Discovering Process Framing for Al-Augmented BPM Systems in a Multi-Process Setting

Anti Alman^{1,*}, Izack Cohen², Avigdor Gal³, Fabrizio Maria Maggi⁴ and Marco Montali⁴

¹University of Tartu, Tartu, Estonia ²Bar-Ilan University, Ramat Gan, Israel ³Technion – Israel Institute of Technology, Haifa, Israel ⁴Free University of Bozen-Bolzano, Bolzano, Italy

Abstract

A core component of AI-Augmented Business Process Management Systems (ABPMS) is *framing* which gives the system process-awareness and defines the boundaries in which the system must operate. When developing an ABPMS, framing can be created manually based on domain knowledge or automatically discovered from prior process executions. Existing process discovery approaches work well for the latter, assuming the individual processes can be identified and analyzed in isolation. However, this is not guaranteed if prior executions were recorded by a system that was not process-aware. In such cases, the bare minimum requirements for process discovery, namely ordered activity sequences grouped by a case identifier, may still be met. For example, treatment activities in a hospital can still be grouped by the patient identifier regardless of how many individual treatment processes (e.g., due to comorbidities) the patient underwent in the same time frame. In this paper, we present a process discovery approach for the above setting. Our focus is on discovering concurrently executed procedural processes, in the presence of shared activities, while the approach itself relies on interpreting declarative process discovery results.

Keywords

process discovery, process framing, concurrent processes, multi-model paradigm, Petri net, Declare

1. Introduction

A recent research manifesto on AI-Augmented Business Process Management Systems (ABPMS) outlines a vision in which traditional BPMS lifecycle is augmented with AI technologies to allow the system to *"reason about the current state of the process (or across several processes) to determine a course of action that improves the performance of the process"* [1]. In that vision, a core component of any ABPMS is *framing*, which provides the system process-awareness and defines the boundaries in which the system must operate to achieve its goals. Framing can be expressed in a variety of business process modeling languages, and even hybrid combinations of procedural and declarative languages are acknowledged as a possible option [1].



PMAI@ECAI24: International ECAI Workshop on Process Management in the AI era, October 19, 2024, Santiago De Compostela, Spain

^{*}Corresponding author.

[🛆] anti.alman@ut.ee (A. Alman); izack.cohen@biu.ac.il (I. Cohen); avigal@technion.ac.il (A. Gal);

maggi@inf.unibz.it (F. M. Maggi); montali@inf.unibz.it (M. Montali)

^{© 0000-0002-5647-6249 (}A. Alman); 0000-0002-6775-3256 (I. Cohen); 0000-0002-7028-661X (A. Gal); 0000-0002-9089-6896 (F. M. Maggi); 0000-0002-8021-3430 (M. Montali)

^{© 0 2024} Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Despite the importance of framing, little indication of how it would be created for an ABPMS is given in [1]. Broadly speaking, we see two options here. The framing could be modeled manually during the analysis and development of the ABPMS, or it could be automatically discovered from the recordings of past executions of the related processes using process mining techniques. There are numerous process discovery algorithms available for this purpose, all of which assume input data in the form of an event log.

In general, event logs are only required to contain sequences of activity executions grouped by case identifiers, where each unique case identifier is assumed to distinguish separate executions of the same underlying process. For example, each treatment instance of a heart attack (regardless of the variance between instances) is required to have a unique case identifier and is assumed to be in an event log containing only instances of other a heart attack treatments. We refer to the latter as the *process uniqueness assumption*, and it can be easily met if process executions have been recorded by a Process-Aware Information System (PAIS) [2]. In absence of process-awareness, event logs could still be created on the basis of more general identifiers such as personal codes of the patients. However, this may result in mixing activities of multiple processes (e.g., heart attack may occur at the same time with a trauma injury [3]), and therefore the process uniqueness assumption would no longer be guaranteed to hold.

In this paper we present a process discovery approach that can handle separately defined processes being seemingly executed with the same case identifier, thus foregoing the process uniqueness assumption. Furthermore, this approach can handle semi-concurrent process executions (e.g., one process being paused to address an immediate concern in another process) and interdependencies between processes (e.g., one process requiring input from another process or a single activity progressing multiple processes simultaneously). The former means that we can no longer rely on directly-follows relations, which are the basis of most current process discovery algorithms, while the latter necessitates handling of synchronizations between otherwise separate processes. In contrast to existing approaches, the proposed approach first discovers a constraint-based declarative representation of the behavior in the given event log, which is then processed to provide a procedural model of the underlying processes and their interactions.

In the context of ABPMS, the proposed approach enables the automated discovery of (at least partial) framing based on past activity executions recorded by non-process-aware systems. While the discovery of the complete framing cannot be guaranteed from the past executions alone (e.g., some currently allowed behavior might not be executed in the past), the proposed approach should nevertheless be a valuable tool for the analysis and development of ABPMS.

In the remainder of this paper, Section 2 outlines our proposed approach and demonstrates its output in a multi-process setting. Section 3 provides a comparison to two existing process discovery approaches. Sections 4 and 5 highlight limitations and related works respectively. Finally, Section 6 concludes the paper by outlining directions for future work.

2. Approach

Our approach relies on two modeling languages, namely, DECLARE [4] and Petri nets [5]. The former is used for discovering and working with specific types of eventually-follows relations, while the latter is used to represent the final discovered model. Next, we describe the eventually-

follows relations used in our approach (Section 2.1), the patterns for combining sets of these relations into Petri net fragments (Section 2.2), and the procedure for creating a complete Petri net from these fragments (Section 2.3). Finally, we conclude this section by demonstrating the output of our approach in a multi-process setting (Section 2.4).

2.1. Discovering Eventually-Follows Relations

As the first step, we use Declare Miner [6] to discover a DECLARE model of the following relations between the activities in the given event log (with A and B serving as activity placeholders):

- Succession(A, B) A occurs if and only if it is eventually followed by B;
- Response(A, B) If A occurs then it will be eventually followed by B;
- Precedence(B, A) If A occurs then B must have occurred some time before;
- Not Co-Existence(A, B) If A occurs then B will never occur (and vice-versa);
- Co-Existence(A, B) If A occurs then B will eventually also occur (and vice-versa);
- Not Succession(A, B) If A occurs then B can no longer occur.

Intuitively, Succession identifies potentially optional activities that always occur in the same order, Response identifies mandatory followers of potentially optional activities, and Precedence identifies prerequisite activities of potentially optional activities. Additionally, Not Co-Existence identifies mutually exclusive activities, Co-Existence without (Not) Succession identifies parallel activities, and Not Succession together with Co-Existence identifies any fixed activity orders.

All aforementioned relations are discovered between all pairs of activities assuming there is at least one positive example and no counter examples for the given relation with the given activity pair. Additionally, to simplify processing, an artificial start and end activity is added to each execution trace, ensuring the discovery of at least one Response and one Precedence (or Succession in case of a mandatory activity) per each activity. Finally, the discovered relations are pruned [7] to remove relations which are implied by other relations (e.g., if both Succession and Response hold for the same pair of activities, then the Response relation is removed).

2.2. Patterns

Our approach uses four pairs of patterns, each of which corresponds to the manifestation of some procedural behavior (e.g., sequence flow, choice, etc.) in the DECLARE modeling language. All pairs are symmetric (similarly to Alpha Miner [8]), such that one pattern of the pair handles procedural split gateways and the other handles the corresponding join gateways. Next, we describe the pattern pairs as they would apply to the relations of a single activity, while more complex processing is discussed in Section 2.3. The relations for each pattern are given on the left side of its figure, while the matching Petri net fragment is given on the right (e.g., Fig. 1(a)).

Simple sequence flow and parallelism. Patterns 1a and 1b (Fig. 1) are used for simple sequence flows and parallelism. For example, Succession(A, B) is interpreted as a Petri net fragment where the transition A has one output place that provides a token for the transition B. Multiple Succession relations originating from the same activity are interpreted as the start of



Figure 1: Patterns handling simple sequence flows and parallelism based on Succession relations.



Figure 2: Patterns for handling exclusive choices based on Precedence, Response, and Not Co-Existence relations.

parallelism (i.e., AND-split), while multiple Succession relations leading to the same activity are interpreted as the end of parallelism (i.e., AND-join).

Exclusive choice. Patterns 2a and 2b (Fig. 2) are used for mutually exclusive choices. The beginning of a mutually exclusive choice (i.e., XOR-split) is matched by pattern 2a, where the relations intuitively mean that B, C, ... can be executed only if A has occurred some time before (due to the Precedence relations), and at most one of B, C, ... can be chosen (due to Not Co-Existence relations). Meanwhile, pattern 2b matches the end of a mutually exclusive choice (i.e., XOR-join), with the intuition being that B, C, ... must be eventually followed by D (due to the Response relations). By default, both patterns add a silent transition into to the corresponding Petri net fragments to allow all of the mutually exclusive activities to be skipped. However, if the frequency of the mutually exclusive activities is equivalent to the frequency of A in pattern 2b, then the silent transition is omitted.

Optional parallelism. Patterns 3a and 3b (Fig. 3) match a special case of parallelism where the entire parallel structure may be skipped. The intuitive meaning of Precedence and Response relations is the same as in patterns 2a and 2b, however, the Co-Existence relation (instead of Not Co-Existence) means that if any of B, C, ... is executed, then all of them must be executed. We note that, activity frequencies do not need to be considered here, since equivalent frequencies, would lead to Succession relations instead of Precedence and Response relations.

Inclusive choice. Patterns 4a and 4b (Fig. 4) are used, effectively as fallback patterns, for cases where a single activity has multiple outgoing Precedence or incoming Response relations, but patterns 2a, 2b, 3a, and 3b do not apply. Intuitively, pattern 4a corresponds to the beginning of an inclusive choice (OR-split), while pattern 4b corresponds to the end of an inclusive choice (OR-split). In both patterns, the activities B, C, ... are placed in parallel together with an option to skip each individually, thus also allowing cases where none of B, C, ... are executed.



Figure 3: Patterns for handling optional parallelism based on Precedence, Response, and Co-Existence relations.



Figure 4: Patterns for handling inclusive choices and optional activities based on Precedence and Response relations.



Figure 5: Source Petri net and the corresponding discovered Declare model with pruning applied (Not Succession and Co-Existence relations omitted for understandability).

2.3. Creating the Procedural Process Model

The procedural behavior of a given event log can be reconstructed, given the eventually-follows relations from Section 2.1 and the patterns from Section 2.2. The basic idea is to begin with a Petri net containing only the artificial start (cf. Section 2.1) and then applying the patterns to it in a specific order. In doing so, additional transitions are added to the net, each of which will be processed in the same way, until one transition is created (and processed) per each activity. This results in a single Petri net representing the behavior of the underlying processes and the interactions between them. The corresponding procedure is summarized in Algorithm 1.

As an example, we use an event log containing the executions of Fig. 5(a) (with no violations and at least one occurrence of each activity). From this log, the Declare Miner will discover the relations presented in Fig. 5(b). To construct a corresponding procedural process model we start

with an empty Petri net, into which we add a transition start with one input place. This leads to processing the relations of the activity start, which has only the relation Succession(start, A). Therefore, we can directly apply pattern 1a to add a transition A into the Petri net, together with a place that gets a token from transition start and provides it to transition A.

Next, we move to processing activity A. For this activity, we have multiple outgoing relations:

- Succession(A, D) matches pattern 1a directly and would lead to correct placement of D in the Petri net;
- Relations between A, C1, and C2 are a direct match to pattern 2a, but would lead an incorrect placement of the corresponding XOR-split directly after A.
- Relations between A, B1, and B2 are a direct match to pattern 2a, but B2 is also in a Not Co-Existence relation with B3, which also matches pattern 2a;

We employ the following three additional rules to handle this situation:

Rule 1. If the outgoing relations of an activity match multiple patterns, then this is solved through parallelism. In the given example, A would get one output place providing a token for D (as per pattern 1a), and one additional output place per each application of pattern 4a. Furthermore, the patterns are considered in the order of 1a, 1b, 2a, 3a, 4a, 2b, 3b, 4b. The main purpose of this ordering is to apply more specific patterns first (for example, the Co-Existence relations in pattern 3a make it a more specific version of pattern 4a);

Rule 2. If there is any fixed order between outgoing or incoming activities, then only the relations to temporally "closest" activities are considered. In the given example, if any of B1, B2 or B3 occurs, then it must be before either C1 or C2 occurs. This is determined based the existence of Not Succession and Not Co-Existence between the pairs of activities. For example, the Declare Miner would find a relation Not Succession(C1, B1) (omitted from Fig. 5(b)) without finding relation Not Co-Existence(C1, B1). This indicates that B1 and C1 may occur in the same trace, and, if they do, then C1 must occur after B1. As a result, we can ignore outgoing relations to C1, and also to C2, while processing A, as both will transitively appear at some point after B1. An analogous comparison is later also required for the incoming activities of E;

Rule 3. If pattern 2a (or 2b) can be applied to sets of partially overlapping activities, then pattern 2a (or 2b) will be applied once per each of these sets. This rule is necessary because of the non-transitive nature of the Not Co-Existence relation, meaning that Not Co-Existence must be observed between all pairs of activities to be placed after XOR-split (or before XOR-join). Given Fig. 5(b), the pattern 2a could be applied to { A, B1, B2 } and { A, B2, B3 }, with B2 being the overlapping activity. In this case, pattern 2a must be applied to both sets of activities, such that A produces a token into a place providing it to B1 and B2 (first application of pattern 2a), and A also produces a token into a place providing it to B2 and B3 (second application of pattern 2a). In more complex cases, this can be solved by finding all maximal Not Co-Existence cliques among the activities (e.g., by using [9]) and applying pattern 2a (or 2b) once per each clique.

This leads to transitions B1, B2, B3, and D being added to the Petri net, all of which will be processed one by one based on the above rules and the corresponding patterns.

Finally, the Petri net (\mathcal{PN}) returned by Algorithm 1 is post-processed to handle optional behavior. More specifically, all patterns, except for 1a and 1b, can produce silent transitions, which either do not have any output places (indicating the start of some optional behavior, i.e.,

skip start) or do not have any input places (indicating the end of some optional behavior, i.e., *skip end*). For each skip start, we first find all outgoing activities from places that may provide a token for that skip start. For each of these activities, we then find the outgoing Precedence relations. This indicates all the activities that must also be skipped if the given skip start transition is fired. Then, we do the same for each skip end while finding the incoming activities from all places into which the skip end produces a token. For each of these activities, we find the incoming Response relations, which gives a set of activities that must also be skipped if the given skip end transition is fired. Then, each skip start is merged with the skip end for which the sets of skipped activities are equivalent.

	Algorithm 1 Petri net construction from activity relations.
	input :relations \mathcal{R} // as discovered in Section 2.1 output:Petri net \mathcal{PN}
1	$\Theta := (1a, 1b, 2a, 3a, 4a, 2b, 3b, 4b) //$ see Section 2.2 and Rule 1 in Section 2.3
2	$\Sigma^U := \{\text{start}\} / /$ initialize Σ^U with the artificial activity 'start'
3	$\mathcal{PN} := \{p_0 \land p_0 \bullet = start\} // initialize \ \mathcal{PN} with place p_0 and transition start$
4	while $\Sigma^U eq \emptyset$ do
5	$\alpha := \alpha \in_R \Sigma^U$ // choose random activity from Σ^U
6	$\mathcal{R}^U := \{\mathcal{R}_i L(\mathcal{R}_i) \ni \alpha\}$ // consider only the relations containing α
7	$\mathcal{R}^U \leftarrow \{\mathcal{R}^U_i \mathcal{R}^U_i \text{ temporally closest to } \alpha\} // \text{ see rule 2 in Section 2.3}$
8	for each $ heta \in \Theta$ do
9	$\mathcal{PN}^U := \theta(lpha, \mathcal{R}^U) / /$ instantiate pattern θ using α and \mathcal{R}^U
10	$\Sigma^U \leftarrow \Sigma^U \cup \{L(\mathcal{PN}^U) \setminus L(\mathcal{PN})\} // \text{ add activities not yet in } \mathcal{PN} \text{ to } \Sigma^U$
11	$\mathcal{PN} \leftarrow \mathcal{PN} \cup \mathcal{PN}^U$ // extend \mathcal{PN} (merging same-labeled transitions)
12	$\mathcal{R}^U \leftarrow \{\mathcal{R}^U_i \mathcal{R}^U_i \text{ not captured in } \mathcal{PN}^U\} / / \text{ remove relations added to } \mathcal{PN}$
13	end
14	$\Sigma^U \leftarrow \Sigma^U \cup \{L(\mathcal{R}^U) \setminus L(\mathcal{PN})\} // \text{ if any remaining relations refer to activities not in } L(\mathcal{PN})$
	then add these activities to Σ^U
15	$\Sigma^U \leftarrow \Sigma^U \setminus \alpha //$ finished processing α (α now in $L(\mathcal{PN})$)
16	end
17	$\mathcal{PN} \leftarrow \{\mathcal{PN} \cup p_f \bullet p_f = end\} / / add final place$
18	return \mathcal{PN} // Petri net of activity relations

Given the relations in Fig. 5(b), Algorithm 1 together with the rules outlined in in this section will precisely reconstruct the Petri net shown in Fig. 5(a).

2.4. Discovery Results in Multi-Process Setting

To demonstrate the result of Algorithm 1 in multi-process setting, we use a prototype implementation of our approach (publicly available at: https://github.com/antialman/multi-model-miner/).

For input we use [10] to generate an event log of the concurrent execution of Figs. 5(a), 6(a) and 6(b). For these models, we note that C2 is present in both Fig. 5(a) and Fig. 6(a), thus making it a shared activity between the processes. As a result, C2 can only be executed if both C1 and C3 are not executed. Furthermore, Fig. 6(b) refines this interplay, by specifying that if C3 (in Fig. 6(a)) is executed then D (in Fig. 5(a)) must be executed afterwards. The process model discovered by our approach form the generated event log is shown in Fig. 7.

The behavior of the discovered model is correct, and, furthermore, the individual input processes (Figs. 5(a) and 6(a)) are fairly well recognizable in the resulting model (Fig. 7), especially when ignoring the artificial start and end events (highlighted by number 1). In that case, the



Figure 6: Additional process specifications to demonstrate discovery of hybrid model-sets.



Figure 7: Process model of the concurrent execution of Figs. 5(a), 6(a) and 6(b). Numbers 1–3 highlight regions that are further discussed in text.

two processes would only be connected by the transition C2 (highlighted by number 2), and the place 30 (highlighted by number 3). The transition C2 is correctly placed to represent that the corresponding activity is shared between the input processes. Meanwhile, place 30 represents the additional interplay refinement Response(C3, D) in Fig. 6(b).

With regards to the latter, it must be noted that there is a silent transition that is connected to place 30 and does not have any input places. This silent transition is caused by Response(C3, D) enforcing behavior which matches pattern 4b, without there being any corresponding match for pattern 4a (i.e., pattern 4b created a silent transition to end optional behavior, but no start for that optional behavior was created). As a result an unbounded number of tokens can be produced into place 30, which, while undesirable, does not enable any unintended behavior. Nevertheless, such cases of declarative behavior are something we plan to handle in the future.

3. Comparison to Contemporary Approaches

In this section we demonstrate that contemporary process discovery approaches are likely to give incorrect results in our setting, when considering shared activities and other relations between processes. More specifically, we reuse the event log generated in Section 2.4 for discovering process models with Flexible Heuristics Miner [11] and Split Miner [12].

The model discovered by Flexible Heuristics Miner (Fig. 8) is relatively close in terms of the its behavior, but not in terms of structure. The overlapping XOR between B1, B2 and B3, while difficult to see, is captured correctly. The relation between activities C1, C2, and C3 is also identified; however, this results in the inability to execute, for example, both C3 and B1 without deadlocking the model. Furthermore, the interplay relation between C3 and D is not captured.

The Split Miner (Fig. 9) achieves better results, coming fairly close to the structure of the initial models. The overlapping XOR between B1, B2 and B3 is captured correctly and easy to see. Also, C3 does lead into D, thus accurately reflecting the corresponding interplay relation.



Figure 8: Flexible Heuristics Miner with default settings.



Figure 9: Split Miner with maximum sensitivity to parallelism and all nodes and arcs displayed.

However, this model has two main drawbacks. First, it is difficult to interpret the gateways and thus the behavior of the model. For example, the AND-splits after A and C3 seemingly lead to the same XOR-join. Secondly, the initial input models, while relatively accurately represented, are also relatively strongly intertwined, making it difficult to disentangle the individual processes.

We also note that, while not presented in this paper, both Flexible Heuristics Miner and Split Miner performed well when the individual processes did not have shared activities and there were no interplay refinements (such as Fig. 6(b)) between the processes. In these cases, the individual processes were discovered as parallel branches of a single larger process model.

4. Current Limitations

The approach presented in this paper has some limitations we plan to tackle as future work. The most important among these are handling of repetitions and improving tolerance to noise.

Repetitions (i.e., self-loops, short-loops, long-loops) were not considered in this paper, however, we note that, almost all patterns presented in Section 2.2 be used within larger outer-loops with little to no adjustments. Furthermore, given that a loop can be viewed as a choice between executing an iteration or continuing beyond the loop, patterns 2a and 2b will match the entry and exit points of loops, respectively.

Our approach, as presented in this paper, is likely to be sensitive to noise in the event log. For example, a single counter example to Succession(A, B) would lead to discovering Co-Existence(A, B) instead. Here, we could consider adopting the causality measure of Heuristics Miner [13], but, it would likely require some modifications as it is designed for directly-follows relations.

Finally, the approach could be simplified by using target-branched DECLARE discovery [14], which can discover Response and Precedence relations on disjunctions of activities. For example, in Fig. 5 this would lead to discovering relations such as Response(A, (B1 or B2)), thus making the discovery of Not Co-Existence(B1, B2) unneeded and simplifying patterns 3a and 3b.

5. Related Work

We are not aware of any approaches that explicitly address business process discovery in multi-process setting nor that forego the process uniqueness assumption.

However, a similar setting can be found in the context of *local processes* [15] where the goal is to discover common (and possibly concurrent) sub-behaviors within one process. Conceptually, this could be reformulated such that the sub-behaviors to be discovered would instead be seen as parallel processes, for which the process identifier is missing. However, [15] is not developed with our setting in mind and, as shown in [16], produces very small models (often 2-3 activities).

In relation to local processes, we also highlight region-based discovery techniques, which use region theory for discovering characteristic regions that match the behavior seen in the log. These regions, formulated as Petri nets, represent partial views of the behavior captured in the log that are integrated to a complete process model [17, 18]. While our approach shares some mutual ideas with region-based discovery, we rely on DECLARE discovery and hybrid semantics providing us the flexibility of representing any relation between (or within) the processes declaratively, should the relation be difficult to incorporate into a Petri net.

There are also some parallels with multi-agent systems (MAS), where larger systems are modeled through decentralized decision units referred to as agents. While the focus of MAS is more on the behavioral rules of the agents rather than the related business processes, there is some research on integrating MAS with process discovery techniques, e.g., the work in [19].

The combination of Petri nets and DECLARE is also used in some other hybrid process discovery approaches, such as [20, 21], however none of them are designed to tackle our setting. For a broader overview of various hybrid approaches we refer to [22].

It is feasible that the approach presented in this paper, and the discovery of ABPMS framing in general, could be further enhanced by leveraging additional domain knowledge. This idea is underexplored thus far, but ontological domain knowledge has been leveraged in process discovery to, for example, guide event abstraction [23].

Finally, we highlight that a core part of our approach is (partially) transforming (patterns of) DECLARE constraints into Petri nets. A more general version of this transformation has already been explored in the literature, with [24] providing a full lexicon of constructs for DECLARE constraints. However, due to the generality of that work, it leads to significantly more complex Petri net structures, requiring, for example, weighted, reset and inhibitor arcs. We have instead side-stepped this complexity by considering only a relatively small number of specific patterns consisting of only a limited set of DECLARE constraint types.

6. Conclusion

In this paper, we focused on the discovery of ABPMS framing in a multi-process setting. More specifically, we proposed a process discovery approach that can handle interleaved executions of multiple processes based on some shared identifier (e.g., patient id). This enables automated discovery of (at least partial) ABPMS framing based on the activity executions recorded in non-process-aware systems. The proposed approach makes it possible to discover the individual processes as well as the relations between them in a single model. While the approach currently

has some limitations related to process repetitions and noise tolerance, we nevertheless believe it is a solid foundation for future works in the same direction, especially given that, in this paper, we rely only on the temporal orderings of activities in the given event log.

The next major extension of the work presented in this paper will be incorporating the core ideas of the recently proposed multi-model paradigm for BPM [25] and the related semantics [10]. In essence, this means the discovery of multiple separate process models (corresponding to the multi-process setting) from a single event log, together with models defining their interactions. This may be achieved by identifying and adopting suitable graph decompositioning and clustering techniques to further analyze the current output Petri net. For example, the result presented in this paper (Fig. 7) could be relatively easily decomposed by identifying a minimal set of nodes, which, when removed, would make the Petri net graph disjoint. However, further research in that direction is still required.

In addition, we plan to explore the inclusion of domain knowledge and data perspective in our approach. This could be used to further guide the process discovery towards better separating the individual processes. For instance, knowing ontological relations between activities and which resources execute them may provide hints about the nature of the underlying processes. Finally, our approach would benefit from an empirical evaluation based on real-life datasets.

Acknowledgments

A. Alman was supported by the Estonian Research Council grant PRG1226. F.M. Maggi was supported by the UNIBZ project PRISMA.

References

- [1] M. Dumas, F. Fournier, L. Limonad, A. Marrella, M. Montali, J. Rehse, R. Accorsi, D. Calvanese, G. De Giacomo, D. Fahland, A. Gal, M. La Rosa, H. Völzer, I. Weber, AI-augmented business process management systems: A research manifesto, ACM Trans. Manag. Inf. Syst. 14 (2023) 11:1–11:19.
- [2] W. M. P. van der Aalst, Process-aware information systems: Lessons to be learned from process mining, Trans. Petri Nets Other Model. Concurr. 2 (2009) 1–26.
- [3] F. B. Irfan, R. I. G. D. J. Consunji, R. Peralta, A. El-Menyar, L. B. Dsouza, J. M. Al-Suwaidi, R. Singh, M. Castrén, T. Djärv, G. Alinier, Comparison of in-hospital and out-of-hospital cardiac arrest of trauma patients in qatar, Int. J. Emerg. Med. 15 (2022) 1–8.
- [4] M. Pesic, H. Schonenberg, W. M. P. van der Aalst, DECLARE: full support for looselystructured processes, in: EDOC, IEEE Computer Society, 2007, pp. 287–300.
- [5] M. de Leoni, P. Felli, M. Montali, A holistic approach for soundness verification of decisionaware process models, in: ER, volume 11157 of *LNCS*, Springer, 2018, pp. 219–235.
- [6] F. M. Maggi, C. Di Ciccio, C. Di Francescomarino, T. Kala, Parallel algorithms for the automated discovery of declarative process models, Inf. Syst. 74 (2018) 136–152.
- [7] F. M. Maggi, R. P. J. C. Bose, W. M. P. van der Aalst, A knowledge-based integrated approach for discovering and repairing declare maps, in: CAiSE, volume 7908 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 433–448.

- [8] W. M. P. van der Aalst, T. Weijters, L. Maruster, Workflow mining: Discovering process models from event logs, IEEE Trans. Knowl. Data Eng. 16 (2004) 1128–1142.
- [9] C. Bron, J. Kerbosch, Finding all cliques of an undirected graph (algorithm 457), Commun. ACM 16 (1973) 575–576.
- [10] A. Alman, F. M. Maggi, M. Montali, A. Rivkin, Generating event logs from hybrid process models, in: Business Process Management Workshops, volume 492 of *Lecture Notes in Business Information Processing*, Springer, 2023, pp. 289–301.
- [11] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. P. van der Aalst, Data-driven process discovery - revealing conditional infrequent behavior from event logs, in: CAiSE, volume 10253 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 545–560.
- [12] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, A. Polyvyanyy, Split miner: automated discovery of accurate and simple business process models from event logs, Knowl. Inf. Syst. 59 (2019) 251–284.
- [13] A. J. M. M. Weijters, W. M. P. van der Aalst, Rediscovering workflow models from eventbased data using little thumb, Integr. Comput. Aided Eng. 10 (2003) 151–162.
- [14] C. Di Ciccio, F. M. Maggi, J. Mendling, Discovering target-branched declare constraints, in: BPM, volume 8659 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 34–50.
- [15] N. Tax, N. Sidorova, R. Haakma, W. M. P. van der Aalst, Mining local process models, J. Innov. Digit. Ecosyst. 3 (2016) 183–196.
- [16] M. Brunings, D. Fahland, B. F. van Dongen, Defining meaningful local process models, Trans. Petri Nets Other Model. Concurr. 16 (2022) 24–48.
- [17] J. Carmona, Projection approaches to process mining using region-based techniques, Data Mining and Knowledge Discovery 24 (2012) 218–246.
- [18] M. Solé, J. Carmona, Region-based foldings in process discovery, IEEE Transactions on Knowledge and Data Engineering 25 (2011) 192–205.
- [19] R. H. Bemthuis, M. Koot, M. R. K. Mes, F. A. Bukhsh, M. Iacob, N. Meratnia, An agent-based process mining architecture for emergent behavior analysis, in: EDOC Workshops, IEEE, 2019, pp. 54–64.
- [20] J. De Smedt, J. De Weerdt, J. Vanthienen, Fusion miner: Process discovery for mixedparadigm models, Decis. Support Syst. 77 (2015) 123–136.
- [21] F. M. Maggi, T. Slaats, H. A. Reijers, The automated discovery of hybrid processes, in: BPM, volume 8659 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 392–399.
- [22] A. A. Andaloussi, A. Burattin, T. Slaats, E. Kindler, B. Weber, On the declarative paradigm in hybrid business process representations: A conceptual framework and a systematic literature study, Inf. Syst. 91 (2020) 101505.
- [23] S. Montani, G. Leonardi, M. Striani, S. Quaglini, A. Cavallini, Multi-level abstraction for trace comparison and process discovery, Expert Syst. Appl. 81 (2017) 398–409.
- [24] J. De Smedt, S. vanden Broucke, J. Weerdt, J. Vanthienen, A Full R/I-net Construct Lexicon for Declare Constraints, Technical Report, KU Leuven, 2015.
- [25] A. Alman, F. M. Maggi, S. Rinderle-Ma, A. Rivkin, K. Winter, Towards a multi-model paradigm for business process management, in: CAiSE, volume 14663 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 178–194.