we are dealing with a one-word language (a single sequence), it is obvious that any such language can be obtained by shuffling any of its non-empty subsequences. Viewed as a special case of the problem addressed in Ref. [21], our instance is trivial but the algorithm presented there, addressing a related but different problem, does not seem to help solving ours.

Another field posing apparently related questions is bioinformatics. However, the main difference here concerns the fact that every instance of our problem contains a single sequence to be covered completely by a set of patterns, while bioinformatics typically searches for common fragments shared by several sequences. The techniques are therefore quite different and do not seem to be directly transferable.

1.2. Previous work

Among works most closely related to the current one, we can mention Ref. [22] which introduced an Expectation–Maximization approach to estimate a Markov model from an unlabeled event log. The approach includes a greedy algorithm that can be used to assign a case id to each event. This greedy algorithm finds a single solution, which is often a local maximum of the likelihood function. Also, the resulting number of patterns depends on the Markov model itself and on the way the greedy algorithm decides where to terminate the occurrence of one pattern and begin another occurrence of the same or a different pattern.

Here, we are interested in traversing the complete search space in order to enumerate all possible solutions with a given number of patterns. We are also concerned with the concept of *minimum description length* (MDL) [23], so we define the optimal solution(s) as the one(s) with a minimal set of patterns. As in the example of Table 1, it is easier to describe the sequence based on 2 patterns rather than 3 or 4 patterns. The principle of MDL has already been used in process mining to evaluate the quality of mined models [24]. Here we use it as a guiding principle while looking for solutions across the entire search space.

Table 1
Some of the solutions for the sequence *aacbabadaccddacd*.

Solutions with 2 patterns	Solutions with 3 patterns	Solutions with 4 patterns
$\langle acd^4, ba^2\rangle$	$\langle ad^4, bc^2, ca^2\rangle$	$\langle ad^2, ba^2, ca^2, dc^2\rangle$
$\langle ab^2, acd^4\rangle$	$\langle ab^2, acd^2, cda^2\rangle$	$\langle ac^2, ba^2, cd^2, da^2\rangle$
	$\langle abdc^2, ad^2, ca^2\rangle$	$\langle ab^2, ac^2, ad^2, cd^2\rangle$
	…	…
(2 solutions)	(44 solutions)	(12 solutions)