Merging Event Logs for Process Mining with Hybrid Artificial Immune Algorithm

Yang Xu [1], **Qi Lin**[1], **Martin Q. Zhao**[2]
[1]School of Software Engineering, South China University of Technology, Guangzhou, China
[2]Computer Science Department, Mercer University, Macon, Georgia, USA

Abstract - *Current process mining techniques and tools are based on a single log file. In an actual business environment, however, a process may be supported by different computer systems. So it is necessary to merge the recorded data into one file, and this mission is still challenging. In this paper, we present an automatic technique for merging event logs with an artificial immune algorithm combed with simulated annealing. Two factors that we used in the affinity function, occurrence frequency and temporal relation can express the characteristics of matching cases more accurate than some factors used in other solutions. By introducing simulated annealing selection and an immunological memory to strengthen the diversity of population, the proposed algorithm has the ability to deal with local optima due to pre-mature problem of artificial immune algorithms. The proposed algorithm has been implemented in the ProM platform. The test results show a good performance of the algorithm in merging logs.*

Keywords: Process Mining, Event Log Merging, Artificial Immune

1 Introduction

Process mining makes use of recorded historical process data in the form of so called event logs to discover and analyze as-is process models [1]. In an intra- or inter-organizational setting, log data are distributed over different sources, each of which encompasses partial information about the overall business process. However, currently available process mining techniques, such as Heuristics Miner [2], Generic Miner [3], Fuzzy Miner [4], and Conformance Checking [5] require these data first to be merged into a single event log. Merging logs for an extended analysis of the entire process becomes a challenging task.

The most remarkable research on merging the log files is made by Claes [6], in which an artificial immune system is used. Their method assumes the existence of an identical case ID among various logs, or a case ID as an attribute in each event log with which events in different logs related to the same case can be matched. When two traces have the same case identifier, it is easy to establish the correlation directly. In real-life processes, however, each sub-process often employs different case ID. It is not easy to keep the case ID same in different logs. In addition, as with many applications of genetic algorithm, the artificial immune algorithm also has a problem of low performance due to tendency to converge towards local optima.

In this paper we present an automatic technique for merging event logs, which is based on a hybrid artificial immune algorithm combined with simulated annealing. In the method, we optimize the affinity function with two factors: occurrence frequency of execution sequences and temporal relations between process instances. An immunological memory is used to keep solutions with highest affinity score and apply simulated annealing to strengthen the diversity of later generations so as to alleviate the problem of local optima.

The rest of the paper is structured a follows. Section 2 clarifies the concepts about merging log data. Section 3 discusses related work. Our approach and the log merging algorithm are introduced in section 4. Section 5 evaluates our approach. Finally, conclusions are given in Section 6.

2 Related concepts

Before we introduce our approach, we first need to clarify some concepts related to event log merging.

Definition 1 (**Process**) A process is a tuple $P = (A, I, O, A_0, A_e)$, where,

- A is a finite set of activities,
- $I: A \rightarrow \mathcal{P}(\mathcal{P}(A))$ is the input condition function. $\mathcal{P}(A)$ denotes the powerset of some set A.
- $O: A \rightarrow \mathcal{P}(\mathcal{P}(A))$ is the output condition function,
- $A_0 \in A$ is start point of the process, $I(A_0) = \{\emptyset\}$,
- $A_e \in A$ is end point of the process, $O(A_e) = \{\emptyset\}$

For a process $P' = (A', I', O', A'_0, A'_e)$, if $A' \subseteq A$, and $I' \subseteq I$, $O' \subseteq O$, then call P' is a sub-process of P, denoted as $P' \subseteq P$.

Definition 2 (**Execution sequence**) Given a process $P = (A, I, O, A_0, A_e)$, a sequence $\sigma \in A^*$ is called a execution sequence, if and only if, for $n \in IN$, there exist $a_1, \cdots, a_n \in A$ and $a_1 = A_0$, $a_n = A_e$, such that $\sigma = a_1 \cdots a_n$, and, for

all i with $1 < i < n$, I_i and O_i are the input condition function and output condition function of a_i respectively, $I_i \subseteq \mathcal{P}(\mathcal{P}(\{a_1, \cdots, a_i\}))$, and $O_i \subseteq \mathcal{P}(\mathcal{P}(\{a_i, \cdots, a_n\}))$.

Definition 3 (**Log schema**) A log schema S is a finite set $\{D_1, \cdots, D_n\}$, where D_i with $(i = 1, \cdots, n)$ is called as attribute.

Definition 4 (**Event log**) Given a log schema S for a process P, an event log is a tuple $L = (S, P, E, \rho)$, where, $E \subseteq D_1 \times D_2 \times \cdots \times D_n$ is a finite set of events $\{D_1, \cdots, D_n\}$, and ρ is a mapping from the set E to the process P, that is $\rho: E \rightarrow A$.

In the process mining domain, an executed activity of a process, called an ***event***, is recorded in event log. An event $(d_1, \cdots, d_n) \in D_1 \times D_2 \times \cdots \times D_n$ is an interpretation over the set of activities A associating the log schema, i.e. an instance of the log schema. Actually, the event is the smallest unit in event log. A record in log is the description of an event with a group of attribute values, for example, execution time and executor of an activity, which are defined in a log schema. To be convenient, an event log can be briefly regarded as a set of events, denoted by $L(P)$. The denotation of $e.d$ represents the value of attribute d while e denotes an event.

Definition 5 (**Case**) Given an event log L, a set of events $\omega \subseteq E$ is called a case of L, denoted as $\omega \in L$, if and only if, the following conditions are met,

- Each event in ω appears only once. That is, for event $e_i, e_j \in \omega$ with $1 \leq i, j \leq |\omega|, e_i \neq e_j$,
- The events in ω are ordered, and
- ω is one of the instances, or the actual execution of an execution sequence.

A case also has attributes. For example, the order of events in a case is an attribute of the case. The duration of a case from the time of the start event to the end event is another case attribute. The denotation of $\omega.a$ represents the value of attribute a of the case ω.

Obviously, an event log can also be regarded as a finite set of cases. When we call a log as event log, it means that the log has been structured for process mining from the raw log, and all the cases have been identified.

Definition 6 (**Occurrence frequency**) Given a process P and its event log L, let T be a finite set of all execution sequences of P, $\mathcal{F}: T \rightarrow \text{IN}$ is a mapping about T over L. For $\forall \sigma \in T$, $\mathcal{F}(\sigma)$ is the number of occurrences of σ.

Definition 7 (**Mergable log**) Let two processes P_1, P_2 with $P_1 \subseteq P$, $P_1 \subseteq P$, and $L(P_1) \cap L(P_2) = \emptyset$. $L(P_1)$ can be merged with $L(P_2)$, if $\forall \omega^{(P_1)} \in L(P_1)$, $\exists \omega^{(P_2)} \in L(P_2)(\omega^{(P_1)} \cup \omega^{(P_2)} \in L(P))$.

Basically, event log merging consists of two steps: (i) correlate cases of both logs that belong to the same process execution and (ii) sort the activities in matched cases into one case to be stored in a new log file. The main challenge is to find the correlated cases in both logs that should be considered related to the same entire case. In this paper we limit ourselves to this challenge and we take the following assumptions:

- Logs from various systems can be preprocessed to a uniform format [7].
- All cases in logs have been identified [8], and
- Every event in logs has the attribute of reliable and comparable timestamp such that events can be easily sorted according to the values of timestamps. That is, $(\omega, \leq_{timestamp})$ is a partially ordered set, for $e_i, e_j \in \omega, 1 \leq i < j \leq |\omega|$, $e_i \leq_{timestamp} e_j$.

3 Related works

Merging logs is the problem of log pre-processing for process mining. Other researches on pre-processing of logs focus on uniform format [7] and event correlation [8], while the actual merging process has not been widely addressed.

Several approaches have been proposed in recent years. Rule-based log merging method [9] provides a tool with which a set of general merging rules are used to construct a set of specific matches between cases from two event logs. However, the tool only provides the descriptive language for the rules. The rules and decisions for specific logs have to be made by users who need to have enough knowledge about both the business process and the rule language. Text mining-based merging method [10] uses text mining techniques to merge logs with many to many relations. In this approach, similarity of attribute values is calculated and cases are matched according to the assumption that matching cases would have more common words in their attribute values than non-matching cases. The approach still has to face the situation in which identical word has possibly different semantics in two logs.

Artificial immune system-based method [6] is the most remarkable approach for merging logs. This approach exploits the artificial immune algorithm [11] which is inspired by the biological immune system to merge two mergeable logs. Judging whether two cases from two different logs belong to the same process execution becomes a process in which the strength of the binding of antigen and antibody is evaluated. The strength is related to the affinity between an antigen and the antibody in the binding, i.e. between two cases. When the affinity reaches certain threshold value, the two cases can be regarded as belonging to the same process execution. The underlying principle is the Clonal Selection Principle, which requires the matching process undergo iterations of clonal selection, hypermutation and receptor editing until a certain stop condition is met. The algorithm starts from a random population of solutions. Each solution is a set of matches

between the cases in both logs. The affinity of each matching is calculated and the total is summed for the solution. The higher the affinity value of a matching, the higher the likelihood for that matching being cloned, the fewer the chances for being mutated and edited. When a stop condition is met through the evolution of several generations, the solution with highest affinity is the proposed solution.

Obviously, the calculation of affinity is a very important impact factor on the performance of the algorithm. The authors assume identical case ID as an attribute in the event log with which the affinity is calculated. Actually, it is meaningless for the affinity calculation when different case IDs are exploited in the real life logs. On the other hand, the artificial immune algorithm also has a problem of tending to converge towards local optima. No discussion about the problem is present in their papers.

In summary, the problem of merging logs for process mining has not been addressed so far. Our work in this paper is trying to fill the gap in this research area.

4 Merging event log

4.1 Overview

Basically, correlating cases from two logs is the process in which finding the best solution among all possible solutions with matched cases. We believe that using only one factor, e.g. timestamp of events or case id, to judge whether two cases from the different logs belonging to the same process execution is not reliable due to the complexity of business processes and their execution context, as well as the imperfect logs caused by operational faults of information systems and other unexpected events. For this kind of search and optimization problems, evolutionary algorithms, such as genetic algorithm and artificial immune system, are the appropriate selection. In this paper, we choose artificial immune system with an immunological memory as the foundation. In this section we describe the key steps of our log merging approach.

The approach starts from a random population of solutions. The solutions are sorted according to their affinities. The top p percent solutions with higher affinity are selected to construct an initial population for subsequent immune process. Every generation of population has to go through four steps: clonal selection, hypermutation, annealing operation and diversity strengthening. The algorithm then iterates over these steps until a stop condition is met. Here, we set a fixed amount of iterations as the stop condition.

The solutions in each population are sorted according to their affinity. The top ranking solutions with the highest scores are selected (cloned) to construct the next generation. Then, each solution is mutated to build a new population. The amount of mutations on each solution also depends on its affinity: the higher the affinity, the less mutation.

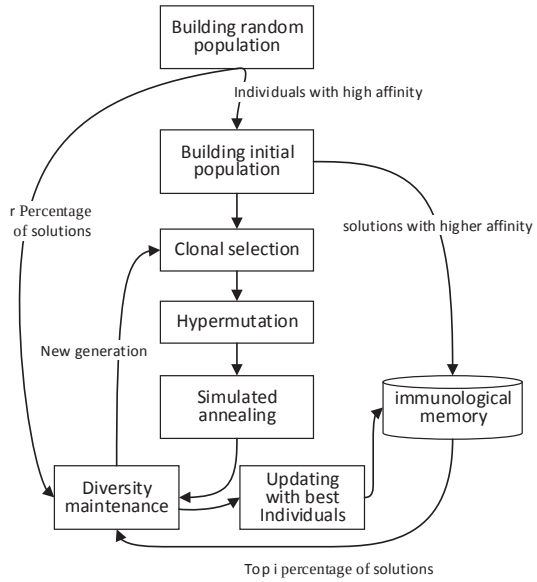


Fig. 1. The processing framework of merging logs

Like genetic algorithm, premature convergence may often occur in artificial immune search and make the merging process slow down. Diversity maintenance in every generation is the key for avoiding premature convergence or persistent degradation. We introduce simulated annealing in artificial immune algorithm to guarantee that not only better solutions with higher affinity than last generation are kept, but also some deteriorative solutions at a certain probability are accepted to generate the new population. On the other hand, we set an immunological memory to keep solution with the highest affinity in every generation. When building a new generation, the solution in the memory is selected to strengthen the diversity of antibodies.

The most important design choice in the algorithm is the affinity function, which determines the affinity of a solution. The higher the affinity, the greater the probability of merging two logs. Every step is influenced by the affinity as the evaluation of affinity is used throughout the whole algorithm.

In the affinity function, two important factors related to attributes of case, *occurrence frequency* and *temporal relation*, are used. If two cases from different logs belong to the same whole process execution, the occurrence frequencies of their corresponding *execution sequences* are statistically equivalent. If the difference between *occurrence frequencies* of two execution sequence exceeds a threshold value, the corresponding cases are not considered from the same process execution. The factor of *temporal relations* describes the time overlap between two cases. In this paper, we assume that (1) before merging two logs, we know which corresponding process starts first; and, (2) if two processes whose logs can be merged, one process that starts before the beginning of the other process triggers the latter. It should have some time overlap between the processes. Hence, the time overlap can be used to check whether two cases are matched on time property, described in detail below.

4.2 Affinity function

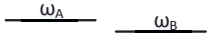
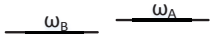
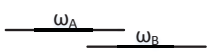
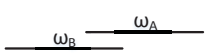
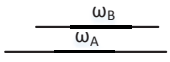
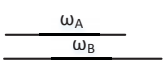
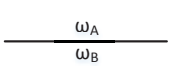
Because more than one factor can indicate that two cases should be matched, the affinity is evaluated with a number of indicator factors.

$$f = \alpha_1 \sum AOF_i + \alpha_2 \sum OLT_j + \alpha_3 \sum EAV_k \quad (1)$$

Here, AOF_i indicates (approximate) equivalent occurrence frequency between two execution sequence of two logs. Given case $\omega_A \in L(P_1)$, $\omega_B \in L(P_2)$ and $P_1, P_2 \subseteq P$, if ω_A and ω_B are matched, i.e. $\omega_A \cup \omega_B \in L(P)$, then the Occurrence frequency of the execution sequence of ω_A is equivalent to, or close to that of ω_B. This factor has a greater positive effect on the affinity value, indicating that two logs have greater probability to be mergeable when the situation appears more.

OLT_j indicates the temporal relations between two processes. If two logs are mergable, one corresponding process should be triggered by the other. The triggered one should start after the beginning of the other. Therefore, there is some time overlap between the two cases. Table 1 shows the types of temporal relations between two cases. Note that we have to make a decision on which case starts earlier before the temporal relation is used to calculate the affinity. In table 1, ω_B is triggered by ω_A.

Table 1. Temporal relations between two cases

Relations	Illustrations	Match	Comments
$\omega_A < \omega_B$		N	No overlap - No relation of triggering between ω_A and ω_B
$\omega_B < \omega_A$		N	
$\omega_A \preccurlyeq \omega_B$		Y	Partial overlap - ω_B is triggered by ω_A.
$\omega_B \preccurlyeq \omega_A$		N	Partial overlap - ω_A is triggered by ω_B.
$\omega_A \ni \omega_B$		Y	Full contain - ω_B is triggered by ω_A and ends before ω_A.
$\omega_B \ni \omega_A$		N	Full contain - ω_A ends before ω_B.
$\omega_A \parallel \omega_B$		N	Parallel between ω_A and ω_B

In our affinity calculation, only two temporal relations, Partial overlap and Full containment, make contributions. It seems that we only need to take into account the situations of $\omega_A \preccurlyeq \omega_B$ and $\omega_A \ni \omega_B$, since only these two indicate the matching between ω_A and ω_B. However, it is possible that the relations $\omega_A \preccurlyeq \omega_B$ and $\omega_B' \preccurlyeq \omega_A'$ in which ω_A and ω_A' have the same execution sequence as well as ω_B and ω_B', are both appear in the relation statistics. In this situation, the rule that the relations $\omega_A \preccurlyeq \omega_B$ and $\omega_A \ni \omega_B$ increase the affinity while $\omega_B \preccurlyeq \omega_A$ and $\omega_B \ni \omega_A$ decrease the affinity.

EAV_k indicates matching values of event attributes between two logs. In real life processes, it often happens that some values (e.g. "invoice code") are passed from event to event between two cases belonged to two different logs but identical whole process. Hence, we can calculate the amount of matches of event attribute values to indicate the affinity of two logs.

The α_1, α_2 and α_3 are the weight of the factors with $\alpha_1 + \alpha_2 + \alpha_3$=1. Every weight can be adjusted by the user with his knowledge of business process to adapt the algorithm to concrete problem.

Of course, some other factors of case attributes or event attributes, such as identical case ID, can be added into the affinity function. Some factors representing business properties are allowed to be considered to adapt the evaluation of the affinity to actual business. These factors need to be designed by business specialists.

4.3 Immunological memory

The immunological memory is set to keep solutions with the highest affinity score in every population. The solutions in the memory are selected to generate next generation of population. The initialization of the memory occurs when initial population is constructed. The top t percent solutions are selected into the memory. And these solutions are updated with those with much higher affinity when new candidate solutions produced by mutation and picked up by the simulated annealing.

4.4 Simulated Annealing

Simulated annealing is a probabilistic method based on the physical process of metallurgical annealing for approximate global optimization in a large search space [11]. Here, we exploit it to make the artificial immune merging jumping out from the trap of local optima. When a population undergoes mutation, we evaluate the new candidate solutions and get the best one whose affinity is the highest in the population. If it has a higher affinity value than the highest from the previous population, the simulated annealing accepts it. Otherwise, it is accepted into this new population with a certain probability calculated according to the equation (2).

$$Pr = exp(|\Delta Z|/kT) \quad (2)$$

Here, ΔZ is the affinity difference between the new candidate solution and the best solution in last generation. T represents the current temperature, and k is a constant. The algorithm for accepting candidates is shown in algorithm 1.

Algorithm 1: Accept new populations by simulated annealing

```
Input: G(i) //the current population
       P(i) //the best solution of the current population G(i)
       B(i) //the worst solution in the immunological memory Bank[]
outout: G(i+1) // next population
T ← affinity(P(i)) - affinity(B(i)); // initial temperature
T_min ← 0.0;  //temperature for stopping
while G(i) do
   calculate each individual's affinity in G (i);   // equation (1)
   if P(i) > B(i) do
      update Bank[] with the best individual in G(i);
   if terminal condition is true then
      return
   G '(i+1);    //clone and mutation for new population
   P'(i+1);    //the best individual in G '(i+1)
   if ( T > T_min ) do
      ΔZ  ← affinity(P'(i+1)) - affinity (P(i));
      if ( ΔZ>= 0 ) do
         G(i+1) ←G '(i+1);   //accept new population
      else
         if ( exp(ΔZ / kT) > random(0,1) ) do
            G(i+1) ←G '(i+1);    //accept new population
            T ← r * T;     // temperature decrease
         else
            G(i+1) ← G(i);    // reject new population
```

4.5 Diversity maintenance

The solutions are selected by simulated annealing into the next population. This new population has to contain as many elements as the previous one and for this reason new solutions are picked from two other sources. One is the solutions from the random population. The other is the solutions in the immunological memory. All solutions of the random population and the immunological memory have a chance to be selected for the new generation, but the solutions with a higher fitness score still get a higher chance than the solutions with a lower fitness score. The amount of solutions from the three sources is calculated with equation (3).

$$size_{pop} = N \times n + I \times i + R \times r \qquad (3)$$

In equation (3), *N* represents the amount of solutions in new population accepted by the simulated annealing. *I* represents the amount of solutions from the immunological memory. And *R* represents the amount of solutions from the random population. The *n*, *i*, *r* are the are the percentages of N, I, R respectively, and can be decided by users.

4.6 Complexity analysis

The time overhead of our approach depends on size of logs. We assume that there are n_1 and n_2 cases respectively in two logs, with an average of m events each case and an average of r attributes each event.

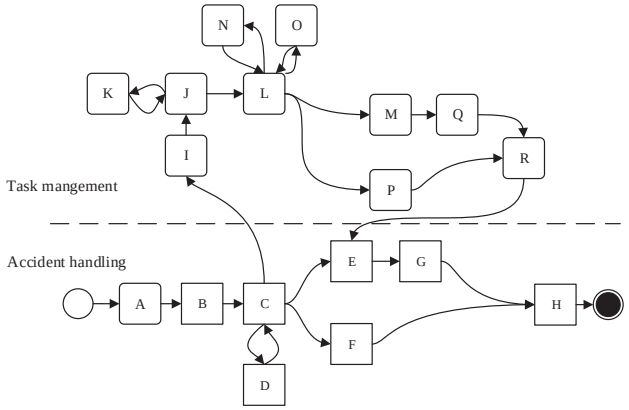
In the step of initial population, n solutions have to be built ($n = \max\{n_1, n_2\}$). The affinity of every candidate solution is calculated according to formula (1). The overhead of the initialization is $T_1(n) = O(nmr)$, i.e. $O(n)$..

Assume the population evolves s generations before stop. The overhead in clone and mutation steps including the affinity evaluation is $T_2(n) = O(sk^2n^2)$. Here, k is the number of mutation operation. If each solution in every generation only mutate once which is the best case, the overhead is $T_2(n) = O(sn^2)$ with $k = 1$. In the step of generating new population, the overhead of simulated annealing selection, affinity evaluation and update of immunological memory are all $O(n)$.

In summary, the complexity of the algorithm is $O(sk^2n^2)$ which depends on the amount of cases n, the amount of mutation operation k and the number of generation s. Ideally, the complexity is $O(sn^2)$.

5 Evaluation

The proposed algorithm has been implemented and evaluated in a controlled experiment using logs from two information systems: incident management system and task management system. These two systems support the implementations of processes of accident handling procedure (for short P_A) and task management procedure (for short P_B), which are sub-processes of Incident Lifecycle Management for IT operation and maintenance. The model we used in our experiments is shown in figure 2.



A: Register Acc. B: Dispatch Acc. C: Treat Acc. D: suspend Acc. E: Solve Acc. F: Cancel Acc. G: Confirm Acc. H: Close Acc. I: Create task J: Assign task K: Suspend task L: Execute task M: Complete task N: Transmit task O: Request assist. P: Cancel task Q: Evaluate task R: Close task

Fig. 2. The processing framework of merging logs

In the model, after an accident has been designated to somebody to handle (activity node C in P_A), P_B is triggered with a new task being created (activity node I in P_B). Note that when a task is closed (activity node R in P_B) , P_B provide a feedback to the activity E in P_A to continue the accident processing. Therefore, the temporal relationship between cases of the two processes is Full containment.

Table 2. Information of logs

Group	Logs	#Cases	#Event
G1	log1 vs. log2	1024 vs. 1024	5790 vs. 7942
G2	Log3 vs. log4	2048 vs. 2048	11598 vs. 15847
G3	Log5 vs. log6	4096 vs. 4096	23401 vs. 31903

In our experiments, we design three groups of tests, each group having two logs for merging: log1 vs. log2, log3 vs. log4 and log5 vs. log6. The logs in three groups are different in log characteristics, such as the number of cases and the number of events, as shown in table 2. We compare the Claes's algorithm (AIA) and the proposed algorithm (SA+AIA) in two aspects: merging quality and merging efficiency. We use the success rate of merging to measure merging quality. The success rate of merging is a ratio between number of successful match of cases C_s and all of cases in two logs C_a and C_b , as shown in equation 4.

$$R_s = 2\sum C_s / \left(\sum C_a + \sum C_b\right) \qquad (4)$$

The execution time is a significant indicator of the merging efficiency. Due to the stochastic nature of the algorithms, the experimental results are assessed with average number of successful matched cases (ACMC) and average execution time (AET). The ACMC and AET of each test are averaged over 3 independent runs, in which we use the same logs and the same setting. The tests run on a 6GB RAM 2.2GHz laptop. The parameters of the algorithms are set as follow: the size of random population is 100; the percentage for initial population: 60%; the weight of the factors α_1, α_2 and α_3 are 40, 30, 30; and the constant k for annealing is 0.95.

Table 2. Test results

	ALG	ACMC	Merging success rate	#Generations before stop	AET (secs)
G1	AIA	949	92.68%	10000	92
	SA+AIA	957	93.36%	6367	72
G2	AIA	1883	92.03%	10000	197
	SA+AIA	1907	93.12%	7362	153
G3	AIA	3735	91.12%	10000	409
	SA+AIA	3785	92.41%	7299	328

In general, the results indicate that the proposed algorithm performs better than AIA algorithm with merging success rate of merging of at least 90%, as shown in figure 3. The results in Table 3 show that the proposed algorithm has reduced the number of iterations significantly over the AIA algorithm. The time overhead is reduced by an average of 21% over the AIA algorithm, as shown in figure 4. This means that the proposed algorithm can accelerates convergence to the global optimum. Note that, however, the performance of the algorithm is sensitive to the amount of logs. The success rate has a downward trend and the time overhead increases with the increasing of the amount of logs. It is reasonable to believe the trend will be more obvious with the increasing complexity of process and size of logs. Figure 5 shows the mining result based on the merged log with Alpha method in ProM. The mined model is the same as the model shown in Figure 2.

In summary, the proposed approach has better performance than the exsiting AIA method in merging logs.

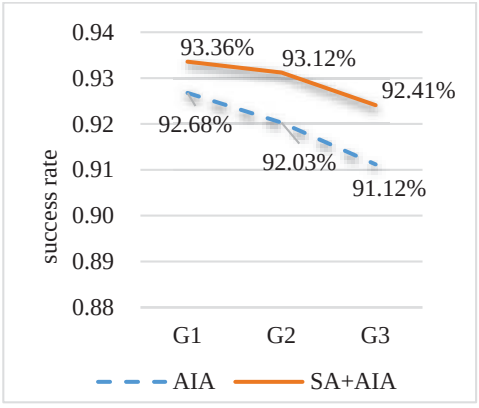


Fig. 3. Test Results of Merging quality

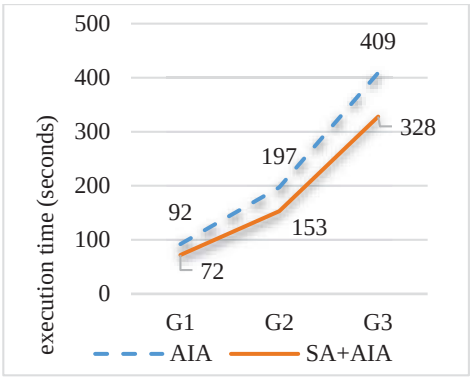


Fig. 4. Test Results of Merging efficiency

6 Conclusions

In this paper we presented a solution to merging log files from different systems into a single file so that the existing process mining techniques can then be used. The proposed algorithm is inspired by artificial immune algorithms and simulated annealing algorithms. An affinity function is the key in every step in this algorithm. The set of factors used in the affinity function, occurrence frequency and temporal relation can express the characteristics of matching cases more accurate than some factors, e.g. case ID used in other solutions. Another unique feature of the proposed algorithm is its ability to deal with local optima due to pre-mature problem of artificial immune algorithms by introducing simulated annealing selection and immunological memory to strength the diversity of population. The proposed algorithm has been implemented as a plugin into the ProM platform. The test result shows a good performance of the algorithm in merging logs.

It has been found that the performance of the proposed approach, especially time overhead, is sensitive to the size of the log files, same as other approaches. In real life business, massive logs are produced every day. It is a big challenging to improve the merging efficiency for massive log data. Distributed or parallel merging approach for massive log will be explored in our future research.

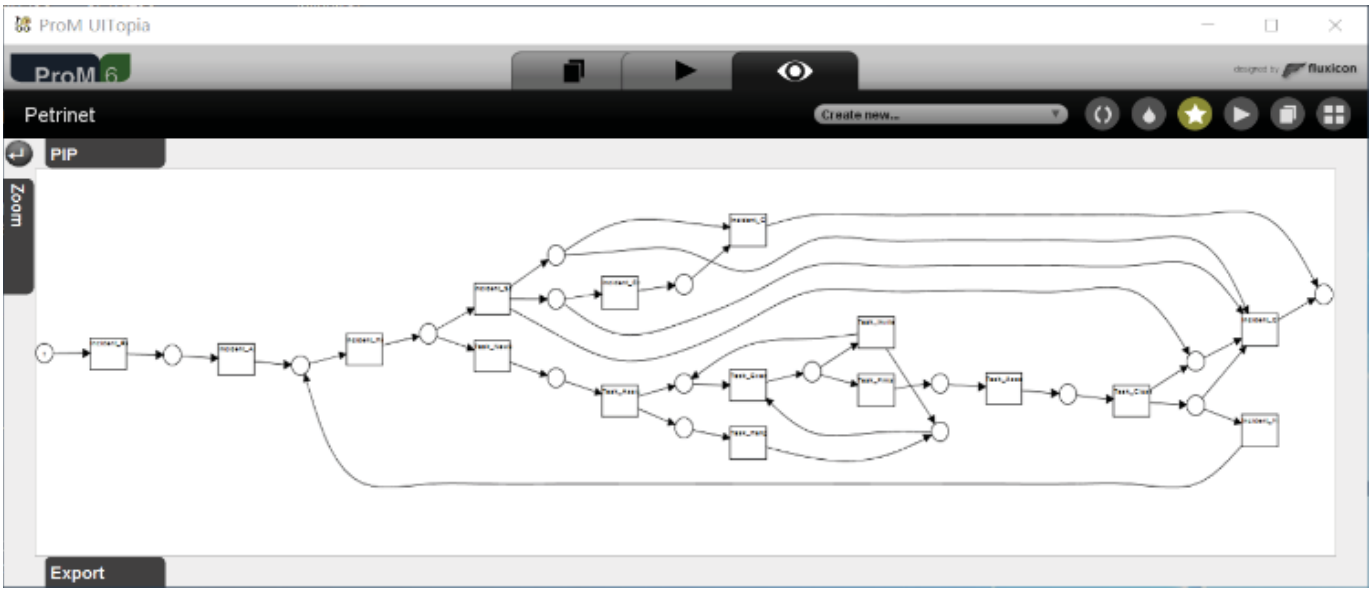


Fig. 5. The Model mined in ProM

Acknoweledge

This work was supported in part by the National Natural Science Foundation of China under Grant 71090403, the Guangdong Provincial S&T Planned Projects under Grant 2014B090901001/2015B010103002 and Special Funds on "985 Project" Disciplinary Construction in School of Software Engineering of South China University of Technology under Grant x2rjD615015III.

References

[1] W. M. P. van der Aalst, "Process Mining: Discovery, Conformance and Enhancement of Business Processes". Springer Berlin Heidelberg, 2011.

[2] A. J. M. M. Weijters , J. T. S. Ribeiro. "Flexible Heuristics Miner". Proccedings. of 2011 IEEE Symposium on Computational Intelligence and Data Mining, Paris, pp. 310—317, April 2011.

[3] A. K. A. de Medeiros, A. J. M. M. Weijters, W. M. P. van der Aalst. "Genetic process mining: an experimental evaluation". Data Mining and Knowledge Discovery, vol. 14, issue 2, pp. 245—304, April 2007.

[4] B. F. van Dongen, A. Adriansyah. "Process Mining: Fuzzy Clustering and Performance Visualization". Business Process Management Workshops, Springer Berlin Heidelberg, pp.158—169, September 2010.

[5] W. M. P. van der Aalst, A. Adriansyah, B. F. van Dongen, "Replaying history on process models for conformance checking and performance analysis", Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 2, issue 2, pp.182—192 , April 2012

[6] J. Claes, G. Poels, "Merging Computer Log Files for Process Mining: an Artificial Immune System Technique", Procceding of the 9th International Conference on Business Process Management Workshops, France, pp. 99—110, August 2011.

[7] H. M. W. Verbeek, Joos C. A. M. Buijs, Boudewijn F. van Dongen, W. M. P. van der Aalst. "XES, XESame, and ProM 6". Information Systems Evolution, Springer Berlin Heidelberg, pp. 60—75, June 2010

[8] H.R. Motahari-Nezhad, R. Saint-Paul, F. Casati, B. Benatallah, "Event correlation for process discovery from web service interaction logs ". The International Journal on Very Large Data Bases, Springer New York, vol.20, no.3, pp. 417—444, June 2011

[9] J. Claes, G. Poels, "Merging Event Logs for Process Mining: A Rule Based Merging Method and Rule Suggestion Algorithm". Expert Systems with Applications, vol.41, issue.16, pp.7291—7306, November 2014.

[10] L. Raichelson, P. Soffer. "Merging Event Logs with Many to Many Relationships". Business Process Management Workshops, the series Lecture Notes in Business Information Processing, Springer International Publishing, vol. 202, pp.330—341, April 2015.

[11] E. K. Burke, G. Kendall, "Search Methodologies-Introductory Tutorials in Optimization and Decision Support Techniques (2nd Ed) ". Springer New York, 2014.