

Merging Event Logs with Many to Many Relationships

Lihi Raichelson and Pnina Soffer

Department of Information Systems, University of Haifa, Haifa 31905, Israel
LihiRaOs@gmail.com, Spnina@is.haifa.ac.il

Abstract. Process mining techniques enable the discovery and analysis of business processes, identifying opportunities for improvement. However, processes are often comprised of separately managed procedures that have separate log files, impossible to mine in an integrative manner. A preprocessing step that merges log files is quite straightforward when the logs have common case IDs. However, when cases in the different logs have many-to-many relationships among them this is more challenging. In this paper we present an approach for merging event logs which is capable of dealing with all kinds of relationships between logs, one-to-one or many-to-many. The approach matches cases in the logs, using temporal relations and text mining techniques. We have implemented the algorithm and tested it on a comprehensive set of synthetic logs.

Keywords: Process Mining, Multiple Instances, Merging Log Files, End-To-End Process.

1 Introduction

Process mining techniques are used for discovery and analysis of the actual business processes from the event logs of the systems that support and manage them [1][2]. However, process mining usually considers a single event log, while different process procedures often use different systems and thus have separate logs. To provide a full analysis, process mining should be applied to a log containing all relevant activities of the end-to-end process flow. Such log can only be obtained after identifying and merging related log files from different distributed systems corresponding to the same process.

Existing methods that compose logs [12] either assume identical case ID or the existence of one log's case ID as an attribute in the other log, and simple relationships between procedures (one-to-one or one-to-many). Often, real-life processes include procedures that stand for multiple instances and hence may have complex relations among them (e.g. many-to-one or many-to-many). In such situations, each procedure employs a different case ID, not necessarily directly corresponding to cases in the other procedures. Merging the logs of these procedures becomes a challenging task, which is still mandatory for an extended analysis of the entire process.

In this paper we present an automatic technique for merging event logs, which does not assume any specific relationship between the procedures and is hence applicable

for one-to-one as well as for many-to-many relationships, where no unique identifier correlates the cases in the logs. To illustrate the problem, consider an example process, where different organizational units can directly place orders for office supplies through an ordering system. Consolidated deliveries are received at a warehouse, where they are registered and distributed to the ordering units. We consider the ordering process as the main process, and the delivery handling as the sub-process. Table 1 shows a simplified log of the main process, while a simplified log of the warehouse procedure is given in Table 2.

Table 1. Simplified log of the main process of ordering goods

Order	Timestamp	User	Activity	Item no	Department
3001	02/02/14 10:12	Ilana	open order	1234 1235	dep. 89
3001	02/02/14 10:13	Tsvi	approve order	1234 1235	dep. 89
3001	02/02/14 13:16	Ilana	check status	1234 1235	dep. 89
3001	03/02/14 16:18	Ilana	receive item	1234	dep. 89
3001	04/02/14 16:35	Ilana	receive item	1235	dep. 89
3001	04/02/14 16:36	Ilana	close order	1234 1235	dep. 89
3002	02/02/14 10:30	Sigal	open order	1234 1236	dep. 79
3002	02/02/14 10:31	Rachel	approve order	1234 1236	dep. 79
3002	02/02/14 15:31	Sigal	check status	1234 1236	dep. 79
3002	03/02/14 16:19	Sigal	receive item	1234	dep. 79
3002	04/02/14 16:35	Sigal	receive item	1236	dep. 79
3002	04/02/14 16:36	Sigal	close order	1234 1236	dep. 79

Table 2. Simplified log of the sub process of delivery handling

Delivery	Timestamp	User	Activity	Item no	Department
5001	03/02/14 11:45	Mosh	receive item	1234	
5001	03/02/14 16:15	Mosh	send to dep.	1234	dep. 89
5001	03/02/14 16:16	Mosh	send to dep.	1234	dep. 79
5002	04/02/14 12:46	Mosh	receive item	1235 1236	
5002	04/02/14 16:30	Mosh	send to dep.	1235	dep. 89
5002	04/02/14 16:31	Mosh	send to dep.	1236	dep. 79

In the example, both processes employ multiple instance procedures. Furthermore, the lower-level instances in both processes refer to ordered items. However, the grouping of items to cases is different for the main and sub-process. In the main process, the grouping is by order (serving as case ID), and in the sub process the grouping is by delivery (case ID), where items ordered by different departments are supplied together. As a result, multiple cases of the main process may correspond to multiple cases of the sub process. Merging the logs would enable mining the end-to-end process, and particularly tracing the low-level instances (ordered items). The unified log would employ a single case ID for all the events that relate to the same order. In the absence of a common case ID, the main challenge is to identify the events in both logs that should be considered related to the same case.

The rest of the paper is structured as follows. Section 2 presents the approach and the log merging algorithm, demonstrating it using the running example; Section 3 reports the evaluation performed to a set of synthetic logs; Section 4 discusses related work. Finally, conclusions are given in Section 5.

2 Merging event logs

For the above described challenge we suggest an approach of an automatic merge of log files by finding a match between each case in the main process and corresponding cases in the sub process. To handle many-to-many relationships between the cases of the different logs, we duplicate cases of the sub process and merge them with every relevant case in the main process, revealing the end-to-end process flow.

The overall idea is shown in Fig. 1. An illustration of a main log includes cases which may have multiple instances is presented in Fig. 1(a). Those instances are grouped together in the sub process, so instances from case 1 in the main process are included in case 1 and case 2 of the sub process. Our approach, illustrated in Fig 1(b), generate new case IDs for the unified log file, where each case of the main process corresponds to a unique new case id, and cases of the sub process can be duplicated (e.g. case 1 of the sub process) in order to reflect the relationships to the cases of the main process.

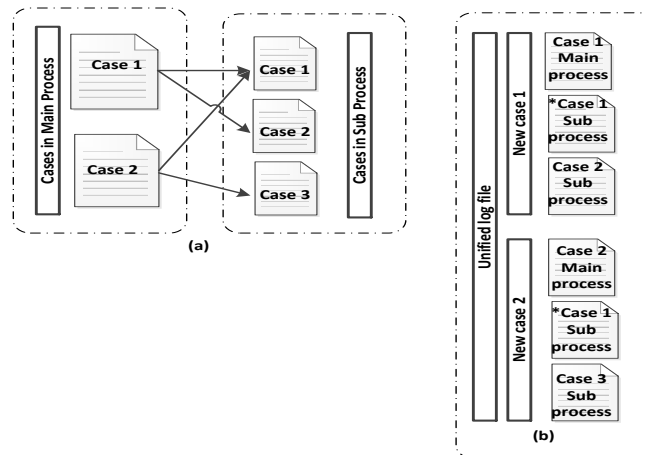


Fig. 1. An illustration of merging cases from different event logs

2.1 Approach overview

As a preliminary step, the boundaries of the end-to-end process need to be determined and all related procedures identified. As a consequence, a set of logs from various systems can be identified and preprocessed to a uniform format [14]. In this paper we limit ourselves to merging two logs, one related to a main process and the other to a sub-process. However, this can be performed repeatedly to a hierarchy of sub-processes. Note that the identification of a main and sub process might not always be clear. In what follows, we indicate clear temporal relationships that should hold between the process considered as “main” and the one considered as “sub” for the sake of the merging.

As explained, we seek matching cases in the two logs, taking the following assumptions. (1) Both logs are taken from synchronized systems and consist of reliable and comparable timestamps. (2) Both logs might include multiple instances, thus four types of relationships between the logs are possible: one-to-one, one-to-many, many-to-one, and many-to-many. (3) The attribute values in the logs use uniform terms. Note that this assumption can be relaxed by using synonym detecting tools (e.g. Wordnet), but this is not included in our current scope. (4) The attribute values may include free text. The approach should be capable of dealing with logs where the reference to the other log is unstructured and given as free text. (5) The log of the sub process might include cases initiated by other processes. Hence, it is acceptable that not all its cases would be matched to cases of the main process and appears in the unified log file. In other words, the unified log should not necessarily contain all the cases of the sub process log.

Finding a match between log cases is trivial when the logs have a one-to-one relationship and both have the same case ID. In this case, matching is immediate and the logs can be merged directly. For logs whose case ID is not identical, cases are matched based on (i) similarity of *attribute values*, and (ii) appropriate *temporal relations*.

Similarity of attribute values can easily be established if we know in advance which attributes should hold similar values in the two logs. This is often the case, and can rely on domain knowledge. However, according to our assumptions, the logs might contain free-text attributes, and these might contain the information which is relevant for the match.

To accommodate for such situations, we assume that matching cases would have more common words in their attribute values than non-matching cases. For example, consider possible free text in the main log “we wish to order two monitors of models 1234 and 1235”, and free text in the sub-process log “ordered by department 89”. While generally similar text might appear in many entries across the logs, the specific item ids and department ids - representing the actual case properties - will be common only to the matching cases; hence their common words count is expected to be higher than those of non-matching cases. We hence calculate a similarity score based on the number of common words in the total text associated with the cases. However, some words can be common to all cases (e.g., stop words like "the", or activity names). These should not affect the similarity score. To this end, we use a text mining technique, the Term Frequency-Inverse Document Frequency (tf-idf) [18]. tf-idf relates to the frequency of a word in a given text (case attributes) as compared to its appearance in other texts (other cases). It allows filtering out words that are common to all cases and would not be a good indicator of similarity of two specific cases. The remaining words are extracted to a "bag of words" for each case, and similarity score of two cases is calculated as the count of unique words that are common to them.

In our running example the bag of unique words of the 1st case in the main log (Table 1) would be {3001, Ilana, 1234, 1235, 89} and the bag of unique words of the 1st case in the sub log (Table 2) would be {5001, Mosh, 1234, 89, 79}. The similarity score would hence be 2.

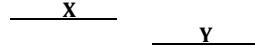
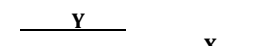
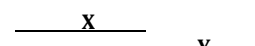
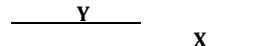
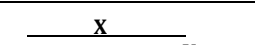
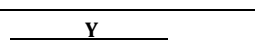
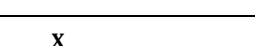
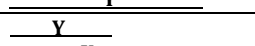
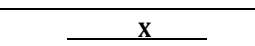
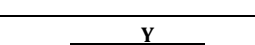
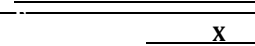
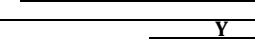
Note that it is common for two logs to include similar but not identical terminology (e.g., concatenation of several attribute values from the other log). This kind of

relations can usually be indicated a-priory based on domain knowledge, so the log can be pre-processed accordingly.

Appropriate temporal relations – here we make two requirements. First, since the sub process is triggered by the main process, it should start after the beginning of the main process. Second, we require the sub process to provide some feedback to the main process, and hence it should have some time overlap with the main process and cannot start after the main process has ended. Note that in general it is possible that a sub process will appear in the log as starting after the main process has ended (e.g., recorded manually, mistakenly closed cases). Those cases are exceptions and not considered as correct matches in our automatic solution.

To formalize these requirements in terms of temporal relations we rely on Allen's interval algebra [4]. Table 3 specifies for each of Allen's temporal relation types whether it meets the requirements and can be considered a match.

Table 3. The 13 temporal relations with respect to case matching

Case in Main Process Vs. Case in Sub Process	Relation: X=Main; Y=Sub	Match (Y/N)	Illustration	Comments
Main Process takes place <i>before</i> Sub Process	$X < Y$	NO match		missing feedback from sub-process (Y) to main process (X)
Sub Process takes place <i>before</i> Main Process	$Y > X$	NO match		Main process starts after sub process; No triggering
Main Process <i>meets</i> Sub Process	$X m Y$	NO match		Missing feedback
Sub Process <i>meets</i> Main Process	$Y m X$	NO match		Main process starts after sub process; No triggering
Main Process <i>overlaps</i> with Sub process	$X o Y$	Positive Match		Meets requirements
Sub Process <i>overlaps</i> with Main process	$Y o X$	NO match		Main process starts after sub process; No triggering
Main process <i>starts with</i> Sub process	$X s Y$	NO match		Start at the same time, no triggering
Sub process <i>starts with</i> Main process	$Y s X$	NO match		Start at the same time, no triggering
Main process start-end <i>during</i> Sub process	$X d Y$	NO match		Main process starts after sub process; No triggering
Sub process start-end <i>during</i> Main process	$Y d X$	Positive Match		Meets requirements
Main process start during and <i>finishes with</i> Sub process	$X f Y$	NO match		Main process starts after sub process; No triggering
Sub process start during and <i>finishes with</i>	$Y f X$	Positive Match		Meets requirements

with main process				
Main process equal to Sub process	$X = Y$	NO match	$\frac{X}{Y}$	Start at the same time no triggering

2.2 Algorithm

Following the above discussion the algorithm takes two logs, one of the main process and the other of the sub process, and generates a unified log file. The algorithm, depicted in Listing 1, addresses only situations where the main process and the sub process have different case IDs (n cases in main log, m cases in sub log). The algorithm uses the following main variables and functions:

**Word_Set (case)*: a set holding all the values of attributes in all events of a case (bag of unique words).

**Function *Match_time (case1, case2)*: checks the temporal relation between case1 and case2 and returns TRUE if they meet temporal requirements

***Function *check_Match_Score [word_set(case 1), word_set(case 2)]*: returns an integer value *Match_Score*($MS_{n,m}$) – the calculated similarity score of attribute values between the cases.

The algorithm generates a new case ID for each case of the main process, and calculates match scores for every case combination that meets the temporal requirements. For every case of the main process (and corresponding New_Case ID) it selects the sub-process cases whose match score is maximal (and above zero), possibly creating duplications of sub-process cases. Finally, it merges the logs accordingly.

Listing 1.

Algorithm: Check match score between cases in cross logs for potential merging

```

1:   Set *word_set {main log(case_id)} == extract all unique values for every case id in main log
2:   Set *word_set {sub log(case_id)} == extract all unique values for every case id in sub log
3:
4:   For all case_id ∈ main log do
5:     For all case_id ∈ sub log do
6:       If ** match_time {case1 (main_log); case2 (sub_log)} then
7:         ***  $MS_{n,m}$  == check_Match_Score {word_set (case1); word_set (case2)}
8:         end if
9:       end for
10:    end for
11:
12:   For all case_id ∈ main log do
13:     Generate new_case_id
14:     For all case_id ∈ sub log do
15:       Return case_id { $MS_{n,m} > 0 \ \&\& \ \text{Max}(MS_{n,m})$ }
16:       merge {case_id(main_log), case_id(sub_log)}
17:     end for
18:   end for
19:   return UnifiedLogFile

```

2.3 Running Example

A merged log of one New Case for our running example (Tables 1 and 2) is given in Table 4. While being related to a New ID, the events keep record of the reference log where they originated and the respective original case ID. The new case relates to an end-to-end process of one order (3001) placed by one department (89) for two items (1234 and 1235). Since handling the delivery of these items is grouped at the warehouse with items ordered by another department (79), the new case of the merged log includes the respective events as well.

Table 4. Illustration of the merged log of the running example.

New ID	Ref log	ID	Timestamp	User	Activity	Item no	Department
3001-A	main log	3001	02/02/14 10:12	Ilana	open order	1234 1235	dep. 89
3001-A	main log	3001	02/02/14 10:13	Tsvi	approve order	1234 1235	dep. 89
3001-A	main log	3001	02/02/14 13:16	Ilana	check status	1234 1235	dep. 89
3001-A	sub log	5001	03/02/14 11:45	Mosh	receive item	1234	
3001-A	sub log	5001	03/02/14 16:15	Mosh	send to dep.	1234	dep. 89
3001-A	sub log	5001	03/02/14 16:16	Mosh	send to dep.	1234	dep. 79
3001-A	main log	3001	03/02/14 16:18	Ilana	receive item	1234	dep. 89
3001-A	sub log	5002	04/02/14 12:46	Mosh	receive item	1235 1236	
3001-A	sub log	5002	04/02/14 16:30	Mosh	send to dep.	1235	dep. 89
3001-A	sub log	5002	04/02/14 16:31	Mosh	send to dep.	1236	dep. 79
3001-A	main log	3001	04/02/14 16:35	Ilana	receive item	1235	dep. 89
3001-A	main log	3001	04/02/14 16:36	Ilana	close order		dep. 89

3 Evaluation

The proposed algorithm has been implemented and evaluated in a controlled experiment using synthetic logs. The use of synthetic logs for evaluating the algorithm enables a fully controlled experiment, since (a) the correct match between cases is known in advance, allowing an accurate measurement of precision and recall [10][15] of the results, (b) when generating the logs, a full coverage of relationship types (one-to-one up to many-to-many) and temporal relations between the logs can be ensured, and (c) it is also possible to control the amount of text-related noise (additional irrelevant text) in the attribute values.

We generated event logs specifying a process model for each of the introduced relationships (i.e. one-to-one, one-to-many, many-to-one, many-to-many) as shown in table 5. As a result, four logs of a main process (similar to the running example) and

corresponding logs of sub processes were generated. The logs includes up to 260 cases, each case consisting of 3-7 events. The generated synthetic logs included the following mandatory fields (possibly with multiple instances): case ID (order number vs. delivery number), timestamp, user, department number, item number, and activity. The logs were generated by simulating the end-to-end process with the three possible temporal relations between the main and the sub-process, namely (1) sub process during main process (2) main process overlaps with sub process (3) sub process finishes main process (see Table 3).

Note that the logs included cases of the end-to-end process with similar attribute values. The corresponding cases in the main and sub log could have a relatively high match score, but should not be candidate for merging due to inappropriate temporal relations (e.g., main process before sub process or sub process before main process).

Table 5. Synthetic logs generated for the evaluation

Logs Relationship	Main Log – number of cases	Sub Log – number of cases	Unified log – expected number of cases
One-To-One (OTO)	130	130	130
One-To-Many (OTM)	130	260	260
Many-To-One (MTO)	260	130	260
Many-To-Many (MTM)	260	260	520

Applying the algorithm to all four versions of log combinations resulted in a perfect unified log with 100% recall and precision. Recall was calculated as the proportion of correctly matched cases from all the positives matches, and precision was calculated as the proportion of correctly identified matched cases from the total identified matches.

Yet, the investigated logs included only fully structured data with no free text. To evaluate the ability of the algorithm to handle noisy free text, we relied on the structure and context of real-life logs, with a 200 words free text attribute in the main log and three five - words free text attributes in the sub-log. Following this, we (1) added to the main logs a free text attribute of up to 200 words, and (2) added to the sub logs three free text attributes with up to five words each. The text for the attributes in all logs was randomly selected from the free text attributes of the real-life log. Note that the additional free text attributes served as noise, and were not supposed to determine the match. The results obtained by applying the algorithm to the logs that include free text are given in Table 6 and graphically presented in Fig. 2.

Table 6. The table present recall and precision calculations vs. relationship types.

Logs relationship	True positives (<i>tp</i>) - correctly identified	False positives (<i>fp</i>) - incorrectly identified	False negative (<i>fn</i>) - incorrectly rejected	Recall	Precision	F-measure
OTO	126	16	4	97%	89%	93%
OTM	244	28	16	94%	90%	92%
MTO	239	11	10	96%	96%	96%
MTM	501	35	15	97%	93%	95%

In general the results indicate that the algorithm performs well, with recall of at least 94% and precision of at least 89%. Recall was higher than precision in most cases (except for MTO, where they were equal). It should also be noted that no substantial trade-off was observed between precision and recall, and that no relationship type was identified as "easier match" with superior performance over the others, with insignificant differences of F measure values: 92% to 96%.

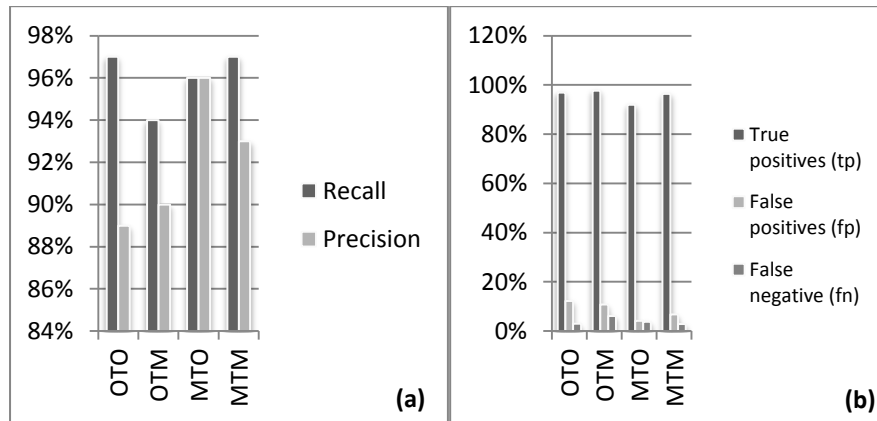


Fig. 2. Evaluation results: (a) precision/recall for all types of relationships (b) true positives/false positives/false negatives for all types of relationships

Note that despite the encouraging results, this evaluation is still too limited to draw general conclusions, but it is reasonable to believe based on these results that the performance of the algorithm is not sensitive to the type of relationship between the logs. Sensitivity to the amount and distribution of free text in the logs is yet to be tested. Last, the ability of the algorithm to handle real life complexity should be tested using a real-life log. To obtain an initial indication of the scalability of the algorithm and the required processing time when handling realistically large logs, we applied it to real-life logs taken from the 2014BPI challenge. The main process log contains 26,876 instances and the sub process log contains 21,960 instances. With this data, a unified log was obtained in 1 minute and 54 seconds. With a real-life log, however, precision and recall would only be estimated, as the real matches would not be known a-priori.

4 Related Work

Process mining uses event logs data in order to discover, monitor and improve the actual processes in an organization from an event log commonly available in information systems [1]. However, it should be applied to a single event log [1][3]. In case the addressed business processes are conducted at different systems, the pre-processing of the log, which usually aims at obtaining a "clean" and uniformly formatted dataset [18], should also combine the logs of the systems into a single one.

Merging logs as a preparatory step to process mining has not been widely addressed. The work which is closest to the one we propose is presented in [12], whose approach for matching the cases in the different logs is based on genetic algorithms. The main difference in the problem addressed is that they assume that each case in the sub-process relates to exactly one case in the main process, thus many-to-many relationships are not addressed. Another difference is that free-text data is not addressed by [12]. Another closely related work is [8], which addresses conformance checking of processes comprised of various sub-processes (proclets), each possibly having a separate log. Differently than our work, their basic assumption is that the connection between the sub-processes is known. Rather than merging the logs before mining, they mine each sub-process separately and then combine the process models.

The problem we address is related to mining hierarchical process models, since our aim is to facilitate the discovery of an end-to-end process comprised of sub-procedures. Mining hierarchical models has received research attention [6][11][13][20]. In most of these works, mining applies to a single log, and various approaches are taken to determine the hierarchy and to overcome abstraction challenges [6][11][20], attempting to cluster events into sub-processes [8][10]. In our case, we only lay the ground for mining by providing a unified log that can be mined. Since we keep the information of the original log each event is taken from, hierarchical mining would be easier and would not need techniques for discovering this information.

Process hierarchy is an inherent part of artifact-centric processes [5][7][19]. Artifact-centric processes are centered around an artifact, which encompasses data and life-cycle models. Artifact lifecycle is modeled by a Guard-Stage-Milestone (GSM) model [19], which provides a natural hierarchy of the model. Artifacts also enact and trigger other artifacts in a hierarchical manner, often including multiple instances. The triggered artifacts might be managed separately by different information systems and have separate logs. Following this, mining artifact lifecycle is of much relevance. Indeed, [17] and [16] address the possibility of many-to-many relationships between artifacts, and hence they abandon the case as a basis for mining. Yet, they assume a single log where all events are recorded. With this log, the effort is to relate each event to a relevant artifact so artifact-related logs are created, and then mining can produce a GSM model of these artifacts.

In summary, to the best of our knowledge the problem of merging logs with many-to-many relationships in preparation for process mining has not been addressed so far. Additionally, while text processing has received much attention in various areas, process mining mostly does not currently relate to free text data in logs.

5 Conclusions

Process mining techniques, which are useful for discovery and analysis of actual processes, rely on a single case ID that classifies all events into process cases. Hence, in common situations where processes include separately managed procedures with separate logs, some preprocessing is required for producing a unified log with unique

case IDs. Existing techniques are able to do this only when the main and sub-process have one-to-one or one-to-many relationships. This paper proposes an algorithm which can produce a unified log for all relationship types and specifically for the many-to-many case, where each log has non-matching case IDs. Another unique feature of the proposed algorithm is its ability to handle logs that contain unstructured and free-text data by using text mining techniques.

These capabilities enable mining complex and distributed processes that often exist in organizations and analyzing their flow. Process improvement opportunities that would emerge from such analysis would address the end-to-end flow rather than local views that relate to the individual sub-processes comprising the overall one. In particular, lack of coherence, correlation and synchronization among the different parts of the process can be identified. However, while the unified log provides a good support to overall flow analysis, it is less suitable for other analysis types. In particular since it includes duplicate sub-process activities that are associated with several main cases, it does not support analysis of activity frequencies and resource load.

Two other limitations of the algorithm are the assumption of comparable terminology and attribute values across logs, and the assumption of synchronized systems, producing timestamps along one time-line (as opposed to, e.g., systems in different time zones). The former can be overcome by using synonym-detection tools like Wordnet; the latter can be overcome if the lack of synchronization is consistent (different time zones). Then timestamps can be modified consistently as a preparatory step, to provide a uniform time-line for all logs. However, if lack of synchronization is not consistent and the time-gap between the systems might unexpectedly change, temporal relations between cases cannot be determined and the algorithm will not produce correct results.

Future research includes a number of directions. First, additional evaluation on real-life logs would be beneficial in order to better test the performance of the developed algorithm. Second, using the merged logs for the end-to-end process discovery might require some specialized visual representation. Visualizations used by current process discovery techniques are not fully supportive of the many-to-many relationship along process hierarchy, and might not provide the full benefits of process visualizations that apply to simply structured processes. Last, match results can be improved through interaction with the user, who can evaluate the matching of specific cases based on domain knowledge. Based on the user feedback, machine learning techniques can be used for improving the match results.

References

1. Van der Aalst, WMP.: *Discovery, Conformance and Enhancement of Business Processes*. Springer, Heidelberg, (2011)
2. Van der Aalst, WMP, et al.: *Process mining manifesto*. Business process management workshops. Springer Berlin Heidelberg, (2012)
3. Van der Aalst, WMP, Weijters, T., & Maruster, L.: *Workflow mining: Discovering process models from event logs*. Knowledge and Data Engineering, IEEE Transactions on, 16(9), 1128-1142, (2004)

4. Allen, J. F.: Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11), 832-843, (1983)
5. Nigam, A., & Caswell, N. S.: Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3), 428-445, (2003)
6. Baier, T., & Mendling, J.: Bridging abstraction layers in process mining by automated matching of events and activities. In *Business Process Management* (pp. 17-32). Springer Berlin Heidelberg (2013)
7. Cohn, D., & Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 32(3), 3-9 (2009)
8. Ferreira, D., Zacarias, M., Malheiros, M., & Ferreira, P.: Approaching process mining with sequence clustering: Experiments and findings. In *Business Process Management* (pp. 360-374). Springer Berlin Heidelberg (2007)
9. Fahland, D., De Leoni, M., Van Dongen, B. F., & Van Der Aalst, W. M.: Conformance checking of interacting processes with overlapping instances. In *Business Process Management* (pp. 345-361). Springer Berlin Heidelberg (2011)
10. Günther, C. W., Rozinat, A., & Van Der Aalst, W. M.: Activity mining by global trace segmentation. In *Business process management workshops* (pp. 128-139). Springer Berlin Heidelberg (2010)
11. Greco, G., Guzzo, A., & Pontieri, L.: Mining hierarchies of models: From abstract views to concrete specifications. In *Business Process Management* (pp. 32-47). Springer Berlin Heidelberg . (2005)
12. Claes, J., & Geert P.: Integrating computer log files for process mining: a genetic algorithm inspired technique. *Advanced Information Systems Engineering Workshops*. Springer Berlin Heidelberg, (2011)
13. Li, J., Bose, R. J. C., & Van Der Aalst, W. M.: Mining context-dependent and interactive business process maps using execution patterns. In *Business Process Management Workshops* (pp. 109-121). Springer Berlin Heidelberg (2011)
14. Raichelson, L. & Soffer, P.: Unifying Event Logs To Enable End-To-End Process Mining. In: *Proceeding of the 7th Israel Association for Information Systems (ILAIS) Conference*, July 2013.
15. Moghnieh, A., & Blat, J.: The potential of Recall and Precision as interface design parameters for information retrieval systems situated in everyday environments (2011)
16. Nooijen, E. H., van Dongen, B. F., & Fahland, D.: Automatic discovery of data-centric and artifact-centric processes. In *Business Process Management Workshops* (pp. 316-327). Springer Berlin Heidelberg. (2013).
17. Popova, V., Fahland, D., & Dumas, M.: Artifact lifecycle discovery. *arXiv preprint arXiv:1303.2554* (2013)
18. Rajaraman, A., & Ullman, J. D.: *Mining of massive datasets*. Cambridge University Press (2012)
19. Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., Heath III, F. T., & Vaculin, R.: Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *Proceedings of the 5th ACM international conference on Distributed event-based system* (pp. 51-62). ACM (2011)
20. Yzquierdo-Herrera, R., Silverio-Castro, R., & Lazo-Cortés, M.: Sub-process discovery: Opportunities for process diagnostics. In *Enterprise Information Systems of the Future* (pp. 48-57). Springer Berlin Heidelberg (2013)