Research Article

Set-Based Differential Evolution Algorithm Based on Guided Local Exploration for Automated Process Discovery

Si-Yuan Jing ^{1,2}

¹School of Computer Science, Leshan Normal University, Leshan 614000, China

²Key Lab of Internet Natural Language Processing of Sichuan Provincial Education Department, Leshan Normal University, Leshan 614000, China

Correspondence should be addressed to Si-Yuan Jing; siyuan-jing@hotmail.com

Received 15 October 2019; Revised 25 December 2019; Accepted 3 February 2020; Published 12 March 2020

Guest Editor: Tomas Veloz

Copyright © 2020 Si-Yuan Jing. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Evolutionary algorithm is an effective way to solve process discovery problem which aims to mine process models from event logs which are consistent with the real business processes. However, current evolutionary algorithms, such as GeneticMiner, ETM, and ProDiGen, converge slowly and in difficulty because all of them employ genetic crossover and mutation which have strong randomness. This paper proposes a hybrid evolutionary algorithm for automated process discovery, which consists of a set-based differential evolution algorithm and guided local exploration. There are three major innovations in this work. First of all, a hybrid evolutionary strategy is proposed, in which a differential evolution algorithm is employed to search the solution space and rapidly approximate the optimal solution firstly, and then a specific local exploration method joins to help the algorithm skip out the local optimum. Secondly, two novel set-based differential evolution operators are proposed, which can efficiently perform differential mutation and crossover on the causal matrix. Thirdly, a fine-grained evaluation technique is designed to assign score to each node in a process model, which is employed to guide the local exploration and improve the efficiency of the algorithm. Experiments were performed on 68 different event logs, including 22 artificial event logs, 44 noisy event logs, and two real event logs. Moreover, the proposed algorithm was compared with three popular algorithms of process discovery. Experimental results show that the proposed algorithm can achieve good performance and its converge speed is fast.

1. Introduction

Process-based information system (PIS), including workflow management system (WfMS), customer relationship management (CRM), enterprise resource planning (ERP), has become the fundamental infrastructure of modern enterprises. PIS can greatly improve the operational efficiency of enterprises. Besides that, it will record the information of business processes, such as name of activities, time of activities happening, life cycle of activities, to form event logs. The XES Standard, published by IEEE in 2016, provides a unified and extensible language to standardize the content and format of event logs. Process mining technique can be used to discover a process model from a XES event log. It is hoped that the mined process model is as consistent as possible with the real business process. The obtained process model can be used to improve business processes of

enterprises, increase production efficiency, and optimize products. For example, ASML employs process discovery technique to optimize the wafer scanner during the production of lithography machines. The ERP system of SAP uses the process discovery technique to assist users design business process, analyze the bottleneck of business, and plan resource. Philips collects the event logs from their medical devices around the world to analyze customers' habits. By this way, they can optimize their medical products and shorten the time of product development.

Generally, there are three major tasks in process mining, including process discovery, conformance checking, and process enhancement [1]. Process discovery aims to obtain a process model, which is as consistent as possible with the real business process. Most studies are focused on mining binary relations of any two activities in event logs from the perspective of control flow. Beyond that, it can also discover

the knowledge contained in the event log from other perspectives, such as organization, time, resource, and so on. Conformance checking is used to measure the deviation of a mined process from a real business process by replaying the event log on the mined process model. This technique can be used for the diagnosis of the process model as well as the analysis of business bottleneck. Process enhancement focuses on changing or extending a prior process model. For example, by using time-stamp in an event log, the model can be enhanced to analyze bottleneck, estimate remaining time, and discover hierarchical process model. In this paper, I only focus on the process discovery technique from the perspective of control flow.

The α -algorithm, proposed by van der Aalst et al., is usually regarded as a milestone in the field of process mining [2]. It models the workflow by Petri net and can effectively find the causal relation, parallel relation, and choice relation between any two activities from the event log. After that, some variants of the α -algorithm were proposed, just like $\alpha+$ algorithm [3] and $\alpha++$ algorithm [4]. However, there are many shortcomings in the α -series algorithms, such as ability to resist noise, alignment-based fitness, and precision. To address these problems, some more efficient algorithms were proposed, such as ILP Miner [5] and inductive miner [6, 7]. The former was proposed recently by van Zelst et al., which is based on integer linear programming. The latter was proposed by Leemans et al. Both of them show good performance when dealt with small event logs.

Evolutionary algorithm is an effective way to solve the problem of process mining. de Medeiros et al. [8] firstly employed the genetic algorithm (GA) in process mining, named as GeneticMiner. By defining good fitness function as well as genetic operators (i.e., crossover and mutation), the GeneticMiner can find a process model, which is consistent with the real process. Cheng et al. [9, 10] indicated that the GeneticMiner cannot effectively discover parallel structures from event logs; therefore, they proposed a hybrid technique, which is based on integration of GeneticMiner, particle swarm optimization, and differential evolution to improve the result of process mining. Vázquez-Barreiros et al. [11] proposed another algorithm, named ProDiGen, which improves the GeneticMiner by introducing hierarchical fitness function to find complete, precise, and minimal structure process models. Buijs et al. [12–14] employed evolutionary tree to present process models and proposed an alignment-based technique to guide the mutation in GA. However, it is unbearable to embed alignment-based local search in mutation operation because it is too time-consuming. In general, the advantage of the genetic algorithm includes good antinoise, and it can deal with a major part of key problems in process mining by a unified framework, such as invisible tasks, non-free-choice structure, and tasks with duplicated name. However, the convergence speed of current GA-based algorithms is too slow because all of them adopt random search.

In this paper, a hybrid evolutionary algorithm for process mining is proposed, named DEMiner. The innovation of our work includes three parts:

- (1) A hybrid evolutionary strategy is proposed in this work. In our method, DEMiner firstly approximates the optimal process model by a set-based DE algorithm; when the prematurity is detected, a guided local exploration method will join in the evolution process to help the algorithm skip out the local optimum.
- (2) Two set-based DE operators, i.e., a set-based mutation operator and a set-based crossover operator, are designed for differential evolution of causal matrix.
- (3) A fine-grained evaluation method is proposed to guide the local exploration by assigning scores to all the nodes in the candidate process models. This method can not only help the population avoid prematurity but also improve the efficiency of the DEMiner.

The rest of this paper is organized as follows. Section 2 introduces some basic knowledge of process mining, such as Petri net and causal matrix, as well as the DE algorithm. Section 3 explains the proposed algorithm in detail. The experiments as well as the analysis of experimental results are given in Section 4. Finally, Section 5 gives conclusions.

2. Preliminaries

2.1. Process Mining. In the problem of process mining, a process model is generally modeled as a place/transition net (abbreviated as P/T Net), which is a variant of classic Petri net. The definition of P/T is given below.

Definition 1 (P/T Net) [2]. A P/T Net is a tuple $N = (P, T, F)$, where P is a finite set of places, T is a finite set of transitions, $P \cap T = \emptyset$, and $F = (P \times T) \cup (T \times P)$ is a finite set of directed arcs.

Let $N = (P, T, F)$ be a P/T Net. Elements of $P \cup T$ are called nodes. A node x is an input node of another node y if $\langle x, y \rangle \in F$. Similarly, a node x is an output node of another node y if $\langle y, x \rangle \in F$. Furthermore, a symbol $\bullet x$ denotes all the input nodes of node x ; that is, $\bullet x = \{y \mid \langle y, x \rangle \in F \wedge y \in P \cup T\}$. Similarly, $x \bullet$ denotes all the output nodes of x . Based on the P/T Net, a formal definition of workflow net (abbreviated as Wf-Net) is given.

Definition 2 (Wf-Net) [2]. Let $N = (P, T, F)$ be a P/T Net, and t is a fresh identifier not in $P \cup T$. N is a Wf-Net if

- (1) P contains an input place i such that $\bullet i = \emptyset$
- (2) P contains an output place o such that $o \bullet = \emptyset$
- (3) $\tilde{N} = (P, T \cup t, F \cup \{\langle t, i \rangle, \langle o, t \rangle\})$ is strongly connected

Figure 1 shows a process model, which is represented by a Wf-Net. The circles denote places, and the squares denote transitions. The transitions in Wf-Net represent the activities (also called tasks) in the real business process. The black dot in the initial place denotes a token. A transition is enabled to be fired if all of the input places contain tokens. The word “fire” means that an activity is ready to be executed. If a

FIGURE 1: An illustrating example of process model.

transition fires, tokens in its input places are removed; meanwhile, tokens are put in its output places. For example, if the transition “A” is fired, the token in place “start” would be removed and the place “P1” and “P2” will get a token, respectively. After that, three transitions “B,” “C,” and “D” would be enabled. Note that “P1” just has one token; in other words, although two transitions (a.k.a. “B” and “C”) are enabled, only one can be fired. Thus, the possible sequences between “A” and “E” include $\{B, D\}$, $\{C, D\}$, $\{D, B\}$, and $\{D, C\}$.

Given a sound Wf-Net $N = (P, T, F)$, we say $\delta \in T^*$ is an event trace and $W \in P(T^*)$ is an event log, which consists of traces. Take the above process model as an example, and it can induce lots of event traces, just like ABDEG, ACDEH, ADBEFCDG, and so on.

The first problem required to be solved in evolution-based process mining is coding of chromosome. Unfortunately, it is hard to directly employ Petri net for evolution. de Medeiros proposed the causal matrix to represent the process model, which has been applied to many evolutionary algorithms of process mining, such as ProDiGen and GeneticMiner. The definition of the causal matrix is given below.

Definition 3 (Causal Matrix) [8]. A causal matrix is a tuple $CM = (A, C, I, O)$, where

- (1) A is a finite set of activities
- (2) $C = A \times A$ is the causality relation
- (3) $I: A \longrightarrow P(P(A))$ is the input condition function
- (4) $O: A \longrightarrow P(P(A))$ is the output condition function

Since we usually need to compare the process model presented by the causal matrix with other models presented by Petri net, a method of mapping the P/T Net to the causal matrix is required. Definition 4 shows a method for converting a P/T Net to a causal matrix.

Definition 4 (Mapping of a P/T Net to a Causal Matrix) [8]. Let $N = (P, T, F)$ be a P/T Net. The mapping of N is a tuple $\Pi_{N \rightarrow CM} = (A, C, I, O)$, where

- (1) $A = T$
- (2) $C = \{(t_1, t_2) \in T \times T \mid t_1 \cdot \cap \cdot t_2 \neq \emptyset\}$

- (3) $I: T \longrightarrow P(P(T))$ such that $\forall_{t \in T} I_{\Pi}(t) = \{p \mid p \in \cdot t\}$
- (4) $O: T \longrightarrow P(P(T))$ such that $\forall_{t \in T} O_{\Pi}(t) = \{p \mid p \in t \cdot\}$

To explain the mapping process, we convert the process model in Figure 1 to a causal matrix, which is shown in Table 1. Take activity “E” as an example, and it has two input places “P3” and “P4.” Furthermore, the input transitions of “P3” are transitions “B” and “C,” and the input transition of “P4” is transition “D,” and thus, $I_{\Pi}(E) = \{\{B, C\}, \{D\}\}$. It should be noted that those activities in the same subset of $I_{\Pi}(t)$ have an OR-join relation, and those different subsets in $I_{\Pi}(t)$ have an AND-join relation. On the contrary, activities in the same subset of $O_{\Pi}(t)$ have an OR-split relation and those different subsets have an AND-split relation. Besides, $I_{\Pi}(t) = \{\}$ demonstrates that the input of the activity is empty and $O_{\Pi}(t) = \{\}$ demonstrates the output of the activity is empty.

2.2. Differential Evolution Algorithm. Differential evolution algorithm (DE), firstly proposed by Das et al. in 1995, is a stochastic method simulating biological evolution, in which the individuals adapted to the environment are preserved through repeated iterations [15]. Compared to other evolutionary algorithms, the DE algorithm has some advantages, such as better global searching ability, fast convergence speed, and strong robustness.

The major steps of DE include mutation, crossover, evaluation, and selection, which is similar with GA. DE starts with a population, which contains N randomly generated individuals (also known as chromosomes). The individual is represented by a vector $\vec{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, \dots, x_{D,i,G}]$, where i denotes the i -th individual, G denotes current generation, and D denotes the dimension of the vector. In the step of mutation, DE generates a *donor* vector $\vec{V}_{i,G} = [v_{1,i,G}, v_{2,i,G}, \dots, v_{D,i,G}]$ for the i -th individual (called *target* vector). It firstly chooses three distinct vectors $\vec{X}_{r_{1,i,G}}$, $\vec{X}_{r_{2,i,G}}$, and $\vec{X}_{r_{3,i,G}}$ from the population. The indices $r_{1,i}$, $r_{2,i}$, and $r_{3,i}$ are mutually exclusive integers randomly chosen from the range $[1, N]$. Then, the difference of any two of these three vectors (i.e., $\vec{X}_{r_{2,i,G}}$ and $\vec{X}_{r_{3,i,G}}$) is scaled by a scalar factor F , and then the scaled difference is added to the last vector

TABLE 1: Causal matrix of the process model.

Activity	$I(\text{Activity})$	$O(\text{Activity})$
A	$\{\}$	$\{\{B, C\}, \{D\}\}$
B	$\{\{A, F\}\}$	$\{\{E\}\}$
C	$\{\{A, F\}\}$	$\{\{E\}\}$
D	$\{\{A, F\}\}$	$\{\{E\}\}$
E	$\{\{B, C\}, \{D\}\}$	$\{\{F, G, H\}\}$
F	$\{\{E\}\}$	$\{\{B, C\}, \{D\}\}$
G	$\{\{E\}\}$	$\{\}$
H	$\{\{E\}\}$	$\{\}$

$\vec{X}_{r_{1,j},G}$; it finally obtains the *donor* vector. Such process can be expressed by the following formula:

$$\vec{V}_{i,G} = \vec{X}_{r_{1,j},G} + F \cdot (\vec{X}_{r_{2,j},G} - \vec{X}_{r_{3,j},G}). \quad (1)$$

To enhance the potential diversity of the population, a crossover operation comes into play after generating the donor vector. The donor vector exchanges its components with the target vector $\vec{X}_{i,G}$ under this operation to form the *trial* vector $\vec{U}_{i,G} = [u_{1,i,G}, u_{2,i,G}, \dots, u_{D,i,G}]$. There are two popular ways for crossover in DE, which are exponential crossover (or two-point modulo) and binomial crossover (or uniform). This paper just introduces the latter, which is given in formula (2), where Cr is the crossover rate and j_{rand} is a random number. The condition $j = j_{\text{rand}}$ guarantees that at least one element of the donor vector will be selected. The obtained trial vector will be evaluated by a predefined fitness function. If the fitness of the trial vector is higher than the fitness of the target vector, the DE algorithm would replace the target vector by the trial vector; otherwise, it keeps the target vector:

$$u_{j,i,G} = \begin{cases} v_{j,i,G}, & \text{if } (\text{rand}_{i,j}[0, 1] \leq Cr \vee j = j_{\text{rand}}), \\ x_{j,i,G}, & \text{otherwise.} \end{cases} \quad (2)$$

3. DEMiner: A Hybrid Evolutionary Algorithm for Process Discovery

3.1. Framework of DEMiner. The GA-based process mining algorithms, including GeneticMiner [8], ProDiGen [11], and ETM [12], suffer from the problem that all of them need hundreds or even thousands of generations to converge to a solution. The reason behind this problem is that the genetic operators follow a completely random way, without taking advantage of the information of the log and the errors of the mined model during the parsing of the traces. The ProDiGen solves this problem in a simple way that it selects an incorrectly parsed activities as the crossing point in the step of crossover. However, it cannot significantly improve the speed of convergence because the process of crossover is also random. The ETM employed the alignment-based technique for local exploration to accelerate the convergence. However, the total running time is also unacceptable because the alignment algorithm is too time-consuming.

In this section, I will introduce a hybrid evolutionary algorithm for automated process discovery, called DEMiner. The main steps of DEMiner are shown in Figure 2. It can be

seen from the figure that the major difference between DEMiner and the traditional evolutionary algorithm is that it needs to select a specific evolutionary strategy in the loop. Step V is a set-based DE algorithm (abbreviated as DE or DE algorithm), which is in charge of fast approximation of the optimal solution. However, the DE algorithm usually falls into local optimum. To overcome the premature convergence, I employ Step VI, which is a guided local exploration algorithm. The local exploration algorithm can take advantage of the error information during the parsing of the log and help DEMiner quickly skip out the local optimum.

The pseudocode of DEMiner is given in Algorithm 1. The algorithm skips out the loop when the number of generation is higher than a predefined threshold *maxGenerations* or the *timesNotChange* is higher than *maxNotChange*. The variable *timesNotChange* records how long the population has not been replaced. In the loop, two statistics, i.e., *meanFitness* and *devFitness*, are used to detect whether the algorithm appears premature convergence. The former is the mean fitness of the population, and the latter is the deviation of the fitness value. If *meanFitness* and *devFitness* are lower than predefined thresholds *MF* and *DF*, at the same time, I think the algorithm is premature. Besides two statistics, a random number *rand* is used in the condition. The reason behind this consideration is that the global searching ability of the local exploration algorithm is lower than the DE algorithm. Sometimes, the local exploration may make an individual move forward, but it may not cause significant changes in two statistics. Therefore, the proposed algorithm will randomly choose a strategy if it falls into local optimum. Next, I will introduce these steps in detail.

3.2. Population Initialization. The population initialization method used in DEMiner follows the heuristic method proposed in [8], which is based on the causal relation between activities. Except that, there are two changes in our method, called gene bank and taboo list, which can improve the performance of DEMiner.

Gene bank is a set of chromosomes (i.e., individuals), including the individuals that are in current population and the individuals that have been eliminated during the evolution. To reduce the cost of memory space, the individuals in gene bank will be serialized; in other words, they will be converted to a simple format. For example, $I_{\text{ind}}(E) = \{\{B, C\}, \{D\}\}$ will be converted to a string " $I(E) = [\{B, C\}, \{D\}]$." If the algorithm generates an individual which has been in gene bank, the individual would be discarded without calculation of its fitness value.

Taboo list keeps a set of historical operations of local exploration. In DEMiner, an important step, called guided local exploration, is employed to search around a specified node. The local exploration requires to randomly select one of the three operations, which are adding an arc, deleting an arc, or redistributing a node. These operations that have been performed, no matter they are useful or useless, are forbidden to be selected again. Every node has a taboo list, and these lists will be initialized to empty at the beginning.

FIGURE 2: Framework of the DEMiner.

```

(1) Initialize population
(2) Evaluate population
(3) Calculate meanFitness and devFitness of the population
(4) generation  $\leftarrow 1$ , timesNotChange  $\leftarrow 0$ 
(5) while generation  $\leq$  maxGenerations && timesNotChange  $\leq$  maxNotChange do
(6)   if meanFitness  $\geq$  MF && devFitness  $\leq$  DF && rand  $\leq$  R do
(7)     Generate the trial individuals by the guided local exploration
(8)   else
(9)     Generate the trial individuals by the DE algorithm
(10)  Evaluate the trial individuals
(11)  if the fitness of the trial individuals is higher that the fitness of the target targets do
(12)    Replace population
(13)    timesNotChange  $\leftarrow 0$ 
(14)  else
(15)    timesNotChange++
(16)  Update meanFitness and devFitness
(17)  generation++

```

ALGORITHM 1: Pseudocode of DEMiner.

3.3. Fitness Function. Generally, two metrics should be considered when evaluating a process model, which are completeness and precision [16]. The completeness quantifies the ability of a discovered process model that it can accurately parse the traces recorded in the event log. A natural way to define a completeness metric is the number of correctly parsed traces divided by the total number of traces. However, such definition is too coarse because it cannot indicate how many parts of an individual are correct when the individual does not properly parse a trace. Consider two process models: one is a totally incorrect process model, and the other just misses an arc; the above method cannot distinguish the two individuals because both of them cannot correctly parse the log. Due to this, I employ the partial completeness given in [8], which takes into account the correctly parsed activities as well as the number of tokens, which are missing or not consumed during the parse. I use the symbol “ C_f ” to denote the completeness metric.

A discovered process model may not be appropriate even if it gets completeness. For example, a flower model can parse arbitrary event logs, but it is useless. Precision is used to quantify the fraction of the behavior allowed by the model, which is not seen in the event log. However, it is hard to give a proper definition of precision because it has to detect all the extra behavior, i.e., possible path in the model but not in the log. In [4], the definition of the precision is $\text{allEnabledActivities}(L, CM)$ divided by $\max(\text{allEnabledActivities}(L, CM))$, where $\text{allEnabledActivities}(L, CM)$ is

the number of enabled activities when a log L is parsed by an model CM . The denominator is a function which returns the max number of enabled activities in the population. It is easy to find that the precision of each individual depends on the rest of the population. In this work, I consider another definition of precision proposed in [11] (see formula (3)). It is easy to find that the more the activities enabled in a process model is, the lower the precision of the process model is:

$$P_f = \frac{1}{\text{all Enabled Activities}(L, CM)}. \quad (3)$$

Generally, I need to assign weighted coefficients to combine the two metrics in a weighted sum [17]. However, it is difficult to combine the two metrics in an appropriate way because the used precision is not normalized. Therefore, a hierarchical method is employed to define the fitness function in this work. Because the completeness is more important than the precision when evaluating a discovered process model, I firstly compare the completeness of two process models; if their completeness is equal, then I compare their precision. By this way, when the completeness of all individuals are equal to 1, the individual that has better precision would win. It is easy to notice that the hierarchical fitness function can be easily extended by other metrics, such as structure complexity and generalization.

3.4. Differential Evolution Algorithm. The DE algorithm contains a loop which goes through all individuals in the population. For each individual (called the *target* individual), it firstly generates a *donor* individual based on three randomly selected individuals (mutation operation), and then it combines the *target* individual and the *donor* individual to get a *trial* individual (crossover operation). It must be emphasized that, due to the obtained *donor/trial* individual may be inconsistent, both of them should be repaired before going to next step. Then, the *trial* individual will be evaluated if it is not in the gene bank. If the fitness value of the *trial* individual is higher than the fitness of the *target* individual, the *target* individual would be replaced by the *trial* individual; meanwhile, the *trial* individual would be added into the gene bank. It can be seen that there are three key steps in the DE algorithm, which are mutation, crossover, and repair. Next, the details of the three steps will be explained.

3.4.1. Mutation. Current set-based evolutionary algorithms usually employ crisp sets to represent the candidate solutions (which are called individuals or chromosomes in GA). For example, Chen et al. [18] proposed a set-based particle swarm optimization algorithm in which the candidate solutions are represented by a set of ordinal pairs. However, the causal matrix is a type of much more complex set. The elements in $I(\text{Activity})$ and $O(\text{Activity})$ are crisp sets, such as $I_{\text{ind}}(E) = \{\{B, C\}, \{D\}\}$. Therefore, the traditional set-based mutation operators cannot be directly used in this work. In Ou-Yang's method [9, 10], the mutation operator randomly selects ingredients from three individuals, and then use them to update the target individual to obtain a donor individual. The advantage of the method is that some good ingredients (e.g., parallel structure) can be directly transplanted to the target individual, which can improve the searching ability of the GeneticMiner. However, Ou-Yang's method cannot be directly employed in this work because the proposed algorithm is entirely based on the DE algorithm. In other words, it needs more flexible mutation operators.

This section introduces two novel operators, which allow the proposed algorithm to perform differential mutation on the causal matrix. The definitions of the two operators are given below.

Definition 5 (Minus Operator between Two Sets). Given a causal matrix $\text{CM} = (A, C, I, O)$ and two sets $S_1, S_2 \in P(P(A))$. The relative complement $S_1 - S_2$ of S_2 in S_1 is defined as

$$S_1 - S_2 = \{e \mid e \in S_1 \wedge e \notin S_2\}. \quad (4)$$

Definition 6 (Plus Operator between Two Sets). Given a causal matrix $\text{CM} = (A, C, I, O)$ and two sets $S_1, S_2 \in P(P(A))$. Then, $S_1 + S_2$ is defined as

$$S_1 + S_2 = \left\{e \mid e \in \{e' / \cup S_2 \mid e' \in S_1\} \vee e \in S_2\right\}, \quad (5)$$

where $\cup$ denotes a generalized union operation and $e' / \cup S_2$ means that it removes the elements in $\cup S_2$ from e' . It is

easy to find that the plus operator will keep the elements in S_2 . The reason behind this consideration is that S_2 in formula (4) is, in fact, the result from formula (3) (a.k.a. the difference of two sets). By this way, it can greatly change the structure of S_1 and enhance the potential diversity of a trial individual.

Figure 3 gives an illustrating example. Given three sets which represent three distinct input sets of activity D , $S_1 = \{\{A, B, C\}\}$, $S_2 = \{\{A, B\}, \{B, C\}\}$, and $S_3 = \{\{A, C\}, \{B, C\}\}$. In terms of the definition of mutation in DE (see formula (1)), it requires to firstly calculate the difference of S_2 and S_3 and then add the difference to S_1 . Based on Definitions 5 and 6, we can get $S_2 - S_3 = \{\{A, B\}\}$ and then $\text{Donor} = S_1 + \{\{A, B\}\} = \{\{A, B\}, \{C\}\}$. Note that the scale factor F is not used in the mutation.

3.4.2. Crossover. The aim of crossover is combining a donor individual and a target individual to generate a trial individual. The trial individual would take the place of the target individual if its fitness is higher than the target individual. There are two kinds of popular crossover methods, which are the exponential and the binomial. I employ the latter in this work. The pseudocode of the binomial operator is shown below. "Cr" is called the crossover rate. The binomial crossover is performed on each of the activity node whenever a randomly generated number "rand" between 0 and 1 is less than or equal to the "Cr" (Algorithm 2). The "r" is a randomly chosen index, which ensures that the trial individual can get at least one component from the donor individual.

3.4.3. Repair. As is known to all, individuals obtained in the iterative process of evolutionary algorithms are always inconsistent. For example, it is possible to get a trial individual in which $I_{\text{ind}}(E) = \{\{B, C\}, \{D\}\}$, but $O_{\text{ind}}(D)$ does not contain activity "E." Besides that, the input of the "start" activity as well as the output of the "end" activity may be not empty. Therefore, a repair operation is required to be performed on the donor individual as well as the trial individual. In the GA-based process mining algorithms, such as GeneticMiner and ProDiGen, the repair operation is simple because the crossover as well as the mutation are performed on a designated point. The repair operation in this work is much more complex because the mutation as well as the crossover operation are performed on all nodes in a causal net. In other words, it is required to repair all nodes in a causal matrix. Before introducing the algorithm of repair, I firstly give the definition of consistence for a causal matrix.

Definition 7 (Consistence of a Causal Matrix). Let $\text{CM} = (A, C, I, O)$ be a causal matrix, and we say it is consistent if

- (1) $I(\text{start}) = \emptyset \wedge O(\text{start}) \neq \emptyset$, where "start" is the beginning activity
- (2) $O(\text{end}) = \emptyset \wedge I(\text{end}) \neq \emptyset$, where "end" is the ending activity

FIGURE 3: Illustrating example of the differential mutation operators.

```

(1) Add the activities A into the trial individual
(2) Generate a random number  $r$  between 1 and  $length(A)$ 
(3) foreach  $t$  in A do
(4)   if  $rand < Cr \parallel i = r$  do
(5)      $I_{trial}(t) \leftarrow I_{donor}(t)$ 
(6)      $O_{trial}(t) \leftarrow O_{donor}(t)$ 
(7)   else
(8)      $I_{trial}(t) \leftarrow I_{target}(t)$ 
(9)      $O_{trial}(t) \leftarrow O_{target}(t)$ 

```

ALGORITHM 2: Pseudocode of the binomial crossover.

- (3) For $\forall t' \in \cup I(t)$, $t \in \cup O(t')$, where $t, t' \in A \wedge t \neq start \wedge t' \neq end$
- (4) For $\forall t' \in \cup O(t)$, $t \in \cup I(t')$, where $t, t' \in A \wedge t \neq end \wedge t' \neq start$

The pseudocode of repair is given in Algorithm 3. Steps 1–6 are in charge of repairing the “start” node as well as the “end” node. It firstly lets the $I_{ind}(start)$ and $O_{ind}(end)$ be empty. Then, it would generate a new input set (output set) for $O_{ind}(start)$ ($I_{ind}(end)$) if they are empty. From Steps 7 to 19, the algorithm goes through all nodes in the causal matrix and repair $I_{ind}(t_1)$ and $O_{ind}(t_1)$, respectively. There are two choices in the repair operation. Take the repair of the input of t_1 for an example; if the output of t_2 does not contain t_1 , it may randomly add t_1 to $O_{ind}(t_2)$ or remove t_2 from $I_{ind}(t_1)$. By this way, we will finally obtain a consistent individual.

3.5. Guided Local Exploration. Although evolutionary algorithms, including the GA algorithm and the DE algorithm, have strong ability of global search, all of them suffer from the problem of premature convergence. In [14], van Eck et al. proposed a local exploration method based on alignment. In the method, a A^* algorithm is employed to find the optimal alignment between a process model and an event trace.

By this way, it can find out the abnormal areas in the process model. However, the alignment-based local exploration has two drawbacks. Firstly, it can only locate the abnormal area but not a node, which is too coarse to guide the exploration. Secondly, although the technique can accelerate the convergence of the GA algorithm, the execution time of the A^* algorithm is so long, which makes the total execution time unbearable.

This paper proposes an efficient and simple method to guide local exploration, which can help DEMiner skip out

```

(1)  $I_{\text{ind}}(\text{start}) \leftarrow \emptyset$ 
(2)  $O_{\text{ind}}(\text{end}) \leftarrow \emptyset$ 
(3) if  $O_{\text{ind}}(\text{start}) = \emptyset$  do
(4)   Generate an output set for  $O_{\text{ind}}(\text{start})$ 
(5) if  $I_{\text{ind}}(\text{end}) = \emptyset$  do
(6)   Generate an input set for  $I_{\text{ind}}(\text{end})$ 
(7) foreach  $t_1$  in  $A$  do
(8)   foreach  $t_2$  in  $\cup I_{\text{ind}}(t_1)$  do
(9)     if  $\cup O_{\text{ind}}(t_2)$  does not contain  $t_1$  do
(10)      if  $\text{rand} < 1/2 \parallel \text{length}(\cup I_{\text{ind}}(t_1)) \leq 1$  do
(11)        Add  $t_1$  to  $O_{\text{ind}}(t_2)$ 
(12)      else
(13)        Remove  $t_2$  from  $I_{\text{ind}}(t_1)$ 
(14)      foreach  $t_2$  in  $\cup O_{\text{ind}}(t_1)$  do
(15)        if  $\cup I_{\text{ind}}(t_2)$  does not contain  $t_1$  do
(16)          if  $\text{rand} < 1/2 \parallel \text{length}(\cup O_{\text{ind}}(t_1)) \leq 1$  do
(17)            Add  $t_1$  to  $I_{\text{ind}}(t_2)$ 
(18)          else
(19)            Remove  $t_2$  from  $O_{\text{ind}}(t_1)$ 

```

ALGORITHM 3: Pseudocode of repair of a candidate individual.

the local optimum and move forward to the global optimum. The method is based on token-based log replay, which is also employed in evaluation of the process model (a.k.a. causal matrix) in this work. The original algorithm for parsing a log on a causal matrix only records three types of information, which are “*allParsedActivities*,” “*allMissingTokens*,” and “*allExtraTokensLeftBehind*.” The *allParsedActivities* denotes the total number of activities which are correctly parsed, the *allMissingTokens* denotes the number of missing tokens in all event traces, and the *allExtraTokensLeftBehind* denotes the number of tokens that are not consumed after the parsing. In our method, besides those, it requires to record the nodes that where the parsing errors happen, including miss tokens during the parsing or left behind tokens after the parsing. By this way, it can achieve fine-grained evaluation of all nodes.

It is easy to find that the proposed method has several advantages. Firstly, it can accurately locate the abnormal nodes and improve the efficiency of the local exploration. Secondly, the time complexity of the proposed method is much lower than the alignment-based method because it does not need extracomputation. The evaluation of nodes can be finished along with the evaluation of the individual. The formulas of fine-grained evaluation are given below, in which $C_f(t)$ represents the score of $I_{\text{ind}}(t)$ and $C_o(t)$ represents the score of $O_{\text{ind}}(t)$:

$$\begin{aligned}
C_f(t) &= \frac{\text{all Missing Tokens}(L, CM, t)}{\text{all Parsed Activities}(L, CM, t)} \\
C_o(t) &= \frac{\text{all Extra Tokens Left Behind}(L, CM, t)}{\text{all Parsed Activities}(L, CM, t)}.
\end{aligned} \tag{6}$$

The pseudocode of the guided local exploration is shown in algorithm 4. Step 2 employs a roulette wheel strategy to randomly select a node for local exploration. The node with a lower score has great probability to be

selected. Step 3 randomly selects a direction for exploration, i.e., “input” or “output.” Steps 4–28 randomly choose a mutation operation, including randomly add an arc to the node, randomly delete an arc from the node, and randomly redistribute the structure of the node. An example for illustrating the redistribution operation is that given $I_{\text{ind}}(E) = \{\{B, C\}, \{D\}\}$, it may get $I_{\text{ind}}(E) = \{\{B, C, D\}\}$ or $I_{\text{ind}}(E) = \{\{B\}, \{C, D\}\}$ after the redistribution. In the algorithm, a *taboo list* is used to record the history of local exploration. Some operations which are useless (a.k.a. cannot make the individual move forward) are recorded in the *taboo list*. By this way, it can improve the efficiency of the local exploration.

4. Experiments

In this section, I give the experiments as well as the analysis. The experiments are focused on two aspects. The first is to evaluate that whether the DE algorithm and the guided local exploration are efficient to accelerate the convergence speed of the proposed algorithm. The second is to evaluate the performance of the proposed algorithm (a.k.a. the DEMiner). Next, the event logs used in the experiments will be introduced.

4.1. Event Logs. In the experiments, 66 event logs were used for evaluation of the proposed algorithm. The event logs can be classified into three groups. The first group contains 22 artificial event logs which are from [8, 19] and can be downloaded from <https://svn.win.tue.nl/repos/prom/DataSets/GeneticMinerLogs/>. The description of the event logs is shown in Table 2. The process models that generate these logs include different structures, such as sequence, choice, parallelism, loops, and invisible tasks. These process models, represented as Petri nets and heuristic nets, can be found in [19].


```

(1) foreach ind in population do
(2)   selected  $\leftarrow$  randomly select an activity from  $ind$ 
(3)   mutationType  $\leftarrow$  randomly select a direction for exploration
(4)   if rand < 1/3 do // randomly add an arc
(5)     if mutationType = "input" do
(6)       precursor  $\leftarrow$  select a precursor of selected activity which is not in  $I_{ind}$ (selected)
(7)       arc  $\leftarrow$  <precursor, selected>
(8)     else
(9)       successor  $\leftarrow$  select a successor of selected activity which is not in  $O_{ind}$ (selected)
(10)      arc  $\leftarrow$  <selected, successor>
(11)     if add arc is allowed in taboo list do
(12)       Add arc into ind
(13)     else go to 5
(14)   else if rand < 2/3 do // randomly delete an arc
(15)     if mutationType = "input" do
(16)       precursor  $\leftarrow$  select an activity from  $I_{ind}$ (selected)
(17)       arc  $\leftarrow$  <precursor, selected>
(18)     else
(19)       successor  $\leftarrow$  select an activity from  $O_{ind}$ (selected)
(20)       arc  $\leftarrow$  <selected, successor>
(21)     if delete arc is allowed in taboo list do
(22)       Delete arc from ind
(23)     else go to 15
(24)   else // redistribute the node
(25)     if mutationType = "input" do
(26)       Redistribute  $I_{ind}$ (selected)
(27)     else
(28)       Redistribute  $O_{ind}$ (selected)

```

ALGORITHM 4: Pseudocode of the guided local exploration.

In the event logs, the traces with same event sequence are grouped together.

The second group, which is used to evaluate the antinoise ability of the DEMiner, contains 44 event logs. These event logs were generated based on the first group of event logs, which contain 5% and 10% noise, respectively. Three different types of operations for noise generation were used, including randomly add an event into a trace, randomly delete an event from a trace, and randomly swap two adjacent events in a trace. To incorporate noise, the traces of the original noise-free logs were randomly selected and then one of the three noise types was applied and each one with an equal probability of 1/3.

The third group includes two real event logs, both of which are downloaded from https://data.4tu.nl/repository/collection:event_logs. The first event log which was named "BPI2013cp" records the process information from the Volvo IT problem management system. It includes 1487 traces as well as 6660 events. The second event log which was named "Sepsis" records the events of sepsis cases from a hospital ERP system. It includes 1050 traces as well as 15214 events.

4.2. Convergence Speed and Running Time. This section introduces the experiments for evaluation of the efficiency of the DE algorithm as well as the guided local exploration method.

This section evaluates whether the DE algorithm and the guided local exploration are efficient to accelerate the convergence speed of the proposed algorithm. Four different strategies are considered in the experiment, which are the DE algorithm without local exploration (denoted as DE), the DE algorithm with random local exploration (denoted as DE + Random Search), the DE algorithm with guided local exploration (denoted as DE + Guided Search), and the GA. It should be explained that (1) the random search used in the second strategy is the genetic mutation in the GeneticMiner, and (2) the third strategy is the DEMiner proposed in this work, and (3) the GA algorithm follows the framework in this work but uses the genetic operators of the GeneticMiner. Three metrics are employed in the experiments, which are completeness, precision, and generation (i.e., number of iterations). To avoid the inaccuracy of the experimental results caused by randomness, each algorithm was run 10 times and the average value of each metric as well as its standard deviation is calculated.

The first group of event logs was used for the evaluation. The computer for experiments is equipped with a 2.5 GHz CPU and 8 GB memory. The parameters setting is shown in Table 3. It should be explained that the population size is the number of activities multiplied by 1~2. The parameter "MF" is set to 0.7 because the local exploration is not hoped to be involved in the search too early. The parameter "DF" is set to 0.2 which is used for detection of premature convergence. In fact, a slight change of these parameters,

TABLE 2: Description of event logs.

Model	# tasks	# traces	# events	Sequence	Choice	Parallelism	Length one loop	Length two loop	Structured loop	Arbitrary loop	Invisible tasks
a5	7	300	2189	✓	✓	✓	✓				
a6nfc	8	300	2040	✓	✓	✓					✓
a7	9	300	2032	✓	✓	✓					
a8	10	300	1803	✓	✓	✓					
a10skip	12	300	2665	✓	✓	✓					✓
al1	9	300	3976	✓	✓	✓				✓	
al2	13	300	5800	✓	✓	✓				✓	
bn1	42	300	11100	✓	✓						
bn2	42	300	24540	✓	✓				✓		✓
bn3	42	300	35527	✓	✓				✓		✓
choice	12	300	2400	✓	✓						
h3p4	12	300	5637	✓	✓	✓			✓		
h6p18	7	300	9844	✓	✓		✓	✓	✓		✓
h6p30	19	300	14851	✓	✓						✓
h6p36	12	300	3000	✓	✓						
h6p41	16	300	3600	✓	✓	✓					
h6p45	8	300	2400	✓		✓					
l1l	6	300	1987	✓	✓		✓				
l2l	6	300	4932	✓	✓			✓			
l2lOpt	6	300	2622	✓	✓			✓			
l2lSkip	6	300	4554	✓	✓			✓			✓
para15	10	300	3000	✓		✓					

TABLE 3: Parameter settings.

Parameter	Setting
Population size	1~2
maxGenerations	400
maxNotChange	20~40
MF	0.7
DF	0.2
R	0.5
Cr	0.6

e.g., the MF is set to 0.6~0.8 and the DF is set to 0.1~0.2, would not affect the performance of the algorithm, including the quality of mining results as well as the convergence speed.

The experimental results are shown in Table 4. For completeness, it can be seen that the four algorithms (from left to right in Table 3) always achieve completeness 1.00 on 1 event logs, 10 event logs, 20 event logs, and 12 event logs, respectively. The result demonstrates that the “pure” DE algorithm cannot achieve the best model; a.k.a., it always falls into local optimum. Then, for precision, the “DE” algorithm achieves the best precision on just one event log. Except that, the “DE + Random Search” and the “GA” achieve the best precision on 2 event logs, and the “DE + Guided Search” achieves the best precision on 14 event logs stably. From the two metrics, it can be seen that the “DE + Guided Search” plays much better than other three strategies. For generation, it requires to exclude the “DE” algorithm because it always suffers from premature convergence. Among the remaining algorithms, it is obvious that the “DE + Guided Search” has the fastest convergence speed and the “GA” has the slowest convergence.

To illustrate the time complexity of the DEMiner, the running time of “DE + Guided Search” is also recorded, which is shown in Figure 4. From the figure, it can be seen that the minimum running time is about 3 seconds (“a6nfc”) and the maximum running time is about 80 seconds (“bn3”). This proves the time performance of the DEMiner.

Based on the above results, some conclusions can be drawn. (1) The “DE” algorithm always falls into local optimum. (2) The “DE + Random Search” and the “GA” can discover process models with similar quality, but the former has faster convergence speed than the latter. Note that both of the two algorithms use random search (i.e., the genetic mutation). The difference is that the “DE + Random Search” employs the DE algorithm. It proves that the DE algorithm can quickly approximate the optimal solution and accelerate the convergence speed. (3) Based on the experimental results of the “DE + Random Search” and the “DE + Guided Search,” it can be seen that the latter achieves much better results than the former. This explains that the guided local exploration is efficient to help the DE algorithm skip out the local optimum and improve the searching ability of the DEMiner.

4.3. Performance on Artificial Event Logs

4.3.1. Setup. This section compares the performance of the DEMiner with three popular process mining algorithms. Through the experiments, I want to evaluate the performance as well as the antinoise ability of the DEMiner. The selected process mining algorithms for comparison include Heuristics Miner (HM) [20], ILP Miner [5], and ETM Miner [12]. Among these algorithms, HM is a popular tool of process mining, which outputs a heuristic net as a mining

TABLE 4: Comparison of different methods.

Event logs	DE			DE + Random Search			DE + Guided Search			GA		
	Cp.	Prec. (10^{-4})	Gen.	Cp.	Prec. (10^{-4})	Gen.	Cp.	Prec. (10^{-4})	Gen.	Cp.	Prec. (10^{-4})	Gen.
a5	0.91±0.02	1.05±0.18	27±3	1.00	2.58±0.04	92±21	1.00	2.64	63±10	1.00	2.60±0.04	88±21
a6nfc	0.85±0.04	1.92±0.36	28±2	0.99±0.01	3.52±0.08	51±12	1.00	3.58±0.02	25±3	0.98±0.02	3.49±0.08	105±36
a7	0.82±0.05	1.82±0.19	31±4	0.93±0.02	2.42±0.12	70±22	0.98±0.02	2.51±0.08	55±13	0.97±0.03	2.37±0.10	90±18
a8	0.92±0.01	2.21±0.17	27±5	0.98±0.02	3.49±0.16	73±18	1.00	3.98	33±4	1.00	3.71±0.15	60±17
a10skip	0.90±0.04	2.68±0.09	30±3	1.00	2.75±0.04	32±6	1.00	2.81	27±4	1.00	2.77±0.04	43±14
all	0.76±0.07	0.92±0.12	35±6	0.98±0.01	1.28±0.17	36±10	1.00	1.57±0.02	28±6	0.99±0.01	1.22±0.19	91±23
al2	0.87±0.03	0.48±0.13	32±5	1.00	0.52±0.08	38±9	1.00	0.63±0.03	49±17	1.00	0.45±0.12	49±9
bn1	0.97±0.02	0.71±0.11	24±2	1.00	0.85	45±11	1.00	0.85	26±5	1.00	0.83±0.02	48±12
bn2	0.99±0.01	0.32±0.05	22±2	1.00	0.37±0.01	27±5	1.00	0.38	25±4	1.00	0.35±0.03	33±8
bn3	0.91±0.04	0.21±0.03	24±2	0.97±0.02	0.23±0.02	61±14	1.00	0.25	44±14	0.94±0.04	0.21±0.03	76±17
choice	0.88±0.04	1.09±0.10	35±5	0.98±0.01	1.92±0.18	49±12	1.00	2.78	41±11	1.00	2.33±0.14	63±10
h3p4	0.85±0.03	0.92±0.17	22±1	1.00	1.21±0.03	30±6	1.00	1.28±0.08	31±7	1.00	1.29±0.05	46±12
h6p18	0.73±0.09	0.35±0.12	42±6	0.93±0.03	0.52±0.06	85±16	0.97±0.02	0.58±0.06	85±32	0.96±0.03	0.53±0.04	122±45
h6p30	0.92±0.03	0.28±0.08	29±3	0.97±0.02	0.35±0.04	42±10	1.00	0.43	42±10	0.99±0.01	0.38±0.04	44±8
h6p36	1.00	2.80	20±0	1.00	2.80	20±0	1.00	2.80	20±0	1.00	2.80	20±0
h6p41	0.85±0.03	1.12±0.06	32±4	1.00	2.12±0.03	64±12	1.00	2.15	34±8	0.98±0.02	2.20±0.04	75±21
h6p45	0.74±0.05	1.19±0.08	39±5	0.94±0.03	2.33±0.15	79±15	1.00	2.79±0.05	46±12	0.98±0.01	2.08±0.18	109±27
l1l	0.68±0.13	0.92±0.19	31±3	0.92±0.04	1.14±0.28	84±12	1.00	2.40	47±7	0.95±0.03	1.58±0.13	94±19
l2l	0.94±0.03	1.14±0.14	22±2	1.00	1.35±0.09	36±8	1.00	1.44	27±5	1.00	1.42±0.10	36±5
l2lOpt	0.81±0.06	1.38±0.18	22±1	1.00	2.71±0.03	36±11	1.00	2.75	25±2	1.00	2.75	48±9
l2lSkip	0.73±0.11	1.02±0.07	25±2	0.99±0.01	1.42±0.07	48±10	1.00	1.60	32±6	1.00	1.14±0.07	54±12
para15	0.71±0.08	1.13±0.16	35±4	0.91±0.04	1.37±0.11	82±18	1.00	1.66±0.06	52±10	0.93±0.04	1.26±0.17	102±31

FIGURE 4: Running time of DEMiner on 22 event logs.

result, and ILP Miner as well as ETM are two state-of-the-art algorithms in the field of process mining. ProM 6.9 which is the most popular process mining platform was used in the experiments [21]. Parameters of the three algorithms were set to default. Because ILP Miner and ETM output a Petri net and a process tree as a mining result, respectively, the obtained models must be converted to causal matrices based on Definition 4. By this way, the four algorithms can be evaluated by a unified way. Specially, if the outputting model contains invisible transitions (e.g., the ETM), it would be converted to causal matrices by hand.

Four metrics defined in [8, 19] were used to evaluate the algorithms in the experiments. The metrics include behavioral precision (Bp), behavioral recall (Br), structural precision (Sp), and structural recall (Sr). The Bp and Br are based on the parsing of an event log by the mined model and the original model. The former detects how much behavior is allowed by the mined model that is not by the original model. The latter detects for the opposite. Moreover, the closer the values of Bp and Br to 1.0, the higher the similarity between the original and the mined models. The Sp and Sr metrics are based on the causality relations of the mined and original models, the former detects how many causality relations the mined model has that are not in the original model, and the latter detects for the opposite. Different from the Bp and Br, the Sp and Sr measure the similarity from the structural point of view. When the

original model has connections that do not appear in the mined model, Sr will take a value smaller than 1, and, in the same way, when the mined model has connections that do not appear in the original model, Sp will take a value lower than 1.0.

4.3.2. Noise-Free Event Logs. First of all, the experiments were performed on 22 noise-free event logs. The results are listed in Table 5, in which the best results on each log are in italics. From the results, it is easy to find that the performance of the DEMiner is slightly better than other three algorithms. The DEMiner achieves the optimal solutions (i.e., four metrics are equal to 1) on 18 event logs. The HM, ILP Miner, and ETM achieve the optimal solutions on 16 event logs, 14 event logs, and 5 event logs, respectively. To compare the four algorithms more intuitively, a combinatorial metric, called average f -score, is designed, which is shown as follows:

$$f\text{score}_{\text{ave}} = \frac{B_p \cdot B_r}{B_p + B_r} + \frac{S_p \cdot S_r}{S_p + S_r}. \quad (7)$$

The results are shown in Figure 5. From the figure, it is easy to find that the DEMiner only lost to other algorithms on four event logs, which are “a5,” “a6nfc,” “a7,” and “h6p18.” Except that, the DEMiner obtained the best results on the remaining 18 event logs. Moreover, the average f -

TABLE 5: Comparison of four algorithms' performance on 22 event logs without noise.

Event logs	DEMiner				HM				ILP Miner				ETM			
	Br	Bp	Sr	Sp	Br	Bp	Sr	Sp	Br	Bp	Sr	Sp	Br	Bp	Sr	Sp
a5	1.00	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.91	1.00	1.00
a6nfc	1.00	0.94	0.93	1.00	0.87	0.68	0.93	1.00	1.00	1.00	1.00	1.00	0.98	0.91	0.93	0.93
a7	1.00	0.83	0.98	0.92	0.86	0.78	0.95	0.92	1.00	0.99	0.95	1.00	0.96	0.92	0.97	0.97
a8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.00	0.95	1.00
a10skip	1.00	1.00	1.00	1.00	1.00	0.91	0.90	1.00	1.00	0.97	1.00	0.95	0.96	0.86	1.00	1.00
al1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
al2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.95	0.88	0.94	0.95
bn1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.97	0.91	0.98	0.98
bn2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	0.98	0.98	0.96	0.98	0.94
bn3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.00	0.98	—	—	—	—
choice	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
h3p4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.93	0.82	0.94	0.82
h6p18	0.96	0.86	0.94	1.00	1.00	1.00	1.00	1.00	1.00	0.80	1.00	0.80	0.95	0.82	0.92	0.92
h6p30	1.00	1.00	1.00	1.00	1.00	0.98	1.00	0.95	1.00	1.00	1.00	1.00	0.92	0.98	0.90	0.91
h6p36	1.00	0.95	0.92	1.00	1.00	0.95	0.92	1.00	1.00	0.95	0.91	1.00	1.00	0.95	0.92	1.00
h6p41	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.93	0.99	0.95	0.95
h6p45	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.98	0.95	0.95
l1l	1.00	1.00	1.00	1.00	0.95	0.97	0.87	1.00	0.95	0.97	0.86	1.00	1.00	1.00	1.00	1.00
l2l	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
l2lOpt	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.91	0.83	0.90	0.90
l2lSkip	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.95	1.00	0.90	1.00	0.95	1.00	0.90
para15	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.93	0.97	0.97

FIGURE 5: Average f -score of four algorithms on 22 event logs without noise.

score achieved by the DEMiner on the rest 4 event logs is over 0.9. It demonstrates that the DEMiner has good performance. Later, a deep analysis is given.

There are four event logs that the DEMiner could not achieve the best results, which are “a5,” “a7,” “h618,” and “h6p36.” The mining results which were repeated most often are shown in Figure 6. In the figure, the incorrect parts have been labeled by red color. Moreover, the dotted lines denote the missing arcs (a.k.a in the original model

but is not discovered by the DEMiner), and the solid line denotes the incorrect connections (a.k.a. structural errors).

In Figure 6(a), it can be seen that the mined model lacks a cycle $\langle E, E \rangle$. In the original model, there are two cycles on the node “E.” The reason behind this phenomenon is that the operators of differential mutation proposed in this work (Definitions 5 and 6) will remove such structure during the evolution. Assume $S1 = \{\{E\}, \{E\}, \{B, C\}\}$ and $S2 = \{\{E\}, \{B\}\}$,

FIGURE 6: Mining results of four event logs: (a) a5; (b) a7; (c) h6p18; (d) h6p36.

then $S1-S2 = \{\{B, C\}\}$; in other words, the two subsets $\{E\}$ would be removed from $S1$ by the minus operation. Furthermore, assume $S1 = \{\{E\}, \{E\}, \{B, C\}\}$ and $S2 = \{\{E\}, \{B, C\}\}$; it is easy to find that the two sets will obtain the same completeness value, but the precision value of $S1$ is lower than that of $S2$ (the former has more enabled activities). In other words, $S1$ would be replaced by $S2$ during the

evolution. Similarly, this phenomenon also appears in the event log “h6p18.”

Next, in the mined model of “a7,” the input set of node “D” is $\{\{2, 4\}, \{3, 7, 8\}\}$, which is $\{\{2\}, \{3, 7\}, \{4, 7, 8\}\}$ in the original model. The reason behind this incorrectness is also the differential mutation operations. It is easy to find that removing the intersection part of two sets is a high

TABLE 6: Comparison of four algorithms' performance on 22 event logs with 5% noise.

Event logs	DEMiner				HM				ILP Miner				ETM			
	Br	Bp	Sr	Sp	Br	Bp	Sr	Sp	Br	Bp	Sr	Sp	Br	Bp	Sr	Sp
a5	1.00	0.84	1.00	0.95	1.00	0.78	0.86	0.95	1.00	1.00	1.00	1.00	0.94	0.84	1.00	0.83
a6nfc	0.98	0.65	0.96	0.74	0.87	0.68	1.00	0.83	0.81	0.75	0.64	0.80	0.84	0.75	0.86	0.95
a7	1.00	0.78	0.95	0.74	0.82	0.78	0.88	0.72	1.00	0.76	0.68	0.64	0.85	0.81	0.78	0.70
a8	1.00	0.74	0.98	0.69	1.00	0.79	0.84	0.80	1.00	0.70	0.78	0.82	0.89	0.81	0.67	1.00
a10skip	1.00	0.83	0.94	0.72	0.85	0.62	0.82	0.71	1.00	0.81	0.94	0.75	0.96	0.66	0.88	0.75
al1	1.00	0.77	0.96	0.94	0.92	0.81	0.82	0.76	1.00	0.81	1.00	0.90	0.91	0.77	0.84	0.85
al2	0.97	0.74	0.99	0.80	0.96	0.67	0.82	0.59	1.00	0.82	0.88	0.73	0.95	0.82	0.94	0.75
bn1	1.00	0.95	1.00	0.92	1.00	0.88	1.00	0.83	1.00	1.00	1.00	1.00	1.00	0.98	0.94	0.94
bn2	1.00	0.92	1.00	0.92	1.00	0.88	1.00	0.82	1.00	0.88	1.00	0.89	0.92	0.83	0.95	0.84
bn3	0.97	0.92	1.00	0.91	0.96	0.87	0.91	0.84	1.00	0.88	0.98	0.94	—	—	—	—
choice	0.98	0.72	0.96	0.84	0.89	0.72	0.77	0.80	1.00	0.75	1.00	0.83	1.00	0.73	0.92	0.86
h3p4	0.96	0.78	0.98	0.76	1.00	0.71	0.94	0.69	1.00	0.75	0.95	0.80	0.96	0.80	0.87	0.82
h6p18	1.00	0.69	1.00	0.84	1.00	0.82	1.00	0.79	0.93	0.78	1.00	0.80	1.00	0.84	0.86	0.82
h6p30	0.98	0.77	0.98	0.83	0.89	0.76	0.81	0.70	1.00	0.81	0.85	0.78	0.94	0.83	0.80	0.80
h6p36	1.00	0.74	0.90	0.86	0.94	0.80	0.83	0.72	1.00	0.75	0.94	0.82	1.00	0.76	0.87	0.81
h6p41	0.99	0.79	1.00	0.80	0.94	0.78	0.95	0.77	0.92	0.80	1.00	0.79	0.85	0.81	0.78	0.82
h6p45	0.97	0.73	1.00	0.72	0.93	0.69	0.86	0.73	0.92	0.77	1.00	0.75	0.90	0.82	0.81	0.76
l1l	1.00	0.80	0.91	0.82	0.84	0.81	0.78	0.83	1.00	0.78	1.00	0.85	0.88	0.81	0.73	1.00
l2l	1.00	0.93	1.00	0.83	0.97	0.92	0.86	0.89	0.94	0.96	1.00	0.87	1.00	0.89	1.00	0.90
l2lOpt	0.99	0.91	0.98	0.95	1.00	0.88	1.00	0.91	1.00	0.90	1.00	0.91	0.91	0.90	0.90	1.00
l2lSkip	1.00	0.90	1.00	0.83	0.98	0.93	0.89	0.90	1.00	0.94	0.97	0.85	0.90	0.87	0.90	0.80
para15	0.98	0.83	1.00	0.82	0.87	0.80	0.78	0.83	0.99	0.92	0.86	0.82	1.00	1.00	1.00	1.00

probability event in the proposed algorithm. Assume two sets $S1 = \{3, 7\}$ and $S2 = \{4, 7, 8\}$, $S1 + S2 = \{\{3, 7\}, \{8\}\}$ in terms of Definition 6. It can be seen that the intersection of two sets is removed. From above analysis, it can be found that the proposed method could not achieve the best results in some rare cases (e.g., two circles on a same node and existing redundant structures). However, from another perspective, it shows that the proposed method prefer the model with low structural complexity.

For “h6p36,” the four algorithms discovered the same model shown in Figure 6(d). The mined model lacks two arcs $\langle KB, NB \rangle$ and $\langle KA, NA \rangle$, which exist in the original model (heuristic net) [19]. Through analysis, I find the original heuristic net is incorrect. From the CPN model of “h6p36,” it can be seen that the model just has two parallel paths starting with “KA” and “KB,” respectively. However, the giving heuristic net has two arcs, i.e., $\langle KA, NA \rangle$ and $\langle KB, NB \rangle$, which may lead to two nonexistent paths $\{\text{Start}, KA, NA, \text{End}\}$ and $\{\text{Start}, KB, NB, \text{End}\}$. Therefore, the mining results of four algorithms can perfectly fit to the event log in fact.

4.3.3. Noisy Event Logs. Next, the experiments were performed on 22 event logs with 5% noise. The experimental results are shown in Table 6. The performance of the HM degrades significantly. Noise also affects other two algorithms, but their performance degradation is smaller than the HM algorithm. Moreover, the ETM can also discover the optimal process models on “para15,” which did not achieve the best result before. Similarly, the average f -score of the

four algorithms are calculated (see Figure 7). It can be seen from the figure that the performance of DEMiner slightly degrades and it achieves the best results on 13 event logs. Moreover, the average f -score of the DEMiner is between 0.8 and 1.0.

Later, the experiments were performed on the event logs with 10% noise. The experimental results are listed in Table 7, and the average f -score of the four algorithms is shown in Figure 8. From the table, we can see that the DEMiner achieves the best results on 14 event logs, and the other three algorithms (from left to right) achieves the best models on 2, 7, and 4 event logs, respectively. We can see that the ETM discovers the optimal model on “a7” and “a8,” which contain 10% noise, but it does not find the optimal model on the same logs that with 5% noise. This is because the two logs are independent; i.e., the inserted noise may be totally different. Through careful comparison of Figures 7 and 8, it can be found that the performance of the DEMiner dose not degrade significantly, and it keeps in a stable level. Based on the experiments on two groups of event logs with noise, a conclusion can be drawn that the DEMiner has good antinoise ability. However, we should notice that the DEMiner cannot discover the optimal solutions on all event logs with noise. This phenomenon demonstrates that the DEMiner cannot yet avoid noise interference.

4.4. Performance on Real Event Logs. This section shows the performance of the DEMiner on two real event logs, i.e., “BPI2013cp” and “Sepsis.” In the experiments, a “start”

FIGURE 7: Average f -score of four algorithms on 22 event logs with 5% noise.

TABLE 7: Comparison of four algorithms' performance on 22 event logs with 10% noise.

Eventlogs	DEMiner				HM				ILP Miner				ETM			
	Br	Bp	Sr	Sp	Br	Bp	Sr	Sp	Br	Bp	Sr	Sp	Br	Bp	Sr	Sp
a5	0.98	0.81	0.96	0.85	0.89	0.90	0.83	1.00	0.96	0.88	0.92	0.84	0.92	0.86	0.92	0.81
a6nfc	1.00	0.84	0.98	0.86	1.00	0.89	0.91	0.93	1.00	0.89	0.92	0.87	0.98	0.91	0.86	0.93
a7	1.00	0.94	1.00	0.80	1.00	0.88	0.94	0.86	1.00	0.95	1.00	0.98	1.00	1.00	1.00	1.00
a8	1.00	0.85	1.00	0.88	0.97	0.91	0.82	0.83	1.00	0.73	0.98	0.93	1.00	1.00	1.00	1.00
a10skip	0.99	0.76	0.97	0.69	0.84	0.73	0.86	0.70	0.91	0.79	0.86	0.73	0.90	0.74	0.82	0.62
al1	0.97	0.73	0.91	0.93	0.91	0.67	0.75	0.60	0.95	0.81	0.85	0.92	0.92	0.73	0.84	0.81
al2	0.92	0.80	1.00	0.75	0.89	0.73	0.74	0.79	0.94	0.79	0.83	0.84	0.88	0.62	0.81	0.80
bn1	1.00	0.78	1.00	0.84	0.92	0.83	0.87	0.75	1.00	0.75	0.90	0.82	0.96	0.69	0.93	0.97
bn2	1.00	0.74	1.00	0.87	0.98	0.74	0.93	0.77	1.00	0.73	0.99	0.82	1.00	0.74	0.95	0.90
bn3	0.97	0.84	0.96	0.79	1.00	0.79	0.91	0.74	0.96	0.84	0.95	0.82	—	—	—	—
choice	1.00	0.77	1.00	0.68	0.95	0.70	0.80	0.83	1.00	0.77	1.00	0.78	1.00	0.72	0.94	0.76
h3p4	0.98	0.70	0.96	0.69	0.95	0.66	0.89	0.63	0.93	0.77	0.92	0.72	0.96	0.70	0.85	0.72
h6p18	0.92	0.64	0.90	0.77	0.96	0.75	0.95	0.78	0.92	0.76	0.92	0.80	0.93	0.74	0.92	0.70
h6p30	0.96	0.77	0.97	0.82	0.96	0.76	0.95	0.78	0.95	0.78	0.96	0.80	0.92	0.80	0.86	0.75
h6p36	1.00	0.89	0.80	0.78	1.00	0.88	0.84	0.76	1.00	0.88	0.81	0.74	1.00	0.91	0.85	0.80
h6p41	0.96	0.77	0.98	0.76	0.91	0.74	0.92	0.77	0.93	0.73	0.95	0.78	0.87	0.80	0.88	0.72
h6p45	1.00	0.82	1.00	0.85	0.94	0.72	0.88	0.90	0.96	0.82	1.00	0.90	0.90	0.74	0.83	0.76
l1l	1.00	0.75	1.00	0.84	0.87	0.75	0.92	0.80	1.00	0.78	0.94	0.85	0.87	0.75	0.83	0.82
l2l	1.00	0.77	0.99	0.80	0.98	0.83	0.91	0.77	0.99	0.82	0.95	0.81	1.00	0.74	0.98	0.83
l2lOpt	1.00	0.88	0.99	0.79	1.00	0.82	1.00	0.73	1.00	0.80	0.98	0.77	0.98	0.85	0.89	0.82
l2lSkip	0.98	0.75	0.96	0.73	0.97	0.80	0.92	0.74	0.98	0.77	0.92	0.80	0.90	0.75	0.96	0.76
paral5	0.95	0.74	0.97	0.77	0.90	0.78	0.84	0.71	0.88	0.82	0.95	0.79	0.82	0.76	0.73	0.63

event as well as an “end” event was added to each trace at the running time. It ensures that each path has the same “start” node and the “end” node. To eliminate the randomness of evolution, the DEMiner was executed 10 times on each event log. The processes of evolution on two event logs are shown in Figures 9 and 10, respectively. The

ordinate denotes the completeness metric, and the abscissa denotes the number of generations, which is scaled from 1 to 100. Here, I only record the completeness value because it is monotonous. On the contrary, the precision value may fluctuate greatly along with the change in the completeness value. From the two figures, it

FIGURE 8: The average f -score of four algorithms on 22 event logs with 10% noise.

FIGURE 9: The completeness curves for 10 runs on BPI2013cp.

can be seen that the DEMiner converges fast on two event logs.

Furthermore, three metrics were employed to evaluate the efficiency of the DEMiner, which are alignment-based fitness [22], alignment-based precision [23], and the combined f -score [24]. The calculating results, including the average value and the standard deviation, are

listed in Table 8. Compared with the results given in [24], it is easy to find that the results obtained by the DEMiner is slightly lower than the result obtained by the ETM on “Sepsis,” and the results obtained by the DEMiner is better than the result obtained by the ETM on “BPI2013cp”. This proves that the DEMiner can play well on the real event logs.

FIGURE 10: The completeness curves for 10 runs on Sepsis.

TABLE 8: The alignment-based fitness and precision of the DEMiner on two real event logs.

Event log	Fitness	Precision	f -score
BPI2013cp	0.97 ± 0.02	0.82 ± 0.06	0.88 ± 0.04
Sepsis	0.88 ± 0.04	0.61 ± 0.03	0.72 ± 0.03

5. Conclusions

This paper proposes a new process mining algorithm, named DEMiner. The proposed algorithm is based on a hybrid evolutionary strategy, which consists of a set-based DE algorithm and a guided local exploration algorithm. Meanwhile, some techniques are employed to improve the efficiency of the DEMiner, such as gene bank, taboo list, and consistence repair. To evaluate the performance, 68 event logs were used in the experiments. Some conclusions can be drawn based on the experimental results:

- (1) Through comparison of the four different strategies (i.e., DE, DE + Random Search, DE + Guided Search, and GA), the “DE + Guided Search” outperforms the rest strategies and it can achieve the best quality solution as well as the fastest convergence speed. Moreover, the results prove that the DE algorithm can rapidly approximate the optimal solution, but it always suffers from premature convergence. The guided local exploration can help the DE algorithm skip out the local optimum and improve the efficiency of the proposed algorithm.
- (2) Through comparing the performance of DEMiner with three popular process mining algorithms (i.e., HM, ILP Miner, and ETM) on 22 noise-free event logs and 44 noisy event logs, it shows that the DEMiner can achieve the best result on most of the event logs. Furthermore, based on the experimental results on 2 real event logs, it can be concluded that the DEMiner can work well on real-world events logs. This proves the effectiveness and the efficiency of the proposed algorithm.

However, we can see that the DEMiner did not successfully discover an optimal process model from any one of the 44 noisy event logs. This demonstrates the drawback of the DEMiner. Besides that, it is hard for the DEMiner to discover some rare structures, such as two circles on a node. This is the future work for us.

Data Availability

The event logs used to support the findings of this study are included within the article, which can be downloaded from <https://svn.win.tue.nl/repos/prom/DataSets/GeneticMinerLogs/>. The corresponding process model can be found in [19].

Conflicts of Interest

The author declares that there are no conflicts of interest.

Acknowledgments

This work was supported by the Project of Science and Technology Bureau of Leshan (Grant no. 18JZD117), the Scientific Research Fund of Leshan Normal University (Grant no. ZZ201822), the Key Projects of Sichuan Provincial Education Department of China (Grant no. 18ZA0239), and the Project of Key Lab of Internet Natural Language Processing of Sichuan Provincial Education Department (no. INLP201903).

References

- [1] W. M. P. van der Aalst, *Process Mining: Data Science in Action*, Springer-Verlag, Berlin, Germany, 2nd edition, 2016.
- [2] W. van der Aalst, T. Weijters, and L. Maruster, “Workflow mining: discovering process models from event logs,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.
- [3] A. Medeiros, B. Dongen, W. M. P. van der Aalst et al., “Process mining for ubiquitous mobile systems: an overview and a concrete algorithm,” in *Proceedings of the Ubiquitous Mobile Information and Collaboration Systems*, pp. 156–170, Riga, Latvia, June 2004.
- [4] L. Wen, W. M. P. van der Aalst, J. Wang, and J. Sun, “Mining process models with non-free-choice constructs,” *Data Mining and Knowledge Discovery*, vol. 15, no. 2, pp. 145–180, 2007.
- [5] S. J. van Zelst, B. F. van Dongen, W. M. P. van der Aalst, and H. M. W. Verbeek, “Discovering workflow nets using integer linear programming,” *Computing*, vol. 100, no. 5, pp. 529–556, 2018.
- [6] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, “Discovering block-structured process models from event logs containing infrequent behaviour,” in *Proceedings of the 2013 Business Process Management Workshops*, pp. 66–78, Beijing, China, August 2013.
- [7] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, “Scalable process discovery and conformance checking,” *Software & Systems Modeling*, vol. 17, no. 2, pp. 599–631, 2018.
- [8] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst, “Genetic process mining: an

- experimental evaluation,” *Data Mining and Knowledge Discovery*, vol. 14, no. 2, pp. 245–304, 2007.
- [9] H.-J. Cheng, C. Ou-Yang, and Y.-C. Juan, “A hybrid approach to extract business process models with high fitness and precision,” *Journal of Industrial and Production Engineering*, vol. 32, no. 6, pp. 351–359, 2015.
- [10] C. Ou-Yang, H.-J. Cheng, and Y.-C. Juan, “An integrated mining approach to discover business process models with parallel structures: towards fitness improvement,” *International Journal of Production Research*, vol. 53, no. 13, pp. 3888–3916, 2014.
- [11] B. Vázquez-Barreiros, M. Mucientes, and M. Lama, “Pro-DiGen: mining complete, precise and minimal structure process models with a genetic algorithm,” *Information Sciences*, vol. 294, pp. 315–333, 2015.
- [12] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst, “A genetic algorithm for discovering process trees,” in *Proceedings of the IEEE Congress Evolutionary Computation*, pp. 1–8, Brisbane, Australia, June 2012.
- [13] J. C. A. M. Buijs, B. F. van Dongen, W. M. P. van der Aalst et al., “On the role of fitness, precision, generalization and simplicity in process discovery,” in *Proceedings of the on the Move to Meaningful Internet Systems Conference*, pp. 305–322, Rome, Italy, September 2012.
- [14] M. L. van Eck, J. C. A. M. Buijs, and B. F. van Dongen, “Genetic process mining: alignment-based process model mutation,” in *Business Process Management Workshops*, pp. 291–303, Springer, Cham, Switzerland, 2014.
- [15] S. Das and P. N. Suganthan, “Differential evolution: a survey of the state-of-the-art,” *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.
- [16] W. van der Aalst, A. Adriansyah, and B. van Dongen, “Replaying history on process models for conformance checking and performance analysis,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 182–192, 2012.
- [17] B. Xu, X. Chen, X. H. Huang, and L. Tao, “A multistrategy-based multiobjective differential evolution for optimal control in chemical processes,” *Complexity*, vol. 2018, Article ID 2317860, 22 pages, 2018.
- [18] W. N. Chen, J. Zhang, H. S. H. Chung et al., “A novel set-based particle swarm optimization method for discrete optimization problems,” *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 2, pp. 278–300, 2010.
- [19] A. K. Alves de Medeiros, *Genetic process mining*, Ph.D. thesis, Eindhoven Universiteit of Technology, Eindhoven, Netherlands, 2006.
- [20] A. J. M. M. Weijters and J. Ribeiro, “Flexible heuristics miner (FHM),” in *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining*, pp. 310–317, Paris, France, April 2011.
- [21] B. van Dongen, A. K. Alves de Medeiros, H. M. W. Verbeek et al., “The prom framework: a new era in process mining tool support,” in *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets*, pp. 444–454, Miami, FL, USA, June 2005.
- [22] A. Adriansyah, B. van Dongen, and W. M. P. van der Aalst, “Conformance checking using cost-based fitness analysis,” in *Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference*, pp. 55–64, Helsinki, Finland, August 2011.
- [23] A. Adriansyah, J. Munoz-Gama, J. Carmona et al., “Alignment based precision checking,” in *Proceedings of the International Workshop on Business Process Intelligence*, pp. 137–149, Tallinn, Estonia, September 2012.
- [24] A. Augusto, R. Conforti, M. Dumas et al., “Automated discovery of process models from event logs: review and benchmark,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 4, pp. 686–705, 2019.