Merging Computer Log Files for Process Mining: an Artificial Immune System Technique

Jan Claes and Geert Poels

Department of Management Information Systems and Operations Management
Faculty of Economics and Business Administration
Ghent University, Tweekerkenstraat 2, 9000 Ghent, Belgium
{jan.claes,geert.poels}@ugent.be

Abstract. Process mining techniques try to discover and analyse business processes from recorded process data. These data have to be structured in so called *computer log files*. If processes are supported by different computer systems, merging the recorded data into one log file can be challenging. In this paper we present a computational algorithm, based on the Artificial Immune System algorithm, that we developed to automatically merge separate log files into one log file. We also describe our implementation of this technique, a proof of concept application and a real life test case with promising results.

Keywords: Business Process Modelling, Process Mining, Process Discovery, Log File Merging

1 Introduction

Process mining techniques [1] are used to discover and analyse business processes in a semi-automatic way. Starting from all kinds of recorded process data (called *log files*) process mining tries to automatically discover the structure and properties of the business processes, which can be visualised in business process models.

Traditionally, business process models were made by domain experts, based on their experience and perceptions in the organisation. This manual task of modelling is subjective and time-consuming. In contrast, process mining techniques start from recorded actual process data and therefore the main benefits of process mining relate to correctness (no errors), completeness (no missing paths) and speed. [2]

However, the first step of gathering the recorded data is still a primarily manual task and thus the results of process mining techniques can be tempered if no optimal set of data is collected.

Three actions have to be taken before process discovery and analysis techniques can be performed: searching for data in the IT support systems, structuring these data (i.e. identifying single process steps (events) and groups of process steps that belong to the same process execution (process instances)), and converting these data to the format required by the process mining tool. If process data are found in different sources, then a fourth action is required: merging the data into one computer log file.

In this paper we present an automated technique for merging already collected, structured and converted process data according to an Artificial Immune System (AIS) algorithm, which is based on the features and behaviour of the vertebrate immune system. By automating this fourth action of the preparation step, we try to broaden the benefits of process mining to an extended part of the overall process mining procedure, because the automation makes the merge step in the preparation phase faster (speed), the use of data from multiple systems is facilitated (completeness) and the way these data are merged is less subjective than when performed manually (correctness).

We start with a description of the problem of log file merging for process mining and discuss related research topics in Section 2. The Artificial Immune System algorithm and its technical implementation details are presented in Section 3. Experiment results of a proof of concept application using a generated test case are described in Section 4. As a minimal form of validation the AIS merging technique was also applied to a real case. The results of this realistic exercise can be found in Section 5. To end the paper, a conclusion is provided in Section 6.

2 Problem Description

2.1 Process Mining

The starting point for process mining techniques is a single computer log file. This file often does not exist at the beginning of a process mining analysis, but must be constructed out of the actual recorded process data. These data have to be *collected* first from databases or files (e.g. SAP audit trails, web service log file) (Fig. 1).

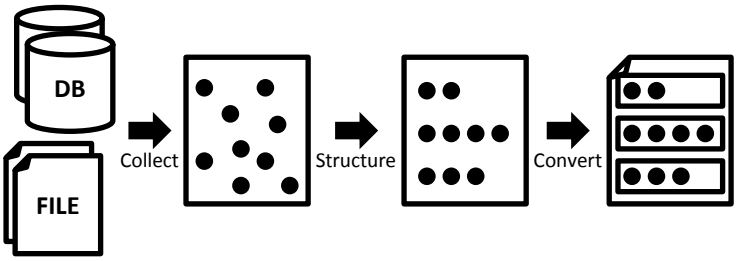

Fig. 1. Preparation steps before process mining techniques can be performed.

When all relevant data are collected, they have to be *structured* before analysis can start. A process is normally executed over and over again and thus the data set contains information of multiple executions of the same process. Different event records that belong to the same execution of a process are grouped into *traces*. Usually, one log file will contain information of only one process, but otherwise the traces are again grouped per process.

A last preparation step, before process mining can be applied, is the *conversion* of the structured data set into the proper format, mostly according to a selected tool. This is a pure syntactical exercise and should be possible in a (semi-)automated way.

2.2 Log File Merging

In the nineties many business process tasks were being automated or supported by IT systems. Many tools were developed and nowadays, in many cases, processes are supported by multiple IT systems with no clear relation to each other [3]. For this reason recorded process data is scattered across several databases or files and merging these data into one consistent data set can be challenging [3].

In this paper we present as a solution for this challenge a technique for automated merging of data from different sources. Our approach is implemented in ProM[1], a well known academic process mining tool, which implies that for our implementation we assume the different data sets are first separately structured and converted to the ProM file format[2] (e.g. with Nitro[3], a tool mainly made for this purpose).

Fig. 2 shows the steps for our solution implementation. First, data is collected, structured and converted into a series of ProM compatible computer log files (e.g. using Nitro). Second, our Artificial Immune System Merger plug-in in ProM is used to merge the files into one computer log file. Third, this log file is used as input to discover and analyse the business process with the other plug-ins included in ProM.

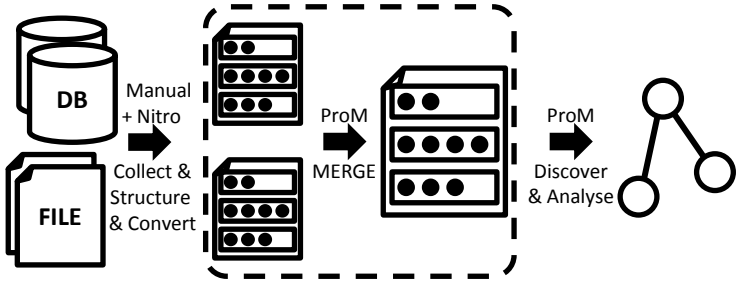

Fig. 2. Merging data of different sources can be performed after structuring and converting to a tool-specific file format. We implemented our merge technique in the ProM analysis tool itself.

2.3 Related Research

We are not aware of any literature reporting on research of automated log file merging techniques. Nevertheless the same kind of problem is studied in other research areas (e.g. data matching [4]).

---

[1] ProM can be downloaded at http://www.processmining.org/prom/downloads.
[2] More information about the ProM file format (xes) can be found at http://xes-standard.org.
[3] Nitro can be downloaded for trial at http://fluxicon.com/nitro.

In the field of process mining similar research is performed for *event correlation* problems [5, 6]. This subarea is concerned about finding a way to automatically structure log files (i.e. to determine which events belong to the same process execution and have to be put together in the same trace in the log files). For example *event clustering* techniques (e.g. [7]) usually calculate a proximity function that is used to decide if events belong together [5]. Where these techniques focus on finding out which *events* belong together, this paper describes a technique to find out which *traces* (groups of events) of different computer log files belong together.

3 Solution Design

The merging of two computer log files consists of two steps: (i) linking together traces of both logs that belong to the same process execution and (ii) merging these traces into one trace to be stored in a new log file. We assume reliable and comparable timestamps are available in the original logs causing the second step to be a simple exercise of chronological ordering of all the events of linked traces into one new trace in the resulting merged log file. Therefore our solution description focuses on the first step of finding traces in both log files that belong together. In our opinion, more than one factor can indicate that two traces should be linked (see 3.2.1). We looked for existing techniques that incorporate multiple indicators in their solution procedure and found our inspiration in the Artificial Immune System algorithm [8].

3.1 Artificial Immune System

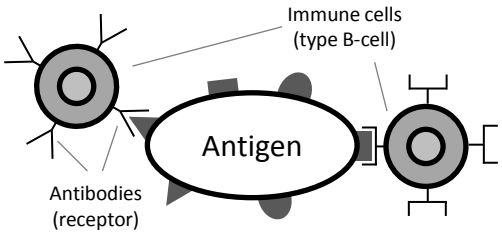

Fig. 3. Antigen (disease causing element) and immune cells (of type B-cell) with their antibodies (receptor molecules). (inspired by Fig. 1(a) in [8])

An Artificial Immune System (AIS) is a computational algorithm inspired by the vertebrate immune system (see Fig. 3). The main task of the vertebrate immune system is to discover and eliminate disease causing elements (called antigens). The cells responsible for this task are called immune cells. There are two types: B-cells recognise antigens by the molecules on their surface and T-cells require other accessory cells that in their place recognise the antigens. Our solution implementation is based on the B-cells which directly recognise antigens. The B-cells are covered with receptor molecules (called antibodies) which can bind with the antigen surface

molecules. The strength of a binding is related to the affinity between an antigen surface molecule and the antibody in the binding. If this affinity reaches a certain threshold value, the immune system is activated and the antigen is destroyed.

The real strength in the immune system lies in the principles of clonal selection, hypermutation and receptor editing. When antibodies connect with antigens with a high affinity, they clone themselves in high volumes. The higher the affinity, the higher the amount of clones. This principle of *clonal selection* causes the immune system to be highly resistant to the found antigens and become 'immune' to them.

After cloning, the antibodies are subject to random changes (*hypermutations*) and a more diverse population of antibodies is created. Because only the ones with the highest affinity with discovered antigens are cloned, the antibody population becomes better in recognising and killing antigens. The hypermutations are random, but the amount of changes depends on the binding affinity: the higher the affinity, the less changes.

Because the cloning, amount of mutations, but also the life span of the antibodies depend on the affinity with an antigen, the antibodies with the lowest affinity tend to leave the population and make room for newly formed antibodies, which is called *receptor editing*.

3.2 Implementation Details

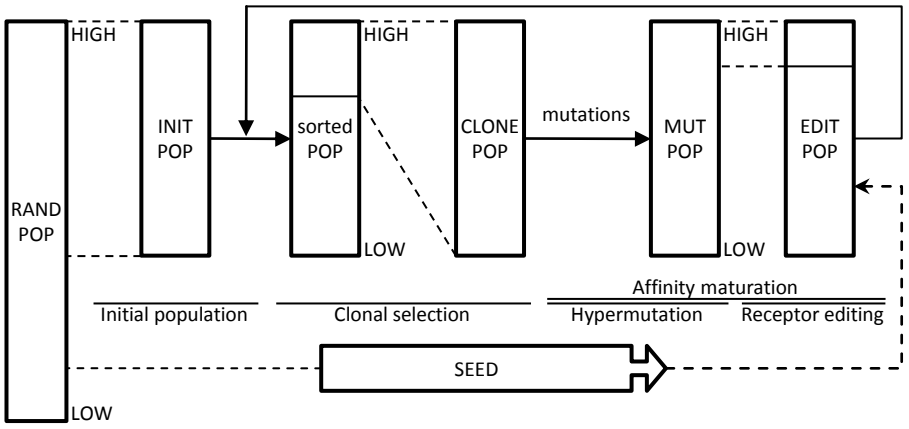

Fig. 4. Steps in our Artificial Immune System algorithm implementation (inspired by Fig. 1 in [9]).

Fig. 4 shows the steps in our AIS algorithm implementation. The algorithm starts with a total random population (RANDPOP) of solutions. Each solution is nothing more than a set of links between (a part of) the traces in both logs. To quantify the affinity of each set of links in the population, a fitness function score is calculated for every solution in the population. The solutions in the random population are next sorted according to their fitness function score. The actual population used throughout the algorithm is smaller than RANDPOP in size and therefore the initial population

(INITPOP) is constructed out of the best solutions in RANDPOP (so with the highest fitness function score). The AIS algorithm then iterates over three steps until a certain stop condition is met: clonal selection, hypermutation and receptor editing.

Clonal selection

The solutions in the population (INITPOP) are sorted according to their fitness function score. The top p_{clone} percentage solutions with the highest scores are selected to build up the next population. This new population (CLONEPOP) contains as much elements as the previous one and for this reason multiple clones of the same element will be included. The chance that a solution from the selected top p_{clone} solutions will be chosen for cloning depends on the fitness function score of that solution.

Hypermutation

Every solution in CLONEPOP is then altered (mutated) to build a new population MUTPOP. The amount of mutations on each solution depends again on the fitness function score, but this time, the higher the score, the less mutations. The amount of mutations for solution *s* (num_s) is calculated with formula (1):

$$num_s = \max\left(1, \frac{\left(100 - 100e^{-0{,}05(bestScoreSoFar - score_s)}\right) numLinks}{100}\right) \quad (1)$$

Each particular mutation on a certain solution follows the next four steps:

- **Indicator factor choice**. The goal of mutations is to find solutions which would get a higher fitness function score. As this score is the sum of a number of indicator factors (see 3.2.1), we choose a factor to be improved by this mutation. In the next three steps we try to improve our solution for this selected indicator factor, but it is possible that the overall score will decrease (due to the other factors in the function). At the start, there is an equal chance for each indicator factor to be selected, but also for a total random change (i.e. not aimed at improving a specific factor) to be selected. If at a time a mutation for a specific indicator did decrease the overall fitness score, this factor gets only half the chance of the other factors to be chosen in the next mutations. If it is again chosen and leads to an overall improvement this time, the chance is reset to be equal to the other factor chances.
- **Action choice**. For most indicator factors there are three possible actions to improve that factor score: add a link between traces, remove a link between traces or alter a link between traces. One of these actions is randomly chosen with equal chances. For some indicators a certain action is useless and has no chance to be chosen for that specific indicator factor (e.g. deleting a link between traces cannot make the number of links with a certain property higher).
- **Candidate choice**. For the selected indicator factor and action a set of candidates is assembled with all links for which the selected indicator factor can be optimised with the selected action. A random link is chosen from this set. Because a previous mutation on the same solution can have diminished the overall fitness function score, priority is given to all touched links in previous mutations on the same solution in the current algorithm iteration.

- **Improvement choice**. For the selected indicator factor, action and candidate link a set of improvements (new links) is built. A random improvement is chosen from this set.

Receptor editing

The solutions in MUTPOP are sorted according to their fitness function score. The top p_{edit} percentage solutions with the highest score are selected to be part of the next population (EDITPOP). This new population has to contain as much elements as the previous one and for this reason new solutions are picked from the initial random population (RANDPOP) to fill up the new population. The chance to be selected is again related to the fitness score of each solution: All solutions of RANDPOP have a chance to be selected for the new generation of POP, but the solutions with a higher fitness score still get a higher chance than the solutions with a lower fitness score.

Stop condition

The resulting population (EDITPOP) serves as input for another cycle of clonal selection, hypermutation and receptor editing. This iterating algorithm stops when a certain stop condition is met. Then the best solution of all generated populations (which is continuously updated) is the proposed solution. In our implementation a fixed amount of iterations can be set (*numIter*). If in *numIterNoOpt* consecutive iterations no improvement of the overall best solution is achieved, the algorithm stops earlier. The algorithm parameters (size of RANDPOP, size of the other populations, p_{clone}, p_{edit}, *numIter* and *numIterNoOpt*) can be modified by the user to be optimised for a certain combination of input logs.

3.2.1 Fitness function

The fitness function determines the affinity of a certain solution. This fitness function score is used throughout the whole algorithm as every step is influenced by the affinity. Because different factors can indicate that traces in both logs belong together, the fitness function is built up from different indicator factors:

$$f = \sum STI_i + \sum SO_i + \sum EAV_i + \sum \mathrm{ET_j} + \sum \mathrm{MT_j} + \sum TD_i \qquad (2)$$

In the next part of this text we will use the terms *first trace* and *second trace*. With these terms we mean a trace from one of both logs and a trace from the other log respectively. Notice that the input order of the log files to be merged is not important.

Same trace identifier (STI_i)

A first indicator for two traces to belong together is if they have the same trace identifier (i.e. the process execution is consistently identified in both logs). In this case the problem is rather trivial, because it's almost certain how to link the traces from the two logs. But this is no reason to exclude the factor from our fitness function. If, exceptionally, two traces with the same trace identifier do not belong

together (e.g. a customer number that matches with an invoice number), then another solution should score higher due to the other indicator factors of the fitness function.

Same originator (SO_i)

A second indicator could be the originator: the resource that is registered for a certain event or trace. If the originator of a certain first trace matches with an originator of a second trace, we assume there is a higher chance that both traces belong to the same process execution and thus should be linked. Notice that if for example all the originators in both files are the same, the overall fitness function score of each solution is high, but compared to each other, there will be no difference for this indicator factor. In this case the factor has little effect, which is desired because the factor is in fact irrelevant.

Equal attribute values (EAV_i)

In many processes a reference number or code is used throughout the entire process. This is most probably the trace id. But maybe other numbers are passed from event to event. If this number is logged, we should search for matching values of event attributes. Note that attribute names do not need to correspond. The name for this number can be different in both logs (e.g. "invoice number" and "reference number") and matching attribute names is more challenging [10]. Also note that some attribute values may have equivalents in lots of traces (for example status *completed*). This would make barely any difference between different solutions, because almost all possible solutions would score higher.

Extra trace (ET_j)

It is possible that a first trace should be linked to multiple second traces (e.g. one order handling causes two deliveries), but we think the number of second traces linked to the same first trace should be rather low. If too many second traces are linked to the same first trace, this indicator factor makes the overall fitness function score decrease (unless other factors have a greater positive effect indicating that there should be more than one trace linked to the current trace).

Missing trace (MT_j)

Analogically, we think a solution with traces of both logs that are not linked is probably less correct. If there are second traces that are not linked to a first trace, this indicator factor makes the overall fitness function score decrease. The combination of this factor and the extra trace factor should lead to an even spread of links between the traces in both logs.

Time difference (TD_i)

A last indication in our implementation for two links to belong together is the time difference. In our opinion smaller time differences are more probable then higher differences, which is represented by a higher score for smaller time differences. The

time difference for a certain link in a certain solution is defined as the difference between the times of the first events of both logs.

Each indicator factor has a weight that can be changed by the user to give the opportunity to influence the algorithm with his insights on the log file merging problem. At the end an overview of the individual indicator scores for the solution is presented to the user which also gives him the chance to gain insight and to start over with new indicator scores.

Some of the factors are calculated for each individual link in the solution (STI_i, SO_i, EAV_i and TD_i). Therefore in RANDPOP we also include two special solutions which we think can be a good starting point for the optimal solution: one for which we linked every first trace to the second trace with which it has the highest individual link score and one for which we linked every second trace to the best first trace.

4 Proof of Concept

We have tested our technique with a simulated example. The benefit of using simulation is that the correct solution (i.e. the process to be discovered) is known. Another advantage is that properties like time difference or noise can be controlled.

The example model we used in our experiments (see Fig. 5) is based on the same example model as in [11]. We generated two log files with 100 random executions of the process where the executions of tasks A, E, and F were logged in a first log file and the executions of tasks B, C and D in the second file. We initially did not include noise, the executions did not overlap in time, and there was no structural unbalance in choosing one or the other path first for the AND-split (B and C) or selecting the path to be followed for the OR-split (A or E). Because there is no unbalance in choosing paths in the OR-split, the second log ends up with about 50 traces. Time differences between consecutive events were also random.

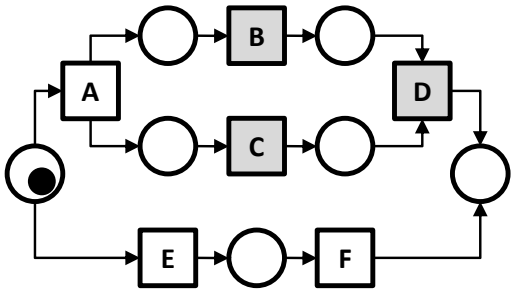

Fig. 5. Process model for our back-end IT support process example

We then also generated different log files with different properties:

- We added noise in the same way as described in [12]. One of four options is randomly selected: (i) delete minimum one and up to one third of events from the beginning of a trace, (ii) from the middle of a trace, (iii) from the end of a trace, or (iv) switch places of two random events in a trace. The noise percentage determines the chance a trace is influenced by noise in one of the four ways.
- Another property we varied is overlap. The overlap percentage determines the chance of each execution to start during the previous execution. With 10% overlap 10% of traces started before the previous ones ended.
- Finally, we repeated each test with two log files without matching trace identifiers.

The results of our tests with matching trace identifiers were perfect (in all our tests a perfect set of links was found and both log files were correctly merged). The results of our tests without matching trace identifiers can be seen in Table 1. Our implementation always found the correct *number* of links, but when traces run partly in parallel, there seems to be too little information left to find the right links. The amount of noise in the logs seems to have little impact on the correctness of the identified links. The duration for all our tests was about 300-400 milliseconds on a 3,45GB RAM 2,39 GHz laptop.

Table 1. Test results for non-matching identifiers with varying noise and overlap percentage (percentage of correct links in relation to total links identified)

No matching id	Overlap						
Noise	0%	10%	20%	50%	75%	100%	Mean
0%	100%	94%	80%	68%	46%	48%	73%
10%	100%	88%	85%	68%	53%	52%	74%
20%	100%	92%	88%	54%	45%	42%	70%
50%	100%	91%	87%	71%	58%	49%	76%
Mean	100%	91%	85%	65%	50%	48%	

5 Validation

We also tested our new AIS algorithm on a real test case in a university in Belgium. The payroll process of a specific type of employees is shown in Fig. 6: Different users register payroll information in an SAP application (step 1), the data is stored in the SAP database (step 2), data is extracted to transfer files (step 3) that are imported and processed in the old salary calculation system in Oracle (step 4 to 6) and get back to the SAP database and applications through other transfer files (step 7 to 9).

The steps where we extracted logs are shown in black circles (step 2, 3, 5, and 7). The constructed log files for step 2, 5 and 7 contained recorded information of 1.032 employees and used the same trace identifier. The constructed log file for step 3 contained information of 8.242 employees and used another trace identifier (although we were able to check our merge solution because the trace identifier used in the other three log files could be derived from an attribute in the log for step 3).

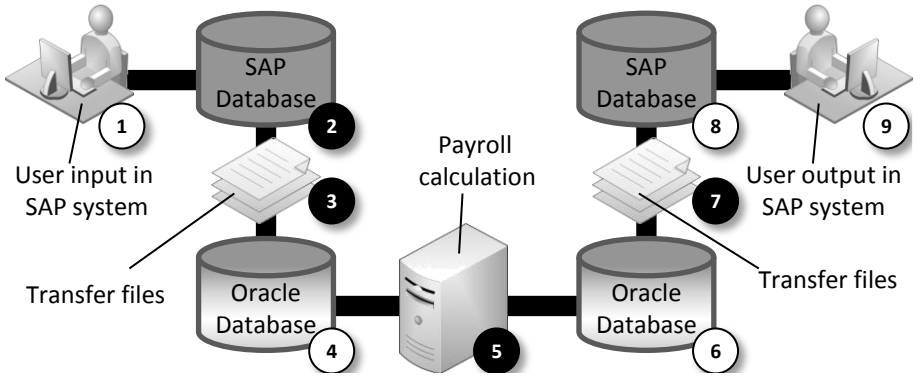

Fig. 6. Main steps in the payroll process

We merged the log from step 2 with the log from step 3, the resulting merged log was then merged again with the log from step 5 and finally the overall resulting merged log was again merged with the log from step 7. The results of these merge exercises are shown in Table 2.

Because the merged log files never have the same trace identifiers, results are average (the amount of links between the files is correct, but there are 32-39% incorrect links). The durations of the merging on a 3,45GB RAM 2,39GHz laptop are also presented in Table 2 (6-10 minutes). We did an extra test to merge only the log files from step 5 and step 7 (with matching trace identifiers) and noted that results were considerably better (4% incorrect links) and faster (53 seconds).

We discovered that for all our tests the fitness function score for the right solution would have been lower than the score of the proposed solution. This suggests our algorithm finds a solution with an optimal score, but the errors were made due to an imperfect fitness function (missing indicator factors, obsolete factors or suboptimal fixed factor scores).

Table 2. Results for our real life test case experiment.

Merging	Number of traces	Number of linked traces	Number of correct linked traces	Duration of merge
2 & 3	1032 & 8242 = 8242	1032/1032 (100%)	700/1032 (68%)	6 min.
23 & 5	8242 & 1032 = 8242	1032/1032 (100%)	697/1032 (68%)	9,5 min.
235 & 7	8242 & 1032 = 8242	1032/1032 (100%)	627/1032 (61%)	10,3 min.
5 & 7	1032 & 1032 = 1032	1032/1032 (100%)	990/1032 (96%)	53 sec.

6 Conclusion

In this paper we presented a technique for log file merging using an Artificial Immune System algorithm. All the steps in this algorithm are influenced by a fitness function,

which determines the quality of discovered parts of the solution. To calculate the fitness function score a set of factors is defined that indicate if parts of the two logs belong together. The sum of all factors has to lead the algorithm to an optimal solution. We implemented the algorithm in ProM, a well known academic process mining tool and tested our solution with a set of generated files with varying characteristics and a real life test case.

One of the indicators to decide if a trace of one log matches with a trace of the other log is the *trace identifier*. If the identifier of two traces is equal, it is almost certain that both traces belong together. For all our tests with matching trace identifiers the log files were correctly merged. If the traces of both logs had different identifiers, our implementation struggled with log files with many overlapping traces, but had few problems with log files with much noise. Our future research includes optimising our implementation (in speed and correctness) and validating our solution with extended case studies.

7 References

1. Van Der Aalst, W.M.P. Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer-Verlag New York Inc (2011)
2. Rozinat, A., Mans, R.S., Song, M., Van Der Aalst, W.M.P. Discovering Simulation Models. Information Systems. 34, 305–327 (2009)
3. Georgakopoulos, D., Hornick, M. An overview of workflow management: from process modeling to workflow automation infrastructure. Distributed and parallel. 3, 119-153 (1995)
4. Shvaiko, P., Euzenat, J. A survey of schema-based matching approaches. Journal on Data Semantics IV. pp. 146-171 (2005)
5. Motahari-Nezhad, H.R., Saint-Paul, R., Casati, F., Benatallah, B. Event correlation for process discovery from web service interaction logs. The VLDB Journal. (2010)
6. De Pauw, W., Hoch, R., Huang, Y. Discovering Conversations in Web Services Using Semantic Correlation Analysis. IEEE International Conference on Web Services (ICWS 2007). 639-646 (2007)
7. Ferreira, D., Zacarias, M., Malheiros, M., Ferreira, P. Approaching Process Mining with Sequence Clustering: Experiments and Findings. Business Process Management. p. 360–374 Springer-Verlag New York Inc (2007)
8. Castro, L.N. de, Timmis, J. Artificial immune systems: A novel paradigm to pattern recognition. Artificial Neural networks in pattern Recognition. 67–84 (2002)
9. Van Peteghem, V., Vanhoucke, M. An Artificial Immune System for the Multi-Mode Resource-Constrained Project Scheduling Problem. 9th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2009. p. 85 Springer-Verlag New York (2009)
10. Wang, J.R., Madnick, S.E. The inter-database instance identification problem in integrating autonomous systems. Data Engineering. p. 46–55 IEEE (2002)
11. Van Der Aalst, W.M.P., Weijters, A.J.M.M. Process Mining: A Research Agenda. Computers in Industry. 53, 231–244 (2004)
12. Weijters, A.J.M.M., Van Der Aalst, W.M.P. Rediscovering Workflow Models from Event-based Data Using Little Thumb. Integrated Computer-Aided Engineering. 10, 151–162 (2003)