DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEVELOPING PROCESS MINING ALGORITHMS FOR FINDING MEANINGFUL PATTERNS

by
İsmail YÜREK

June, 2018
İZMİR

DEVELOPING PROCESS MINING ALGORITHMS FOR FINDING MEANINGFUL PATTERNS

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Engineering

by
İsmail YÜREK

June, 2018
İZMİR

Ph.D. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"DEVELOPING PROCESS MINING ALGORITHMS FOR FINDING MEANINGFUL PATTERNS"** completed by **İSMAİL YÜREK** under supervision of **ASSOC. PROF. DR. DERYA BİRANT** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Assoc. Prof. Dr. Derya BİRANT

Supervisor

Prof. Dr. Alp KUT

Thesis Committee Member

Prof. Dr. Bilge BİLGEN

Thesis Committee Member

Doç. Dr. Rıza Cenk ERDUR

Examining Committee Member

Doç. Dr. Tuğba ÖZACAR ÖZTÜRK

Examining Committee Member

Prof. Dr. Latif SALUM
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Assoc. Prof. Dr. Derya BİRANT, for her support, patient guidance, supervision and useful suggestions throughout this study. Her guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Dr. Alp KUT, and Prof. Dr. Bilge BİLGEN for their insightful comments and encouragement, but also for the hard questions which incented me to widen my research from various perspectives. As well, my sincere thanks goes to Lecturer Dr. Kökten Ulaş BİRANT for his support and valuable guidance.

Also, I would like to thank my family: my parents and to my brother for supporting me spiritually throughout writing this thesis and my life in general.

Most importantly, I would like to offer my special thanks to my wife, Özlem Ece YÜREK, for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without you. Thank you.

İsmail YÜREK

DEVELOPING PROCESS MINING ALGORITHMS FOR FINDING MEANINGFUL PATTERNS

ABSTRACT

Process mining is a technique for extracting knowledge from event logs recorded by an information system. In the process discovery phase of process mining, a process model is constructed to represent the business processes systematically and to give a general opinion about the progressive of processes in the event log. Considering in advance the trend and different features of running process is important. Especially, time management is crucial in designing and conducting business processes.

Every day information systems collect different kind of process instances of a business flow. As time goes on, size of collected data builds up speedily and constitutes a huge volume of data. It is a very challenging task to obtain valuable information and features of processes from such a large volume of data.

This thesis proposes a novel algorithm, Interactive Process Miner (IPM), to create process model based on event logs and, also a new approach that contains three different features; including activity deletion, aggregation and addition operations on the existing process model. The proposed algorithm, IPM, is enhanced by introducing time perspective. Time-oriented IPM algorithm, T-IPM, is capable of predicting the remaining and completion time of each process in a flow.

This thesis also includes the development of a new process mining tool, ProLab, in order to work on large volume of event logs and to handle the execution records of running process instances. Experimental studies demonstrate the capability of IPM and T-IPM algorithms and, also ProLab tool on both real-life and experimental datasets, including low memory usage, modification opportunity and improvement in performance compared to the existing algorithms.

Keywords: Process mining, process model, time prediction, pattern discovery

ANLAMLI ÖRÜNTÜLERİN BULUNMASI İÇİN SÜREÇ MADENCİLİĞİ ALGORİTMALARININ GELİŞTİRİLMESİ

ÖZ

Süreç madenciliği, bir bilgi sistemi tarafından kaydedilen olay kayıtlarından bilgi çıkarmak için kullanılan bir tekniktir. Süreç madenciliğinin süreç keşfi aşamasında, iş süreçlerini sistematik olarak temsil etmek ve olay günlüğündeki süreçlerin ilerleyişi hakkında genel bir fikir vermek için bir süreç modeli oluşturulur. Devam eden sürecin eğilimlerinin ve farklı özelliklerinin önceden bilinmesi önemlidir. Özellikle zaman yönetimi, iş süreçlerinin tasarlanmasında ve yürütülmesinde çok önemlidir.

Her gün bilgi sistemleri bir iş akışının farklı süreç örneklerini toplar. Zaman geçtikçe, toplanan verilerin boyutu hızla artar ve büyük miktarda veri oluşturur. Bu kadar büyük hacimli verilerden, süreçlerin değerli bilgilerini ve özelliklerini elde etmek çok zor bir görevdir.

Bu tez, olay günlüklerine dayalı süreç modeli oluşturmak için Etkileşimli Süreç Madenciliği (ESM) adında yeni bir algoritma önermektedir ve mevcut süreç modelinde aktivite silme, birleştirme ve ekleme işlemlerinden oluşan üç farklı özelliği barındıran yeni bir yaklaşım önermektedir. Önerilen algoritma (ESM), zaman perspektifi dahil edilerek genişletilmiştir. Zaman odaklı ESM algoritması (Z-ESM), bir iş akışındaki her bir sürecin kalan ve tamamlanma zamanını tahmin edebilmektedir

Bu tez aynı zamanda, büyük hacimli olay günlüklerinde çalışmak ve devam eden süreç örneklerinin yürütme kayıtlarını işlemek için yeni bir süreç madenciliği aracının (ProLab) geliştirilmesini de içermektedir. Deneysel çalışmalar, ESM ve Z-ESM algoritmalarının ve ayrıca ProLab aracının hem gerçek yaşam hem de deneysel veri setlerinde düşük bellek kullanımı, modifikasyon fırsatı ve mevcut algoritmalara göre performansta iyileştirme gibi yeteneklerinin olduğunu göstermektedir.

Anahtar kelimeler: Süreç madenciliği, süreç modeli, zaman tahmini, örüntü keşfi

CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER ONE
INTRODUCTION

1.1 General

Process mining is a technical way to acquire information in order to analyze, discover, improve and manage the processes from event logs that contain elaborative materials about the history of business operations. Process mining provides an important opportunity to detect process bottlenecks of a system. It is possible to control, manage and fix all issues after detecting the weak points of the system by applying process mining techniques. A process model represents the dependencies between activities of the process and all the information about them without categorizing which ones are important or not. This situation makes the process model hard to understand and interpret for people.

As time progresses, event logs grow rapidly and create a large volume of data. This large data increase scales up the discovery time and causes performance problems. In general, current process mining techniques analyze on historical data. Incorporation of ongoing or newly completed process records has a major importance in terms of keeping the process model constantly up-to-date. Updating the process model instantly will provide the ability to see the problems in the progressive process records immediately.

Event logs contains varied crucial features of the processes such as name, cost, resource, location, timestamp etc. Time is one of the most important features of processes. Considering with time is vital in understanding, designing/re-designing, operating and managing business processes. Time management plays a major role in controlling lifecycle of processes. It provides continuous improvement of business processes. In order to ensure the improvement of processes, process engineers, controllers, quality managers and directors need to know time-related aspects of business processes in advance. Time prediction algorithms provides insight about the

execution durations of ongoing processes. Also, it helps to detect bottlenecks of processes and to take the proper actions about situation.

In this thesis, our goal is to develop a process mining algorithm which is able to work on large volume of event logs and incorporate the execution records of ongoing processes into discovered process model instantly. The proposed process mining algorithm supports different operations such as adding, deleting, and aggregating activities on the process model to provide an interactive environment which reveals impacts of improvement changes before applying the decisions in real life. One of the most important features of the proposed algorithm is that it quickly and instantly adds the completion and execution time information of newly completed or ongoing processes into the time prediction model. The algorithm calculates the time-related aspects of already-executed portion of a process flow and puts the result of calculations to analysis data to make new predictions based on it.

1.2 Purpose

The aim of this thesis is to develop a new process mining algorithm that runs on large datasets and handles execution records of running instances. The proposed algorithm provides an interactive method that allows users to modify the constructed model by adding, deleting and aggregating the activities to see the impacts of process improvement changes in a simulation environment before applying decisions in real life.

Another aim of this thesis is to develop a time prediction algorithm which calculates the remaining and completion time of each process in a flow and incorporate the execution records of ongoing processes into discovered process model instantly.

There is a need for a software tool in order to run the developed algorithms. This thesis proposes a novel software tool to analyze large volume of event logs and to handle the execution records of running process instances in a short time with low

memory usage, and also support an interactive environment for process mining to give deep insights for event logs.

The proposed process mining algorithm in this thesis is named as IPM (Interactive Process Miner) and the propose time prediction algorithm which is the enhanced version of IPM by introducing time perspective is named as T-IPM (Time-oriented Interactive Process Miner) and, the proposed software tool in this thesis is named as ProLab (Process Laboratory). These abbreviations will be used through the rest of the document to indicate the developed system.

1.3 Novel Contributions of this Thesis

The main contributions of this thesis are on four levels;

First, we developed a novel algorithm, named Interactive Process Miner (IPM), is proposed to create process model based on large volume of event logs. It also proposes three new features for process mining which are process addition, deletion and aggregation.

Second, we enhanced IPM algorithm by introducing time perspective. The time prediction algorithm is named Time-oriented Interactive Process Miner (T-IPM). The enhanced algorithm, T-IPM, is capable of estimating the completion time of the processes that has not started yet and the remaining time of ongoing processes instantly.

Third, we developed a process mining tool, ProLab, is capable of working on big event logs in a short time with low memory usage. The proposed tool can also handle the execution records of ongoing process instances with online approach to make more accurate prediction.

As a result, in this thesis, (i) a novel process mining algorithm, IPM, was proposed, (ii) IPM algorithm was enhanced by introducing time perspective (T-IPM), (iii) a process mining tool, ProLab, was developed.

1.4 Organization of the Thesis

This thesis includes seven chapters and the remaining of this thesis is organized as follows.

In Chapter 2, general information about related works, literature review and field research about process mining are given.

In Chapter 3, background information about process mining; benefits, algorithms, categories, and applications.

In Chapter 4, the new process mining algorithm, IPM, and implementation details are explained.

In Chapter 5, time prediction algorithm, T-IPM, is described and how to construct time prediction model from event logs is explained.

In Chapter 6, the new process mining tool, ProLab, is explained. The tool is tested with some case study samples.

In Chapter 7, experiments were performed for the proposed process mining algorithm, IPM, and the time prediction algorithm, T-IPM, which is enhanced version of IPM algorithm, and also the developed process mining tool, ProLab.

Finally, in Chapter 8, the conclusion remarks and future works are presented.

CHAPTER TWO
RELATED WORK

In this chapter, technical research projects, literature and field reviews are explained and research results supporting desired goals are discussed.

2.1 Literature Review

The concept of process mining started to come up at the end of the 90's. Agrawal, Gunopulos, & Leymann (1998) proposed a new approach which deals with noise and parallel structure to extend the utility of actual workflow systems. Their approach allows the user to use existing event logs to model a given business process as a graph. After that, Cook & Wolf (1998) described different methods for process discovery and to produce formal models based on the actual process executions.

Eder, Panagos, & Rabinovich (1999) emphasizes the importance of time management in workflow-based process management systems. They proposed a framework to compute deadlines of activities to see that all time constraints are satisfied and the end-to-end process deadline is met. They presented ways to check satisfiability of time constraints like lower and upper bound between activities at process build and process instantiation time.

Van der Aalst, Weijters, & Maruster (2004) introduced α-algorithm which is able to discover a large and relevant class of workflow processes. At first α-algorithm analyses the event log, and then constructs various dependency relations between tasks. The aim is to analyze different kinds of workflow logs in the presence of noise and without any knowledge of the underlying process. In the same year, Cook, Du, Liu, & Wolf (2004) worked on discovering concurrent models of system behavior from event traces by using probabilistic techniques. Herbst and Karagiannis (2004) dealt with the duplicate tasks and they proposed an algorithm which is based on inductive approach in two steps: (i) induction and (ii) Stochastic Task Graph (SAG) generation. SAG is then transformed into blocked structured model using a definition

language. They developed a tool, called InWoLvE, which takes many parameters, however it is necessary to give proper parameters to improve mining efficiency and quality. Schimm (2004) proposed an approach to extract accurate model from event logs and to deal with hierarchically structured workflow models that include the splits and joins. He demonstrated his method by an example and also developed a tool for process mining.

Dongen & van der Aalst (2005) defined a standard for storing event log. They introduced a data model and an XML format called MXML (Mining eXtensible Markup Language). In the same year, Eder & Pichler (2005) worked on the notion of probabilistic time management to improve the estimations about remaining duration of a workflow. It is stated that in a workflow different routes may be selected so taking in consideration the probabilities of each path is important. They introduced the probabilistic timed graph which shows the time histograms and branching probabilities.

Weijters, van der Aalst, & de Medeiros (2006) proposed the Heuristics Miner algorithm that discovers main behavior registered in a noisy event log. The algorithm includes different threshold parameters in order to overcome two problems: noise and low frequency behavior. In the same year, Reijers (2006) made a point of challenges of case prediction. Reijiers defines the case prediction difficulties as part of Business Process Management Systems. This paper minds the forecasting of the remaining time to complete a specific case.

Günther & van der Aalst (2007) emphasized existing problems in the traditional process mining techniques when the processes are large and less-structured. To handle the problems, they developed a flexible approach based on their previous works (Weijters & van der Aalst, 2003): Fuzzy Mining. Their approach adaptively analyzes, simplifies and visualizes mined process models based on two metrics: significance and correlation of graph elements. De Medeiros, Weijters, & van der Aalst (2007) used genetic algorithm to mine process models in ProM (process mining) framework and performed experiments on the simulated data. Their results showed that genetic

algorithm found all possible business process models that could parse all the traces in the event log. However, time and space complexity is the main disadvantage of genetic approach. For this reason, Bratosin, Sidorova, & van der Aalst (2007) proposed distributed genetic approach to overcome high computational problem of genetic approach.

Song, Liu, & Liu (2008) used simulated annealing technique in business process mining. Song, Gunther, & van der Aalst (2008) have also proposed a novel approach "trace clustering", in which the event log is split into homogeneous subsets and for each subset a process model is created. Van Dongen, Crooy, & van der Aalst (2008) centered on the remaining cycle time. They computed the remaining cycle time by using non-parametric regression. This paper stated that non-parametric regression is very appropriate when no or very limited precedents are present. By applying this method, predicting the cycle times in any uncertain case in a business process is made possible. They took into account the duration and occurrence of all activities and showed that their approach does better from taking the acreage cycle time minus the already spent time with a real-life example. Verwer, Weerdt, & Witteveen (2008) defined an algorithm which depends on the state combining method for learning a deterministic finite state automation. The algorithm is used for learning a timed model from observations. Schonenberg, Weber, van Dongen, & van der Aalst (2008) suggested a recommendation service. This service is able to work with flexible Process Aware Information Systems (PAIS). It supports end users while process execution is proceeding by giving advices about possible next stages. They created these recommendations which were given by the service depending on similar past process executions. Also, they considered the specific optimization goals while studying on different methods to calculate log-based recommendations.

Leitner et al. (2009) presented an approach to predict Service Level Agreement (SLA) violations at runtime. Measured and estimated facts is used as input to create a prediction model. In the paper, it is stated that the prediction model is based on machine learning regression methods, and trained using historical process instances.

The machine learning method which will be applied can be determined by user via defining an algorithm and the respective parameterization for it.

Van der Aalst, Pesic, & Song (2010) concentrated on the application of process mining to operational decision making. They suggested a framework for time-based operational support and defined a set of new approaches for time-based operational support and implemented them in ProM tool. This work points out that process mining techniques are not only restricted to the past processes but can also be used for the present and future processes. It is said that existing process mining algorithms considers about the historical information, but in this study van der Aalst et al. interested in individual process instances which is still running and incomplete.

Bose, van der Aalst, Žliobaitė, & Pechenizkiy (2011) proposed features and statistical techniques to detect changes and to identify changed regions from a control-flow perspective. Luengo & Sepulveda (2011) extended the work which is used for learning a timed model from observations (Verwer et al., 2008) by adding time feature and the clusters that formed by sharing both a structural similarity and a temporal proximity. Van der Aalst, Schonenberg, & Song (2011) provided a configurable approach to predict the completion time of process by constructing a process model with time information. In the paper, it is stated that they seriously focus on the transition system generation. They used this annotated transition system to predict the remaining flow time of all or some of the process instances. It is presented that the algorithm they proposed to predict the completion of a case performs better than simple heuristics (e.g., always estimating half of the average flow time or the average flow time minus the already elapsed time) and also outperforms regressions models in terms of efficiency and precision. They also highlighted that their approach can be easily extended to predict other features of a case such as the time until a particular event or the occurrence of particular event by annotating the transition system with related information/additional features.

Van der Aalst (2012) emphasized process mining as one of the hot topics in Business Process Management. Three basic types of process mining (discovery,

conformance checking, and enhancement) were presented using a small example and some larger examples were given to illustrate the applicability in real-life settings. Our study described in this thesis focuses on the discovery type of process mining. Van der Aalst and his team developed three process mining tools: Little Thumb, EMiT (enhanced mining tool) and ProM (process mining). Little Thumb can extract workflow nets from noisy and incomplete logs (Weijters & van der Aalst, 2003). EMiT can convey workflow models with Petri nets (van Dongen & van der Aalst, 2004). ProM is a generic open-source framework for implementing process mining projects that includes many packages with many plug-ins (van Dongen, de Medeiros, Verbeek, Weijters, & van der Aalst, 2005).

Fahland & van der Aalst (2013) presented a post-processing approach to control the balance between overfitting and underfitting by simplifying discovered process models. They expressed the discovered process model in Petri net, and their approach can be combined with any process discovery method which generates Petri net. In the same year, Appice, Pravilovic, & Malerba (2013) worked on a process mining approach and used predictive clustering to prepare an execution scenario with a prediction model. This model expresses last events of running cases to forecast the features of coming events. They used predictive clustering tree (PCT) to predict online event elements of any new running case. They implemented their approach in ProM framework and explained its verification and effectiveness with several case studies.

Polato, Sperduti, Burattin, & de Leoni (2014) presented a novel method in order to enhance the quality of prediction by building a process model that is annotated with time and data information to predict the remaining time. In the paper, it is stated that calculation of predicting the remaining time is made by combining the likelihood of all the following activities, given the data collected so far; and the remaining time estimation given by a regression model built upon the data. They considered the data attribute's values to predict the remaining time of a running case.

Aleem, Capretz, & Ahmed (2015) presented the comparison of different process mining approaches in detail. The important point of their paper is that it collects and

shows all efficient and qualitative results of business process mining for researchers. Their article groups the process mining approaches to five sections: deterministic, heuristic, inductive, genetic and clustering-based mining approaches. Cheng & Kumar (2015) proposed a technique to remove noisy traces from event logs by building a classifier and applying classifier rules on event logs. They showed that generated mined models from such preprocessed logs are superior on several evaluation metrics. Fahland & van der Aalst (2015) investigated the problem of repairing discovered process model to align them to reality. They decomposed the event log into several sublogs of nonfitting traces to make conformance checking. Rovani, Maggi, de Leoni, & van der Aalst (2015) presented a methodology in order to analyze medical treatment processes by showing how to apply process mining techniques based on declarative models.

De Leoni, van der Aalst, & Dees (2016) proposed a framework to unify a number of approaches for correlation analysis. They tried to correlate different process characteristics related to different perspective. Mannhardt, de Leoni, Reijers, & van der Aalst (2016) proposed a process mining algorithm to check process conformance with respect to control flow, data dependencies, resource assignments and time constraints. Pika, van der Aalst, Wynn, Fidge, & ter Hofstede (2016) presented an approach and a supporting tool to evaluate the overall risk of process and to predict process outcomes. The approach is based on the analysis of information about process executions recorded in event logs. Tax, Sidorovaa, Haakmab, & van der Aalst (2016) suggested an algorithm named local process model to discover frequent behavioral patterns in event logs. The algorithm focuses on local structures to enable process mining of noisy event logs and extends sequential pattern mining techniques. Bolt, de Leoni, & van der Aalst (2016) came up with a framework to make process mining repeatable and automated for event logs may need to be decomposed and distributed for analysis. They stated the main motivation of their study is the inability to model and execute process mining workflows. Their study establishes the basic building blocks required for process mining and also describes various analysis scenarios to show the feasibility of their approach. Polato, Sperduti, Burattin, & de Leoni (2016) offered three new predictions methods to forecast the remaining time of running cases.

They took into consideration the additional data presented in the event log besides the control flow information. They used machine learning methods so as to build models that are capable of dealing with additional information. In the paper, it is explained that the proposed approach is able to cope with unexpected behaviors or noisy data by checking the closeness between the new trace and the most similar process flows already observed. The suggested algorithms were evaluated on real life data and they showed the performance of algorithms.

Suriadi, Andrewsa, ter Hofstedea, & Wynna (2017) described a set of data quality issues and presented a patterns-based approach to clean noisy event logs. Mitsyuk, Shugurov, Kalenkova, & van der Aalst (2017) suggested a tool to generate event logs from Business Process Model and Notation (BPMN) and they implemented script-based gateways and choice preferences to manage control flow. Bolt, de Leoni, ter Hofstedea, & van der Aalst (2017) proposed an approach to address the problem of comparing different variants of the same process and to detect differences in behavior and business rules. They used transition systems which were annotated with measurements to model behavior and to underline differences.

Alizadeh, Lu, Fahland, Zannone, & van der Aalst (2018) recommended an approach to enable the identification of deviations by reconciling the data and process perspectives. They linked data and control flow for conformance checking. In their study it is stated that the proposed approach is capable of identifying deviations in both data usage and control flow, while providing the purpose and context of the identified deviations.

Differently from the previous studies, this thesis proposes a novel algorithm, IPM, to create process model, and also provides three new features (addition, deletion and aggregation) to support an interactive environment for process mining (Yürek, Birant, & Birant, 2018). This thesis also proposes another novel algorithm, T-IPM, to predict the remaining and completion time of processes. We also developed a process mining tool, ProLab, is capable of working on big event logs in a short time with low memory

usage (Yürek & Birant, 2018). The proposed tool can handle the execution records of ongoing process instances with online approach to make more accurate prediction.

2.2 Field Review

Academic and commercial tools have been developed for process mining techniques. While ProM tool is used for academic purposes, Disco and Celonis tools are commercial software products developed by different companies. Table 2.1 presents the comparison of process mining tools.

The proposed tool, ProLab, uses streaming method to read event log files, while other tools work by loading the event logs into memory. Their methods lead to the problem of memory insufficiency in very large volume of event logs. All the tools visualize the representation of generated process model after analyzing the event logs. Disco, Celonis and ProLab tools play with the adjustment of visual settings, allowing quick access to desired information. All applications, except ProM, support to make visual adjustments and give detailed statistics of event logs on a dashboard page by using graphics and data tables so that the desired information can be accessed quickly.

Table 2.1 Comparison of process mining tools

	ProM	**Disco**	**Celonis**	**ProLab (our tool)**
Streaming event logs	No	No	No	Yes
Model visualization	Yes	Yes	Yes	Yes
Visualization settings	No	Yes	Yes	Yes
Insights	No	Yes	Yes	Yes
Interactive environment	No	No	No	Yes
Offline fashion	Yes	Yes	Yes	Yes
Online fashion	No	No	No	Yes

In contrast to other tools, only the proposed tool, ProLab, provides an interactive environment for the users. This interactive environment allows the user to merge, delete, or add a new activity in the event logs. The user can immediately see the effects of this change on the process flows.

While offline fashion is a method to analyze the processes by using the historical event logs, online fashion is called instantaneous analysis of event records formed by ongoing process records. ProLab has the ability to analyze the processes with both offline and online fashion.

CHAPTER THREE
BACKGROUND INFORMATION

Business intelligence is a leading way to use the data stored in information systems. The aim of business intelligence is to improve decision making processes and to deal with challenges such as data explosion and information overflow. Data mining is the analysis of data for discovering relationships and patterns. Process mining uses data mining methods in the context of business process management and enables the application of modern approaches for improving the control of business processes.

3.1 Process Mining

Any enterprise and its decision makers hope to produce more products or provide better service in a shorter time, thus the efficiency and quality of the business operation is crucial to the survival and development of enterprise. But the new business requires enterprises to improve existing business process, therefore data mining and machine learning techniques are introduced to workflow field, i.e. process mining. The basic idea is to extract enterprise operation process from workflow log, excavate valuable objective information, and help to realize business process modeling and recycling, greatly enhancing the enterprises competitiveness in the market.

Data mining is the analysis of data for discovering relationships and patterns. The abstraction of the analyzed data is called as patterns. Abstraction decreases complexity and makes information available for the recipient. However, the extraction of information about business processes is the goal of process mining. Process mining discovers, monitors and improves real processes by extracting knowledge from event logs.

Process mining is also called as workflow mining or process detection, which is an analysis method to construct workflow models automatically through analyzing the event logs. Process mining begins with executive stage, with collecting the performance information as input, generating the workflow model as the output.

Process mining can be considered as a branch of data mining, and it has many similarities with data mining in principle and methods. But it is different from data mining, because the traditional data mining method aims at forecasting system behaviors, while the process mining at constructing whole process models. Figure 3.1 shows the relations between business process management, process mining, and data mining.

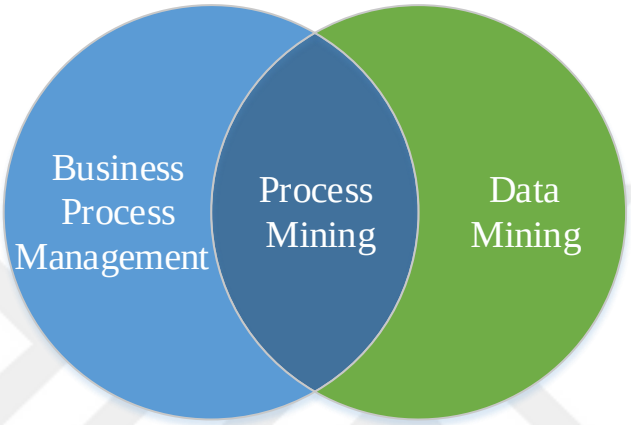


Figure 3.1 Relations between business process management, process mining and data mining

In information systems, the created data during the execution of business processes is used to rebuild process models. These models are beneficial for analyzing and optimizing processes. Process mining is a modern approach and sets up a bridge between data mining and business process management.

3.2 Benefits

Business process management (BPM) is usually a top-down approach. BPM starts by designing a process in a high-level model. It configures a system for managing and controlling the process. This system then coordinates work between the employees, and other resources in an organization.

However, process mining can analyze the processes in a bottom-up fashion. Process mining reduces cost and variation by extracting enterprise operation process from workflow log. Process mining techniques ensure the control in processes and know

what is going on. It improves the quality by comparing processes beyond Key Performance Indicators (KPIs) and makes the processes transparent.

The benefits of process mining can be summarized as follows:

- Find deviations between your plan and reality
- Find out how your process keeps up in reality
- Get objective information on whether it is actually followed as prescribed
- See for the first time how that process is handled in real-life
- Compare processes

3.3 Three main types of process mining

Process mining contributes detailed insights into the process execution by using historical facts recorded by the information system. An overview of process mining is shown in Figure 3.2. The goal of process models is to describe the real-life processes. These process models are applied to configure the information system. While executing the defined process using the information system, historical records of the executed process are stored. The main input of process mining analysis is event logs. Process mining sets up links between the actual observed process execution and the modeled process behavior.

Three major classes of process mining methods can be categorized: (a) the discovery of new process models based only on the event log, (b) conformance verification of the recorded behavior with a provided process model and (c) extension of existing process models using the information from the event log.

Discovery: Traditionally, process mining has been focusing on discovery, i.e., deriving information about the original process model, the organizational context, and execution properties from enactment logs. An example of a technique addressing the control flow perspective is the α-algorithm, which constructs a Petri net model describing the behavior observed in the event log. It is important to mention that there

is no a-priori model, i.e., based on an event log some model is constructed. However, process mining is not limited to process models (i.e., control flow) and recent process mining techniques are more focusing on other perspectives, e.g., the organizational perspective, performance perspective or the data perspective.

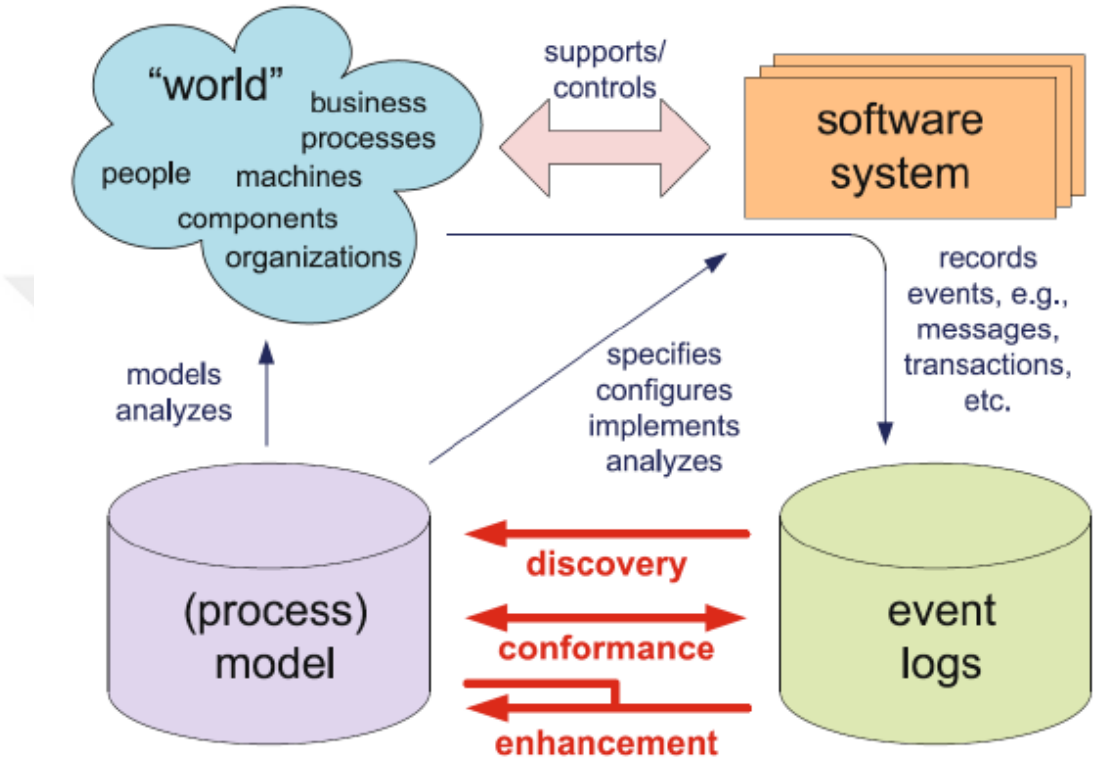


Figure 3.2 Overview of process mining (van der Aalst., 2011)

Conformance: There is an a-priori model. This model is used to check if reality conforms to the model. For example, there may be a process model indicating that purchase orders of more than one million Euro require two checks. Another example is the checking of the so-called four-eyes principle. Conformance checking may be used to detect deviations, to locate and explain these deviations, and to measure the severity of these deviations.

Extension: There is an a-priori model. This model is extended with a new aspect or perspective, i.e., the goal is not to check conformance but to enrich the model with the data in the event log. An example is the extension of a process model with

performance data, i.e., some a-priori process model is used on which bottlenecks are projected.

3.4 Algorithms

The mining algorithm which determines how the process models will be created is the main component in process mining. A broad variety of mining algorithms exists. The following three categories will be discussed in more detail.

- Deterministic mining algorithms
- Heuristic mining algorithms
- Genetic mining algorithms

Determinism means that an algorithm only produces defined and reproducible results. It always presents the same result for the same input. A representative of this category is the α-algorithm (van der Aalst et al., 2004). It was one of the first algorithms that are able to deal with concurrency. It gets an event log as input and evaluates the ordering relation of the events contained in the log.

Also, heuristic mining applies deterministic algorithms but they include frequencies of events and traces for building a process model. A general problem in process mining is the fact that real processes are highly complex and their discovery leads to complex models. This complexity can be decreased by disregarding infrequent paths in the models. HeuristicsMiner algorithm which is proposed to overcome noise and low frequency behavior in noisy event log is a member of this category (Weijters et al, 2006).

Genetic mining algorithms apply an evolutionary method that simulates the process of natural evolution. They are not deterministic. Genetic mining algorithms follow four steps: initialization, selection, reproduction and termination. The idea behind these algorithms is to create a random population of process models and to figure out a satisfactory solution. The algorithms selects individuals iteratively and reproduces

them by crossover and mutation over different. The first population of process models is created randomly and might have little in common with the event log. However due to the high number of models in the population, selection and reproduction better fitting models are created in each generation.

Various advanced mining algorithms exist that can be applied for several objects. Figure 3.3 shows the mined model using the heuristic Fuzzy Miner algorithm (Günther & van der Aalst, 2007). The model does not follow the BPMN notation but instead uses a dependency graph representation. It does not cover any gateway operators but points the dependencies between different activities. The dependency graph depicts for example that A was followed three times by B and two times by C.

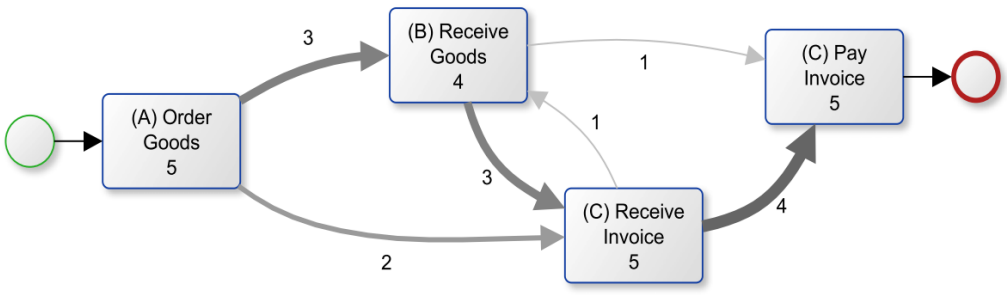


Figure 3.3 Sample process model constructed by using the fuzzy miner (Günther et al, 2007)

3.5 Applications

Process mining can analyze your process in a bottom-up fashion. Figure 3.4 shows the steps of process model discovery. You do not need to have a model of your process to analyze it. Process mining uses the historical data in your information systems. Your information system already records all steps of your process in execution. With process mining, you get a process model from these data. This way, your real process, and actual business rules, can be discovered automatically. Table 3.1 shows a sample event log and Figure 3.5 shows a discovered model which is represented with Petri nets.

Figure 3.4 Steps of process model discovery

An event log keeps the execution history of a process. Table 3.1 shows an excerpt of a sample dataset used for process mining. The sample log stores some execution history of a loan application process. An event log includes data related to a *single process*. Each line in the table shows one event and each column presents an attribute of this event. An event is related to a trace, or *process instance*. The events in Table 3.1 are already grouped by trace and sorted chronologically. The order of events that is recorded for a process instance is called a *trace*.

Table 3.1 Data fields of an event log

Trace id	**Event id**	**Timestamp**	**Activity**	**Resource**
1	35654423	30-12-2010 11:02	Register application	Pete
	35654424	31-12-2010 10:06	Check credit	Sue
	35654425	05-01-2011 15:12	Calculate capacity	Mike
	35654426	06-01-2011 11:18	Check system	Sara
	35654427	07-01-2011 14:24	Reject request	Pete
	35654428	08-01-2011 09:03	Send decision e-mail	Pete
2	35654483	30-12-2010 11:32	Register application	Mike
	35654485	30-12-2010 12:12	Calculate capacity	Mike
	35654487	30-12-2010 14:16	Check credit	Pete
	35654488	05-01-2011 11:22	Accept request	Sara
	35654489	08-01-2011 12:05	Send decision e-mail	Ellen
…	…	…	…	…

As a process modeling language, Petri nets supports concurrency. Petri nets apply a very basic notation of circles representing places and squares representing transitions with arrows connecting them in a bipartite manner. Transitions can represent a task and when executed they consume one token, demonstrated by black dots, from each of their input places and generate a token in each of their output places. In this way, tokens are moved between places, and the allocation of tokens over the places points

different states of the process model. This is called as marking. Special markings are the initial marking, which points how the process starts, and the final marking which points when the Petri net is in a terminate state.

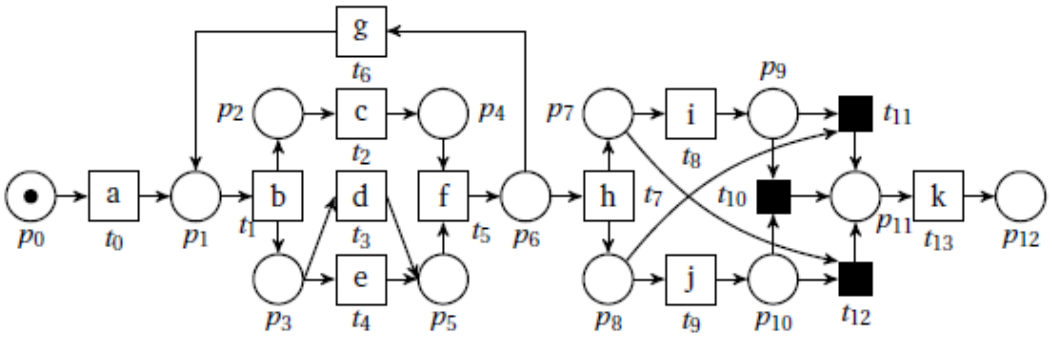


Figure 3.5 Petri net representation of a process model (van der Aalst., 2011)

Process discovery is the major focus of studies in process mining. Process discovery aims to use only the behavior as recorded in the event logs, building a process model describing the underlying behavior. Although this aspect of process mining has received a lot of concentration, and quite a few algorithms presently exist to do this. But, process discovery still remains as a challenge because of time complexity and the cost of computationally expensive algorithms. Therefore, process mining is not applied commonly in industry.

CHAPTER FOUR
INTERACTIVE PROCESS MINER ALGORITHM

In this thesis, we propose a new process miner algorithm titled Interactive Process Miner (IPM). Process mining is a method to discover information from event logs. In process discovery, a process model is constructed to represent the processes based on observed events.

4.1 Proposed Algorithm

First of all, we read process records which are stored in XML format in files with .XES extensions line-by-line via file streaming method. Then, we keep process records a tree-like structure by grouping all the same traces in a path in system memory. By using this tree structure, we create process model of event logs. Table 4.1 represents a sample event log.

Table 4.1 Sample event log for process mining algorithm

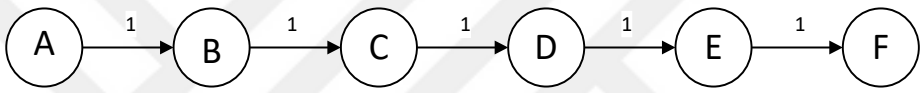
Trace ID	Events
Trace 1	ABCDEF
Trace 2	ABCEF
Trace 3	ACBEF
Trace 4	ACDF
Trace 5	ABCDEF
Trace 6	ABCDEF
Trace 7	ACBEF
Trace 8	ABCEF
Trace 9	ACBEF
Trace 10	ABCEF
Trace 11	ABCDEF
Trace 12	ACDF
Trace 13	ABCEF
Trace 14	ABCDEF

We can separate our process mining progress into four sections. The first section is “Data Model Creation”, the second section is “Dependency Matrix Creation”, the third

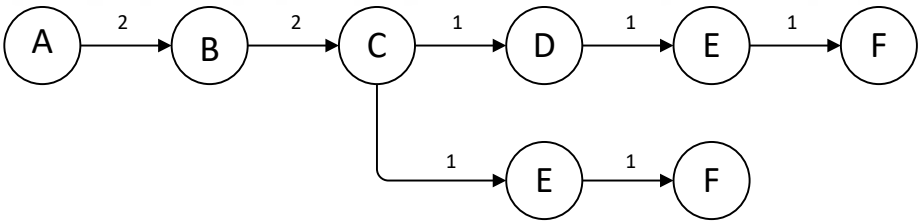
one is “Dependency Graph Creation” and the last section is “Eliminating Low Frequent Traces”. Let us consider data model creation by examining the event log that contains 14 traces listed in Table 4.1

The algorithm creates the data model by reading all the activities in a trace. Read 1st trace that is “ABCDEF”. The first activity is A, then B, then C and it continues. The algorithm increases the count of the edge that represents the relation between two activities each time. B comes after activity A, then the count of the edge between A and B is increased by 1. And the same logic is followed for the other activities.

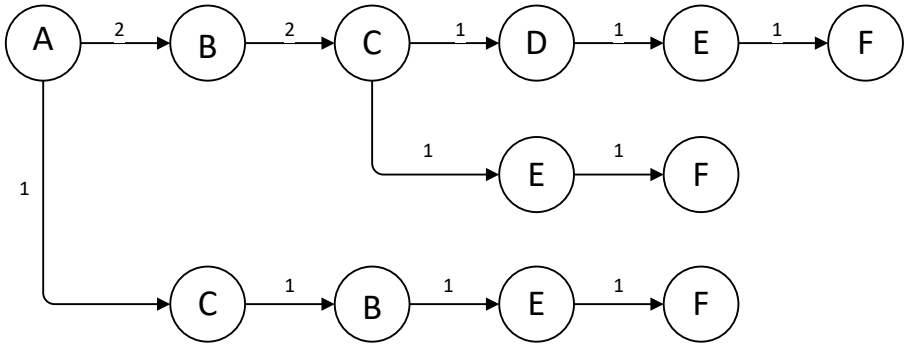
Step 1: Read 1st trace and start constructing the data model.



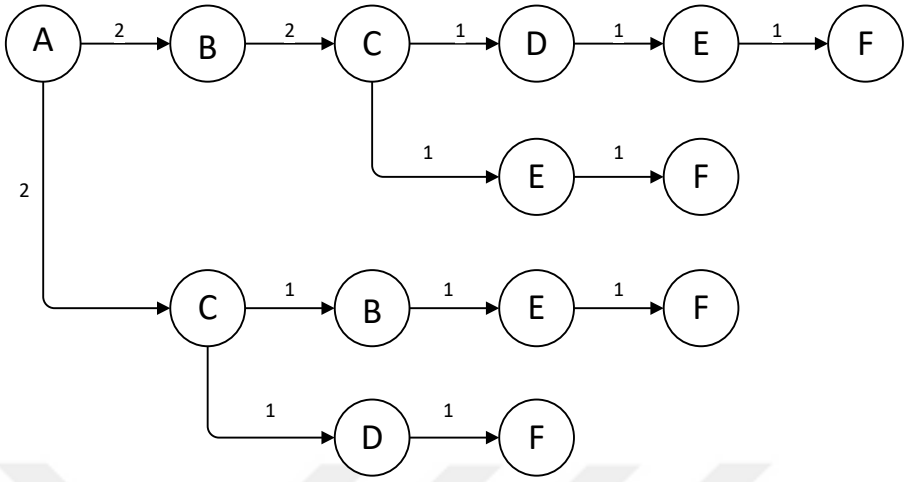
Step 2: Read 2nd trace and update the data model.



Step 3: Read 3rd trace and update the data model.



Step 3: Read 4th trace and update the data model.



Afterwards, all the traces is read in the event log and after the last trace, the final version of data model is constructed.

Last Step: Read 14th trace and update the data model.

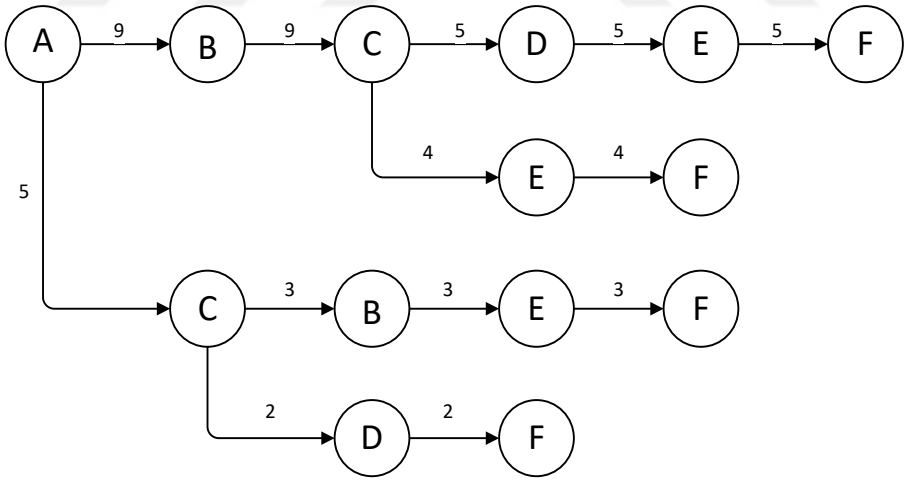


Figure 4.1 shows an example data model constructed from the event log that contains 14 traces listed in Table 4.1 (the output of the first section, Data Model Creation). In our example we have 4 leaf nodes. Let us call them as L0, L1, L2 and L3.

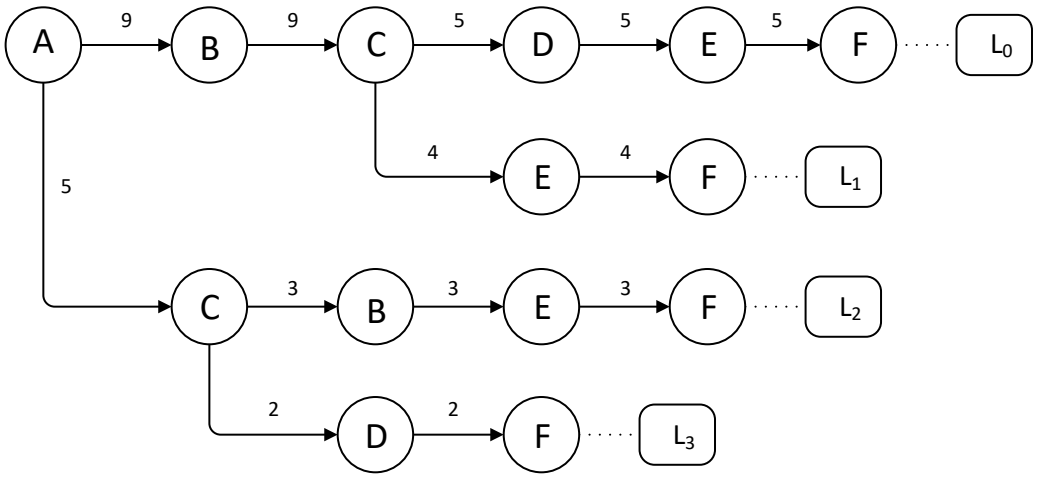


Figure 4.1 Data model of all traces in the event log

Next step is the creation of dependency matrix. The matrix elements represent activities in a trace. The off-diagonal elements are used to indicate dependencies between the activities. We used the output of the first section as input for this second section in order to create dependency matrix. We traverse all the data model which is a tree-like structure, by using in-order algorithm that is one of the depth-first search algorithms. The aim is to reach the leaf node. When we reach to all leaf nodes, tree represents the complete event log because it shows all the traces which are visited so far. We start from the root node and continue to traverse the tree. While traversing all the activities, dependency matrix is created one by one for each activity.

Step 1: We start from the root activity A, and traverse all the paths to reach the first leaf activity. The activities are A → B → C → D → E → F. F is the first leaf $L_{0.}$ Well, how we will know how many times this trace exist in the event log? When we find the leaf activity, it means this trace exists in the event log for the last edge count. In other words, the total number of the trace in the event log is equal to the last edge count of the leaf activity. Then, the number of dependencies is written to the related cell of the matrix. The count of each relation between activities is written to matrix one by one. After first traversal of the path, Table 4.2 represents the created dependency matrix after step 1.

Table 4.2 Dependency matrix after first step

	A	B	C	D	E	F
A	0	5	0	0	0	0
B	0	0	5	0	0	0
C	0	0	0	5	0	0
D	0	0	0	0	5	0
E	0	0	0	0	0	5
F	0	0	0	0	0	0

Step 2: Again, we start from the root activity A, this time we follow a different path to reach the second leaf activity. In the example, the second leaf is $L_{1.}$ We follow the activities A → B → C → E → F to reach the L_1 leaf. When we find the leaf activity, count of the last edge dependency is checked. In this example it is 4, it means trace A → B → C → E → F exists 4 times in the event log. While visiting the tree nodes, when you encounter the same activity, just increase the dependency count of it, otherwise draw a new activity. Table 4.3 represents the created dependency matrix after step 2.

Table 4.3 Dependency matrix after second step

	A	B	C	D	E	F
A	0	5+4	0	0	0	0
B	0	0	5+4	0	0	0
C	0	0+3	0	5	0+4	0
D	0	0	0	0	5	0
E	0	0	0	0	0	5+4
F	0	0	0	0	0	0

Step 3: In the same way, we start from the root activity A, and follow a different path to reach the third leaf activity. In the example, the third leaf is $L_{2.}$ We follow the activities A → C → B → E → F to reach the L_2 leaf. When we find the leaf activity, count of the last edge dependency is checked. In this example it is 3, it means 3 times trace A → C → B → E → F exists in the event log. While visiting the tree nodes, when you encounter the same activity, just increase the dependency count of it, otherwise draw a new activity. Table 4.4 represents the created dependency matrix after step 3.

Table 4.4 Dependency matrix after third step

	A	B	C	D	E	F
A	0	9	0+3	0	0	0
B	0	0	9	0	+3	0
C	0	0+3	0	5	4	0
D	0	0	0	0	5	0
E	0	0	0	0	0	9+3
F	0	0	0	0	0	0

Step 4: The number of dependency matrix created is equal to the number of leaf activities. In this example, we have 4 leaf nodes, so we created 4 different dependency matrices to generate the final version of it. The final version of dependency matrix holds all the relations between all activities in complete event log. Leaf activity L_3 is the last one. The path of it is A → C → D → F. Again the same logic is followed and the final version of the dependency matrix is created. Table 4.5 represents the created dependency matrix after step 4.

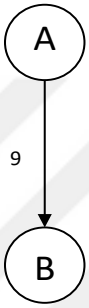
Table 4.5 Dependency matrix after last step

	A	B	C	D	E	F
A	0	9	3+2	0	0	0
B	0	0	9	0	3	0
C	0	3	0	5+2	4	0
D	0	0	0	0	5	0+2
E	0	0	0	0	0	12
F	0	0	0	0	0	0

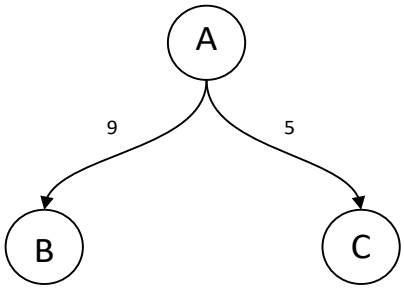
According to the final version of dependency matrix represented in Table 4.5, each dependency between activities is read from left to right. For instance, 1st row and 2nd column of the matrix, it stores the count of dependency between *A* and *B* such as *A* → *B* = 9. Finally, we can get all the dependency counts from the matrix as follows: *A* → *B* = 9, *A* → *C* = 5, *B* → *C* = 9, *B* → *E* = 3, *C* → *B* = 3, *C* → *D* = 7, *C* → *E*= 4, *D* → *E* = 5, *D* → *F* = 2, *E* → *F* = 12. By this way, it is possible to know all the dependencies and the count of them between activities.

We represent process model with a dependency graph. Dependency graph is a directed graph which represents dependencies of activities towards each other. Now, we have all the information about the event log to constitute the process model. Therefore; the third section is to form the dependency graph. Thus far, we know all the dependencies and the count of them between activities. We check the Table 4.5 and start to construct dependency graph.

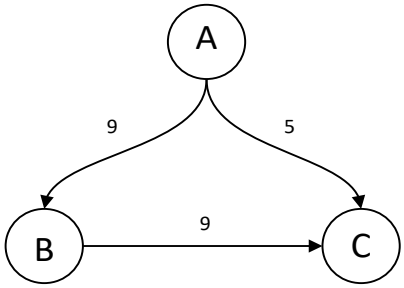
Step 1: Add dependency between *A* and *B* activities.



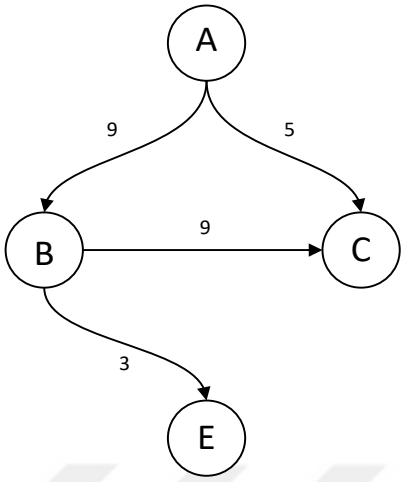
Step 2: Add dependency between *A* and C activities.



Step 3: Add dependency between *B* and *C* activities.



Step 4: Add dependency between *B* and *E* activities.



Afterwards, all the dependencies between activities is added into dependency graph and after the last step, the final version of dependency graph is constructed. Figure 4.2 shows the final version of dependency graph.

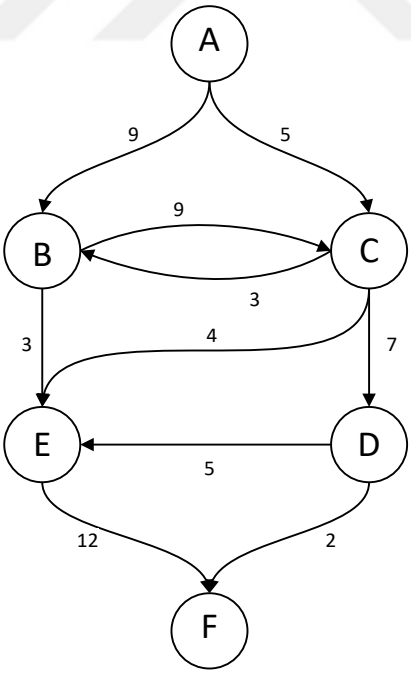


Figure 4.2 Dependency graph representation of process model

The constructed process model contains all the activities in the event log. Showing all the traces and activities of each traces in an event log makes the model complex and hard to understand. Also, event log may be incomplete or may contain noise.

We consider the frequency of traces in the event log to eliminate the effects of incompleteness and noise. Also, we reduce the complexity of the process model to make the model easy to understand. We call the eliminating of traces from process model as a fourth section.

Traces are eliminated from the process model with a defined threshold. This threshold is parametric value and it can take a value between 0 and 1. For instance, if threshold value is set to 0.25, it means the frequency of traces which are under % 25 percentage will be eliminated. In our example, totally we have 14 traces. The trace count which ends with L_0 leaf is 5, and for L_1 is 4, for L_2 is 3 and for L_3 is 2. The %25 of total trace count is 3.5. Then, the traces which their total count is under 3.5 is eliminated. Eventually, the paths which ends with the leaf nodes L_2 and L_3 is eliminated. After elimination process, the dependency graph that represents the process model is recreated. Figure 4.3 shows the filtered process model.

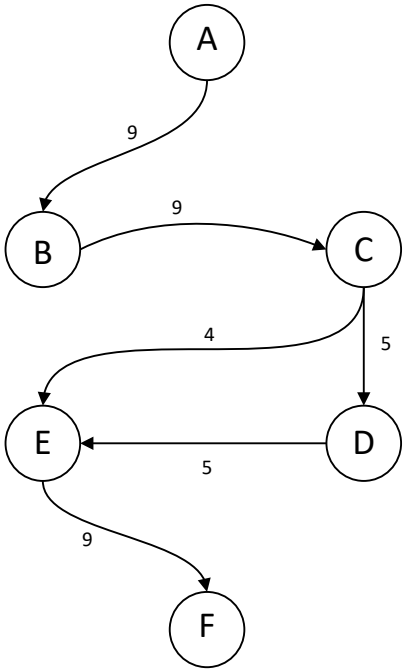


Figure 4.3 Filtered dependency graph representation of process model

After completing the four sections, we will explain a new approach that we applied to existing process model. The new approach contains deletion, aggregation and addition algorithms to modify existing process model. Figure 4.4 shows how IPM algorithm works step-by-step.

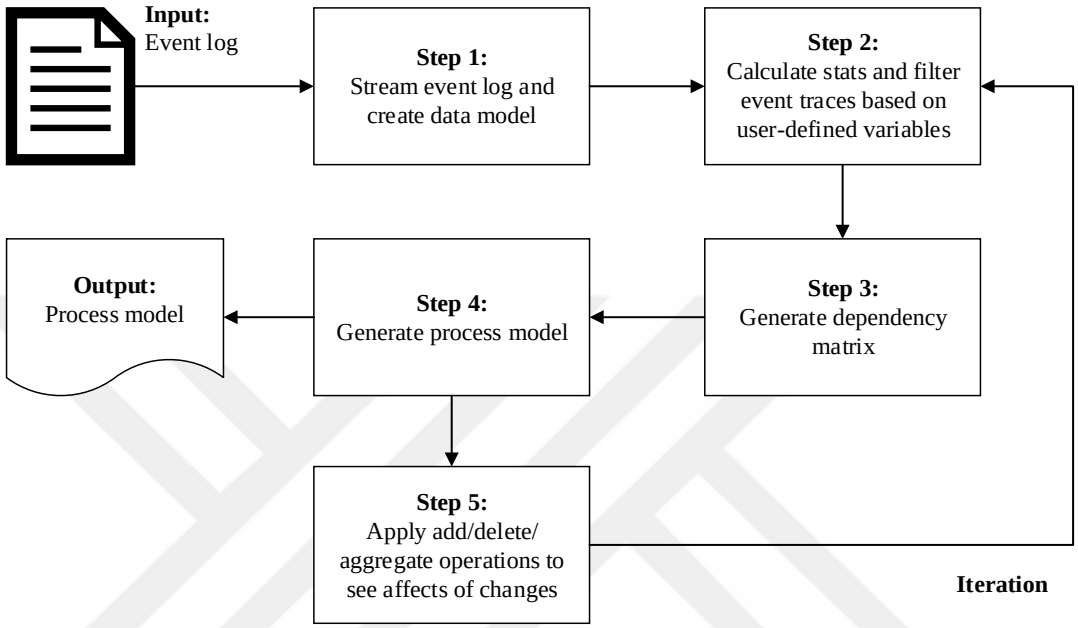


Figure 4.4 Block diagram of IPM algorithm

4.2 Provided approaches for process model modification

Up to this time, we presented all traces in the event log with a constructed process model. With the help of the process model, it is now possible to have a general opinion about the progressive of processes in the event log and we have a chance to see the differences or outliers between the process model we designed and what actually happened in real life. Still, finding answers to some questions is very hard at this stage. For example, what happens if activity *X* is removed from the process model? Or how does the process model look if activities *X* and *Y* are aggregated under another event? Or how process model is affected if a new activity *Z* is added between the activities *X* and *Y*? If we know the answers to these questions, we will have an important opportunity to see the effects of any changes to a real-life system so as to improve it.

Considering this motivation, we provide three new features for process mining which are activity addition, deletion and aggregation.

Available researches put different approaches forward to show an activity under another superior activity or to aggregate different activities whose frequency value is below a certain threshold. However, these operations are done based on some statistical values rather than user interactions. The user who is looking for answers to the above questions should be able to modify the model interactively. In our study, we proposed 3 different approaches including activity deletion, aggregation and addition.

Our aim is to provide an interactive environment for users to apply some changes on the process model and see the effects of these changes to process flow before making any decision about it. It is critical to see the effects of changes before making a decision in real life

4.2.1 Activity Deletion

Picking an activity and removing it from process model is activity deletion operation. This section explains our new approach which is applied to the same sample event log listed in Table 4.1.

Let us proceed the logic behind the algorithm step by step. For instance, what happens, if user wants activity *D* to be deleted from model? The algorithm starts to traverse from the root activity and tries to reach the leaf activities. However, each time it checks if the coming activity is *D* or not. If not, it continues to traverse and adds this path to the dependency matrix; otherwise it stops and deletes the tracked path. In other words, the algorithm does not add the path, which includes deleted activity, to the data model.

Step 1: The algorithm tries to reach L_0, but it faces with activity *D* in the path, stops and removes the related path. Figure 4.5 shows the data model after step 1.

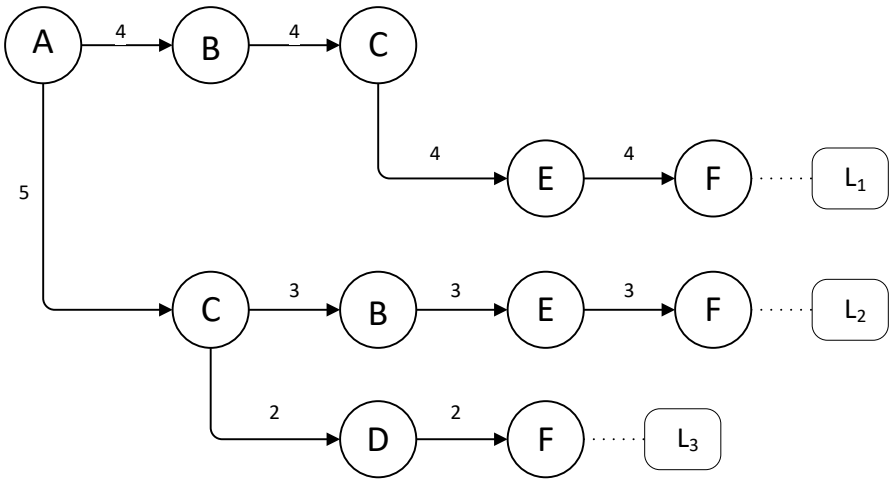


Figure 4.5 Data model after first step of activity deletion

Step 2: Secondly, it tries to reach the leaf L_1 and get there. It means that this path will remain in the process model. Whenever the algorithm reaches a leaf activity, it adds this path to the dependency matrix. Figure 4.6 shows the data model after step 2.

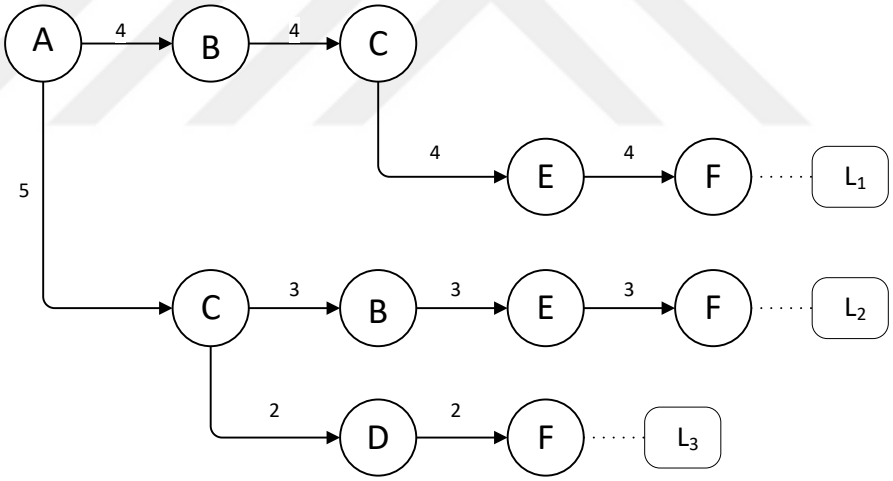


Figure 4.6 Data model after second step of activity deletion

Step 3: It tries to reach L_2 and again get there. Therefore, the third path will remain in the process model and it should be added to the dependency matrix. Figure 4.7 shows the data model after step 3.

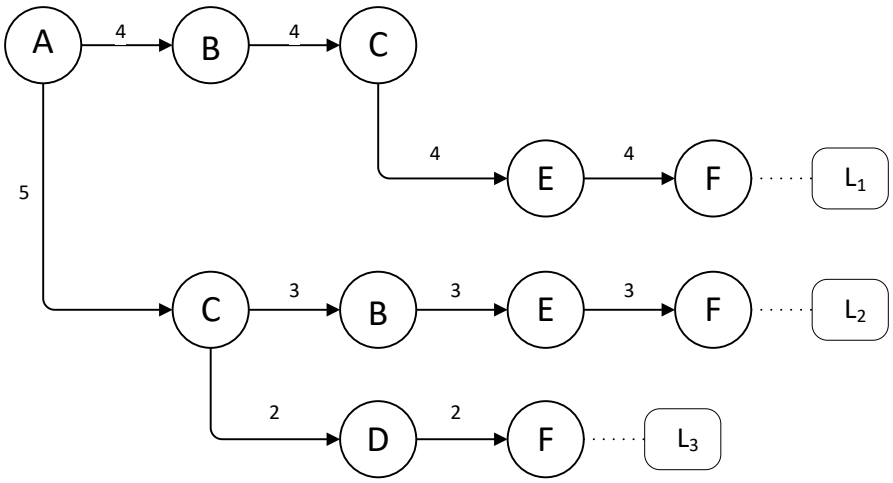


Figure 4.7 Data model after third step of activity deletion

Step 4: Finally, it tries to reach L_3 leaf activity, but it encounters activity *D* in the path, stops and removes the related path. At the end of this step, the algorithm traversed all the leaf activities and the final version of the data model is created as shown in Figure 4.8.

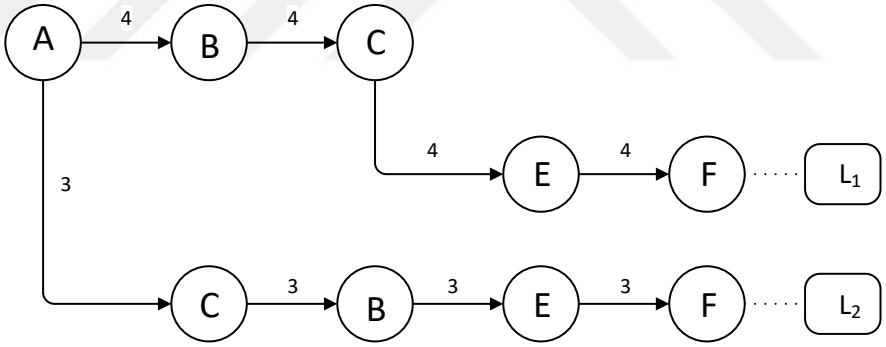


Figure 4.8 Data model after last step of activity deletion

Thus, we can create our new dependency graph from final version of data model. Table 4.6 represents the final version of dependency matrix after activity deletion.

Table 4.6 Dependency matrix after activity deletion

	A	**B**	**C**	**D**	**E**	**F**
A	0	4	3	0	0	0
B	0	0	4	0	3	0
C	0	3	0	0	4	0
D	0	0	0	0	0	0
E	0	0	0	0	0	7
F	0	0	0	0	0	0

All the cells of the matrix is visited and the final version of the dependency graph is constructed. Figure 4.9 shows the final version of dependency graph after activity deletion.

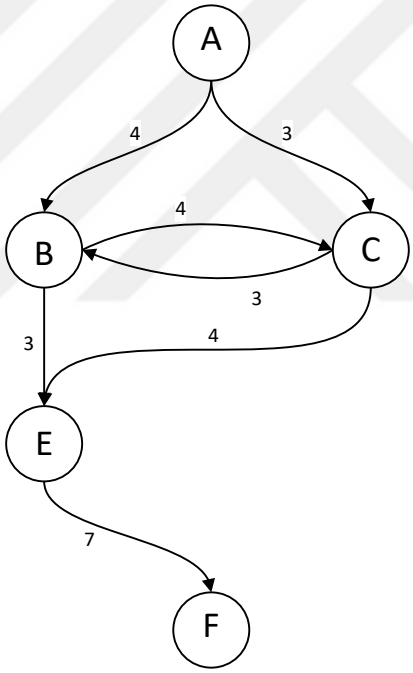


Figure 4.9 Dependency graph after activity deletion

4.2.2 Activity Aggregation

In this section, we are looking for an answer to the question how does the process model look if A_1 and A_2 activities are aggregated under another event such as A_3. At least two activities can be combined and represented as another activity. In the same example, assume that user picks activities *B* and *D* to represent them under another

activity named as *X*. The algorithm starts to traverse from root activity and tries to find activities *B* and *D* to represent as *X* until reaching the leaf activity in the followed path.

Step 1: Data model starts to change while the algorithm traversing all the activities. Firstly, the algorithm tries to reach L_0. Each time, it checks if the coming activity is *B* or *D*. If the activity is one of them, then it replaces the current activity with *X* such as $A \rightarrow X \rightarrow C \rightarrow X \rightarrow E \rightarrow F$ for leaf L_0. After finding the first leaf, it is added to the dependency matrix. In this example, we try to represent *B* and *D* as *X*, so there is no need to add the activities *B* and *D* to dependency matrix. Instead of *B* and *D*, activity *X* should be added. Figure 4.10 shows the data model after step 1.

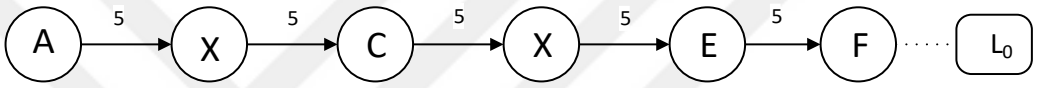


Figure 4.10 Data model after first step of activity aggregation

Step 2: In the second step, it tries to reach L_1. It controls if the coming activity is *B* or *D*. If the activity is one of them, then it replaces the current activity with *X* such as $A \rightarrow X \rightarrow C \rightarrow E \rightarrow F$ for leaf L_1. Figure 4.11 shows the data model after step 2.

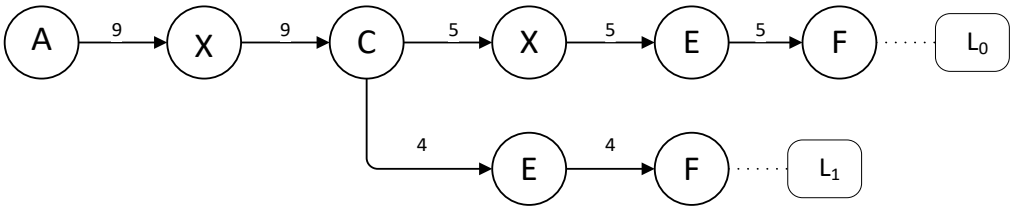


Figure 4.11 Data model after second step of activity aggregation

Step 3: While the algorithm tries to reach leaf L_2, it checks the activities one by one, whether they are *B* or *D*. If the answer is true, then it replaces the current activity with *X* such as $A \rightarrow C \rightarrow X \rightarrow E \rightarrow F$ for leaf L_2. Figure 4.12 shows the data model after step 3.

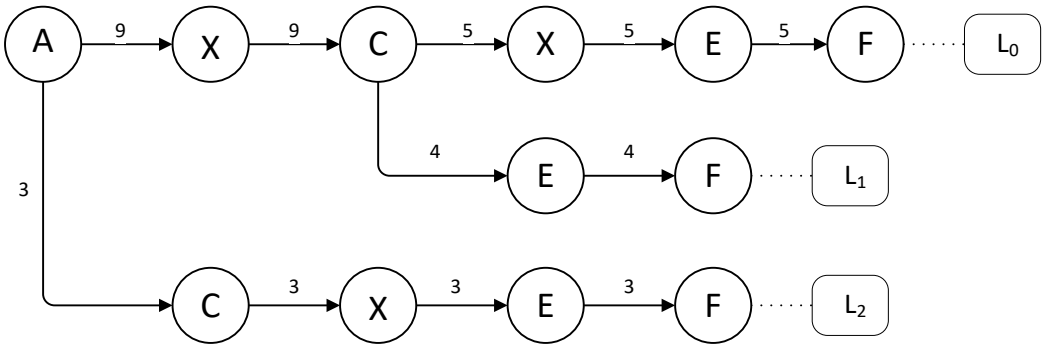


Figure 4.12 Data model after third step of activity aggregation

Step 4: Lastly, the algorithm tries to reach L_3. If it faces with one of the aggregated activities, then it replaces the current activity with new one such as $A \rightarrow C \rightarrow X \rightarrow F$ for leaf L_3. Figure 4.13 shows the data model after step 4.

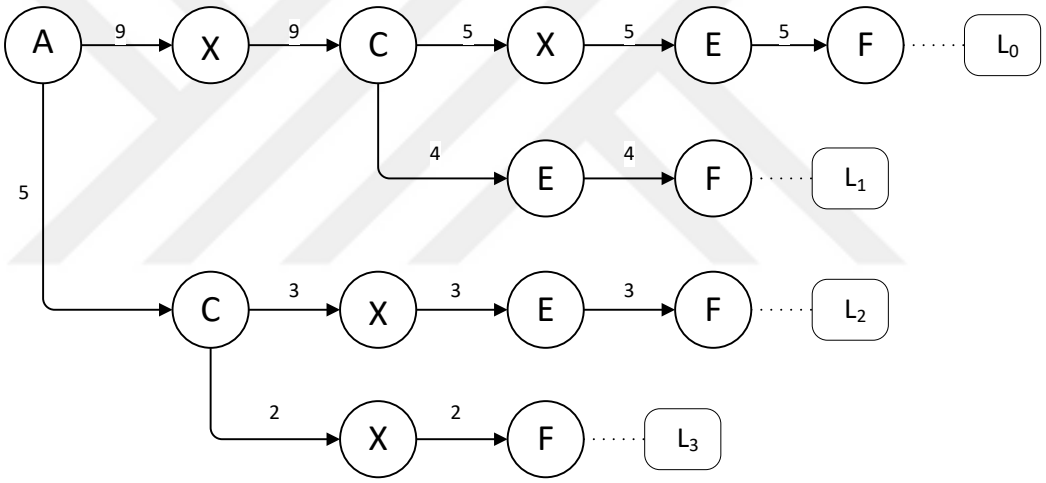


Figure 4.13 Data model after last step of activity aggregation

Now, we can create our new dependency matrix from final version of data model which is reconstructed after activity aggregation. Table 4.7 represents the final version of dependency matrix after activity aggregation.

From the dependency matrix, it is clearly seen that *A* is the root activity, because all the values of the cells are equal to zero vertically. This means that there is no any activity input to *A*. In addition, it is seen that *F* is leaf activity, because all the values of the cells are equal to zero horizontally. This means that there is no activity that outputs from *F*.

Table 4.7 Dependency matrix after activity aggregation

	A	**C**	**E**	**F**	**X**
A	0	5	0	0	9
C	0	0	4	0	10
E	0	0	0	12	0
F	0	0	0	0	0
X	0	9	8	2	0

According to the dependency matrix, dependency graph of new process model can be constructed as shown in Figure 4.14. In this graph, activity *B* and *D* does not exist, because they are aggregated under the new activity *X*, so they are represented as activity *X*.

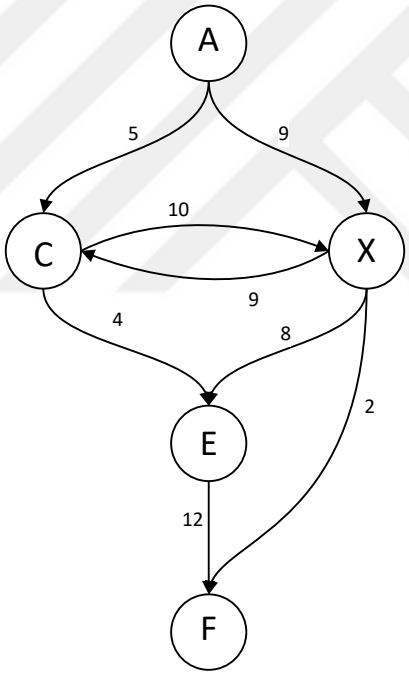


Figure 4.14 Dependency graph after activity aggregation

4.2.3 Activity Addition

In this section, we are looking for an answer to the question how process model is affected if a new activity named as A_3 is added between the activities A_1 and A_2. There are many activity addition combinations. User may want to add a new activity for specific conditions such as if and only if *A* comes after *D*, or if activity *D* is between

E and *F*. The algorithm is able to handle any of these combinations. In the same example, assume that user wants to add a new activity *X* between the activities *C* and *B*. At the same time, user wants to add the same new activity *X* between the activities *C* and *E*.

The algorithm starts to traverse from root activity and tries to find dependencies of *C* → *B* and *C* → *E* to add new activity *X* between them.

Step 1: Firstly, the algorithm tries to reach L_0. Each time, it checks current and next activity. If the current activity is *C* and the coming activity is *B* or *E*, then it inserts the activity *X* between them such as *C* → *X* → *B* or *C* → *X* → *E*. In the first path, there is no sequence that supplies this condition. Therefore, the algorithm does not apply any changes to the path of the leaf L_0. Figure 4.15 shows the data model after step 1.

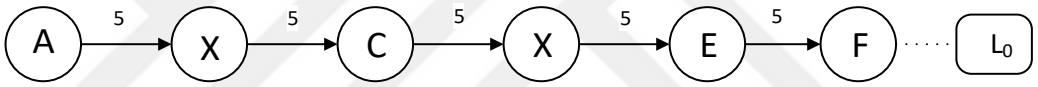


Figure 4.15 Data model after first step of activity addition

Step 2: Secondly, it tries to reach L_1. In the second path, there is a sequence that supplies the desired condition. So, the path of L_1 changes into the following path: *A* → *B* → *C* → *X* → *E* → *F*. After finding the second leaf, it is added to the dependency matrix. In this case, dependency matrix has a new row and column for activity *X*. Figure 4.16 shows the data model after step 2.

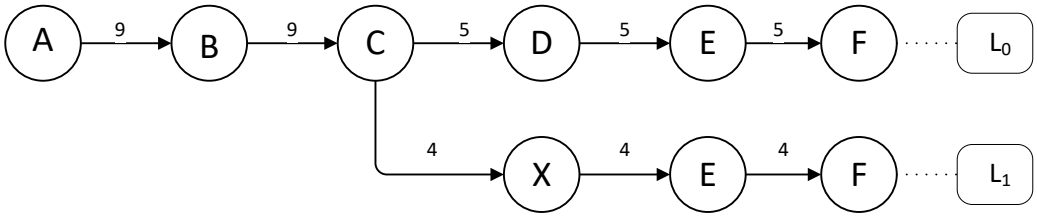


Figure 4.16 Data model after second step of activity addition

Step 3: Thirdly, the algorithm tries to reach L_2. It looks for the condition in the tracked path. In the third path, there is a sequence that supplies the desired condition

($C \rightarrow B$). So, the path of L_2 changes into the following path: $A \rightarrow C \rightarrow X \rightarrow B \rightarrow E \rightarrow F$. After finding the third leaf, it is also added to the dependency matrix. Figure 4.17 shows the data model after step 3.

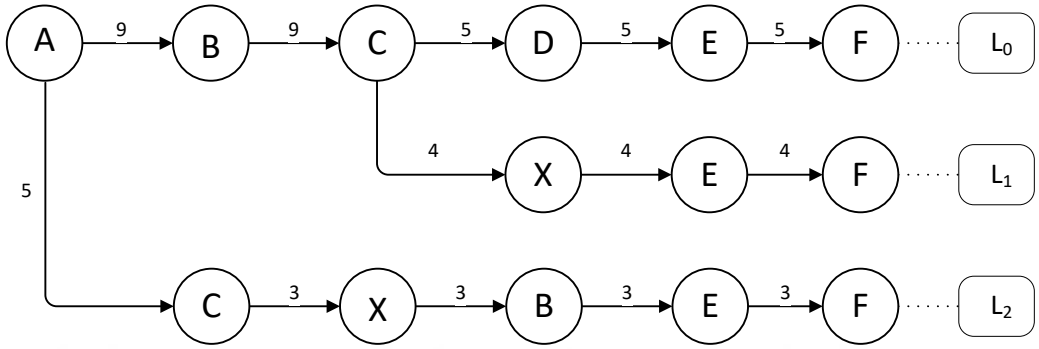


Figure 4.17 Data model after third step of activity addition

Step 4: Lastly, it tries to reach L_3. In the last path, there is no sequence that supplies the desired condition. Therefore, the algorithm does not apply any changes to the path of the leaf L_3. The final version of data model is shown in Figure 4.18.

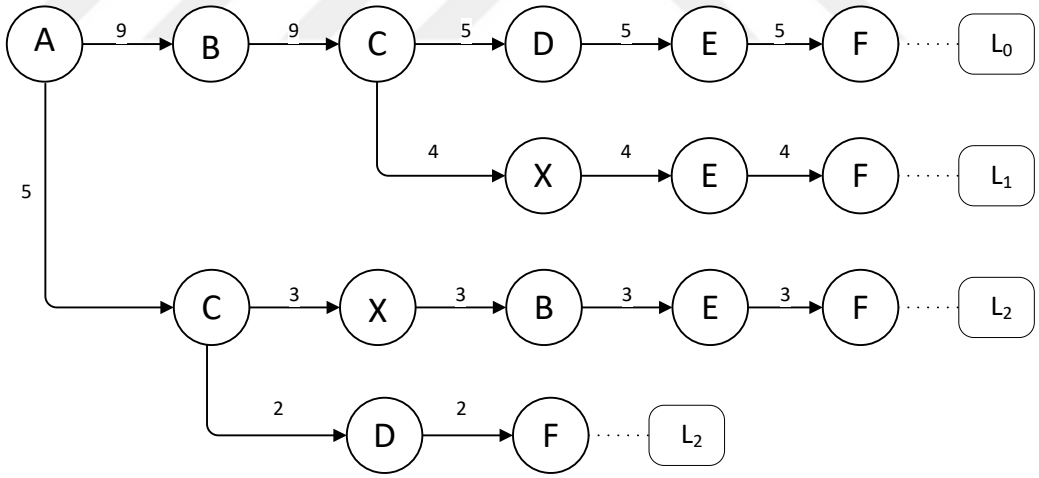


Figure 4.18 Data model after last step of activity addition

Thus, we can create our new dependency matrix from final version of data model which is reconstructed after activity addition. Table 4.8 represents the final version of dependency matrix after activity addition.

Table 4.8 Dependency matrix after activity addition

	A	**B**	**C**	**D**	**E**	**F**	**X**
A	0	9	5	0	0	0	0
B	0	0	9	0	3	0	0
C	0	0	0	7	0	0	7
D	0	0	0	0	5	2	0
E	0	0	0	0	0	12	0
F	0	0	0	0	0	0	0
X	0	3	0	0	4	0	0

According to the dependency matrix, dependency graph of new process model can be constructed as shown in Figure 4.19.

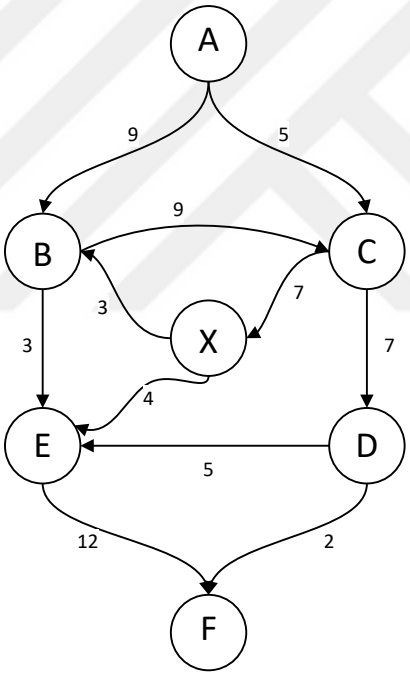


Figure 4.19 Dependency graph after activity addition

4.3 Process Model Modification

In the repair dataset (Bose & van der Aalst, 2010), different activities can be seen such as "Register" (accepting the faulty telephone), "Analyze Defect" (analyzing the telephone fault), "Repair (Simple)" (simple repair for fault of telephone), "Repair (Complex)" (complex repair for fault of telephone), "Test Repair" (testing the

telephone after repairing), "Restart Repair" (starting repairing again if the test fails), "Inform User" (informing the user after repair), and "Archive Repair" (archiving and closing fault record). The repair dataset consists of synthetically created event logs for the telephone repair process. It consists of 1,104 trace and 11,855 events. After performing analysis on the event log of telephone repair process, the created process model can be viewed in Figure 4.20-a.

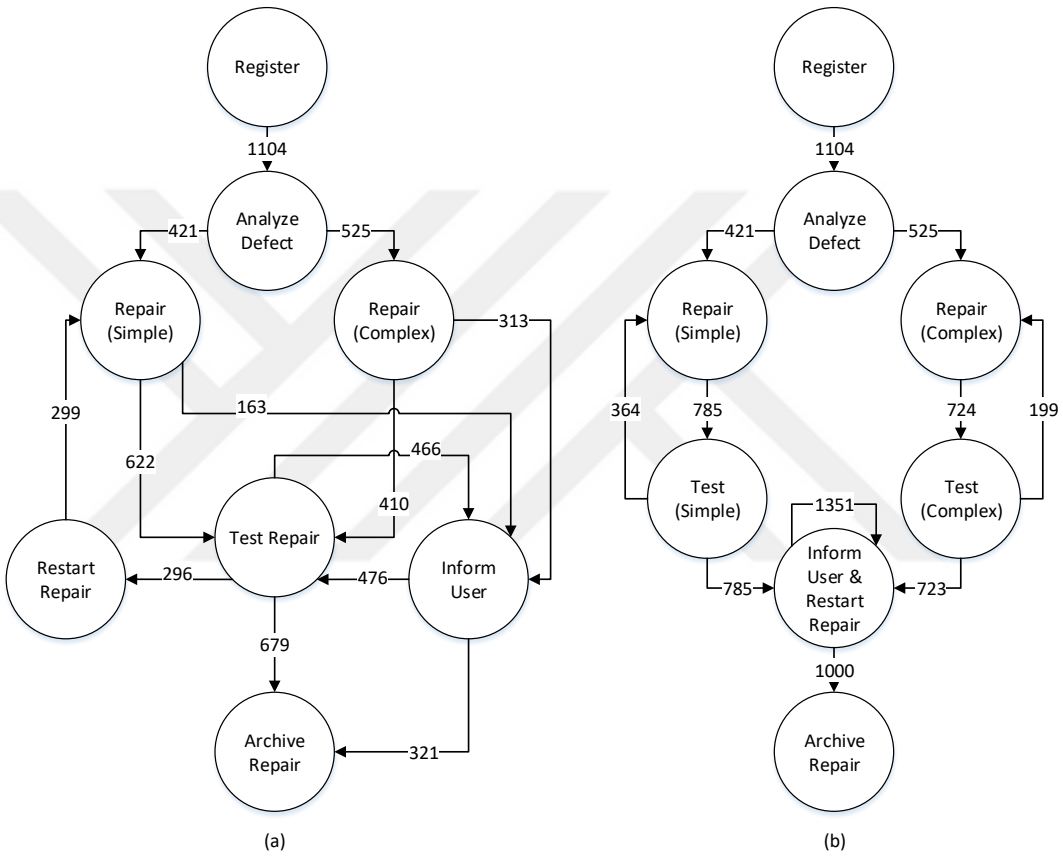


Figure 4.20 Process model modifications

When we look at the process model, it is observed that there is a lot of transitions between events, and as a result of this situation, a complicated process flow is formed. Some improvement changes should be done to reduce this complexity and make the execution of process leaner. We want to make some changes in process flow to achieve this goal. Firstly, “Test (Simple)” and “Test (Complex)” events are added after each repair events to separate test operations. In this case, existing “Test Repair” event is removed from the process flow. Finally, “Inform User” and “Restart Test” events are

aggregated under “Inform User & Restart Repair” event. After applying these changes, the process model is formed as shown in Figure 4.20-b. As stated, we can perform different experiments in a simulation environment to improve the execution of processes. Once the most appropriate model has been identified, the improvement changes start to be implemented in real life. Thanks to the provided IPM algorithm, process improvements can be achieved quickly and truthfully.

CHAPTER FIVE
TIME PREDICTION ALGORITHM

Another aim of this thesis is to develop a time prediction algorithm which calculates the remaining and completion time of each process in a flow and incorporate the execution records of ongoing processes into discovered process model instantly.

We enhanced IPM algorithm by introducing time perspective. The enhanced algorithm is capable of estimating the completion time of the processes that has not started yet and the remaining time of ongoing processes instantly. The time prediction algorithm is named as Time-oriented Interactive Process Miner, T-IPM.

In this section, we will explain how to construct time prediction model from event logs. The purposed algorithm, T-IPM, is able to make analyses on both offline and online execution records. Incorporation of ongoing or newly completed process records which are called as online data has a major importance in terms of keeping the process model constantly up-to-date. Updating the process model instantly will provide the ability to see the last status of process flows.

5.1 Concept of Time Prediction

Event logs contain different features of executed process instances. A sample event log is listed in Table 5.1. The event log contains there traces that have activity, timestamp and life cycle information. Activity, time stamp and life cycle information is important for the proposed time prediction algorithm.

The first step is to create a tree-like data model that includes summarized statistical information about all cases in the event logs. This tree-like data model makes it possible to use large volume of event logs in time prediction algorithm with a low memory usage. Figure 5.1 shows the tree-like data model that is constructed from sample event log.

Table 5.1 Sample event log for time prediction

Trace Id	Activity	Timestamp	Lifecycle
1	A	2018-01-01T12:00	Start
1	A	2018-01-01T18:00	Complete
1	B	2018-01-01T20:00	Start
1	B	2018-01-02T11:00	Complete
1	C	2018-01-02T12:30	Complete
1	D	2018-01-02T14:00	Start
1	D	2018-01-02T15:30	Complete
1	F	2018-01-02T16:00	Complete
2	A	2018-01-03T09:00	Complete
2	B	2018-01-03T11:00	Start
2	B	2018-01-03T15:30	Complete
2	D	2018-01-03T16:00	Start
2	D	2018-01-03T17:00	Complete
2	F	2018-01-03T18:30	Complete
3	A	2018-01-04T11:00	Start
3	A	2018-01-04T11:30	Complete
3	C	2018-01-04T14:00	Complete
3	D	2018-01-04T15:00	Start
3	D	2018-01-04T15:30	Complete
3	F	2018-01-04T15:51	Complete

The proposed algorithm calculates the execution and completion time of the process instances. Execution time refers to the time elapsed between the start time and the end time of an activity. If the start or end time information does not exist in the event logs, then we assume that the value of execution time is zero. Completion time refers to the time elapsed between the end time of the previous activity and the end time of the next activity.

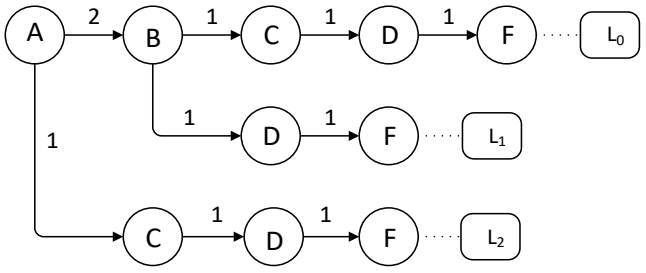


Figure 5.1 Time prediction data model of event log

Figure 5.2 shows the difference between the execution and the completion time of an activity. If the completion time value is greater than the execution time value, this means there is an idle time between two activities. If the completion time value is smaller than the execution time value, this means these two activities progresses in parallel. Figure 5.2 shows a sample process flow that continues like *A* → *B* → *C*. The activity *A* starts at time t_2 and ends at time t_4. The activity *B* starts at time t_3 and ends at time t_6. The activities *A* and *B* continues parallel for a certain period of time. The activity *C* starts at time t_7 and ends at time t_8. After the activity *B* is completed, there is an idle time which start at t_6 and ends at t_7. After the idle time finishes, the activity *C* starts to proceeding.

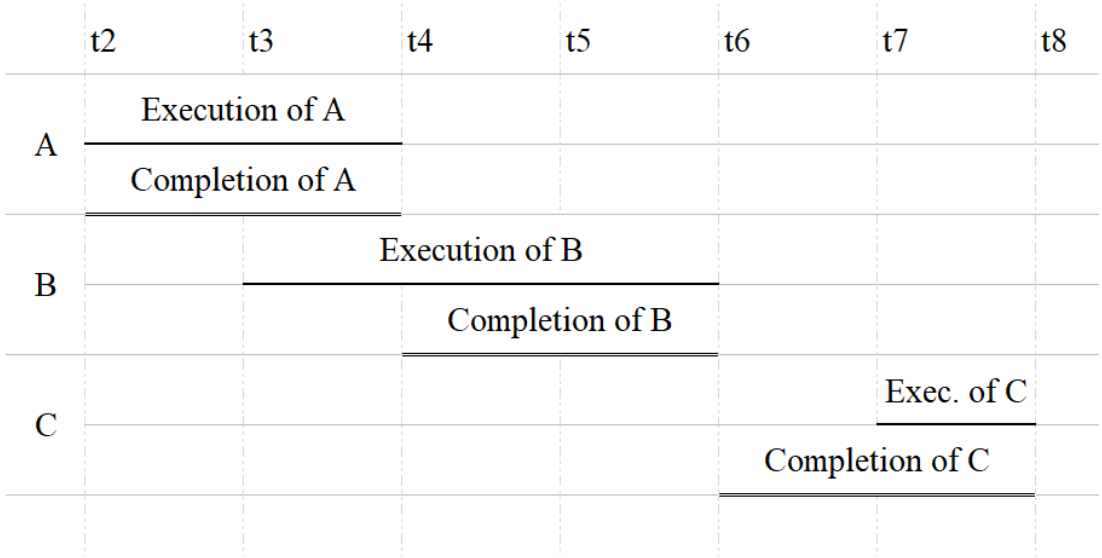


Figure 5.2 The timeline of execution and completion times

5.2 Methodology of Time Prediction

To explain the time calculation from event logs, we need the following notations. Z is a set of activities such as Z = {A, B, C, D, E}. $\sigma \in Z^*$ is a trace of process flow such as σ_1 = ABC, σ_2 = ABCD. $W \subseteq Z^*$ is an event log, i.e., a multiset (bag) of event traces such as W= [ABC, ABCD, ABDE].

Let a, b ∈ Z; $T_{start}(a)$ is the start time of activity *a*. $T_{end}(a)$ is the end time of activity *a*. $T_{et}(a)$ is the execution time of activity *a* and $T_{ct}(a)$ is the completion time

of activity *a*. If there is a trace $\sigma = e_1\ e_2\ e_3 \ldots\ e_n$ and $i \in \{1,2,3,\ldots n\}$ such that $e_i = a$, $\sigma \in W$;

$$T_{et}(a) = T_{end}(a) - T_{start}(a) \tag{4.1}$$

$$T_{ct}(a) = \begin{cases} T_{end}(a) - T_{end}(e_{i-1}), & i > 1 \\ T_{et}(a), & i = 1 \end{cases} \tag{4.2}$$

So far we explained how the execution and completion times of activities are calculated in a process flow. Equation 4.1 is used to calculate execution time value and equation 4.2 is used to calculate completion time value of an event. We will see how the average values are calculated in the next step. Let's assume that $|a \overset{in}{\rightarrow} b|$ is count of incoming edges from activity *a* to activity *b* and $|\overset{in}{a^{\rightarrow}}|$ is count of all incoming edges into activity *a*. We can define avarage time values as follows;

$$AVG_{et}(a,b) = \begin{cases} \frac{\sum_{i=1}^{|a \overset{in}{\rightarrow} b|} T_{et}(b)}{|a \overset{in}{\rightarrow} b|}, & |a \overset{in}{\rightarrow} b| > 0 \\ \frac{\sum_{i=1}^{|\overset{in}{b^{\rightarrow}}|} T_{et}(b)}{|\overset{in}{b^{\rightarrow}}|}, & |a \overset{in}{\rightarrow} b| = 0 \end{cases} \tag{4.3}$$

$$AVG_{ct}(a,b) = \begin{cases} \frac{\sum_{i=1}^{|a \overset{in}{\rightarrow} b|} T_{ct}(b)}{|a \overset{in}{\rightarrow} b|}, & |a \overset{in}{\rightarrow} b| > 0 \\ \frac{\sum_{i=1}^{|\overset{in}{b^{\rightarrow}}|} T_{ct}(b)}{|\overset{in}{b^{\rightarrow}}|}, & |a \overset{in}{\rightarrow} b| = 0 \end{cases} \tag{4.4}$$

Equation 4.3 shows how to calculate the average execution time of $A \rightarrow B$ transition and equation 4.4 shows how to calculate the average completion time of $A \rightarrow B$ transition. Now, let's define time prediction functions.

$$P_{et}(\sigma) = AVG_{et}(\varnothing, e_0) + \sum_{i=1}^{n} AVG_{et}(e_i, e_{i+1}) \tag{4.5}$$

$$P_{ct}(\sigma) = AVG_{ct}(\varnothing, e_0) + \sum_{i=1}^{n} AVG_{ct}(e_i, e_{i+1}) \quad (4.6)$$

The data model is used as input to create time table. To form time table, the algorithm starts from the root activity and continues traversing the tree. While traversing tree, the goal is to reach the leaf activity. When all leaf activities are reached, it means that the constructed tree represents the complete event log. Because, it shows all the traces which are visited so far. While traversing all the activities, time table is created for each activity and 2-length activities. Table 5.2 gives time values of 2-length activities and Table 5.3 gives time values of each activity.

Table 5.2 Time information of 2-length activities

Transition	**#**	**Total Execution Time**	**Average Execution Time**	**Total Completion Time**	**Average Completion Time**
∅ → A	3	390	130	390	130
A → B	2	1170	585	1410	705
A → C	1	0	0	150	150
B → C	1	0	0	90	90
B → D	1	60	60	90	90
C → D	2	120	60	270	135
D → F	3	0	0	141	47

Table 5.3 Time information of activities

Activity	**#**	**Total Execution Time**	**Average Execution Time**	**Total Completion Time**	**Average Completion Time**
A	3	390	130	390	130
B	2	1170	585	1410	705
C	2	0	0	240	120
D	3	180	60	360	120
F	3	0	0	141	47

Let us explain the time estimation algorithm with a sample flow. Assume that a flow of σ1 = ACBDF will take place. This flow is transformed into binary transitions in order to estimate the time. At the end of this operation, binary transitions are obtained in the form ∅ → *A*, *A* → *C*, *C* → *B*, *B* → *D* and *D* → *F*. It is necessary to

find the average times for each binary transitions to calculate the completion time value with equation 4.6. Table 5.4 shows the calculated average time values. If there is a transition in the event log in the form of $a \rightarrow b$, the calculated average value for this pair is obtained from Table 5.2. If there is no such transition, the overall calculated average value for activity b is obtained from Table 5.3.

The average completion time values for each transition pair in the example flow are shown in Table 5.4. These values are summed to estimate the completion time value for the corresponding flow. Equation 4.6 is calculated as 1122 for σ1.

Table 5.4 Average completion time of transitions

Transition	$\|\boldsymbol{a} \overset{\boldsymbol{in}}{\rightarrow} \boldsymbol{b}\|$	$\|\overset{\boldsymbol{in}}{\boldsymbol{a}^{\rightarrow}}\|$	$\boldsymbol{AVG_{ct}(a,b)}$
∅ → A	**130**	130	130
A → C	**150**	120	150
C → B	**-**	**705**	705
B → D	**90**	120	90
D → F	**47**	47	47

Equation 4.5 is used to calculate the execution time value. The average execution time values for each binary transitions in the sample flow are shown in Table 5.5. The estimated value for σ1 is calculated as 645 by taking into account the average time values.

Table 5.5 Average execution time of transitions

Transition	$\|\boldsymbol{a} \overset{\boldsymbol{in}}{\rightarrow} \boldsymbol{b}\|$	$\|\overset{\boldsymbol{in}}{\boldsymbol{a}^{\rightarrow}}\|$	$\boldsymbol{AVG_{et}(a,b)}$
∅ → A	**0**	0	0
A → C	**0**	0	0
C → B	**-**	**585**	585
B → D	**60**	60	60
D → F	**0**	0	0

CHAPTER SIX
A NEW PROCESS MINING TOOL: ProLab

There is a need for a software tool in order to analyze the process records and to extract the statistical information from event logs. This process mining tool should be able to work with low resource consumption on the large amount of event logs. An XML-based file called Mining eXtensible Markup Language (MXML) has been standardized to store event logs which is used in process mining. Thus, a general input format is provided for different process mining techniques and tools. In this thesis, we developed a new process mining tool, ProLab, which is able to work with low resource consumption on huge amount of event logs and to provide an interactive environment for users.

6.1 Block Diagram of Process Mining Tool

The block diagram of the process mining tool we developed is shown in Figure 6.1. Event logs in MXML format are read using the file streaming method, so it is possible to read the event logs without loading the whole file into memory. The statistical information obtained from event logs during streaming is stored in a data structure. This data structure describes the summary information that will be used in process mining.

The process mining algorithm is executed on the data structure that holds the statistical information and a process model is created that represents the process flows. The generated process model is a summary information of the event logs. By visualizing the generated process model, this summary information is displayed to the user with a graphical interface. With the help of settings, users can filter both the event logs and change the visual appearance of the process model. These features allow users to perform a detailed analysis on the event logs.

In addition to visualization of the process model, statistical information about the event logs is presented to the user through a dashboard page with graphics and tables. With this dashboard page, users can get deep insights about the process flows.

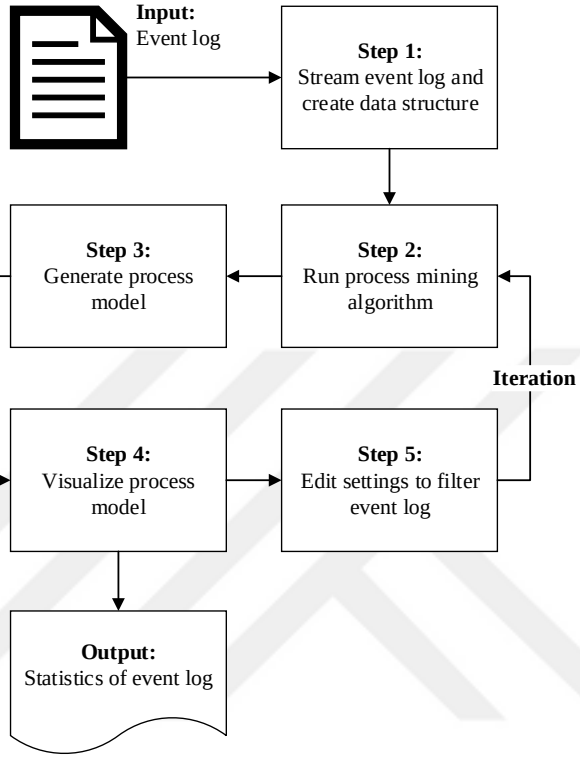


Figure 6.1 Block diagram of process mining tool

6.2 Case Study of Process Mining Tool

Table 6.1 shows an example event logs. Each row in the table represents a completed process flow. A, B, C, D, E, F in the process flow are the names of the activities that can take place in the process execution.

The proposed tool, ProLab, has the option of advanced settings to perform detailed analysis from different point of views. *Case Frequency* and *Activity Frequency* options are available for filtering event logs.

The process instance, ABCDF, in the example event log repeated 5 times and the frequency is 35.71%. The second process instance, ACDEF, repeated 4 times and the

frequency is %28.57. The third process instance, ACBDF, repeated 3 times and the frequency is %21.42. The last process instance, ABEF, repeated 2 times and the frequency is %14.28. When *Case Frequency* value is set to 15%, the process instance, ABEF, will be ignored in analysis. *Activity Frequency* filter allows filtering of activities A, B, C, D, E and F according to the ratio of the total number of activities in the event log.

Table 6.1 Sample event log for process mining tool

Case ID	**Events**
Case 1	ABCDF
Case 2	ACDEF
Case 3	ACBDF
Case 4	ABEF
Case 5	ABCDF
Case 6	ABCDF
Case 7	ACBDF
Case 8	ACDEF
Case 9	ACBDF
Case 10	ACDEF
Case 11	ABCDF
Case 12	ABEF
Case 13	ACDEF
Case 14	ABCDF

The resulting process model that is created based on analysis can be drawn to show the frequency or time information. When it is desired to draw according to frequency value, there are two options. One of these options is *Absolute Frequency* that expresses total activity count and the other one is *Case Frequency* that expresses the activity count per case.

There are 4 different options when it is desired to draw the generated process model according to time value: (i) *total duration* refers to total time, (ii) *mean duration* refers to average time, (iii) *min duration* refers to minimum time and (iv) *max duration* refers to maximum duration in event logs. During visualization, different colors are used,

which are divided into 5 categories for shapes. Thanks to this method, both the colored shapes and the numbers make the process model more understandable.

Figure 6.2 denotes the analysis that is performed on repair data set. The *Case Frequency* value for the analysis is defined as 10%, and the *Activity Frequency* value is defined as 20%. An analysis is performed for the frequency values of the activities and the *Absolute Frequency* option is marked. Frequency values are shown on the activities and on the edges that link the activities. The colors of the activities and edges vary depending on the value of frequency. Moreover, the edge thicknesses also vary depending on the value of dependency size of the activities

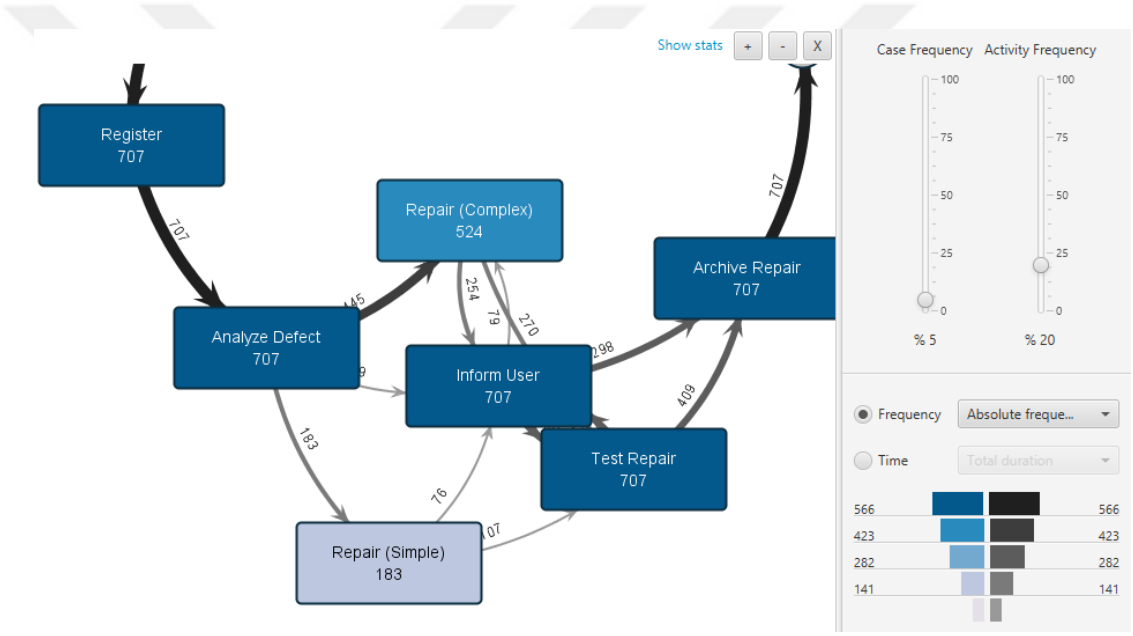


Figure 6.2 Process model visualization in ProLab

CHAPTER SEVEN
EXPERIMENTAL STUDY

The experiments were performed by using 2.4 GHz quad core processor, 16GB RAM. The code was implemented in Java platform. In this thesis, we proposed a process mining algorithm, IPM, and a time prediction algorithm, T-IPM, which is enhanced version of IPM algorithm, and also developed a process mining tool, ProLab. We performed different experimental studies for each one.

7.1 Dataset Description

Datasets of *traffic* (de Leoni & Mannhardt, 2015), *hospital* (van Dongen, 2011), *billing* (Mannhardt, 2017) and *repair* (Bose & van der Aalst, 2010) event logs are used for experimental study. Table 7.1 shows detailed information about the datasets. The hospital dataset includes real-life event logs of the clinical treatment process of an academic hospital in the Netherlands. It consists of 1,143 traces and 150,291 events. Hospital dataset is composed of long and complex event logs, which are defined as Spaghetti. Traffic dataset includes event logs generated by an information system that performs road traffic control. It consists of 150,370 traces and 561,470 events. Billing dataset includes event logs generated by the financial modules of the ERP system of a regional hospital in Netherlands. It consists of 100,000 traces and 451,359 events. The repair dataset consists of synthetically created event logs for the telephone repair process. It consists of 1,104 trace and 11,855 events.

Table 7.1 Characteristics of datasets

Dataset	Traces	Events	Activities	Min Events per Trace	Max Events per Trace
Repair Dataset	1,104	11,855	12	4	24
Hospital Dataset	1,143	150,291	624	1	1,814
Billing Dataset	100,000	451,359	18	1	217
Traffic Dataset	150,370	561,470	11	2	20

7.2 Experimental Results of Process Mining Algorithm

To evaluate process mining algorithm, IPM, the experiments were performed on ProM platform (van Dongen et al., 2005). The first experiment was executed on *hospital* and *traffic* datasets to compare the running time and memory usage of four algorithms: Alpha Miner (van der Aalst et al., 2004), Heuristics Miner (Weijters et al., 2006), Fuzzy Miner (Günther & van der Aalst, 2007), and Interactive Process Miner (IPM - our algorithm). The results of this experiment given in Table 7.2 show the running time and memory usage of IPM algorithm is better than others. IPM created the process model in 2.38 seconds by using 463 MB RAM on hospital dataset and 0.67 seconds by using 36 MB RAM on traffic dataset. When we checked the running times and memory usage of the algorithms, we observed that IPM is the fastest algorithm and has lowest memory consumption comparing to others.

Table 7.2 Experimental results for performance evaluation

	Hospital Dataset		**Traffic Dataset**	
	Running Time(sec)	**Memory Usage (MB)**	**Running Time(sec)**	**Memory Usage (MB)**
Alpha Miner	>120.00	1,503	1.87	675
Fuzzy Miner	39.11	1,806	7.52	789
Heuristics Miner	19.90	1,087	3.62	825
IPM (our algorithm)	2.38	463	0.67	36

The second experiment was executed on *repair* dataset. Multi-Perspective Process Explore plug-in was used to evaluate the quality of created process model (Mannhardt, de Leoni, & Reijers, 2015). This plug-in requires two input files. One of them is the XES file that contains the event log. The other one is the Petri net representation of the process model. This plug-in does not support the conversion of created model to Petri net representation for Alpha Miner and Fuzzy Miner algorithms. For this reason, we only used Heuristic Miner algorithm to compare the results of the second experiment. In second experiment, we focused on fitness and precision metrics to evaluate the success of process model created by the algorithm (Mannhardt et al., 2015). Fitness shows how much of the observed traces in the log are described by the process model

(equation (6.1)). Precision is the ratio between the amount of traces observed in the event log and the amount of traces described by the model (equation (6.2)).

$$fitness(L,N) = \frac{1}{2}\left(1 - \frac{\sum_{\sigma \in L} L(\sigma) \times m_{N,\sigma}}{\sum_{\sigma \in L} L(\sigma) \times c_{N,\sigma}}\right) + \frac{1}{2}\left(1 - \frac{\sum_{\sigma \in L} L(\sigma) \times r_{N,\sigma}}{\sum_{\sigma \in L} L(\sigma) \times p_{N,\sigma}}\right) \quad (6.1)$$

where L is event log, N is process model, σ is trace, L(σ) is the frequency of trace σ, p is produced tokens, c is consumed tokens, m is missing tokens, and r is remaining tokens.

$$precision(P,\varepsilon) = \frac{\sum_{e \in \varepsilon} |obs_p(e)|}{\sum_{e \in \varepsilon} |pos_p(e)|} \quad (6.2)$$

where P is process model, Ɛ is event log, e is event, obsp(e) is the observed behavior as seen in the event log, posp(e) is the possible behavior when event e occurs as allowed by a model.

The framework calculated the fitness 84.8% and the precision 73.9% for created model by IPM as given in Table 7.3. Although fitness values are very close to each other (IPM: 84.8%, Heuristic Miner: 88.7%), there is a significant difference between the precision values of two algorithms (IPM: 73.9%, Heuristic Miner: 57.5%). It is necessary to evaluate these two metrics together. When we look at fitness value, the process model created by IPM contains 84.8% of the event log. On the other side, the process model created by Heuristic Miner contains 88.7% of the event log. When we look at the precision value, 73.9% of the execution variants that we can build by looking at the process model created by IPM are observed in the event log. Conversely, only 57.5% of the execution variants that we can build by looking at the process model created by Heuristic Miner are observed in the event log. Clearly, Heuristic Miner has created a very general process model to conform it to the event log and a big part of execution variants that expressed by process model cannot be observed in the event log. From this point of view, we can say that the model created by IPM is closer to reality and more successful than Heuristic Miner.

In summary, the proposed process mining algorithm, IPM, that runs on large datasets and handles execution records of running instances in a short time with low memory usage. The algorithm provides an interactive method that allows users to modify the constructed model by adding, deleting and aggregating the activities to see the impacts of process improvement changes in a simulation environment before applying decisions in real life.

Table 7.3 Experimental results to evaluate the success of process model

Algorithms	Precision (%)	Fitness (%)
Heuristics Miner	57.50	88.70
IPM (our algorithm)	73.90	84.80

7.3 Experimental Results of Time Prediction Algorithm

To evaluate time prediction algorithm, T-IPM, the experiments were performed on the developed process mining tool, ProLab. The first experiment was executed on *billing*, *traffic* and *repair* datasets to compare the running time and memory usage. The results of the experiment given in Table 7.4 show the running time and memory usage of the proposed algorithm for each datasets.

Table 7.4 Experimental results of performance evaluation for time prediction algorithm

Dataset	Log Size (MB)	Running Time (sec)	Memory Usage (MB)
Repair Dataset	3.31	1	321
Billing Dataset	166	14	735
Traffic Dataset	176	17	826

The proposed algorithm, T-IPM, created the process model in 1 second by using 321 MB RAM on repair dataset and 14 seconds by using 735 MB RAM on billing dataset and 17 seconds by using 826 MB RAM on traffic dataset. When we checked the running times and memory usage, we observed that the size of event logs increased the running time. The factor that affects the amount of memory usage is the complexity of the event logs and the length of each process instance. These two cases are the two most important factors affecting the size of the data structure in which summary information obtained from event logs. The results of the experiments point that it is

possible to perform process analysis on a large volume of data by using limited resources with the proposed time prediction algorithm.

The second experiment was executed on *billing*, *traffic* and *repair* datasets to evaluate the success of time prediction algorithm. The results of the experiment given in Table 7.5 show *Mean Absolute Error (MEA)*, *Root Mean Squared Error (RMSE)*, and *Mean Absolute Percentage Error (MAPE)* of the proposed algorithm for each datasets.

Table 7.5 Experimental results of validation for time prediction algorithm

Dataset	**MAE**	**RMSE**	**MAPE**
Repair Dataset	0.1774	0.2370	16.44 %
Hospital Dataset	391.6208	544.8783	12.80 %
Traffic Dataset	1620.5391	1966.9926	19.38 %

Without considering the direction of values, the average magnitude of the errors in a set of predictions is measured by MAE. MAE is the average of the absolute differences between actual observation and prediction when all individual differences have equal weight.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} | x_i - \hat{x}_i | \tag{6.3}$$

RMSE is the square root of the average of squared differences between actual observation and prediction. It is a quadratic scoring rule. RMSE measures the average magnitude of error.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{x}_i)^2} \tag{6.4}$$

MAPE calculate the average of the percentage error. The size of error is measured in percentage terms.

$$MAPE = \left(\frac{1}{n} \sum_{i=n}^{n} \frac{| x_i - \hat{x}_i |}{x_i} \right) * 100 \tag{6.5}$$

The mean completion times are 1.08 hours, 3056.68 hours, and 8200.29 hours for repair, billing and traffic datasets, respectively. While calculating MAE, RMSE and MAPE values, the unit of time is selected as hour. MAE and RMSE shows the average value of prediction error. It is possible to compare these values by the mean completion times of datasets. MAPE shows the percentage error of prediction. We can say that the accuracy values of prediction are 83.56%, 87.20% and 80.62% for repair, hospital and traffic datasets, respectively. When we analyze these values, it is seen that the proposed algorithm is very successful in time estimation.

7.4 Experimental Results of Process Mining Tool

To evaluate process mining tool, ProLab, the experiment was executed on *hospital*, *traffic* and *repair* datasets to compare the running time and memory usage. The results of the experiment given in Table 7.6 show the running time and memory usage of the developed tool for each datasets.

Table 7.6 Experimental results of performance evaluation for process mining tool

Dataset	Log Size (MB)	Running Time (sec)	Memory Usage (MB)
Repair	3.31	1	321
Hospital	81.40	5	745
Traffic	176.00	17	826

The tool created the process model in 1 second by using 321 MB RAM on *repair* dataset and 5 seconds by using 745 MB RAM on *hospital* dataset and 17 seconds by using 826 MB RAM on *traffic* dataset. When we checked the running times and memory usage, we observed that the size of event logs increased the running time. The factor that affects the amount of memory usage is the complexity of the event logs and the length of each process instance. These two cases are the two most important factors affecting the size of the data structure in which summary information obtained from event logs. The results of the experiments point that it is possible to perform process analysis on a large volume of data by using limited resources with the developed process mining tool.

CHAPTER EIGHT
CONCLUSION AND FUTURE WORK

8.1 Conclusion

This thesis proposes a new process mining algorithm, Interactive Process Miner (IPM), to create process model based on event logs and predict the remaining and completion time of each process in a flow and, also a new approach that contains three different features; including activity deletion, aggregation and addition operations on the existing process model.

We also enhanced IPM algorithm by introducing time perspective. The enhanced algorithm, Time-oriented Interactive Process Miner (T-IPM), is capable of estimating the completion time of the processes that has not started yet and the remaining time of ongoing processes instantly.

In this thesis, we also developed a new process mining tool, ProLab, which has the capabilities of working on a large volume of event logs and handling the execution records of running process instances to create process model in a short time and also supports an interactive environment for process mining to give deep insights for event logs.

The contribution of this thesis can be summarized as the following:

- The proposed algorithms, IPM and T-IPM, are able to analyze historical event logs as well as to incorporate the execution records of ongoing processes into the process model instantly. IPM and T-IPM algorithms support both online and offline process mining fashions.
- IPM and T-IPM enable for modification on the discovered process model. Thus, algorithms provide to observe the effects of possible decisions to be taken in a simulation environment. It has an important feature in order to make the right decision and to observe possible problems in advance

- Experimental studies have proved that IPM and T-IPM algorithms are the fastest algorithm that consumes the least amount of memory and, also has high prediction accuracy.
- Experimental studies have proved that the proposed tool, ProLab, is able to analyze a large volume of event logs.

As a result, in this thesis, (i) a novel process mining algorithm, IPM, was proposed, (ii) IPM algorithm was enhanced by introducing time perspective and was named as T-IPM, (iii) a process mining tool, ProLab, was developed.

8.2 Future Work

In the future, it is possible to enhance IPM implementation by introducing new process mining perspectives such as organizational and resource in user interactive environment.

In addition, a new file format that takes less space for event logs can be created for ProLab tool. Thus, low-cost storage spaces for event logs will suffice. Different visualization techniques such as fish eye and zooming can be supported in ProLab tool to analyze the process model and the statistics of event log in a various perspectives.

Parallel processing can be supported to achieve faster results on large volume of event logs. Event logs can be stored in distributed data processing systems to develop a parallel and scalable process mining system.

Furthermore, in order to make the process flow more understandable, a simulation framework can be developed which the event logs can be played as an animation within a predetermined time period.

REFERENCES

Agrawal, R., Gunopulos, D., & Leymann, F. (1998). Mining process models from workflow logs. *Proceedings of the 6th International Conference on Extending Database Technology*, 467-483.

Aleem, S., Capretz, L. F., & Ahmed, F. (2015). Business process mining approaches: a relative comparison. *International Journal of Science, Technology & Management, 4* (1), 1557-1564.

Alizadeh, M., Lu, X., Fahland, D., Zannone, N., & van der Aalst, W. M. P. (2018). Linking data and process perspectives for conformance analysis. *Computers & Security, 73*, 172-1923.

Appice, A., Pravilovic, S., & Malerba, D. (2013). Process mining to forecast the future of running cases. *2nd International Workshop on New Frontiers in Mining Complex Patterns*, 67-81.

Bolt, A., de Leoni, M., & van der Aalst, W. M. P. (2016). Scientific workflows for process mining: building blocks, scenarios, and implementation. *International Journal on Software Tools for Technology Transfer, 18* (6), 607-628.

Bolt, A., de Leoni, M., ter Hofstedea, A. H. M., & van der Aalst, W. M. P. (2017). Process variant comparison: Using event logs to detect differences in behavior and business rules. *Information Systems, 74* (1), 53-56.

Bose, R. P. J. C., & van der Aalst, W. M. P. (2010). Trace alignment in process mining: Opportunities for process diagnostics. *8th International Conference on Business Process Management*, 227-242.

Bose, R. P. J. C., van der Aalst, W. M. P., Žliobaitė I., & Pechenizkiy, M. (2011). Handling concept drift in process mining. *23rd International Conference on Advanced Information Systems Engineering,* 391-405.

Bratosin, C., Sidorova, N., & van der Aalst, W. M. P. (2007). Distributed genetic process mining. *IEEE Congress on Evolutionary Computation,* 1-8.

Cheng, H. J., & Kumar, A. (2015). Process mining on noisy logs — can log sanitization help to improve performance*? Decision Support Systems, 79,* 138-149.

Cook, J. E., & Wolf, A. L. (1998). Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology, 7* (3), 215-249.

Cook, J. E., Du, Z., Liu, C., & Wolf, A. L. (2004). Discovering models of behavior for concurrent workflows. *Computers in Industry, 53* (3), 297-319.

De Leoni M., Mannhardt F. (2015). Road traffic fine management process. *Eindhoven University of Technology Dataset.* Retrieved May 22, 2018, from https://data.4tu.nl/repository/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.

De Leoni, M., van der Aalst, W. M. P., & Dees, M. A. (2016). A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems, 56,* 235-257.

De Medeiros, A. K. A., Weijters, A. J. M. M., and van der Aalst, W. P. M. (2007). Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery, 15* (2), 245-304.

Eder, J., Panagos, E., & Rabinovich, M. (1999). Time Constraints in Workflow Systems. *11th Conference on Advanced Information Systems Engineering (CAiSE),* 165-280.

Eder, J., & Pichler, H. (2005). Probabilistic calculation of execution intervals for workflows. *12th International Symposium on Temporal Representation and Reasoning (TIME'05)*, 183-185.

Fahland, D., & van der Aalst, W. M. P. (2013). Simplifying discovered process models in a controlled manner. *Information Systems, 38* (4), 585-605.

Fahland, D., & van der Aalst, W. M. P. (2015). Model repair — aligning process models to reality. *Information Systems, 47*, 220-243.

Günther, C. W., & van der Aalst, W. M. P. (2007). Fuzzy Mining: Adaptive process simplification based on multi-perspective metrics. *5th International Conference on Business Process Management*, 328-343.

Herbst, J., & Karagiannis, D. (2004). Workflow mining with InWoLvE. *Computers in Industry, 53* (3), 245-264.

Leitner, P., Wetzstein, B., Rosenberg, F., Michlmayr, A., Dustdar, S., & Leymann, F. (2009). Runtime prediction of service level agreement violations for composite services. *Service-Oriented Computing. ICSOC/ServiceWave Workshops*, 176-186.

Luengo, D., & Sepulveda, M. (2011). Applying clustering in process mining to different versions of business process that changes over time. *BPM 2011 International Workshops*, 153-158.

Mannhardt, F., de Leoni, M., & Reijers, H. A. (2015). The multi-perspective process explorer. *Demo Session of the 13th International Conference on Business Process Management*, 130-134.

Mannhardt, F., de Leoni, M., Reijers, H. A., & van der Aalst, W. M. P. (2016). Balanced multi-perspective checking of process conformance. *Computing, 98* (4), 407-437.

Mannhardt, F. (2017). Hospital Billing - Event Log. *Eindhoven University of Technology Dataset*. Retrieved May 22, 2018, from https://data.4tu.nl/repository/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741.

Mitsyuk, A. A., Shugurov, I. S., Kalenkova, A. A., & van der Aalst, W. M. P. (2017). Generating event logs for high-level process models. *Simulation Modelling Practice and Theory, 74,* 1-16.

Pika, A., van der Aalst, W. M. P., Wynn, M. T., Fidge, C. J., & ter Hofstede, A. H. M. (2016). Evaluating and predicting overall process risk using event logs. *Information Sciences, 352-353,* 98-120.

Polato, M., Sperduti, A., Burattin, A., & de Leoni, M. (2014). Data-aware remaining time prediction of business process instances. *International Joint Conference on Neural Networks (IJCNN),* 38-52.

Polato, M., Sperduti, A., Burattin, A., & de Leoni, M. (2016). Time and activity sequence prediction of business process instances. *Computing,* 1-27.

Reijers, H. A. (2006). Case prediction in BPM systems: a research challenge. *Journal of the Korean Institute of Industrial Engineers, 33* (1), 1-10.

Rovani, M., Maggi, F. M., de Leoni, M., & van der Aalst, W. M. P. (2015). Declarative process mining in healthcare. *Expert Systems with Applications, 42* (23), 9236-9251.

Schimm, G. (2004). Mining exact models of concurrent workflows. *Computers in Industry, 53* (3), 265-281.

Schonenberg, H., Weber, B., van Dongen, B., & van der Aalst, W. M. P. (2008) Supporting flexible processes through recommendations based on history. *6th International Conference on Business Process Management,* 51-66.

Song, W., Liu, S., & Liu, Q. (2008). Business process mining based on simulated annealing. *9th International Conference for Young Computer Scientists,* 135-139.

Song, M., Gunther, C. W., & van der Aalst, W. M. P. (2008). Trace clustering in process mining. *BPM 2008 International Workshops,* 109-120.

Suriadi, S., Andrewsa, R., ter Hofstedea, A. H. M., & Wynna, M. T. (2017). Event log imperfection patterns for process mining: towards a systematic approach to cleaning event logs. *Information Systems, 64,* 132-1564.

Taxa, N., Sidorovaa, N., Haakmab, R., & van der Aalst, W. M. P. (2016). Mining local process models. *Journal of Innovation in Digital Ecosystems, 3* (2), 183-196.

Van der Aalst, W. M. P., Weijters, T., & Maruster, L. (2004). Workflow mining: discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering, 16* (9), 1128-1142.

Van der Aalst, W. M. P., Pesic, M., & Song, M. (2010). Beyond process mining from the past to present and future. *22nd International Conference on Advanced Information Systems Engineering (CAiSE),* 38-52.

Van der Aalst, W. M. P. (2011). *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin-Heidelberg: Springer-Verlag.

Van der Aalst, W. M. P., Schonenberg, M. H., & Song, M. (2011). Time prediction based on process mining. *Information Systems, 36* (2), 450-475.

Van der Aalst, W. M. P. (2012). Process mining: overview and opportunities. *ACM Transactions on Management Information Systems, 3* (2), 1-17.

Van Dongen, B. F., & van der Aalst, W. M. P. (2004). EMiT: A process mining tool. *25th International Conference on Application and Theory of Petri Nets,* 454-463.

Van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H. M. W., Weijters, A. J. M. M., & van der Aalst, W. M. P. (2005). The ProM framework: a new era in process mining tool support. *26th International Conference on Application and Theory of Petri Nets,* 444-454.

Van Dongen, B. F., & van der Aalst, W. M. P. (2005). A Meta model for process mining data. *CAiSE WORKSHOPS,* 309-320.

Van Dongen, B. F., Crooy, R. A., & van der Aalst, W.M.P. (2008). Cycle time prediction: when will this case finally be finished? *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, 319-336.

Van Dongen, B. F. (2011). Real-life event logs - Hospital log. *Eindhoven University of Technology Dataset.* Retrieved May 22, 2018, from https://data.4tu.nl/repository/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54.

Verwer, S., de Weerdt, M. M., & Witteveen, C. (2008). Efficiently learning timed models from observations. *Benelearn Conference,* 75-76.

Weijters, A. J. M. M., & van der Aalst, W. M. P. (2003). Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering, 20* (2), 151-162.

Weijters, A. J. M. M., van der Aalst, W. M. P., & de Medeiros, A. K. A. (2006). Process mining with the HeuristicsMiner algorithm. *Technische Universiteit Eindhoven Technical Report, 166*, 1-34.

Yürek, İ., Birant, D., & Birant, K. U. (2018). Interactive process miner: a new approach for process mining. *Turkish Journal of Electrical Engineering & Computer Sciences, in press*.

Yürek, İ., & Birant D. (2018). ProLab: A new process mining tool. *International Conference on Theoretical and Applied Computer Science and Engineering 2018, in press*.

APPENDICES

APPENDIX: LIST OF ACRONYMS

Acronym	Definition
IPM	Interactive Process Miner
ProLab	Process Laboratory
XML	eXtensible Markup Language
MXML	Mining eXtensible Markup Language
ProM	Process Miner
PAIS	Process Aware Information System
SLA	Service Level Agreement
EMiT	Enhanced Mining Tool
PCT	Predictive Clustering Tree
BPMN	Business Process Model and Notation
BPM	Business Process Management
ERP	Enterprise Resource Planning
XES	eXtensible Event Stream
MEA	Mean Absolute Error
RMSE	Root Mean Squared Error
MAPE	Mean Absolute Percentage Error
KPI	Key Performance Indicator