electronics

MDPI

Article

Timed Genetic Process Mining for Robust Tracking of Processes under Incomplete Event Log Conditions

Yutika Amelia Effendi [1,2] and Minsoo Kim [2,*]

[1] Robotics and Artificial Intelligence Engineering, Faculty of Advanced Technology and Multidiscipline, Airlangga University, Surabaya 60115, Indonesia; yutika.effendi@ftmm.unair.ac.id
[2] Department of Industrial and Data Engineering, College of Engineering, Pukyong National University, Busan 48513, Republic of Korea
* Correspondence: minsky@pknu.ac.kr; Tel.: +82-51-629-6487

Abstract: In process mining, an event log is a structured collection of recorded events that describes the execution of processes within an organization. The completeness of event logs is crucial for ensuring accurate and reliable process models. Incomplete event logs, which can result from system errors, manual data entry mistakes, or irregular operational patterns, undermine the integrity of these models. Addressing this issue is essential for constructing accurate models. This research aims to enhance process model performance and robustness by transforming incomplete event logs into complete ones using a process discovery algorithm. Genetic process mining, a type of process discovery algorithm, is chosen for its ability to evaluate multiple candidate solutions concurrently, effectively recovering missing events and improving log completeness. However, the original form of the genetic process mining algorithm is not optimized for handling incomplete logs, which can result in incorrect models being discovered. To address this limitation, this research proposes a modified approach that incorporates timing information to better manage incomplete logs. By leveraging timing data, the algorithm can infer missing events, leading to process tracking and reconstruction which is more accurate. Experimental results validate the effectiveness of the modified algorithm, showing higher fitness and precision scores, improved process model comparisons, and a good level of coverage without errors. Additionally, several advanced metrics for conformance checking are presented to further validate the process models and event logs discovered by both algorithms.

Keywords: genetic process mining; dual timestamp; incomplete event log; process model; parallel processes; conformance; process mining; process discovery

check for updates

Citation: Effendi, Y.A.; Kim, M. Timed Genetic Process Mining for Robust Tracking of Processes under Incomplete Event Log Conditions. *Electronics* **2024**, *13*, 3752. https://doi.org/10.3390/electronics13183752

Academic Editors: Tania Cerquitelli, Giovanni Malnati and Genoveva Vargas-Solar

Received: 7 August 2024
Revised: 9 September 2024
Accepted: 19 September 2024
Published: 21 September 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In process mining, an event log is a structured collection of recorded events that describes the execution of processes within an organization or system [1,2]. Each event in the log represents a specific action, state change, or activity that occurs during the execution of a process instance [3]. Event logs are crucial in process mining because they capture the sequential steps of processes, ensuring the accuracy and reliability of the analysis derived from process mining techniques [4]. The completeness of these event logs is vital; incomplete event logs can lead to flawed models and misinformed decisions, making completeness a significant focus in process mining research [5].

Real-world event logs are often more complex than the simplified case studies used in research. The former involve numerous cases, activities, timestamps, and intricate dependencies [6]. While this research acknowledges the complexity of real-world logs, it simplifies the analysis to highlight the core properties of event logs, focusing on essential elements like Case ID, Activity, and Timestamp [2]. These elements are fundamental for reconstructing process models and analyzing behaviors [1].

A persistent challenge in process mining is the handling of missing event logs [7]. Missing entries can result from system errors, manual oversight, technical limitations, or

data corruption [8]. The absence of these logs can undermine the integrity of process models, leading to incorrect conclusions about the efficiency and effectiveness of the underlying processes. Systems that run periodically, whether synchronously or asynchronously, are particularly prone to time gaps [9]. Synchronous systems operate on a fixed schedule, while asynchronous systems run based on specific triggers; both types can experience intervals where no events are logged. These gaps may be due to the maintenance periods, system downtimes, manual data entry errors, and irregular operational patterns which contribute to incomplete event logs [10,11].

Despite expectations of robust and detailed logging capabilities in modern software, some organizations still rely on systems that operate periodically and asynchronously [12,13]. This approach allows them to maintain control over their processes while avoiding the delays associated with obtaining a complete event log. In scenarios where manual periodic event logging occurs, such as in port container terminals [14] and hospitals [15], the research focuses on filling gaps created by the temporary absence of activities until the actual logs are entered or updated. Once the complete event log is available, the process discovery algorithm can be reapplied to discover a process model based on the updated data. This iterative approach is repeated whenever an incomplete event log occurs during periodic system operation, ensuring continuous model accuracy and completeness.

Addressing time gaps is essential for maintaining the integrity of event logs and constructing accurate process models. Time gaps can result in incomplete data, making it challenging to create comprehensive models. To mitigate these gaps, it is often necessary to infer missing logs by making educated guesses based on previous complete event logs. This process involves utilizing historical data and patterns to reconstruct a more accurate representation of the process.

The primary goal of this research is to enhance the performance and robustness of process models by transforming incomplete event logs into complete ones, ensuring that the derived models are both accurate and reliable. To achieve this, sophisticated algorithms are needed to intelligently fill in the missing data. One such algorithm is the genetic process mining algorithm, chosen for its ability to generate and evaluate multiple candidate solutions concurrently [1]. This algorithm evolves process models through crossover and mutation, optimizing a fitness function to match the observed data. Its global optimization strategy searches for the models that best fit incomplete and noisy data, increasing the likelihood of the identification and filling of gaps in the event log data [16]. By considering a broad range of possible process flows and sequences, the genetic process mining algorithm can effectively recover missing events and enhance the completeness of event logs.

However, traditional genetic process mining algorithms are not specifically optimized for handling incomplete logs. Missing events can create incorrect sequences in the discovered model, making it less representative of the actual process. Therefore, this research proposes an optimized approach to genetic process mining, one that incorporates timing information, referred to as the timed genetic process mining algorithm. By using timing data, the modified algorithm can infer where events should be, even if they are missing from the log. This approach allows for a process reconstruction which is more accurate, making the algorithm more robust in dealing with incomplete data by filling in gaps based on expected time intervals.

By integrating timing information, the proposed modifications aim to create more robust and accurate process models capable of handling incomplete event logs more effectively. This will lead to better-informed decisions and more efficient processes within organizations. Through various experiments, we aim to validate the effectiveness of this modified algorithm, as well as the process model and event log generated from it. Our analysis demonstrates that this approach accurately generates a complete event log and process model from an incomplete log, showing commendable structural and behavioral appropriateness. Furthermore, our research validation compares the modified algorithm's results with those of the original genetic process mining algorithm and base processes from

the organization, assessing structural appropriateness—such as process model comparison, process model coverage, and structural matching traces—and behavioral appropriateness, including fitness and precision scores.

The document is organized as follows: Section 2 delves into the research background; Section 3 introduces the experimental design; Sections 4 and 5 showcase comparisons of algorithms for recovering missing events and provide in-depth comparisons of experiments and analysis, respectively; and Sections 6 and 7 present advanced metrics for conformance checking and the conclusion.

2. Research Background

2.1. Process Mining

In Industry 4.0, recording nearly all process activities is now possible. This recorded data, whether analyzed in real time or offline, allows for detailed scrutiny tailored to specific needs—a practice known as process mining. Process mining combines data mining and process modeling, providing a bridge between data science and business process management [1]. The primary objectives of process mining include discovery, conformance, and enhancement [17]. Discovery focuses on uncovering and visualizing the actual processes as they occur, often creating a process model based on observed event data [2,17]. Conformance involves comparing the discovered process model with a predefined model to identify deviations and ensure compliance with specified procedures [18,19]. Enhancement aims to improve existing process models by incorporating additional information from event logs to optimize performance, efficiency, and understanding of the processes [1,20].

However, process mining is impossible without proper event logs [1,2]. It is a data analysis technique used to extract insights from event logs generated by information systems. It involves discovering, monitoring, and improving real processes by extracting knowledge from the logs which capture the sequence of activities in an organization's processes. Therefore, through event logs and the three aforementioned techniques, process mining helps organizations understand their processes better, identify inefficiencies, ensure compliance, and enhance overall performance based on empirical data.

2.2. Event Log Completeness

Event log completeness is a crucial concept in process mining, one which involves the analysis of event logs to extract process-related information. Completeness in this context refers to the extent to which the event log accurately and comprehensively reflects the underlying business process [4,7,21]. This includes several aspects: trace completeness, event completeness, attribute completeness, frequency completeness, and temporal completeness. Ensuring event log completeness is fundamental to the reliability and validity of process mining analyses, allowing for accurate process discovery, conformance checking, and enhancement, leading to more effective business process management and improvement [22].

Ensuring event log completeness faces several challenges. Inadequate or faulty data collection mechanisms can result in missing or incomplete data [23]. Processes often span multiple systems, and lack of integration can lead to incomplete event logs [24]. Manual data entry errors and non-compliance with logging protocols can affect completeness [25]. Additionally, technical limitations, such as storage capacity and system performance, can constrain the completeness of logs [26]. Policy and compliance issues also play a role; inconsistent adherence to data logging policies and procedures can lead to missing information, making the logs incomplete [27].

To enhance event log completeness, it is essential to improve data collection mechanisms and ensure all relevant systems are integrated. Regular audits of event logs should be conducted to identify and address gaps in data collection [28]. Staff should be trained on the importance of complete data logging and compliance with protocols [29]. Utilizing advanced data collection and integration tools can automate and improve the completeness of event logs [30].

Consider a hospital's patient management system, in which the event log captures all patient-related activities from admission to discharge, including diagnostics, treatments, and consultations. In a scenario of incomplete event logs, if the hospital uses multiple systems for different departments (e.g., emergency, radiology, and surgery) that are not fully integrated, a patient's transfer from the emergency department to radiology might not be recorded properly. This leads to gaps in the event log, and when analyzing the log for process improvement, it might appear that certain diagnostic procedures were not conducted or there might be discrepancies in the time taken for patient transfers. Consequently, incorrect conclusions about process efficiency and areas needing improvement might be drawn.

In another scenario, incomplete event logs due to human factors can be observed. Nurses and doctors manually enter data into the system, and due to heavy workloads, they occasionally forget to log patient interactions or input incorrect data. Important events, such as medication administration or patient consultations, might be missing or inaccurately recorded. When conducting an analysis to ensure compliance with treatment protocols, it might falsely appear that some patients did not receive the required medications on time, leading to potential misinterpretations of staff performance and patient-care quality.

The relationship between event log completeness and the occurrence of incomplete event logs is evident in how missing or incorrect data can skew process analysis and decision-making. Addressing challenges related to inadequate data collection mechanisms, lack of system integration, human errors, technical limitations, and compliance issues is crucial. Ensuring comprehensive event logs enables accurate and effective process mining and improvement initiatives.

2.3. Dual-Timestamp Event Log

Initially, event logs often record only one timestamp per event [1,2]. Although analyzing the time between two process steps is feasible with a single timestamp, accurately determining the duration of each activity is not possible [14]. This limitation makes the processing time appear instantaneous. To address this, it is essential to have both starting and ending timestamps for each activity, leading to the concept of an event log with dual timestamps.

There has been previous research related to the concept of dual timestamps. For instance, studies [31–33] exemplify this concept. Research efforts [31,32] introduced a time-based discovery algorithm that uses the non-linear dependence principle. This algorithm employs the concept of a Gantt chart to represent dual timestamps. Furthermore, study [33] explored the use of threshold intervals to identify parallels in the heuristic miner.

Event logs with dual timestamps provide a richer, more detailed dataset that significantly enhances the accuracy and effectiveness of process discovery algorithms. They offer better insights into the temporal aspects of processes, improve the handling of concurrency, and support analysis and visualization techniques which are more advanced [34]. Dual timestamps allow for the precise representation of activities that may overlap in time, providing a clearer picture of parallelism and concurrency, which helps in identifying and modeling concurrent processes more accurately [35]. By enabling the analysis of activity durations, dual timestamps are crucial for understanding process flow, identifying bottlenecks, and measuring performance [36]. They also provide a more detailed sequencing of events, allowing for the establishment of a more accurate order of activities. Detailed analysis of resource utilization is made possible by knowing exactly when each activity starts and ends, helping to determine resource allocation, utilization rates, and potential conflicts. Dual timestamps enable the identification of idle or waiting times between activities, pinpointing inefficiencies and aiding in the understanding of delays [37]. Many advanced process mining techniques, such as conformance checking and performance analysis, benefit from this richer dataset, allowing for better analysis of deviations and accurate measurement of performance [38]. Process models generated from dual-timestamp logs can be visualized with more detail, showing the sequence, duration, and overlap of activities, leading to more

informative and actionable reports. This capability is especially beneficial for capturing and analyzing complex processes with multiple concurrent and overlapping activities, resulting in process models which are more accurate and reliable and reflect the true nature of the business process.

Based on the advantages of dual-timestamp event logs, we use them in our study. However, the original form of genetic process mining, when discovering a process model, generally relies on a sequential approach rather than a timed approach. This means that the focus is on the order and sequence of activities rather than the specific timing or duration of each activity. It also includes the use of single-timestamp event logs as input. In the sequential approach, the primary goal is to reconstruct the sequence of activities in the process model from the event log, ensuring that the discovered model accurately reflects the observed order of activities. This involves analyzing the patterns and frequencies of activity sequences to infer the most likely process flow. On the other hand, a timed approach would consider the exact timestamps, durations, and potentially overlapping or concurrent activities. Therefore, a modification of genetic process mining is needed to accept dual-timestamp event logs as input and consider the temporal information of each event, including potentially parallel processes.

2.4. Behavioral and Structural Appropriateness

Behavioral and structural appropriateness are two key concepts in process mining [31,39], part of conformance [1,14,18,19] and used to evaluate the quality and alignment of a process model with the actual event log data. Together, these concepts ensure that a process model is both behaviorally and structurally aligned with the real-world processes it aims to represent, leading to more accurate, reliable, and useful process models in process mining.

2.4.1. Behavioral Appropriateness

Behavioral appropriateness measures the extent of the behavior permitted by the model that was never actually utilized in the observed process executions within the log [31]. The goal is to model the process as accurately as possible. If the model is overly general and permits more behavior than necessary, it becomes less informative because it no longer accurately describes the actual process and may allow for unwanted execution sequences [19].

In practice, this conformance issue can be examined from two perspectives [39]. Firstly, the "extra" behavior allowed by the model might represent, for instance, an alternative path designed for exceptional situations that did not occur during the period when the log was recorded, indicating that the event log is incomplete. Secondly, the model might indeed be overly generic, allowing for situations that never occur in reality. An expert in the relevant area would be needed to distinguish between these two scenarios. Hence, appropriate metrics are required.

Based on [14,31,39], the key aspects of behavioral appropriateness include how well the process model mirrors the actual behavior observed in the event log, concerning both sequence and parallel relationships. It assesses whether the model can reproduce the sequences and parallels of activities found in the log and avoid those that are not observed.

2.4.2. Structural Appropriateness

Constructing the model of a business process in a concise and meaningful way through measurement is challenging. The perceived suitability of a model often depends on subjective preferences and is typically aligned with the model's specific purpose. Factors like the granularity of workflow actions can only be determined by an experienced human designer [19]. Structural appropriateness usually pertains to the control of flow perspective, with multiple syntactic ways available to represent the same behavior in a process model, including duplicate tasks, invisible tasks, and implicit places [31].

To address structural appropriateness independently of the model's actual behavior, it is advisable to adhere to certain design guidelines that specify the preferred methods for expressing specific behavioral patterns and penalize deviations from these guidelines [39]. These design guidelines will vary across different process modeling notations and may depend on individual or corporate preferences.

Based on [14,19,31,39], structural appropriateness focuses on the structural aspects of the process model. It evaluates whether the model's structure is reasonable and aligns well with the logic and constraints of the real-world process.

2.5. Process Discovery

The essence of process discovery involves extracting valuable insights and knowledge from event logs produced by information systems [1,2,17]. There are several methods available for process discovery, such as the alpha miner, the heuristic miner, genetic process mining, the fuzzy miner, and the inductive miner. The methods represent various approaches to extracting process models from event logs, each with its own strengths and suitability, depending on the characteristics of the data and the goals of the analysis. The alpha miner [1,40–42] focuses on constructing a Petri net by identifying frequent patterns of behavior in the event log, making it suitable for straightforward processes with clear sequences. The heuristic miner [1,43,44] employs a more flexible approach by considering dependencies between activities and is adept at handling noise and inconsistencies in event data. Genetic process mining [1,16] applies evolutionary algorithms to explore and optimize process models, which is useful for complex processes with multiple variants. The fuzzy miner [1,45,46] incorporates fuzzy logic to accommodate uncertain or imprecise event data, providing process models which are more nuanced. Lastly, the inductive miner [1,46,47] uses a tree-based approach to infer process models from event logs, emphasizing simplicity and clarity in representation, which can be beneficial for understanding and communicating process flows. Each method offers distinct advantages in uncovering and representing process behavior, catering to different scenarios and analytical needs in process mining.

Genetic process mining excels in discovering process models from event logs due to its ability to handle complexity, variability, and data quality issues effectively [1,16]. Its exploratory nature, coupled with its optimization capabilities and scalability, positions genetic process mining as a robust approach for extracting actionable insights from process data, compared to other methods like the alpha miner, heuristic miner, fuzzy miner, and inductive miner.

This study uses, from among the five mentioned process discovery algorithms for recovering missing events from incomplete event logs, genetic process mining due to its ability to maintain diversity and explore alternative solutions. It inherently maintains a diverse population of potential solutions (process models) through mechanisms like crossover and mutation. This diversity allows it to explore different hypotheses and scenarios, which is crucial when dealing with incomplete event logs where certain events or sequences may be missing or ambiguous.

Unlike some other process mining methods, which may rely on specific assumptions or heuristics about process behavior [1], genetic process mining can generate and evaluate multiple candidate solutions concurrently [16]. This capability increases the likelihood of identifying and filling gaps in the event log data by considering a broader range of possible process flows and sequences. Furthermore, it is adaptive and iterative. It continuously refines and adjusts process models based on feedback from the event log data, gradually converging towards more accurate representations of the underlying processes. This adaptability is particularly advantageous in scenarios in which initial event logs may be sparse or inconsistent, or contain errors, allowing the algorithm to iteratively improve the completeness and reliability of the discovered process models over time.

Figure 1 outlines a genetic process mining approach to discover a process model from an event log. Algorithm A1 in Appendix A is an implementation of Figure 1. Initially, a

population of potential process models is initialized. Each model's fitness is evaluated based on how well it matches the observed behavior in the event log. The process then enters a loop in which, until certain termination criteria are met (such as a maximum number of iterations or satisfactory fitness level), parents are selected from the population based on their fitness scores. These selected parents undergo genetic operations—crossover and mutation—to produce offspring, representing new potential models. The fitness of these offspring is evaluated against the event log data to determine how accurately the offspring capture the process behavior. The least-fit models in the population are then replaced with the offspring, with the aim of improving the overall quality of models over successive iterations. Once the termination criteria are satisfied, the best-performing model in the population is selected as the discovered process model, representing the most accurate depiction of the process observed in the event log.

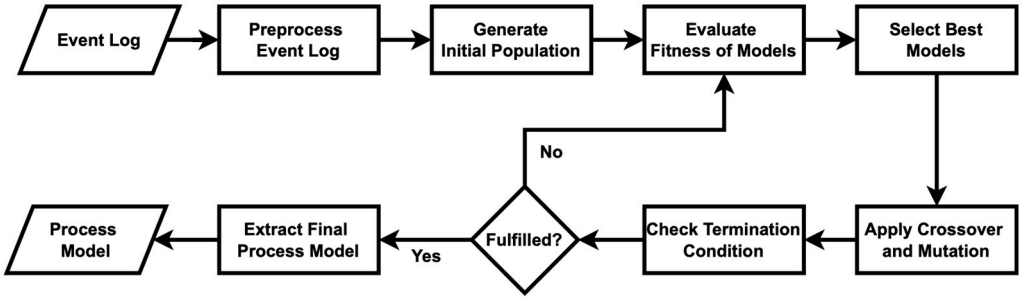


Figure 1. Flowchart of the original genetic process mining algorithm employed to discover a process model.

3. Experimental Setup

We conducted experiments using real data obtained from a private hospital in Surabaya, Indonesia. The data utilized in our study were collected between March and April 2023. Figure 2 illustrates the complete event log from the hospital. The event log, as recorded in the hospital, is already in a dual-timestamp format, with case IDs (Case Identifiers) that distinguish one instance of a process from another and activity names. In this study, activities are the same as events. The event log also reveals the presence of both sequential and parallel processes, as seen from the timestamps of each activity. There are 50 cases and 13 activities involved in this research.

However, during March and April 2023, several activities were manually recorded by hospital workers, and the data were entered into the hospital information system after working hours, at around 6 P.M. In this study, this scenario—in which the hospital information system does not fully record executed processes because some events are recorded asynchronously—results in incomplete event logs. Therefore, during March and April 2023, there are 10 cases identified as incomplete event logs. The goal of the genetic process mining algorithm is to track the runtime and sequences of process activities that happened for the data recorded outside the system, referred to as recovering missing events. Figure 2 also illustrates the incomplete event log from the hospital. It shows that there is 1 missing event in HPE019, which is activity "Patient Assignment", and 3 missing events in HPE033, comprising the activities "Therapy Sessions", "Payment Processing", and "Discharge Summary".

In this experiment, we applied the original genetic process mining algorithm under its standard conditions, using the same dataset as in Figure 2 but including only the starting timestamp, and without the ending timestamp. This approach aligns with the concept of the original algorithm, as well as most process discovery algorithms, which typically use a single timestamp—specifically the starting time for each event in the event log [1,2]. Moreover, the computer system configuration used for the experiments includes a Windows 10 Ultimate 64-bit operating system with 8 GB of memory. Table 1 outlines the parameters

used to test the dataset, for both the original genetic process mining algorithm and the timed genetic process mining algorithm.

COMPLETE EVENT LOG

Case ID	Activity	Start Timestamp	End Timestamp
HPE004	Patient Check-In	2023-03-12 08.13.00	2023-03-12 08.21.00
HPE004	**Patient Assignment**	**2023-03-12 08.20.21**	**2023-03-12 08.47.21**
HPE004	Insurance Verification	2023-03-12 08.26.24	2023-03-12 08.39.24
HPE004	Initial Assessment	2023-03-12 08.58.43	2023-03-12 09.07.43
HPE004	Diagnostic Tests	2023-03-12 09.03.49	2023-03-12 09.29.49
HPE004	Doctor Consultation	2023-03-12 09.11.16	2023-03-12 09.27.16
HPE004	Treatment Planning	2023-03-12 09.52.08	2023-03-12 10.18.08
HPE004	Medication Administration	2023-03-12 09.52.26	2023-03-12 10.04.26
HPE004	Therapy Sessions	2023-03-12 09.48.38	2023-03-12 10.02.38
HPE004	Generate Bill	2023-03-12 10.25.03	2023-03-12 10.49.03
HPE004	Payment Processing	2023-03-12 10.45.13	2023-03-12 10.57.13
HPE004	Discharge Summary	2023-03-12 10.21.00	2023-03-12 10.30.00
HPE004	Patient Discharged	2023-03-12 10.57.13	2023-03-12 11.19.13
HPE044	Patient Check-In	2023-04-14 08.18.57	2023-04-14 08.39.57
HPE044	Insurance Verification	2023-04-14 08.27.31	2023-04-14 08.49.31
HPE044	Patient Assignment	2023-04-14 08.13.22	2023-04-14 08.35.22
HPE044	Initial Assessment	2023-04-14 09.16.05	2023-04-14 09.46.05
HPE044	Diagnostic Tests	2023-04-14 08.49.40	2023-04-14 09.17.40
HPE044	Doctor Consultation	2023-04-14 09.11.14	2023-04-14 09.40.14
HPE044	Treatment Planning	2023-04-14 10.11.30	2023-04-14 10.22.30
HPE044	Medication Administration	2023-04-14 10.03.00	2023-04-14 10.22.00
HPE044	**Therapy Sessions**	**2023-04-14 10.00.42**	**2023-04-14 10.07.42**
HPE044	Generate Bill	2023-04-14 10.41.04	2023-04-14 10.52.04
HPE044	**Payment Processing**	**2023-04-14 10.25.45**	**2023-04-14 10.54.45**
HPE044	**Discharge Summary**	**2023-04-14 10.46.01**	**2023-04-14 10.57.01**
HPE044	Patient Discharged	2023-04-14 10.57.01	2023-04-14 11.19.01

MISSING

MISSING

MISSING

INCOMPLETE EVENT LOG

Case ID	Activity	Start Timestamp	End Timestamp
HPE019	Patient Check-In	2023-03-19 08.15.22	2023-03-19 08.31.22
HPE019	Insurance Verification	2023-03-19 08.25.07	2023-03-19 08.53.07
HPE019	Initial Assessment	2023-03-19 09.09.38	2023-03-19 09.14.38
HPE019	Diagnostic Tests	2023-03-19 09.20.33	2023-03-19 09.40.33
HPE019	Doctor Consultation	2023-03-19 09.09.39	2023-03-19 09.33.39
HPE019	Treatment Planning	2023-03-19 09.54.46	2023-03-19 10.24.46
HPE019	Medication Administration	2023-03-19 10.06.26	2023-03-19 10.21.26
HPE019	Therapy Sessions	2023-03-19 10.07.09	2023-03-19 10.31.09
HPE019	Generate Bill	2023-03-19 10.56.48	2023-03-19 11.22.48
HPE019	Payment Processing	2023-03-19 10.52.01	2023-03-19 11.08.01
HPE019	Discharge Summary	2023-03-19 10.58.27	2023-03-19 11.12.27
HPE019	Patient Discharged	2023-03-19 11.22.48	2023-03-19 11.31.48
HPE033	Patient Check-In	2023-04-02 07.50.02	2023-04-02 08.00.02
HPE033	Insurance Verification	2023-04-02 08.00.02	2023-04-02 08.30.02
HPE033	Patient Assignment	2023-04-02 08.22.30	2023-04-02 08.51.30
HPE033	Initial Assessment	2023-04-02 09.07.32	2023-04-02 09.18.32
HPE033	Diagnostic Tests	2023-04-02 08.54.15	2023-04-02 09.15.15
HPE033	Doctor Consultation	2023-04-02 09.14.09	2023-04-02 09.34.09
HPE033	Treatment Planning	2023-04-02 09.44.36	2023-04-02 10.08.36
HPE033	Medication Administration	2023-04-02 09.57.47	2023-04-02 10.27.47
HPE033	Generate Bill	2023-04-02 10.42.10	2023-04-02 11.05.10
HPE033	Patient Discharged	2023-04-02 11.32.35	2023-04-02 11.42.35

Figure 2. Dual-timestamp event log of the hospital, showing both complete and incomplete logs.

Table 1. Parameters used in the experiments.

Algorithms	Maximum Generation Limit	Size of Population
The original genetic process mining	50	50
The timed genetic process mining	50	50

To validate our proposed methodology, we measure the behavioral and structural appropriateness of both the event log and process model results from the original genetic process mining and the timed genetic process mining against the base event log and process model from the hospital. In our research, the metrics for measuring behavioral appropriateness are fitness and precision. Meanwhile, the metrics for measuring structural appropriateness are process model comparison, process model coverage, and similarity score for trace comparisons using the sequence-matcher method.

In short, fitness measures how well the process model can reproduce the behavior observed in the event log [1,14,35]. A high fitness score indicates that the model can replay all or most of the traces from the event log. In other words, it shows the extent to which the process model can accommodate the recorded events. Meanwhile, the precision determination measures how much additional behavior is allowed by the process model while not being observed in the event log [1,14,26]. A high precision score indicates that the process model does not allow for much behavior beyond what is seen in the event log. Equation (1) is used to calculate fitness, and Equation (2) is used to calculate precision.

$$Fitness = \frac{Correct\ Relations\ (TP)}{Total\ Base\ Relations\ (TP + FN)} \tag{1}$$

$$Precision = \frac{Correct\ Relations\ (TP)}{Total\ Discovered\ Relations\ (TP + FP)} \tag{2}$$

where

- *True Positives* (*TP*/Correct Relations): relations correctly identified in both the base and discovered models;

- *False Positives* (*FP*): relations identified in the discovered model but not in the base model;
- *False Negatives* (*FN*): relations present in the base model but not identified in the discovered model.

As for the process model comparison, we evaluate node correspondence, edge correspondence, flow sequence and frequency, and visual structure. For process model coverage, we calculate the coverage and error percentage of the discovered relations. To calculate the coverage and error percentage scores for each discovered model, we need to use Equations (3) and (4), respectively.

$$Coverage\ Score = \left(\frac{Number\ of\ correctly\ discovered\ base\ relations}{Total\ number\ of\ base\ relations} \right) \times 100 \quad (3)$$

$$Error\ Score = \left(\frac{Number\ of\ incorrectly\ discovered\ relations}{Total\ number\ of\ discovered\ relations} \right) \times 100 \quad (4)$$

4. Comparison of Algorithms for the Recovery of Missing Events

Before presenting our comparison focused on genetic process mining algorithms, we first discuss the results of a process model generated by another process mining algorithm: the inductive miner. The inductive miner is currently regarded as one of the most powerful process discovery algorithms [1,46]. It is typically used to discover process models from complete event logs with a single timestamp, but it does not effectively handle missing events or incomplete traces in its standard form [47].

We used the incomplete event log shown in Figure 2 as input for the inductive miner, and the resulting process model is displayed in Figure 3. The discovered process model is based solely on the incomplete event log, as the algorithm lacks the capability to infer the missing events. Based on Figure 3, several relations are missing from the model. While the inductive miner performs well with event logs that have a clear and repetitive structure, it faces challenges when dealing with highly incomplete event logs. The algorithm relies heavily on identifying common patterns, so when there is significant incompleteness and many missing events, the models it generates may not accurately represent the underlying process. In such cases, the algorithm might oversimplify the process or overlook less-frequent but critical behaviors. Therefore, for recovering highly incomplete event logs and obtaining accurate process models, the genetic process mining algorithm is more suitable.

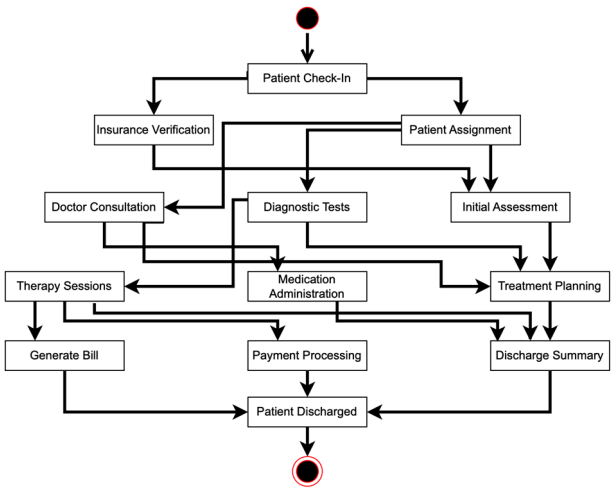


Figure 3. Process model discovered by the inductive miner algorithm using an incomplete event log.

4.1. Original Genetic Process Mining Algorithm

Figure 4 outlines a genetic algorithm designed to recover missing activities in an event log. Algorithm A2 in Appendix A begins by initializing a population of event logs, in which

each individual is a copy of the original event log with missing activities inserted at random positions. This ensures diversity in the initial population. This approach also includes step to sort each case in the event log by timestamp to ensure activities are in chronological order. The fitness of each individual is then evaluated based on the number of correctly recovered activities, compared to the original log. The top individuals, determined by their fitness scores, are selected to form the basis of the next generation. Crossover is performed between pairs of selected parents to create offspring by the exchange of case activities, introducing variability. Mutation is applied to these offspring by randomly changing activities within cases, ensuring further diversity. This process iterates for a specified number of generations, continuously evolving the population towards better solutions. The individual with the highest fitness in the final generation is considered the best solution, representing the event log with the most accurately recovered missing activities. Finally, the complete event log is sorted and displayed after adjusting the timestamp to avoid the duplicate, and the process model is visualized to show the discovered sequence of activities.

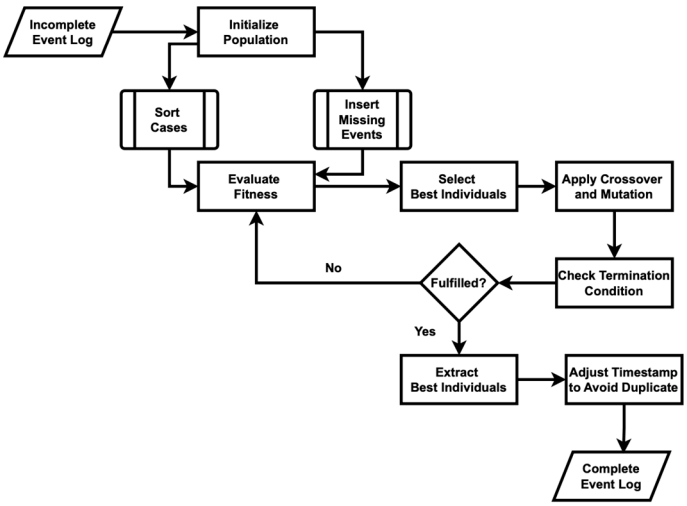


Figure 4. Flowchart of the original genetic process mining algorithm used to recover missing events.

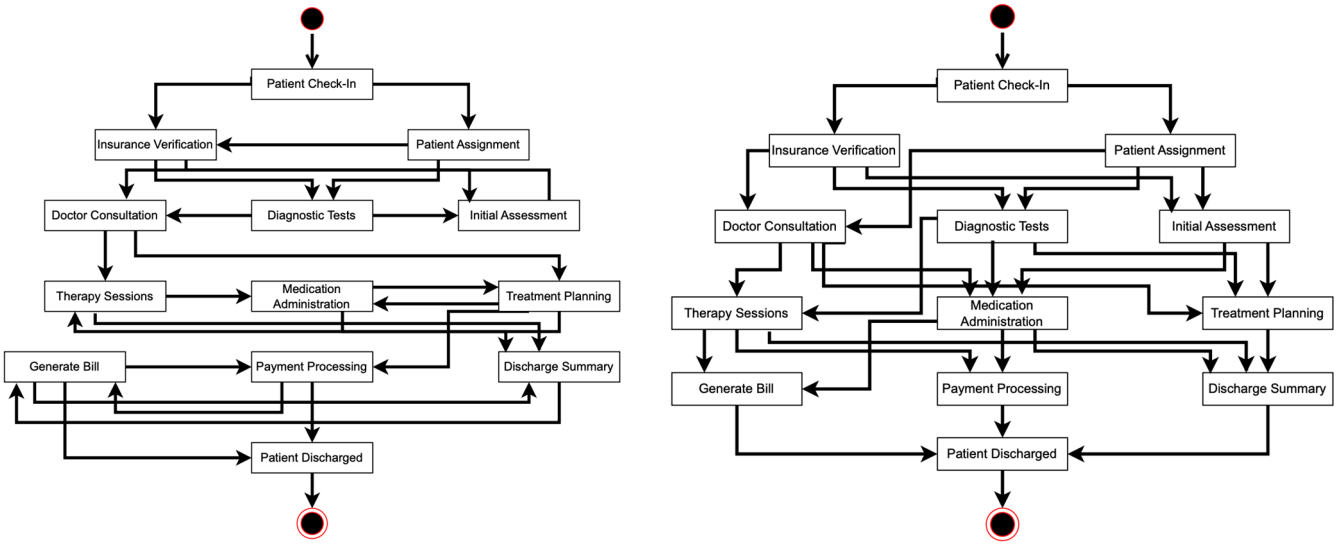
We ran Algorithm A2 using the data presented in Figure 2, utilizing only the starting timestamp, which resulted in a complete event log. The original algorithm was successful in recovering missing events from the incomplete event logs. However, because the original algorithm relies solely on a sequential approach, the timestamps for each recovered activity are often inaccurate and require adjustments to correct the initially generated timestamps. This corrected event log is then used as input to discover a process model. Figures 5 and 6a display the complete event log and the corresponding process model, respectively. According to Figure 5, the missing event "Patient Assignment" in HPE019 and the three missing events "Therapy Sessions", "Payment Processing", and "Discharge Summary" in HPE033 from Figure 2 have been fully recovered.

This algorithm has a time complexity that is primarily driven by its population initialization, fitness evaluation, and iterative evolution processes. The complexity of initializing the population is $O(P{\cdot}N{\cdot}(C\ log\ C + M{\cdot}C))$, where P is the population size, N is the number of cases, C is the average number of events per case, and M is the number of unique activities. During each generation, the fitness of each individual is evaluated with a complexity of $O(N{\cdot}M)$; this is followed by a sorting and selection of the top individuals, which contributes an additional $O(P\ log\ P)$. The crossover and mutation operations performed on the selected individuals have complexities of $O(N)$ and $O(N{\cdot}C\ log\ C)$, respectively. Given these steps, the overall time complexity of the genetic process in this algorithm over G generations is $O(G{\cdot}P{\cdot}N{\cdot}(M + C\ log\ C))$. This indicates that this algorithm's performance is significantly influenced by the number of generations, population size, and size of the event log. Therefore, the algorithm's complexity is primarily characterized by polynomial

time algorithm in Big-O notation. This means that while the algorithm may become computationally expensive as the input size grows, its growth rate is still polynomial, and therefore manageable for a wide range of practical applications.

CaseID	Activity	Timestamp
HPE019	Patient Check-In	2023-03-19 08.15.22
HPE019	**Patient Assignment**	**2023-03-19 08.20.33**
HPE019	Insurance Verification	2023-03-19 08.25.07
HPE019	Initial Assessment	2023-03-19 09.09.38
HPE019	Doctor Consultation	2023-03-19 09.14.44
HPE019	Diagnostic Tests	2023-03-19 09.20.33
HPE019	Treatment Planning	2023-03-19 09.54.46
HPE019	Medication Administration	2023-03-19 10.06.26
HPE019	Therapy Sessions	2023-03-19 10.31.09
HPE019	Payment Processing	2023-03-19 10.52.01
HPE019	Generate Bill	2023-03-19 10.56.48
HPE019	Discharge Summary	2023-03-19 10.58.27
HPE019	Patient Discharged	2023-03-19 11.22.48
HPE033	Patient Check-In	2023-04-02 07.50.02
HPE033	Insurance Verification	2023-04-02 08.00.02
HPE033	**Discharge Summary**	**2023-04-02 08.06.34**
HPE033	Patient Assignment	2023-04-02 08.22.30
HPE033	**Therapy Sessions**	**2023-04-02 08.31.49**
HPE033	Diagnostic Tests	2023-04-02 08.54.15
HPE033	Initial Assessment	2023-04-02 09.07.32
HPE033	Doctor Consultation	2023-04-02 09.14.09
HPE033	Treatment Planning	2023-04-02 09.44.36
HPE033	Medication Administration	2023-04-02 09.57.47
HPE033	Generate Bill	2023-04-02 10.42.10
HPE033	**Payment Processing**	**2023-04-02 11.05.35**
HPE033	Patient Discharged	2023-04-02 11.32.35

Figure 5. Complete event log of the hospital recovered using the original algorithm.



(**a**) (**b**)

Figure 6. Process model generated using complete event log from: (**a**) the original genetic process mining; and (**b**) The base model from the hospital.

Moreover, to evaluate our study, we need a base process model. We use the original and complete data from the hospital, as shown in Figure 2, as input in order to find an accurate base process model. Figure 6b shows the original process model from the hospital, which we use as our base process model for research analysis and validation.

Based on Figure 6b, the base process model has 26 relations, while the original algorithm discovers 24 relations. We then match the discovered relations from the original algorithm to the base model relations, identifying correct flows, incorrect flows, and missing flows. The original algorithm discovers 11 correct relations, 13 incorrect relations, and 15 missing relations. In our study, correct relations are those that appear in both the base model and the discovered model, aligned in terms of both nodes and direction. Incorrect relations are those that appear in the discovered model but do not have a corresponding relation in the base model. Missing relations are those specified in the base model but absent in the discovered model.

We calculate the fitness and precision scores as part of behavioral appropriateness and present them in Table 2. Based on the analysis of Table 2, we can take note of the performance of the original genetic process mining in terms of behavioral appropriateness for discovered process models and event logs. The original method identifies only 11 correct relations. Additionally, the original method has 13 false positives, indicating that it incorrectly identified several relations that were not present in the base model. Regarding false negatives, the original method missed 15 relations, showing that it was not able to capture all relevant relations from the base model, resulting in an inaccurate process model.

Table 2. Results for behavioral appropriateness for the original algorithm.

Metric	Relations Discovered by Original Genetic Process Mining
Correct/True Positives (*TP*)	11
False Positives (*FP*)	13
False Negatives (*FN*)	15
Fitness	0.423
Precision	0.458

Moreover, the fitness score calculated using Equation (1) of the original method is 0.423. This fitness score reflects the fact that the original algorithm does not provide a better representation of the base model's behavior, indicating that its discovered model does not align well with the actual process. Finally, the precision score of the original method, calculated using Equation (2), is 0.458, indicating that less than half of the discovered relations are accurate.

4.2. Timed Genetic Process Mining Algorithm

This modified algorithm is essentially an extension of the original genetic process mining method. In this modification, we make changes in two key areas: (1) utilizing dual-timestamp event log as input, and (2) modifying the steps of the original algorithm to be aware of timestamps, using them to recover missing events and discover a more accurate process model. The modified steps of the original algorithm are as follows:

1. Initialize population: generate an initial population of random process models based on the event log structure, with events containing starting and ending timestamps.
2. Fitness evaluation: include time-aware fitness measures that consider both starting and ending timestamps. This involves calculating the sequence, temporal, and concurrency fitness for each model in the population.
3. Selection, crossover, and mutation: apply genetic operations, with an emphasis on maintaining temporal consistency.
4. Iterate, evolve, terminate, and select best model: continue evolving the population until convergence or a stopping criterion is met. Select the best model from the population based on the highest fitness score.

By incorporating dual timestamps (starting and ending times), the modified genetic process mining algorithm can infer the likely presence of missing activities based on observed gaps between timestamps. This additional temporal information provides insights

into where and when activities are likely to have occurred, thereby improving the accuracy of the discovered process model.

Figure 7 shows the algorithm steps employed to recover missing events, which begin with reading the dual-timestamp event log data from an input file and ensuring the timestamps are in the correct date/time format. The average duration for each activity is then calculated to provide a basis for evaluating temporal gaps. The algorithm employs a fitness function that assesses a sequence of activities by comparing the gaps between the ending and starting times of consecutive activities against the expected average duration. The algorithm operates by first generating an initial population of potential sequences for the missing activities. Each sequence is evaluated for its fitness, and the best-performing sequences are selected for crossover and mutation to create a new generation. Crossover involves exchanging segments between pairs of parent sequences to produce children, while mutation introduces random changes to ensure diversity in the population. This process of selection, crossover, and mutation continues iteratively for a specified number of generations. In each iteration, the algorithm selects the best individual sequence from the population based on its fitness score. This individual represents the most likely sequence of missing activities. The algorithm ensures that the identified activities fit within the temporal constraints of the event log. The final best sequence from the last generation is then inserted into the event log, resulting in a complete and accurate representation of the process. The implementation of Figure 7 in code is shown in Algorithm A3 in Appendix A.

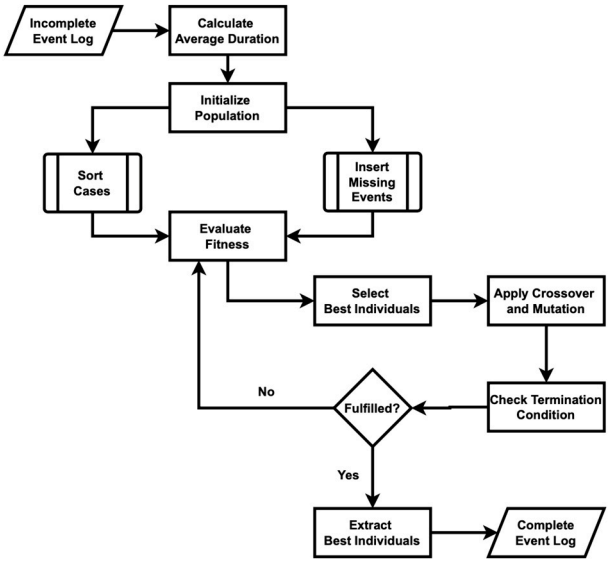


Figure 7. Flowchart of the timed genetic process mining algorithm used to recover missing events.

Algorithm A3 is run with the data in Figure 2 as input, and the result presents the complete event log. Our modified algorithm successfully recovers missing events from the incomplete event logs, along with their timestamps. The generated event log is used as an input to discover a process model. Figures 8 and 9 show the complete event log and the process model, respectively. Figure 8 shows that the one missing event in HPE019, which is activity "Patient Assignment", and the three missing events in HPE033, comprising activities "Therapy Sessions", "Payment Processing", and "Discharge Summary" identified in Figure 2 are now fully recovered.

The algorithm's Big-*O* complexity is also characterized as polynomial time complexities. This algorithm also exhibits a time complexity that is affected by the population size, sequence lengths, and the number of generations, but with some differences in the specifics of its operations. The initial preprocessing steps, including the reading of data and calculation of average activity durations, have a complexity of $O(N)$. The core genetic algorithm iterates over each unique case in the event log (K cases), sorting events and handling missing activities with a complexity of $O(N\ log\ N)$ for sorting and $O(M)$ for

identifying missing activities. The population initialization for each case is $O(P{\cdot}A)$, where A represents the number of missing activities. During each generation, fitness evaluation across the population has a complexity of $O(P{\cdot}S)$, with S being the average sequence length, and the crossover and mutation processes contribute $O(P{\cdot}(S+1))$. Thus, the overall time complexity for this algorithm is $O(K{\cdot}(N\ log\ N + G{\cdot}P{\cdot}S))$. This indicates that the modified algorithm's complexity is primarily influenced by the number of cases, sequence lengths, and the iterative genetic process across generations, which is similar to the original algorithm, but with variations in specific operations and their impacts.

CaseID	Activity	Start Timestamp	End Timestamp
HPE019	Patient Check-In	2023-03-19 08.15.22	2023-03-19 08.31.22
HPE019	Insurance Verification	2023-03-19 08.25.07	2023-03-19 08.53.07
HPE019	**Patient Assignment**	**2023-03-19 08.58.07**	**2023-03-19 09.08.07**
HPE019	Initial Assessment	2023-03-19 09.09.38	2023-03-19 09.14.38
HPE019	Doctor Consultation	2023-03-19 09.09.39	2023-03-19 09.33.39
HPE019	Diagnostic Tests	2023-03-19 09.20.33	2023-03-19 09.40.33
HPE019	Treatment Planning	2023-03-19 09.54.46	2023-03-19 10.24.46
HPE019	Medication Administration	2023-03-19 10.06.26	2023-03-19 10.21.26
HPE019	Therapy Sessions	2023-03-19 10.07.09	2023-03-19 10.31.09
HPE019	Payment Processing	2023-03-19 10.52.01	2023-03-19 11.08.01
HPE019	Generate Bill	2023-03-19 10.56.48	2023-03-19 11.22.48
HPE019	Discharge Summary	2023-03-19 10.58.27	2023-03-19 11.12.27
HPE019	Patient Discharged	2023-03-19 11.22.48	2023-03-19 11.31.48
HPE033	Patient Check-In	2023-04-02 07.50.02	2023-04-02 08.00.02
HPE033	Insurance Verification	2023-04-02 08.00.02	2023-04-02 08.30.02
HPE033	Patient Assignment	2023-04-02 08.22.30	2023-04-02 08.51.30
HPE033	Diagnostic Tests	2023-04-02 08.54.15	2023-04-02 09.15.15
HPE033	Initial Assessment	2023-04-02 09.07.32	2023-04-02 09.18.32
HPE033	Doctor Consultation	2023-04-02 09.14.09	2023-04-02 09.34.09
HPE033	Treatment Planning	2023-04-02 09.44.36	2023-04-02 10.08.36
HPE033	Medication Administration	2023-04-02 09.57.47	2023-04-02 10.27.47
HPE033	**Therapy Sessions**	**2023-04-02 10.04.47**	**2023-04-02 10.30.10**
HPE033	Generate Bill	2023-04-02 10.42.10	2023-04-02 11.05.10
HPE033	**Discharge Summary**	**2023-04-02 11.05.35**	**2023-04-02 11.12.35**
HPE033	**Payment Processing**	**2023-04-02 11.17.35**	**2023-04-02 11.27.35**
HPE033	Patient Discharged	2023-04-02 11.32.35	2023-04-02 11.42.35

Figure 8. Complete event log of the hospital recovered using the modified algorithm.

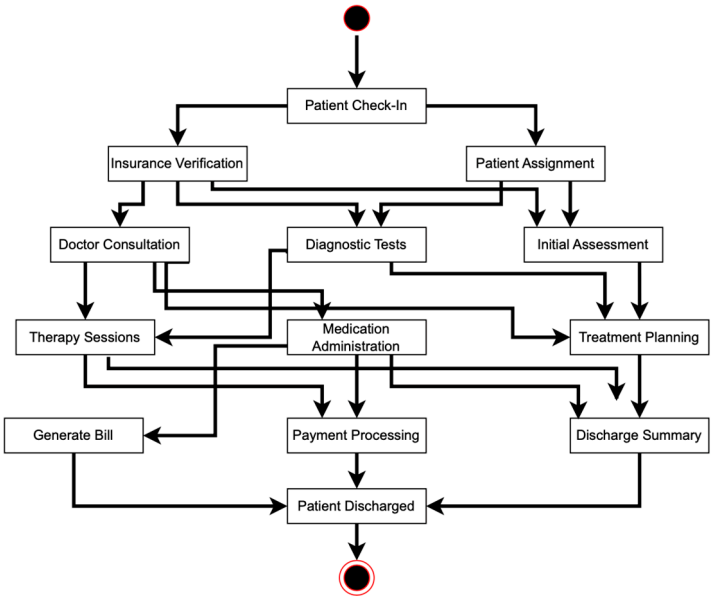


Figure 9. Process model generated using complete event log from the modified algorithm.

Based on the Big-*O* complexity analysis for both algorithms, the modified algorithm generally has a better (lower) complexity than the original algorithm. Therefore, the modified algorithm is more efficient, particularly when the number of unique activities or the average number of events per case in the original algorithm is high, resulting in higher complexity. The modified algorithm's reliance on sequence lengths and simpler operations makes it preferable in scenarios where the event log data is large or the number of activities is substantial, allowing it to perform better in terms of computational complexity.

We also calculated the fitness and precision scores using Equations (1) and (2) as part of behavioral appropriateness, and these are presented in Table 3. Analyzing Table 3, and referring to Table 2 for comparison, allows us to draw several conclusions about the performance of the timed genetic process mining method in terms of behavioral appropriateness for discovered process models and event logs.

Table 3. Results for behavioral appropriateness for the modified algorithm.

Metric	Relations Discovered by Timed Genetic Process Mining
Correct/True Positives (*TP*)	22
False Positives (*FP*)	0
False Negatives (*FN*)	4
Fitness	0.846
Precision	1.000

Firstly, the timed genetic process mining method significantly outperforms the original method in identifying correct relations, discovering 22 correct relations, compared to the original method's 11. This indicates a clear advantage of the timed method in accurately capturing the true behavior of the process. Additionally, the timed method shows notable improvement in avoiding false positives, reporting zero false positives, meaning it did not identify any incorrect relations that did not exist in the base model. In contrast, the original method had 13 false positives, indicating it incorrectly identified several relations not present in the base model. The timed method also excels in minimizing false negatives, missing only four correct relations, whereas the original method missed fifteen. This suggests that the timed method is better at capturing all relevant relations from the base model, resulting in a more complete and accurate process model.

Moreover, the fitness score of the timed method is significantly higher, at 0.846, compared to the original method's score of 0.423. This higher fitness score reflects the fact that the timed method provides a better representation of the base model's behavior, showing that its discovered model aligns more closely with the actual process. Finally, the precision score of the timed method is perfect, at 1.000, meaning all discovered relations are correct. In contrast, the original method's precision score is 0.458, indicating that less than half of the discovered relations are accurate. This highlights the superior accuracy of the timed method in process model discovery.

In conclusion, the timed genetic process mining method demonstrates much higher effectiveness and reliability compared to the original method. It excels in identifying correct relations, avoiding incorrect ones, and capturing the true behavior of the process, making it a more robust tool for process model discovery and event log reconstruction.

To evaluate our research, as explained in Section 3, we measured the behavioral and structural appropriateness of the discovered process models and event logs. Based on Figures 6 and 9, we can list the discovered relations from each process model. The base process model has 26 relations, the modified algorithm finds 22 relations, and the original algorithm discovers 24 relations. We matched the discovered relations from both algorithms to the base model relations, showing that our modified algorithm presents more correct flows and fewer incorrect and missing flows compared to the original algorithm. The results show that the modified algorithm presents twenty-two correct relations, zero incorrect relations, and four missing relations. Meanwhile, the original algorithm discovers 11 correct relations, 13 incorrect relations, and 15 missing relations. All of the relations are presented in Figure 10.

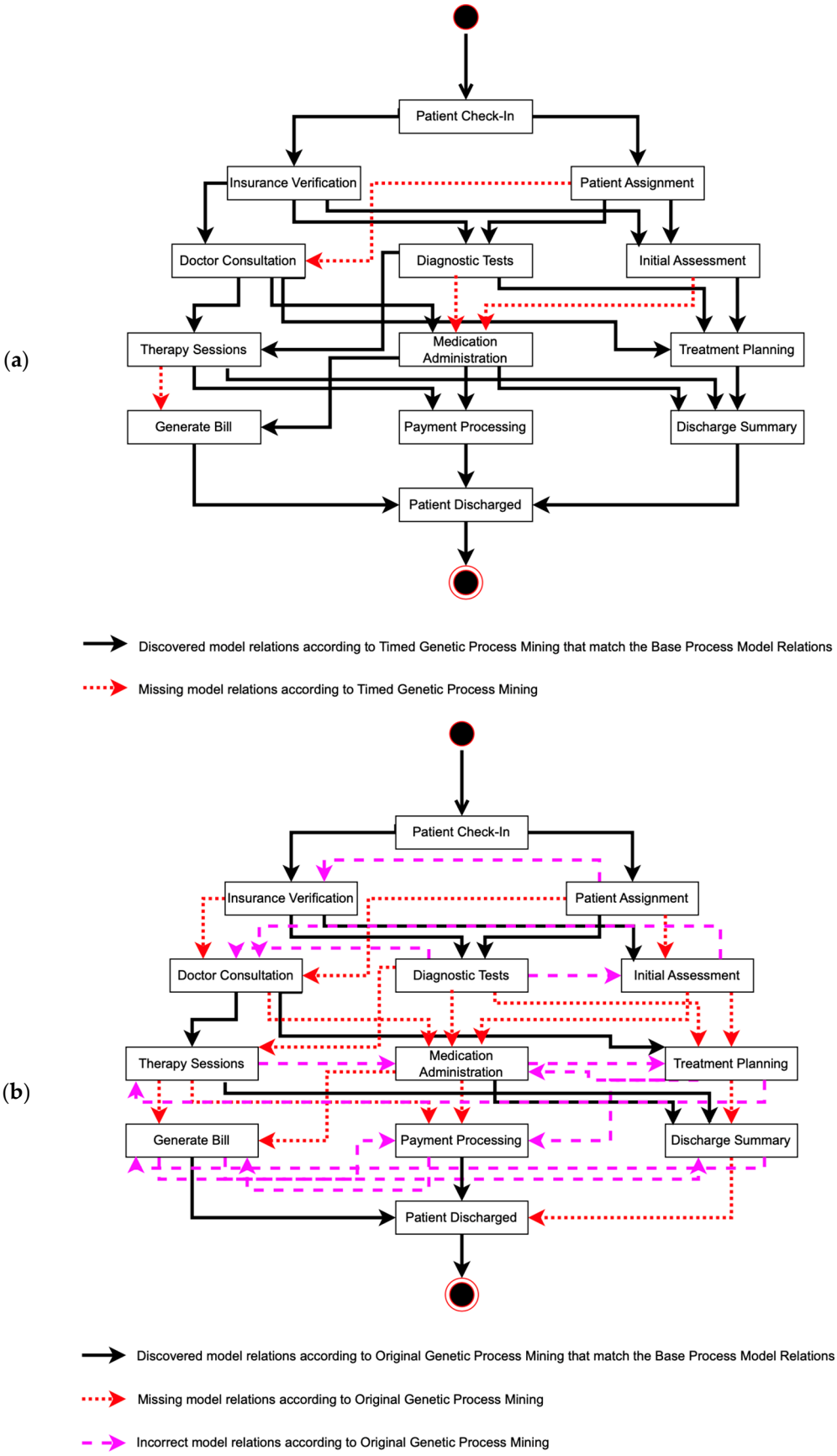


Figure 10. Discovered, incorrect, and missing relations, compared to the base process models' relations, from (**a**) timed genetic process mining and (**b**) original genetic process mining.

5. In-Depth Comparison of Experiments and Analysis

Furthermore, we calculate the process model coverage using Equations (3) and (4) as part of structural appropriateness and present the results in Table 4. Based on the data presented in Table 4, we can analyze and draw conclusions about the process model coverage of the timed genetic process mining compared to the original genetic process mining. In terms of discovered relations, the timed genetic process mining identified 22 relations, while the original genetic process mining discovered 24 relations. However, the quality of these discovered relations varies significantly between the two methods. The timed algorithm correctly identified all 22 of its discovered relations, whereas the original algorithm correctly identified only 11 out of its 24 discovered relations. Additionally, the timed algorithm did not have any incorrect relations, achieving perfect accuracy in identifying relevant relations from the base model. In contrast, the original algorithm had 13 incorrect relations, indicating a significant number of false positives in which relations were identified that did not exist in the base model.

Table 4. Results for process model coverage.

Relations Discovered by Timed Genetic Process Mining	Relations Discovered by Original Genetic Process Mining
Coverage Score: 84.60% Error Score: 0.00%	Coverage Score: 42.30% Error Score: 54.20%

In examining the coverage score, which measures how well the discovered model captures the base model's behavior, it can be seen that the timed algorithm achieved a coverage score of 84.60%, compared to the original algorithm's coverage score of 42.30%. This shows that the timed algorithm is much more effective at covering the behavior of the base model. Furthermore, the error score, which indicates the proportion of incorrect relations, was 0.00% for the timed algorithm, reflecting its high accuracy. In contrast, the original algorithm had an error score of 54.20%, highlighting the presence of several incorrect relations in its discovered model. In conclusion, the data in Table 4 demonstrates that the timed genetic process mining algorithm significantly outperforms the original genetic process mining algorithm in terms of process model coverage. The timed algorithm not only identifies correct relations with higher accuracy but also avoids incorrect relations, resulting in a more precise and reliable representation of the base model. The higher coverage score and error score of zero further validate the superior performance of the timed method in process model discovery.

In addition, Table 5 presents the analysis for process model comparisons in terms of node correspondence, edge correspondence, flow sequence and frequency, and visual structure of the process models. This analysis also shows that our modified algorithm can result in a process model with a visual structure similar to that of the base process model of the hospital.

According to all comparison results, the timed genetic process mining approach, which utilizes timestamp information, provides better results in both structural and behavioral appropriateness, compared to the original genetic process mining approach, which relies on a sequential approach. This improvement is largely due to the ability of the dual-timestamp approach to capture temporal relationships between events in a more detailed and accurate manner.

In terms of behavioral appropriateness, the dual-timestamp approach allows for capturing more nuanced temporal relationships between events. By considering both starting and ending timestamps, the modified algorithm can better discern the true sequence of activities and their dependencies in the process, as well as the possibility of parallel occurrences. This typically leads to process models that are more accurate and that reflect the actual behavior observed in the event log. In contrast, the sequential approach used by the original algorithm might overlook complex dependencies or concurrent activities

that occur within the process. It may struggle to accurately reconstruct the exact behavior, especially when there are overlaps or variations in activity durations.

Table 5. Discovered traces from each generated complete event log.

	Process Model Discovered by Timed Genetic Process Mining	**Process Model Discovered by Original Genetic Process Mining**
Node correspondence	Both discovery models include all nodes from the base model. (13 events)	
Edge correspondence	- Correct flows: 22 - Incorrect flows: 0 - Missing flows: 4	- Correct flows: 11 - Incorrect flows: 13 - Missing flows: 15
Flow sequence and frequency	- Matches many correct flows but is not able to discover all flows. - Frequencies of some flows differ from the base model.	- Matches many correct flows, but there are also many additional/incorrect flows. - Frequencies of some flows differ from the base model.
Visual structure	Has a structure closer to the base model with fewer missing flows, making it visually more similar to the base model, including both sequence and parallel processes.	Has a structure with additional incorrect flows, making it visually less similar to the base model.

Regarding structural appropriateness, the dual-timestamp approach in the modified algorithm often results in process models that are more structured and coherent. This is because it can differentiate between parallel activities, distinguish between activities based on their lifecycle stages, and generally provide a clearer representation of the process flow. While sequential mining, as in the original algorithm, can be effective for simple processes with straightforward sequences, it may produce models which are less structured for processes that involve concurrency or complex dependencies. This can lead to models that are harder to interpret or less accurate in capturing the actual structure of the process.

Overall, by leveraging dual timestamps, the timed genetic process mining algorithm can potentially recover missing events more accurately. It enhances the algorithm's ability to reconstruct the chronological order of activities and their relationships, thereby improving both behavioral fidelity and structural clarity. While the original algorithm is straightforward and easier to implement, it may not perform as well when faced with event logs containing concurrency or overlapping activities, or events with varying lifecycle stages.

In summary, timed genetic process mining with a dual-timestamp approach presents better results in terms of both structural and behavioral appropriateness, compared to original genetic process mining with a sequential approach. The dual-timestamp approach enhances the algorithm's capability to handle complex process behavior and structural intricacies, leading to more accurate and robust process models.

While this research acknowledges the complexities of real-world event logs compared to the simplified case studies, we simplify the analysis to focus on the essential properties of event logs, namely, Case ID, Activity, and Timestamp, as these three elements are crucial for reconstructing process models and analyzing behaviors, providing a foundational understanding necessary for effective process mining techniques [1,2]. This study also examines the structural or sequential ordering of activity execution, which is vital for enhancing the ability to recover missing events from periodic and asynchronous systems. By analyzing the sequence and structure of activities, the research shows how incomplete logs can still yield accurate process models. This capability is particularly important in environments where events are logged periodically or asynchronously, resulting in gaps in the recorded data. The study's findings suggest methods for bridging these gaps by

generating temporary event logs using the proposed algorithm until the actual logs are entered into the system.

6. Advanced Metrics for Conformance Checking

In this section, we discuss several advanced metrics for conformance checking to validate the process models and event logs discovered by both algorithms. Although the main goal of this research is to recover missing events in the event log by filling gaps created by the temporary absence of activities until the actual logs are later entered into the system, comparing a process model with an event log is also an important aspect of research validation. This comparison helps determine how well the observed reality (as captured in the event log) aligns with the expected or designed behavior (as specified in the process model).

Firstly, we analyze the traces discovered by both algorithms, in comparison to the base traces of the hospital, using Python's "difflib" module, specifically the sequence matcher [48,49]. Figure 11 illustrates the sequence matcher used in this study. The process of calculating custom similarity scores between traces begins with the loading of the data, which consist of lists of traces from the base model, the modified algorithm, and the original algorithm. Each trace is then split into sequences of activities, allowing for a detailed breakdown into manageable and comparable segments. The core of the algorithm involves the computation of similarity scores between the base traces and the traces generated by both algorithms using a similarity function. These computed similarity scores are subsequently stored and analyzed. The process concludes with the final similarity scores, which are made available for further analysis. The results indicate that the modified algorithm produces higher similarity scores than the original algorithm, as shown in Table 6.

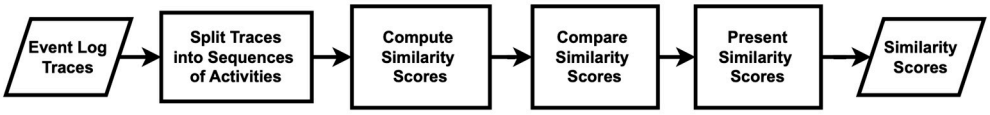


Figure 11. Flowchart of the measure of trace similarity scores using the sequence-matcher method.

Table 6. Results of advanced metrics for conformance checking.

Conformance Metrics	Timed Genetic Process Mining	Original Genetic Process Mining
Trace similarity scores using sequence matcher	0.9320	0.9070
Trace similarity scores using Levenshtein distance	0.6804	0.6728
Cost-based fitness analysis	0.7572	0.7464
Probabilistic trace alignment	100%	80%

Secondly, trace similarity is measured using the Levenshtein distance, which calculates the minimum number of insertions, deletions, and substitutions required to transform one sequence into another using dynamic programming [50]. Figure 12 illustrates the method used to generate trace similarity scores with this approach. The algorithm begins by calculating the Levenshtein distance between individual traces to determine their similarity. It then computes a similarity score based on this distance, normalizes it to a range between 0 and 1, and uses these scores to build a similarity matrix for all models. This matrix provides insight into how closely the traces from different models resemble each other. The results show that the modified algorithm achieves higher similarity scores, compared to the original algorithm, as demonstrated in Table 6.

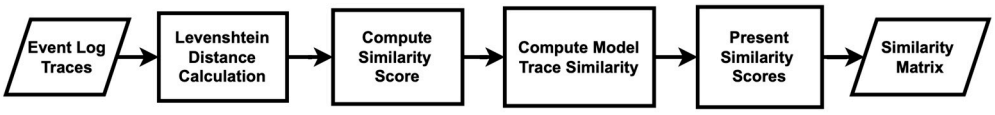


Figure 12. Flowchart of the measure of trace similarity scores using the Levenshtein distance.

Thirdly, we calculate conformance checking using cost-based fitness analysis, as outlined in [51]. This structured approach assesses how well an event log aligns with a process model. Figure 13 illustrates the cost-based fitness analysis used in this study. The fitness metric measures the degree to which observed behavior conforms to modeled expectations, incorporating a cost-based dimension to account for deviations such as skipped or inserted activities. The preparation involves defining the event logs and process models for both algorithms. Following the preparation, the analysis involves defining cost functions for insertions and skips, replaying the event log against the process model to identify deviations, and applying the A* algorithm to find the optimal alignment with minimal cost. The fitness metric is then calculated as one minus the ratio of the total deviation cost to the maximum possible cost, with a higher value indicating better conformance. By analyzing the results and repeating the process for all traces in the event log, a comprehensive fitness score can be derived.

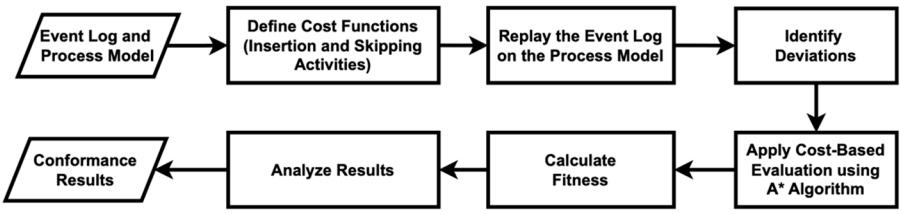


Figure 13. Flowchart of the measure of conformance checking scores using the cost-based fitness analysis.

In this research, we calculate the cost functions for skipping and inserting activities based on historical data analysis to determine the importance and impact of these activities within the process. The costs for inserting each activity are specified as follows: "Patient Check-In" and "Patient Assignment" each have a cost of 1, "Insurance Verification" and "Initial Assessment" each have a cost of 2, and "Therapy Sessions" and "Treatment Planning" are each assigned a cost of 3. If an activity is not listed, a default insertion cost of 1 is applied. Similarly, the skipping costs for various activities are defined: "Patient Check-In" has a cost of 1; "Patient Assignment" and "Insurance Verification" each have a cost of 2; "Initial Assessment", "Diagnostic Tests", and "Doctor Consultation" also each have a cost of 2; "Therapy Sessions", "Treatment Planning", "Medication Administration", and "Discharge Summary" each have a cost of 3; and "Generate Bill", "Payment Processing", and "Patient Discharged" are each assigned a cost of 1. If an activity is not specified in this cost mapping, a default cost of 1 is used. Table 6 presents the conformance results for both algorithms.

Lastly, ref. [52] introduces a conformance checking approach based on trace alignments using Stochastic Workflow Nets (SWNs). Figure 14 shows the flowchart of the method used in this study. Unlike traditional conformance checking methods that provide a numerical score, this approach ranks trace alignments based on two factors: the cost of alignment and the probability of the model trace generated by the stochastic model. The alignment cost is calculated using the Levenshtein distance. This cost is then weighted by the probability of each model trace.

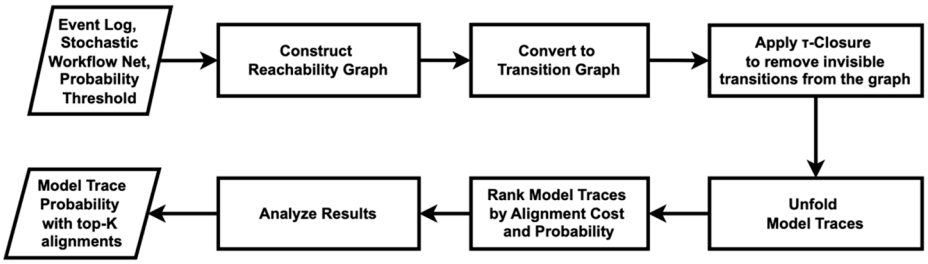


Figure 14. Flowchart of the measure of conformance checking scores using the probabilistic trace alignment.

The algorithm in this study begins by defining the SWN using a set of transitions and their associated probabilities, which are derived from historical data. It then constructs a reachability graph to represent all possible states and transitions within the process. This reachability graph is converted into a transition graph, where a τ-closure is applied to remove any invisible transitions (τ-transitions), simplifying the model's behavior for clearer analysis. The algorithm unfolds all possible model traces from the transition graph that meet a certain probability threshold—set at 0.1 for this study—to ensure that only the most probable traces are considered for alignment with the event log traces.

To assess conformance, the algorithm formulates the problem as a k-Nearest Neighbors (kNN) search to find the top-k alignments between the log traces and the model traces. These alignments are then ranked using two strategies: a brute-force method for exact ranking and an approximate method that employs a KD-Tree (K-Dimensional Tree) to reduce computation time while still providing useful results. In this research, the number of top alignments to return is set to $k = 5$. The result is a set of top-ranked alignments that provide insight into how closely the observed event log traces conform to the expected behavior defined by the process model. Table 6 shows the results for both algorithms.

The data in Table 6 compares the performance of two genetic process mining approaches: timed genetic process mining and original genetic process mining, evaluated using several advanced conformance metrics. These metrics include trace similarity scores using a sequence matcher, trace similarity scores using Levenshtein distance, cost-based fitness analysis, and probabilistic trace alignment.

The first metric, trace similarity scores using a sequence matcher, assesses the similarity function by computing scores between the base traces and the traces from both algorithms. The timed genetic process mining approach achieved a higher similarity score of 0.9320, compared to 0.9070 for the original genetic process mining. This result indicates that the timed genetic process mining provides better alignment to the observed traces than does the original genetic process mining. The second metric, trace similarity scores using Levenshtein distance, further evaluates trace alignment by quantifying the "distance" between two traces by measuring the number of insertions, deletions, and substitutions needed to transform one trace into another. Here, timed genetic process mining also outperforms original genetic process mining, with a score of 0.6804 versus 0.6728. Although the difference is slight, it suggests a marginally better performance by the timed approach in capturing trace similarities.

Furthermore, the third metric, cost-based fitness analysis, measures how well the model conforms to the observed traces by penalizing deviations based on a cost function. Timed genetic process mining achieved a score of 0.7572, which was slightly higher than the original genetic process mining score of 0.7464. This indicates that the timed genetic process mining is somewhat better at fitting the model to the event logs while minimizing deviations. The final metric, probabilistic trace alignment, evaluates the model's ability to account for various trace variations and their likelihood within the event logs. Timed genetic process mining achieved a perfect score of 100%, outperforming the original genetic process mining, which scored 80%. This significant difference highlights the superiority of the timed genetic process mining in modeling the probabilistic nature of trace variations.

In summary, across all conformance metrics, timed genetic process mining demonstrates better performance than original genetic process mining. The improvements suggest that incorporating time aspects into the genetic process mining technique enhances its ability to create a more accurate and better-fitting model for the event logs in this context.

7. Conclusions

This research proposes a modified approach to the genetic process mining algorithm, aiming to recover missing events from incomplete event logs. The modified algorithm, referred to as timed genetic process mining, extends the original method with two key enhancements, using dual-timestamp event logs as input and incorporating timestamp awareness into the algorithm's steps to recover missing events and generate a complete event log, as well as to discover a more accurate process model.

Our experimental results demonstrate the commendable behavioral and structural appropriateness of both the event log and process model results compared to the original genetic process mining method. The modified approach yields higher fitness and precision scores, visually similar process model comparisons, and good coverage with no errors in process models. Additionally, we discuss several advanced metrics for conformance checking, including trace similarity scores using a sequence matcher, trace similarity scores using Levenshtein distance, cost-based fitness analysis, and probabilistic trace alignment, to validate the process models and event logs discovered by both algorithms. In brief, timed genetic process mining outperforms original genetic process mining across all conformance metrics. Possible future work could involve the integration of machine learning techniques to enhance the prediction and recovery of missing events, as well as the refining of the fitness evaluation criteria to incorporate more complex temporal and causal relationships.

Author Contributions: This research represents the collective intellectual effort of the entire team. Y.A.E. and M.K. conceptualized the study design; Y.A.E. developed the methodology, and worked on the software, visualization, and writing—original draft preparation; Y.A.E. and M.K. conducted research validation and formal analysis; M.K. worked on writing—review and editing, supervision, and funding acquisition. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant, funded by the government of the Republic of Korea (MSIT) (RS-2023-00242528).

Data Availability Statement: Experimental data and code related to this paper can be obtained by contacting the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

Algorithm A1. Pseudocode of genetic process mining used to discover a process model.

```
Initialize population P of process models
Evaluate fitness of each model in P based on event log data
while termination criteria not met do:
    Select parents from P based on fitness
    Apply crossover and mutation operators to create offspring
    Evaluate fitness of offspring based on event log data
    Replace least-fit models in P with offspring
end while
Select best model from P as the discovered process model
```

Algorithm A2. Genetic process mining algorithm used to recover missing events.

```
# Function to initialize population of event logs
def initialize_population(event_log, population_size = 50):
    population = []
    unique_activities = event_log['Activity'].unique()
    for _ in range(population_size):
        individual_log = event_log.copy()
        for case_group in individual_log grouped by 'CaseID':
            sort case_group by 'Timestamp'
            case_activities = list of 'Activity' from case_group
            case_timestamps = list of 'Timestamp' from case_group
            existing_activities = set of case_activities
            missing_activities = unique_activities − existing_activities

            # Function to insert missing activities
            for missing_activity in missing_activities:
                insertion_position = random position in case_activities
                insert missing_activity at insertion_position
                insert corresponding timestamp at insertion_position
            updated_rows = DataFrame with updated case_activities and case_timestamps
            append updated_rows to individual_log
        append sorted individual_log to population
    return population
# Function to evaluate fitness score of an individual event log
def evaluate_fitness_score(individual_log, reference_event_log):
    unique_activities_set = set of unique activities in reference_event_log
    fitness_score = 0
    for case_group in individual_log grouped by 'CaseID':
        reference_activities = set of activities in reference case_group
        recovered_activities = set of activities in current case_group
        fitness_score += number of recovered_activities in unique_activities_set −
reference_activities
    return fitness_score
# Function to select the fittest individuals from the population
def select_fittest(population, fitness_scores, selection_count):
    sorted_indices = indices of fitness_scores sorted in descending order
    fittest_individuals = top selection_count individuals from population using sorted_indices
    return fittest_individuals
# Function to perform crossover between two parents to produce offspring
def perform_crossover(parent1, parent2):
    offspring1, offspring2 = copies of parent1 and parent2
    for case_id in parent1:
        if random value > 0.5:
            temp_case1 = parent2's case with case_id
            temp_case2 = parent1's case with case_id
            replace offspring1's case with temp_case1
            replace offspring2's case with temp_case2
    return offspring1, offspring2
# Function to perform mutation on an individual event log
```

Algorithm A2. *Cont.*

```
def perform_mutation(individual_log, mutation_rate = 0.1):
    unique_activities = unique activities in individual_log
    for case_group in individual_log grouped by 'CaseID':
        if random value < mutation_rate:
            sort case_group by 'Timestamp'
            case_activities = list of activities in case_group
            case_timestamps = list of timestamps in case_group
            mutation_position = random position in case_activities
            new_activity = random choice from unique_activities
            replace activity at mutation_position with new_activity
            updated_rows = DataFrame with updated case_activities and case_timestamps
            update individual_log with updated_rows
    return sorted individual_log
# Genetic algorithm to recover missing activities
def genetic_process_recovery(event_log, generations = 50, population_size = 50, mutation_rate =
0.1):
    population = initialize_population(event_log, population_size)
    for _ in range(generations):
        fitness_scores = list of fitness scores for each individual_log in population
        selected_population = select_fittest(population, fitness_scores, selection_count =
population_size // 2)
        next_generation = []
        while len(next_generation) < population_size:
            parent1, parent2 = random sample of 2 from selected_population
            offspring1, offspring2 = perform_crossover(parent1, parent2)
            perform_mutation(offspring1, mutation_rate)
            perform_mutation(offspring2, mutation_rate)
            next_generation.append(offspring1)
            next_generation.append(offspring2)
        population = next_generation
    best_individual_log = individual_log with highest fitness score in population
    return best_individual_log
```

Algorithm A3. Timed genetic process mining algorithm used to recover missing events.

```
# Function to read data from input file
def read_data(file_path):
    event_log['Start Timestamp'] = pd.to_datetime(event_log['Start Timestamp'])
    event_log['End Timestamp'] = pd.to_datetime(event_log['End Timestamp'])
    event_log.columns = ['CaseID', 'Activity', 'Start Timestamp', 'End Timestamp']
# Calculate average duration of each activity
average_durations = {}
grouped_event_log = event_log.groupby('Activity')
for activity, group in grouped_event_log:
    durations = group['End Timestamp'] − group['Start Timestamp']
    average_duration = durations.mean()
    average_durations[activity] = average_duration
# Fitness function to evaluate a sequence based on temporal gaps
def evaluate_fitness(sequence, start_times, end_times):
    fitness_score = 0
    for i in range(len(sequence) − 1):
        if i < len(start_times) − 1:
            expected_gap = average_durations[sequence[i + 1]]
            if start_times[i + 1] − end_times[i] >= expected_gap:
                fitness_score += 1
    return fitness_score
```

Algorithm A3. *Conts.*

```
# Function to perform crossover between two parents
def perform_crossover(parent1, parent2):
    if len(parent1) > 2 and len(parent2) > 2:
        crossover_point = random.randint(1, len(parent1) − 2)
        offspring1 = parent1[:crossover_point] + parent2[crossover_point:]
        offspring2 = parent2[:crossover_point] + parent1[crossover_point:]
        return offspring1, offspring2
    else: return parent1, parent2
# Function to perform mutation on a sequence
def perform_mutation(sequence, activity_pool):
    if len(sequence) > 1:
        mutation_index = random.randint(0, len(sequence) − 1)
        sequence[mutation_index] = random.choice(activity_pool)
    return sequence
# Function to create an initial population of sequences
def initialize_population(size, missing_activities):
    population = []
    for _ in range(size):
        sequence = random.sample(missing_activities, len(missing_activities))
        population.append(sequence)
    return population
# Genetic algorithm to recover missing activities in an event log
def genetic_process_recovery(event_log, generations = 50, population_size = 50):
    all_activities = event_log['Activity'].unique().tolist()
    recovered_log = []
    for case_id, case_group in event_log.groupby('CaseID'):
        case_group = case_group.sort_values(by = 'Start Timestamp')
        case_activities = case_group['Activity'].tolist()
        start_times = case_group['Start Timestamp'].tolist()
        end_times = case_group['End Timestamp'].tolist()
        if case_activities[0] != all_activities[0]:
            initial_activity = all_activities[0]
            case_activities.insert(0, initial_activity)
            avg_duration = average_durations[initial_activity]
            start_times.insert(0, start_times[0] − avg_duration)
            end_times.insert(0, start_times[0] + avg_duration)
        if case_activities[−1] != all_activities[−1]:
            final_activity = all_activities[−1]
            case_activities.append(final_activity)
            avg_duration = average_durations[final_activity]
            start_times.append(end_times[−1])
            end_times.append(start_times[−1] + avg_duration)
        existing_activities = set(case_activities)
        missing_activities = [activity for activity in all_activities if activity not in
existing_activities]
        # Initialize population
        population = initialize_population(population_size, missing_activities)
        for _ in range(generations):
            # Evaluate fitness of each individual in the population
            fitness_scores = [evaluate_fitness(individual, start_times, end_times) for individual
in population]
```

Algorithm A3. *Conts.*

```
            # Select the best individuals
            max_fitness = max(fitness_scores)
            selected_population = [population[i] for i in range(len(population)) if
fitness_scores[i] == max_fitness]
            # Generate new population through crossover and mutation
            new_population = []
            while len(new_population) < population_size:
                parent1, parent2 = random.sample(selected_population, 2)
                offspring1, offspring2 = perform_crossover(parent1, parent2)
                new_population.extend([
                    perform_mutation(offspring1, missing_activities),
                    perform_mutation(offspring2, missing_activities) ])
            population = new_population
        # Initialize variable to keep track of the best individual and its fitness score
        best_individual = None
        max_fitness_score = -float('inf')
        # Iterate through each individual in the population
        for individual in population:
         # Calculate the fitness score for the current individual
         fitness_score = evaluate_fitness(individual, start_times, end_times)
         # Update the best individual if the current fitness score is higher than the max fitness
score
         if fitness_score > max_fitness_score:
             max_fitness_score = fitness_score
             best_individual = individual
```

References

1. Van der Aalst, W.M.P. Process mining: Data science in action. In *Process Mining: Data Science in Action*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 140–178. [CrossRef]
2. Process Mining Book. Available online: https://fluxicon.com/book/read/dataext/ (accessed on 1 September 2024).
3. Van Midden, Y. Using process mining and event log analysis for better business strategy decision-making. In Proceedings of the 35th Twente Student Conference on IT, Enschede, The Netherlands, 2 July 2021.
4. Yang, H.; van Dongen, B.F.; ter Hofstede, A.H.M.; Wynn, M.T.; Wang, J. Estimating completeness of event logs. *BPM Rep.* **2012**, *1204*, 12.
5. Wang, L.; Fang, X.; Shao, C. Discovery of Business Process Models from Incomplete Logs. *Electronics* **2022**, *11*, 3179. [CrossRef]
6. Butt, N.A.; Mahmood, Z.; Sana, M.U.; Díez, I.d.l.T.; Galán, J.C.; Brie, S.; Ashraf, I. Behavioral and Performance Analysis of a Real-Time Case Study Event Log: A Process Mining Approach. *Appl. Sci.* **2023**, *13*, 4145. [CrossRef]
7. Li, C.; Ge, J.; Wen, L.; Kong, L.; Chang, V.; Huang, L.; Luo, B. A novel completeness definition of event logs and corresponding generation algorithm. *Expert Syst.* **2020**, *37*, e12529. [CrossRef]
8. Bowman, S. Impact of electronic health record systems on information integrity: Quality and safety implications. *Perspect. Health Inf. Manag.* **2013**, *10*, 1c.
9. Laplante, P.A.; Ovaska, S.J. *Real-Time Systems Design and Analysis*; IEEE: Piscataway, NJ, USA, 2012; Volume 3, pp. 154–196.
10. Potter, S.; Nieh, J. Reducing downtime due to system maintenance and upgrades. In Proceedings of the 19th Large Installation System Administration Conference, San Diego, CA, USA, 4–9 December 2005; Volume 19, pp. 1–15.
11. Berman, B.A.; Dismukes, R.K.; Jobe, K.K. *Performance Data Errors in Air Carrier Operations: Causes and Countermeasures*; National Aeronautics and Space Administration, Ames Research Center: Moffett Field, CA, USA, 2012; pp. 7–11.
12. Cascio, W.F.; Montealegre, R. How Technology Is Changing Work and Organizations. *Annu. Rev. Organ. Psychol. Organ. Behav.* **2016**, *3*, 349–375. [CrossRef]
13. Kock, N. Asynchronous and distributed process improvement: The role of collaborative technologies. *Inf. Syst. J.* **2001**, *11*, 87–110. [CrossRef]
14. Effendi, Y.A.; Minsoo, K. Refining Process Mining in Port Container Terminals Through Clarification of Activity Boundaries With Double-Point Timestamps. *ICIC Express Lett. Part B Appl.* **2024**, *15*, 61–70. [CrossRef]
15. Nguyen, O.T.; Alishahi Tabriz, A.; Huo, J.; Hanna, K.; Shea, C.M.; Turner, K. Impact of Asynchronous Electronic Communication-Based Visits on Clinical Outcomes and Health Care Delivery: Systematic Review. *J. Med. Internet Res.* **2021**, *23*, e27531. [CrossRef]
16. De Medeiros, A.; Weijters, A.; Van der Aalst, W.M.P. *Using Genetic Algorithms to Mine Process Models: Representation, Operators and Results*; Beta Working Paper Series, WP 124; Eindhoven University of Technology: Eindhoven, The Netherlands, 2004.

17. Huser, V. Process Mining: Discovery, Conformance and Enhancement of Business Processes. *J. Biomed. Inform.* **2012**, *45*, 1018–1019. [CrossRef]
18. Jans, M.; De Weerdt, J.; Depaire, B.; Dumas, M.; Janssenswillen, G. Conformance Checking in Process Mining. *Inf. Syst.* **2021**, *102*, 101851. [CrossRef]
19. Buijs, J.C.A.M.; van Dongen, B.F.; van der Aalst, W.M.P. On the role of fitness, precision, generalization and simplicity in process discovery. In *On the Move to Meaningful Internet Systems: OTM 2012*, 2nd ed.; Meersman, R., Panetto, H., Dillon, T., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., Cruz, I.F., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7565, pp. 305–322.
20. De Leoni, M. Foundations of Process Enhancement. In *Process Mining Handbook. Lecture Notes in Business Information Processing*; Van der Aalst, W.M.P., Carmona, J., Eds.; Springer: Cham, Switzerland, 2022; Volume 448. [CrossRef]
21. Ayo, F.E.; Folorunso, O.; Ibharalu, F.T. A probabilistic approach to event log completeness. *Expert Syst. Appl.* **2017**, *80*, 263–272. [CrossRef]
22. Yang, H.; Wen, L.; Wang, J. An approach to evaluate the local completeness of an event log. *IEEE Int. Conf. Data Min.* **2012**, *12*, 1164–1169. [CrossRef]
23. Kang, H. The prevention and handling of missing data. *Korean J. Anesthesiol.* **2013**, *64*, 402–406. [CrossRef]
24. Marin-Castro, H.M.; Tello-Leal, E. Event Log Preprocessing for Process Mining: A Review. *Appl. Sci.* **2021**, *11*, 10556. [CrossRef]
25. Palaniswamy, S.R.; Jain, V.; Chakrabarti, D.; Bharadwaj, S.; Sriganesh, K. Completeness of manual data recording in the anaesthesia information management system: A retrospective audit of 1000 neurosurgical cases. *Indian J. Anaesth.* **2019**, *63*, 797–804. [CrossRef]
26. Kent, K.; Souppaya, M. Guide to computer security log management. *NIST Spec. Publ.* **2006**, *800–892*, 1–72.
27. Basin, D.; Klaedtke, F.; Marinovic, S.; Zalinescu, E. Monitoring compliance policies over incomplete and disagreeing logs. In *Runtime Verification*; Qadeer, S., Tasiran, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2013. [CrossRef]
28. Event Logging and Auditing. Available online: https://www.nzism.gcsb.govt.nz/ism-document/pdf/Section/15629 (accessed on 21 July 2024).
29. Arain, M.A.; Tarraf, R.; Ahmad, A. Assessing staff awareness and effectiveness of educational training on IT security and privacy in a large healthcare organization. *J. Multidiscip. Healthc.* **2019**, *12*, 73–81. [CrossRef]
30. Pan, Y.; Zhang, L. Automated process discovery from event logs in BIM construction projects. *Autom. Constr.* **2021**, *127*, 103713. [CrossRef]
31. Sutrisnowati, R.A.; Bae, H.; Dongha, L.; Minsoo, K. Process model discovery based on activity lifespan. *Int. Conf. Technol. Innov. Ind. Manag.* **2014**, 137–156. Available online: https://scholar.google.com/citations?view_op=view_citation&hl=en&user=zbYb2_4AAAAJ&citation_for_view=zbYb2_4AAAAJ:WF5omc3nYNoC (accessed on 6 August 2024).
32. Sarno, R.; Kartini; Wibowo, W.A.; Solichah, A. Time Based Discovery of Parallel Business Processes. In Proceedings of the International Conference on Computer, Control, Informatics and Its Applications (IC3INA), Bandung, Indonesia, 5–7 October 2015; pp. 28–33. [CrossRef]
33. Sarno, R.; Haryadita, F.; Sunaryono, D.; Munif, A. Model discovery of parallel business processes using modified Heuristic Miner. In Proceedings of the 2015 International Conference on Science in Information Technology (ICSITech), Yogyakarta, Indonesia, 27–28 October 2015; pp. 30–35. [CrossRef]
34. Sarno, R.; Wibowo, W.A.; Kartini; Amelia, Y.; Rossa, K. Determining process model using Time-based Process Mining and control-flow pattern. *Telkomnika (Telecommun. Comput. Electron. Control)* **2016**, *14*, 349–3591. [CrossRef]
35. Sungbum, P.; Young Sik, K. A Study of Process Mining-based Business Process Innovation. *Procedia Comput. Sci.* **2016**, *91*, 734–743. [CrossRef]
36. Liu, D.; Guo, Y.; Huang, S.; Wang, S.; Wu, T. Dynamic production bottleneck prediction using a data-driven method in discrete manufacturing system. *Adv. Eng. Inform.* **2023**, *58*, 102162. [CrossRef]
37. Elkhuizen, S.G.; Burger, M.P.M.; Jonkers, R.E.; Limburg, M.; Klazinga, N.; Bakker, P.J.M. Using Business Process Redesign to Reduce Wait Times at a University Hospital in the Netherlands. *Jt. Comm. J. Qual. Patient Saf.* **2007**, *33*, 332–341. [CrossRef]
38. Dumas, M.; Van der Aalst, W.M.P.; Ter Hofstede, A.H.M. *Process-Aware Information Systems: Bridging People and Software through Process Technology*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2005.
39. Rozinat, A.; van der Aalst, W.M.P. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.* **2008**, *33*, 64–95. [CrossRef]
40. De Medeiros, A. *Process Mining: Extending the α-Algorithm to Mine Short Loops*; BETA Working Paper Series, WP 113; Eindhoven University of Technology: Eindhoven, The netherlands, 2004.
41. Effendi, Y.A.; Sarno, R. Modeling parallel business process using modified time-based alpha miner. *Int. J. Innov. Comput. Inf. Control* **2018**, *14*, 1565–1579. [CrossRef]
42. Mikolajczak, B.; Chen, J.L. Workflow Mining Alpha Algorithm—A Complexity Study. In *Intelligent Information Processing and Web Mining. Advances in Soft Computing*; Kłopotek, M.A., Wierzchoń, S.T., Trojanowski, K., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; Volume 31. [CrossRef]
43. Sarno, R.; Effendi, Y.A.; Haryadita, F. Modified time-based heuristics miner for parallel business processes. *Int. Rev. Comput. Softw. (IRECOS)* **2016**, *11*, 249–260. [CrossRef]

44. Porouhan, P.; Jongsawat, N.; Premchaiswadi, W. Process and deviation exploration through Alpha-algorithm and Heuristic miner techniques. In Proceedings of the 2014 Twelfth International Conference on ICT and Knowledge Engineering, Bangkok, Thailand, 18–21 November 2014; pp. 83–89. [CrossRef]
45. Effendi, Y.A.; Sarno, R.; Marsha, D.V. Improved fuzzy miner algorithm for business process discovery. *Telecommun. Comput. Electron. Control* **2023**, *19*, 1830–1839. [CrossRef]
46. Siek, M. Investigating inductive miner and fuzzy miner in automated business model generation. In Proceedings of the 3rd International Conference on Computer, Science, Engineering and Technology, Changchun, China, 22–24 September 2023; Volume 2510. [CrossRef]
47. Pohl, T.; Pegoraro, M. *An Inductive Miner Implementation for the PM4PY Framework*; i9 Process and Data Science (PADS); RWTH Aachen University: Aachen, Germany, 2019.
48. Difflib—Helpers for Computing Deltas. Available online: https://docs.python.org/3/library/difflib.html (accessed on 2 September 2024).
49. SequenceMatcher in Python. Available online: https://towardsdatascience.com/sequencematcher-in-python-6b1e6f3915fc (accessed on 2 September 2024).
50. van Dongen, B.F.; Carmona, J.; Chatain, T. A unified approach for measuring precision and generalization based on anti-alignments. In *Business Process Management, 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, 18–22 September 2016*; Proceedings; Springer: Berlin/Heidelberg, Germany, 2016; pp. 39–56.
51. Adriansyah, A.; van Dongen, B.F.; van der Aalst, W.M.P. Conformance Checking Using Cost-Based Fitness Analysis. In Proceedings of the 2011 IEEE 15th International Enterprise Distributed Object Computing Conference, Helsinki, Finland, 29 August–2 September 2011; pp. 55–64. [CrossRef]
52. Bergami, G.; Maggi, F.M.; Montali, M.; Peñaloza, R. Probabilistic Trace Alignment. In Proceedings of the 2021 3rd International Conference on Process Mining (ICPM), Eindhoven, The Netherlands, 31 October—4 November 2021; pp. 9–16. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.