Mining High-Quality Business Process Models from Real-Life Event Logs

Dissertation

FOR THE DEGREE OF
Doctor of Natural Sciences (DR. RER. NAT.)

Yaguang Sun

University of Augsburg
Department of Computer Science
Software Methodologies for Distributed Systems

NOVEMBER, 2017

Supervisor	**Prof. Dr. Bernhard Bauer**, Department of Computer Science University of Augsburg, Germany
Advisor	**Prof. Dr. Robert Lorenz**, Department of Computer Science, University of Augsburg, Germany
Defense	08 February 2018
Copyright	

Abstract

In the last years, Business Process Mining (BPMI) has become a very important research topic in academia. In the industry, also more and more big companies are starting to use such a technique to help them understand how their business processes are implemented in reality and locate the inefficient and noneffective part in their business processes. Traditional BPMI research topic can be classified into three sub-topics: Business Process Model Discovery (BPMD), conformance checking and process extension. However, as one of the most significant branch in the BPMI research area, the present BPMD techniques meet great challenges when mining process models from real-life event logs. "Spaghetti-like" process models are often generated. Such models are normally inaccurate and very complex. The main reason is that in the real world many businesses are often executed in highly flexible environments, e.g., healthcare, customer relationship management (CRM) and product development. As a result, the event logs that stem from such flexible environments often contain dense distribution of cases with a high variety of complex behaviours. In this thesis, we explore the approaches and techniques to help existing BPMD techniques generate accurate and simple process models when mining real-life event logs.

The approaches and techniques presented in this thesis mainly inherit the basic ideas of three classical strategies proposed in the literature for assisting the BPMD techniques in mining process models with high quality which are Mining Algorithm Enhancement-Based Strategy (MEBS), Model Division-Based Strategy (MDS) and Model Abstraction-Based Strategy (MAS). Moreover, the proposed techniques are also carefully designed so as to overcome the weaknesses of the current realisations of the three strategies. The main contributions of this thesis are as follows:

1. For the MEBS, we have developed a new technique named HIF which is able to help existing BPMD techniques overcome their limitations on their expressive ability. The working principle is that HIF can locate the inexpressible process behaviours in the given event logs and then transform them into expressible behaviours for the utilised BPMD techniques.

2. For the MDS, we have developed two trace clustering techniques named TDTC and CTC and one multi-label case classification technique named MLCC. The techniques TDTC and CTC are devised to optimise the accuracy and complexity of the potential sub-process models of each trace cluster during the runtime so as to assure the quality of the generated sub-models. The technique MLCC is able to combine the domain knowledge from the process experts so as to make a more meaningful division of the raw cases

from a specific event log.

3. For the MAS, we have developed a mined model abstraction technique named GTCA which utilises a new model abstraction strategy proposed by us. Through this strategy, GTCA is capable of generating an abstraction process model with higher fitness and lower complexity which cannot be ensured by existing realisations of the MAS. Furthermore, trace clustering technique is employed by GTCA for optimising the quality of the found sub-process models.

Acknowledgements

I would like to thank all the people who have ever supported me on writing my thesis:

- First of all, I really appreciate the opportunity offered by my supervisor Prof. Dr. Bernhard Bauer for doing the research in the area of Business Process Mining in the University of Augsburg. His kind guidance, support and care are the keys for the completion of this thesis.

- I am also grateful to Prof. Dr. Robert Lorenz who would like to be advisor of my thesis.

- I wish to express my gratitude to my colleagues from the Software Methodologies for Distributed Systems group who have created great academic environment which I enjoyed very much. I will never forget those insightful discussions with them which have helped me recognise the nature of the problems I was facing.

- I want to thank all the other people who I met in the international conferences and have proposed a lot of valuable suggestions for the contents of my thesis.

- I would like to thank all my friends who kept encouraging me whenever I met obstacles in the process of research. Your company also have given me strength to overcome the difficulties encountered in my life.

- Especially, I want to thank my parents Yiying Liu and Yi Sun. They have sacrificed a lot on the cultivation of my positive outlook and values from which I will benefit in my entire life.

Contents

1

Introduction

Today, companies are putting increasing emphasis on constant Business Process Improvement (BPI) [1–3] with the purpose of enhancing self-competitive capabilities under the contemporary changeable marketing environment. For instance, to provide customers with consistently reliable and low-cost products and services, current multinational corporations like Wal-Mart, FedEx and McDonald's spare no efforts to continually improve a broad range of internal operation processes. The entry point for BPI is usually a deep analysis of enterprise business processes together with the process execution-related data recorded by the Process-Aware Information Systems (PAIS) [4] such as Workflow Management (WFM), Enterprise Resource Planning (ERP), Supply Chain Management (SCM) and Customer Relationship Management (CRM) systems in which the business processes are implemented. The analysis results obtained cannot only trigger the process improvement or reengineering [5, 6] by helping companies identify the causes of malfunctions and bottlenecks of their business processes, but can foster innovations in the companies through facilitating decision-makings as well. Furthermore, companies are able to predict future business problems and supervise the consistency between operational processes and business strategies by analysing their business processes [7].

A process cannot be improved or redesigned before it is understood. Business process models play an important role in analysing and improving enterprise business processes because they are capable of giving the process analysts a better understanding of how the companies' business processes are executed. In [8, 9], the functions of business process models for corporations are well summed up in eight perspectives:

- *Insight*: process models can help business analysts understand how the companies run their businesses.

- *Discussion*: process models can serve as a common unified language and methodology for stakeholders and business analysts to exchange views and information about business processes.

- *Documentation*: process models can help employees comprehend their jobs easily and also support compliance initiatives such as Six Sigma and ISO 9000 quality management.

- *Verification*: process models can be analysed for detecting errors in systems or procedures (e.g., underlying deadlocks).

- *Performance analysis*: process models can support various kinds of process simulations which help business managers to ensure that their processes are optimised and implementing correctly.

- *Animation*: process models enable end users to "play out" different scenarios and thus provide feedback to the designers.

- *Specification*: process models can help define the modules of PAIS before they are carried out and can hence be regarded as a bridge between developers and end users.

- *Configuration*: process models can be employed to configure PAIS.

Business process models are often designed in enterprises manually. However, artificial model building has been regarded as a complex, time consuming and error prone task. To create an elaborate model that describes the flow of work properly, a modeler usually needs to spend a long time discussing with staff members and managers so as to master every single detail of the business process at hand. Meanwhile, the modeler should also have a deep knowledge about the utilised modeling language. Additionally, current enterprise business processes are often implemented in flexible environments to encourage process evolution for coping with the changes of exterior markets [10–14]. As a result, handmade process models often cater too much to the norms to describe the actual processes [15, 16]. The discrepancies between the manually modeled process behaviours and the realistic execution behaviours of processes will reduce the credibility of artificially designed models.

BPMD [8, 9, 17, 18] that is an important research branch of BPMI [19–22] provides companies with an automatic way to acquire the models of their business processes. During the last years, many outstanding process model discovery techniques have been developed in the literature, e.g., the α algorithm [23], ILP Miner [24], Heuristics Miner (HM), Inductive Miner [25–27], Fuzzy Miner [28] and Genetic Miner [29]. These techniques are able to extract the process models by employing the ordering relations of activities recorded in the so-called "event logs" without human interventions. Typically, there are several advantages for mining the business process models automatically: (i) the process model mining procedure is much more timesaving than the manual model building procedure, (ii) due to the leads utilised to construct the models are from the real implemen-

tation of enterprise business processes, the mined process models are not biased by the modelers' perceptions or canonical behaviours but can express the realistic process behaviours, (iii) the mined process models provide a basis for Delta analysis, i.e., detect discrepancies between the handmade process models and the realistic process models [9].

However, in the real world many business processes are often executed in highly flexible environments, e.g., healthcare, CRM and product development [30]. As a result, the current BPMD techniques might generate inaccurate and impalpable analysis results while dealing with event logs (real-life logs) stemming from such flexible environments. For instance, "spaghetti-like" business process models might be generated by existing BPMD algorithms with an input of real-life event log. Such models are often inaccurate and too complex to be well interpreted. The problem is largely due to the dense distribution of complex process behaviours in the real-life event logs. Accordingly, three main strategies have been proposed in the academia to solve this problem: the Mining Algorithm Enhancement-Based Strategy (MEBS), the Model Division-Based Strategy (MDS) [31,32] and the Model Abstraction-Based Strategy (MAS) [33,34].

Real-life event logs often contain complex process behaviours which might be far beyond the expressive ability of the utilised BPMD algorithms. If this is the case, inaccurate process models will be generated. The MEBS aims at improving the expressive ability of existing BPMD algorithms or developing new algorithms that are able to model more complex workflow patterns. The MDS strategy tries to divide the original event log into several sublogs where each sublog consists of similar process behaviours. Afterwards, by applying BPMD algorithms on each sublog, more accurate and comprehensible (simpler) sub-models can be obtained. The trace clustering technique [31,35,36] is a classical application instance of such strategy. The MAS makes the assumption that the raw models mined from event logs may contain low level subprocesses which can be found in the event logs and abstracted into high level activities so that the low level process behaviours can be hidden in the high level activities. Thus, more accurate and simpler high level process models can be obtained. In practical applications, we discovered that each of the three strategies has its own applicable situation determined by the structural features of the implemented processes and the process execution information recorded in event logs. For example, the MEBS is able to provide end users with a holistic view of their business processes but cannot deal with the problem of high complexity of the mined models. The MDS can solve both the problems of low accuracy and high complexity of the mined models by sacrificing the integrity of process models. Additionally, the limitation of MDS will be revealed while dealing with event logs containing massive trace behaviours (as proved in Chapter 6). The MAS is capable of dealing with extremely unstructured process behaviours [28,33]. However, the process models mined by MAS usually cannot be directly implemented [28]. Given a specific context, an appropriate strategy should be selected to solve the relevant problem.

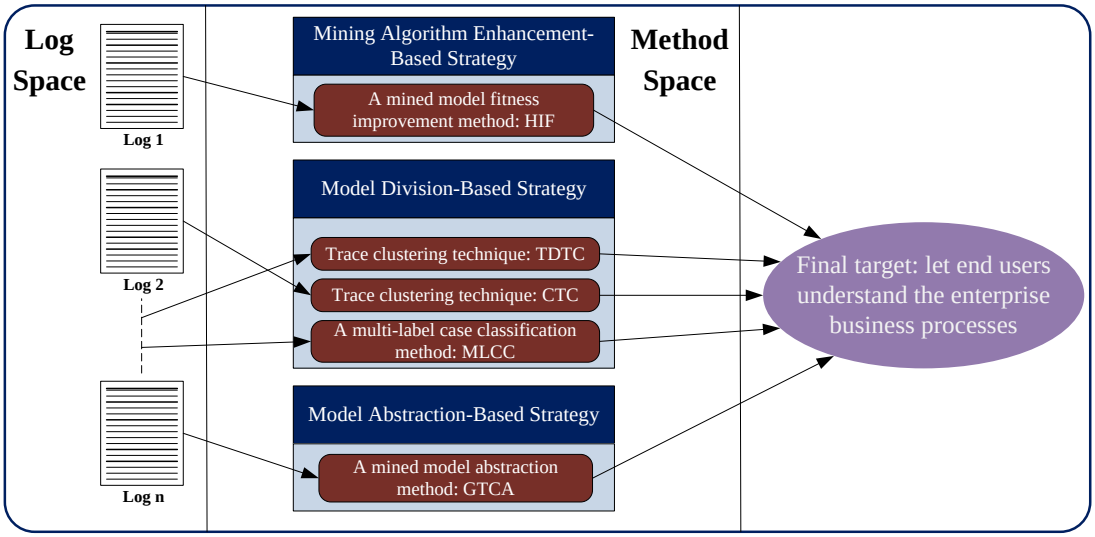


Figure 1.1.: Illustration of the basic idea for solving the problem of BPMD under flexible environment.

In this thesis, we address the challenge of mining enterprise business process models under flexible environments by providing a series of efficient techniques for MEBS, MDS and MAS. As shown in Figure 1.1, all the created techniques for the three strategies serve for one single ultimate target: let end users of our techniques acquire a better insight of their business processes. Each of the developed techniques has its own scope of application (will be introduced in detail in the following chapters) characterised by the features of the business processes recorded in the related event logs waiting to be dealt with.

1.1. Problems and Challenges

Currently, many classical methods for realising MEBS, MDS and MAS have been put forward in the literature. However, there still exist several problems about these methods which need to be resolved so as to improve the practicability of the three strategies. This section describes thorny problems and derives concrete challenges for realising the three strategies.

Mining Algorithm Enhancement-Based Strategy

BPMD technique is the basic tool for transforming the implicit process behaviours hidden in the event logs into abstract models. Many advanced BPMD algorithms [24, 25, 28, 29, 37–40] utilising different approaches and modeling languages have

been presented in the past. Nevertheless, each of these developed algorithms has its own feature reflected by its representational bias, i.e., the category of process behaviours that can be expressed by the algorithm [9]. Creating an approach to represent all kinds of workflow patterns therefore cannot be accomplished.

Problems

- Sometimes, the business processes executed under flexible environment may exhibit very complex behaviours which might be far beyond the expressive ability of the utilised BPMD algorithms. As a result, mining the event logs that record the behaviours of these business processes will generate low-fitness process models.

- Existing process model discovery algorithms don't have effective processing mechanisms to detect and deal with the process behaviours which they cannot express in the real-life event logs.

- Most techniques attached to the MEBS are developed to help mine more fitted models expressed by Petri net. However, few efforts have been made to help the HM get better mining results. According to [36], the HM is one of the most popular BPMD tools for mining real-life event logs and developing an auxiliary method to help it mine high-fitness process models is far from trivial.

Challenge 1. Discover and handle the inexpressible process behaviours recorded in event logs from flexible environment for HM so as to help HM generate better process models.

Model Division-Based Strategy

Event logs usually record all sorts of historical information about the execution of business processes. Such kind of historical information is often organised in the form of process instances. For each process instance, the relevant events together with the values of data fields (e.g., timestamp, cost and resource) linked to these events might also be recorded. The trace clustering technique that derives from the MDS is able to employ the process execution information recorded in an event log in dividing this log into several sub-logs where the similar process instances will be clustered into the same sub-log automatically. Then, more accurate and simpler sub-models can be obtained by using the BPMD algorithms to mine the sub-logs generated.

Problems

- Most currently available trace clustering algorithms focus mainly on the discovery of various kinds of process behaviours while the quality of the underlying sub-model for each sub-log learned is not taken into account. Hence, high quality sub-process models from these trace clustering methods cannot be assured.

- Trace clustering is an unsupervised learning technique and lack domain knowledge. As a result, it is unable to indicate which process behaviours found are crucial or which behaviours are wanted by customers for splitting the original set of process instances. Additionally, treating all of the process behaviours equally may not generate a correct or meaningful separation of process instances.

Challenge 2. Optimise the accuracy and simplicity of the potential sub-process model for each sub-log generated by clustering technique at runtime.

Challenge 3. Combine the domain knowledge from business experts for splitting the process instances in event logs.

Model Abstraction-Based Strategy

Sometimes, extremely complex and inaccurate process models might be output by existing process model discovery algorithms due to the huge amount of low-level activities recorded in the event logs. Furthermore, the business process models composed by low-level activities are often too fine grained to be well comprehended by the process analysts. There is a demand to merge these bottom activities into high-level activities so that the mined process models can be expressed at a suitable level of abstraction [41, 42]. Several methods based on the idea of MAS have been put forward in the literature. The proposed methods are able to discover meaningful sub-processes constructed by the bottom activities and abstract the found sub-processes into high-level activities in the event logs. Afterwards, more meaningful, accurate and simpler abstract models can be generated by executing the BPMD algorithms on the event logs with high-level activities.

Problems

- Most existing model abstraction-based approaches focus mainly on the discovery of sub-processes and cannot assure the quality of the abstract process

models generated.

- Occasionally, the sub-processes which might be interesting to the process analysts may also suffer from inaccuracy and high complexity. Most of the existing methods don't provide a mechanism to deal with the underlying low-quality sub-process models that might be generated.

Challenge 4. Develop a new mined model abstraction mechanism which considers the optimisation of the quality for both the potential abstract process models and the sub-process models discovered at runtime.

1.2. Objectives, Approach and Contributions

In the previous section, we introduced the problems and challenges encountered by the MEBS, the MDS and the MAS. This section identifies the objectives to solve the problems and challenges for the three strategies. We also depict the approaches we take to achieve these objectives and list the contributions of this thesis.

Mining Algorithm Enhancement-Based Strategy

The HM is well designed to deal with real-life event logs and has a good performance in most cases. Nevertheless, its weaknesses will be revealed while dealing with complex process behaviours that it cannot express. As described in *Challenge 1*, this problem confronted by HM should be solved by providing an inexpressible process behaviour handling mechanism.

Objective 1. Create an inexpressible process behaviour handling mechanism to help HM mine fitting process models from real-life event logs.

Approach. We develop a heuristic method named HIF to address this objective. The proposed method follows the divide-and-conquer approach which means that each time our method discovers and resolves the inexpressible process behaviours related to a single activity from the relevant event log. The whole problem is then solved after the found inexpressible process behaviours related to each activity is handled.

Contributions

- An effective method for extracting and organising the process behaviours from real-life event logs is put forward. This method provides a basis for the later inexpressible process behaviour detecting and handling process.

- A priori method is developed to locate the inexpressible process behaviours from the given event logs for HM.

- An inexpressible process behaviour handling mechanism is devised which is capable of transforming the found inexpressible process behaviours into expressible behaviours for HM.

- A case study is carried out in which our method successfully helps the HM mine high-fitness process models from five real-life event logs. A comparison between our method HIF and other classical BPMD techniques is also made through implementing the related methods on several example event logs.

Model Division-Based Strategy

An effective way to deal with an inaccurate and complex process model mined from real-life event log is to divide this model into separated sub-models where each sub-model is more accurate and simpler. Trace clustering technique is devised to implement this job. According to *Challenge 2* it has to be possible for the trace clustering technique to assure the quality of the generated sub-models. Sometimes, the business process experts know the best way to split a complex and inaccurate process model mined according to their domain knowledge. If this is the case, it should be possible to combine the domain knowledge from these experts for the division of the original models mined (see *Challenge 3*).

Objective 2. Develop new type of trace clustering methods that are able to optimise the accuracy and simplicity of the underlying sub-process models during runtime.

Approach. A novel top-down algorithm named TDTC and a compound trace clustering algorithm named CTC for clustering traces are designed to achieve *Objective 2*. The presented approach TDTC first divides the raw model mined into two sub-models which have the highest average quality (expressed by model accuracy and complexity) among all the possible pair of sub-models that can be obtained by dividing the raw model. Then, it continues to deal with the generated sub-models recursively until a stopping criterion is satisfied. Unlike TDTC, the

technique CTC tries to optimise the accuracy and complexity of the potential sub-process models separately by combining the basic idea from TDTC and the mined model fitness improvement technique HIF mentioned above.

Contributions

- The trace clustering problem is surveyed from a new perspective and redefined as an issue of searching for a global optimal solution for splitting the original model in a solution space.
- We formalise the definitions related to trace behaviours. Then, several different kinds of trace behaviours are defined for helping cluster the traces from real-life event logs.
- A top-down algorithm TDTC and a compound algorithm CTC based on the different kinds of trace behaviours defined by us are put forward which identify the optimal solution for a given trace clustering problem.
- A case study is carried out in which our methods are compared with six classical trace clustering methods. The comparison results show that our methods perform better than the other methods on the given real-life event logs.

Objective 3. Develop a new application instance of the MDS which is able to employ the domain knowledge from process experts to split the raw process models.

Approach. Our method named MLCC developed for achieving *Objective 3* is based on classification technique. Not like clustering technique, classification is a supervised technique that can make use of the domain knowledge from business experts seamlessly for the mined model division problem.

Contributions

- We demonstrate and formalise the problem of multi-label case classification.
- A systematic method MLCC based on sequential pattern mining technique [43] is put forward for utilising the case attribute trace for classifying cases.
- A case study executed on a hospital event log proves the effectiveness of our technique.

Model Abstraction-Based Strategy

The target of MAS is to generate a high quality abstract process model by hiding the low level sub-models in the original model mined from the relevant event log. Sometimes, the process experts may also take interests in the behaviour of the sub-models found. According to *Challenge 4*, the method that inherits the idea of the MAS should assure the quality of the abstract model and sub-models generated at the same time.

Objective 4. Develop a new model abstraction-based technique which optimises the accuracy and simplicity for both the abstract model and the sub-models during the runtime.

Approach. To realise *Objective 4*, an approach named GTCA is proposed by us which merges the characters from both the graph clustering technique [44] and the trace clustering technique. The graph clustering technique assists in generating a high quality abstract model while the trace clustering technique guarantees the qualities of the discovered low level sub-models.

Contributions

- A new strategy based on graph clustering technique and trace clustering technique for abstracting the raw models mined from real-life event logs is created.
- A three-stage model abstraction approach GTCA based on the proposed strategy is developed for mining high quality abstract models together with their related low level sub-models.
- A case study implemented on three real-life event logs testifies the efficiency of our method.

1.3. Publications

Parts of this thesis have been previously published in the following peer-reviewed publications:

1. Yaguang Sun and Bernhard Bauer. "A Novel Fitness Improvement Method for Mined Business Process Models". In *Proceedings of the CAiSE'16 Forum at*

the 28th International Conference on Advanced Information Systems Engineering, Ljubljana, Slovenia.

2. Yaguang Sun and Bernhard Bauer. "A Novel Heuristic Method for Improving the Fitness of Mined Business Process Models". In: *Sheng Q., Stroulia E., Tata S., Bhiri S., (eds) Service-Oriented Computing* (ICSOC 2016), volume 9936 of Lecture Notes in Computer Science, pages 537-546.

   The contents of Chapter 3 are based on the first two papers. In the first paper we proposed the basic principle for locating and transforming the inexpressible process behaviours for the utilised BPMD techniques in real-life event logs. In the second paper we put forward the technique HIF for helping HM mine process models with high fitness based on the principle proposed in the first paper.

3. Yaguang Sun and Bernhard Bauer. "A Novel Top-Down Approach for Clustering Traces". In: *Zdravkovic J., Kirikova M., Johannesson P. (eds) Advanced Information Systems Engineering* (CAiSE 2015), volume 9097 of Lecture Notes in Computer Science, pages 331-345.

4. Yaguang Sun, Bernhard Bauer and Matthias Weidlich. "Compound Trace Clustering to Generate Accurate and Simple Sub-Process Models". Accepted for publications by the 15th International Conference on Service-Oriented Computing (ICSOC 2017)

5. Yaguang Sun and Bernhard Bauer. "Function-Based Case Classification for Improving Business Process Mining". In *Proceedings of the 17th International Conference on Enterprise Information Systems* (ICEIS 2015), pages 251-258.

   The contents of Chapter 4 are based on the third, the fourth and the fifth paper. In the third paper, the trace clustering technique TDTC is proposed based on which we then developed another kind of trace clustering technique CTC that is published in the fourth paper. In the fifth paper, we presented the multi-label case classification technique MLCC.

6. Yaguang Sun and Bernhard Bauer. "A Graph and Trace Clustering-Based Approach for Abstracting Mined Business Process Models". In *Proceedings of the 18th International Conference on Enterprise Information Systems* (ICEIS 2016), pages 63-74.

   The contents of Chapter 5 are based on the sixth paper in which the details about the mined model abstraction technique GTCA is elaborated.

All the papers listed above are mainly written by me and commented by the co-authors with whom the basic ideas of these papers are deeply discussed.

2

Basics

Business process mining research field is mainly built on two cornerstones: process modeling (analysis) [45–48] and data mining [49–51]. Even though pure process mining approaches are rarely directly applied to process model discovery or conformance checking (the two main topics of process mining which will be introduced later in this chapter), they play an important role of helpers and resource of ideas for various kinds of business process mining tasks. For example, clustering technique [52–54] from data mining domain can be utilised to help existing process model discovery techniques mine more accurate and comprehensible process models and classification technique [55–58] can assist in analysing the routing rules (decision mining [59, 60]) in the mined process models. In this chapter, the basic knowledge relevant to process mining techniques presented in this thesis is elaborated. In the meantime, the similar methods put forward in the literature which solve the same problems as our techniques are designed to tackle are also retrospected detailedly. In Section 2.1, the correlative data mining approaches (i.e., classification, clustering and sequential pattern mining [61, 62] techniques) are briefly reviewed. In Section 2.2, the elementary concepts and approaches related to process mining are introduced.

2.1. Data Mining Techniques

To bridge the gap between data and information (knowledge), advanced data mining methods have been widely researched in the last a few years which are able to turn the huge amount of data collected and stored in large data repositories into valuable knowledge so as to help the decision makers make important decisions [63]. Generally, the preparing steps such like data cleaning, data integration, data selection and data transformation should be implemented before a particular data mining task [64]. After the data mining step, the knowledge mined from the data set should also be evaluated on its interestingness [65] and then visualised and exhibited to the policy makers. In this section, we mainly concentrate on the classification (Section 2.1.1), clustering (Section 2.1.2) and sequen-

tial pattern mining (Section 2.1.3) techniques from data mining domain because these techniques are most correlative to the process mining approaches proposed in this thesis.

2.1.1. Classification Technique

Classification is a kind of supervised technology (such kind of technology needs domain knowledge to be given in advance) which tries to build models by employing training data set for the later predication of data categories (classes) [57, 66]. Such models are also referred to as classifiers. Table 2.1 displays the standard format of the training data. It can be seen that a training data set contains several data items where each item is a tuple represented by n attributes. Additionally, every item in the training data set has a class label attribute (discrete-valued and unordered) to which a value is manually assigned.

Table 2.1.: Standard training and testing data format for classification technique.

Item	**Attribute 1**	**Attribute 2**	**...**	**Attribute n**	**Label**
1	XX	XXX	...	XXXX	YES
2	XX	XXX	...	XXXX	NO
3	XX	XXX	...	XXXX	NO
4	XX	XXX	...	XXXX	YES
5	XX	XXX	...	XXXX	YES
...	...	...	...	...	...

A data classification process normally consists of two steps: the classifier learning step and the data-category predicting step. The classifier learning step can be regarded as a step of creating a mapping or function, say $y = f(X)$, where y represents the category of a given data item X [50]. The created function can be expressed by classification rules [67, 68], Decision Tree (DT) [69, 70] or other related mathematical formulas. In the second step (i.e., the category predicating step), the accuracy of the classifiers [71–74] generated in the first step are estimated first by making use of the testing data sets (the testing data and the training data have the same formats). If the accuracy of the classifiers is deemed acceptable, they can be utilised to make predication of the classes of the future data items of which the labels are unknown. In the following part of this subsection, we will only introduce the DT classification approach because such an approach is one of the most classical classification approaches and also a benchmark to which the newly designed classification methods are often compared. The readers who are interested in the other classification techniques such like AdaBoost, Naive Bayesian (NB) and SVM can refer to [75–84].

DT classification approach is well known by its high accuracy and intelligibility. There exist several kinds of DT-based classification algorithms such as

ID3 [85], C4.5 [86, 87] and CART [85]. The primary difference between these algorithms is that they use different ways to calculate the attribute selection measure (ASM) [50]. Algorithm 2.1 [50] shows the basic idea of DT classification approach.

Algorithm 2.1 Decision Tree approach (DT)

Input: a training data item set TD, the set of attributes A for the data item in TD, an attribute selection function $\widetilde{S}$ for choosing the best attribute to partition the present data items into classes.

1: create a node N
2: if all the data items in TD belong to the same class c, then
3: return N and label N with class c
4: if A is empty, then
5: return N and label N by the class with maximal number
6: run $\widetilde{S}(TD, A)$ to find the best splitting criterion scr related to attribute $a \in A$
7: label N with the name of attribute a
8: if scr is discrete-valued, then
9: remove a from A
10: for each condition $scr[v]$ in scr
11: let $TD_v \in TD$ be the set of all items satisfying $scr[v]$
12: if TD_v is empty, then
13: attach a leaf to N and label it by the class with maximal number in TD
14: else attach the node returned by $DT(TD_v, A, \widetilde{S})$ to N

Output: a decision tree N.

The steps 1–5 of Algorithm 2.1 are straightforward. Step 6 tries to find the present best splitting criterion from the attribute set A so as to make each partition TD_v for each branch as pure as possible (the attribute selection function $\widetilde{S}$ will be introduced later). If scr is discrete-valued, then remove attribute a from A which means that an attribute will only be considered once for each partition task (step 8–9). Afterwards, the Algorithm 2.1 recursively processes each generated partition TD_v and attach the returned node to the root node N.

To explain the attribute selection function $\widetilde{S}$, we first introduce an attribute selection metric named Information Gain (IG). Let TD be a training data set, A be the attribute list for TD, $a \in A$ be an attribute, C be the set of labels for TD, p_i be the probability of a data item from TD that belongs to the class $c_i \in C$, IG is defined by the following formulas [50]:

$$Gain(a) = Info(TD) - Info_a(TD) \tag{2.1}$$

$$Info(TD) = -\sum_{i=1}^{|C|} p_i log_2 p_i \tag{2.2}$$

$$Info_a(TD) = \sum_{v=1}^{n} \frac{|TD_v|}{|TD|} \times Info(TD_v) \tag{2.3}$$

The $Info(TD)$ that appears in Equation 2.2 stands for the expected amount of information required to categorise the data item in TD (it should be noticed that $Info(TD)$ also represents the *entropy* [88] of TD). Equation 2.3 (assume that attribute a has n values) calculates the expected information needed to classify the data items in TD based on the n subsets generated by partitioning TD according to the n values of a. For each subset TD_v, the calculated $Info(TD_v)$ is also weighted by $\frac{|TD_v|}{|TD|}$. The smaller the value of $Info_a(TD)$ is, the purer the subsets generated by splitting TD according to attribute a are. The IG of attribute a $Gain(a)$ indicates the reduction of the amount of information for classifying the data items in TD gained by partitioning TD utilising attribute a. The attribute selection function $\widetilde{S}$ always detects and chooses an attribute $a \in A$ with the largest IG and this is the basic principle of how $\widetilde{S}$ works. The algorithm ID3 uses IG for $\widetilde{S}$. However, C4.5 utilises Gain Ration (GR) [86] and CART utilises Gini Index (GI) [85] for the attribute selection function $\widetilde{S}$ (GR and GI can be seen as two improved versions of IG).

Now, let's make use of an example training data set as shown in Table 2.2 to illustrate the mechanism of DT approach. Each item in Table 2.2 represents the record for a certain person and the attribute of an item includes the age, income and the credit level of the person. If a person buys an apartment his (or her) related data item in Table 2.2 is marked by 'yes', or the data item is labelled by 'no'. The IG of each attribute from Table 2.2 is first calculated by $\widetilde{S}$, where $Gain(Age)$ = 0.2272, $Gain(Income)$ = 0.3127[1] and $Gain(CreditLevel)$ = 0.0132. Figure 2.1 shows the incomplete DT and sub-training data set generated by utilising the attribute $Income$ as the best attribute.

[1]According to Table 2.2, $Info(TD) = -0.5 \times log_2 0.5 - 0.5 \times log_2 0.5$=1. The attribute $Income$ is chosen as the best attribute by $\widetilde{S}$ because it leads to the highest IG. Furthermore, three selection criteria are found by $\widetilde{S}$, the first one is $Income = low$, the second one is $Income = high$ and the third one is $Income = medium$. For the first criterion, $Info(TD_1) = -\frac{1}{7} \times log_2 \frac{1}{7} - \frac{6}{7} \times log_2 \frac{6}{7}$ = 0.5917. Likewise, $Info(TD_2) = -\frac{4}{5} \times log_2 \frac{4}{5} - \frac{1}{5} \times log_2 \frac{1}{5}$ = 0.7219 and $Info(TD_3) = -\frac{3}{4} \times log_2 \frac{3}{4} - \frac{1}{4} \times log_2 \frac{1}{4}$ = 0.8113. As a result, $Info_{Income} = \frac{7}{16} \times Info(TD_1) + \frac{5}{16} \times Info(TD_2) + \frac{4}{16} \times Info(TD_3)$ = 0.6873.

Table 2.2.: An example training data set.

Item ID	Age	Income	Credit Level	Class: buy apartment
1	$age < 25$	low	fair	no
2	$25 \leq age < 30$	high	excellent	no
3	$age < 25$	high	fair	yes
4	$age \geq 30$	low	fair	no
5	$age \geq 30$	medium	fair	yes
6	$25 \leq age < 30$	medium	excellent	no
7	$25 \leq age < 30$	low	fair	no
8	$age < 25$	low	fair	no
9	$age < 25$	low	fair	no
10	$age \geq 30$	medium	excellent	yes
11	$age \geq 30$	high	excellent	yes
12	$age < 25$	low	fair	no
13	$25 \leq age < 30$	low	fair	yes
14	$age \geq 30$	high	excellent	yes
15	$age \geq 30$	medium	fair	yes
16	$25 \leq age < 30$	high	fair	yes

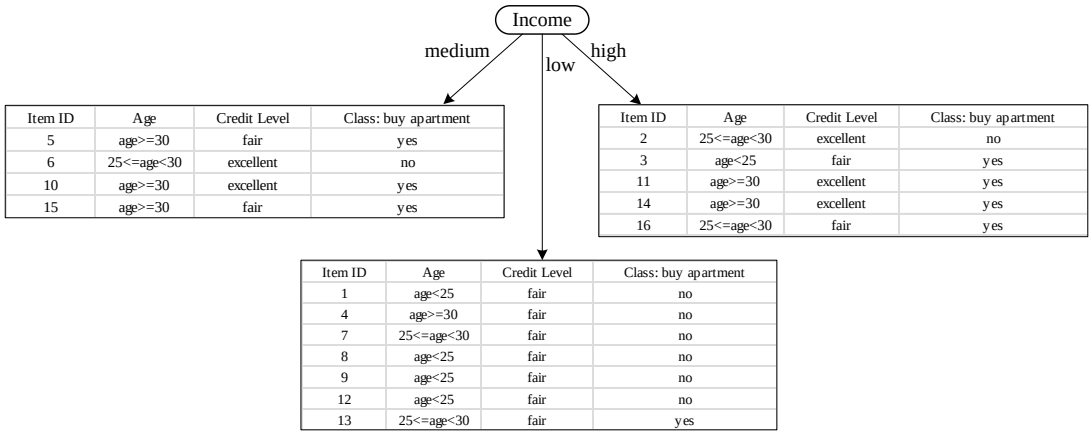


Figure 2.1.: The incomplete decision tree and sub-training data sets generated after processing the first selected attribute: Income.

2.1.2. Clustering Technique

As introduced in Section 2.1.1, classification is a supervised technique which needs domain knowledge for building the classification rules [89]. In this subsection, we introduce the unsupervised classification technique, called clustering [90–92] which aims at dividing a group of objects into a number of subgroups, where the similarity between objects within the same subgroup is kept as large as possible and the dissimilarity between objects from different subgroups should be kept as large as possible. In the literature, traditional clustering technique is usually classified into four categories [50]: partitioning approach [93], hier-

archical approach [94–97], density-based approach [98–102] and grid-based approach [103]. Due to the topic of this thesis is mainly about BPMI, thereby only the idea about the partitioning (Section 2.1.2.1) and hierarchical (Section 2.1.2.2) clustering approach will be introduced here. On the one hand, the partitioning approach can give the reader a straightforward view about what clustering technique is about. On the other hand, the trace clustering methods TDTC and CTC proposed in Chapter 4 are based on the idea of hierarchical clustering approach.

In addition, a special branch in the clustering research area, the graph clustering [44] technique is also reviewed in Section 2.1.2.3 because the mined model abstraction method GTCA put forward in Chapter 5 is built on such technique. Nevertheless, it should be noticed that the graph clustering technique relevant to this thesis is the technique for grouping the vertices in a given graph into clusters which should not be confused with the traditional clustering technique of classifying a set of graphs based on the structural features of these graphs into different subsets.

2.1.2.1. Partitioning Clustering Approach

Partitioning clustering approach is able to divide the original set of objects into a fixed number of clusters in a flat manner (which means that all the generated clusters are at the same level) [104–110]. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of objects waiting to be clustered, where each object $x_i = (x_{i1}, x_{i2}, \ldots, x_{im})$ from X is represented by a m-dimension vector of attributes. As described in [50], the (hard) partitioning clustering approach searches for a l-partition of X, represented by $C = \{c_1, c_2, \ldots, c_l\}$, where the following three conditions should be met: (1) $\forall c_k \in C$ such that $c_k \neq \varnothing$ (2) $\bigcup_{i=1}^{l} c_i = X$ (3) $1 \leq p < q \leq l$ such that $c_p \bigcap c_q = \varnothing$. As indicated in the third condition, most partitioning clustering approach-based algorithms adopt an exclusive separation strategy [90] which means that each object only pertains to a single cluster. However, there exists another object division strategy with which one object is allowed to be assigned to more than one clusters and the fuzzy clustering techniques [111] utilise such a strategy. One of the best-known partitioning approach-based clustering techniques is K-means [104–107] algorithm which will be described in the rest of this subsection for letting the readers who are at a introductory level get acquainted with clustering technique.

K-means is a very simple clustering technique and can be easily implemented to handle a number of practical issues [93]. It is based on an iterative optimisation strategy that starts with k-initial centroid [50] (a centroid is a central point for a cluster). Algorithm 2.2 [50] exhibits the main steps of K-means.

As depicted in Algorithm 2.2, the K-means algorithm requires a pre-specified

Algorithm 2.2 K-means

Input: the number k of clusters, a set X of data objects.

1: randomly choose k objects from X as initial center for the k clusters
2: **repeat**
3: assign each object $x_i \in X$ to a particular cluster $c_j \in C$
4: update the center for each cluster
5: **until** all the clusters become stable

Output: k clusters of data objects.

number k for clusters as input. Then, k objects are randomly chosen from the original set X and each of them is assigned to a particular cluster as center object (step 1). Afterwards, K-means algorithm finds each object $x_i \in X$ a unique cluster $c_j \in C$ (step 3). How is this cluster finding process implemented? The K-means algorithm first calculate the distance d_{pq} between the present object $x_p \in X$ and each cluster $c_q \in C$. Next, x_p is assigned to the cluster c_m which has the shortest distance to x_p .The K-means algorithm employs the Euclidean Distance (ED) [112] between the object x_p and the center object co_q of cluster c_q to stand for the distance between x_p and c_q. Let $x_p = (x_{p1}, x_{p2}, \ldots, x_{pm})$, $co_q = (co_{q1}, co_{q2}, \ldots, co_{qm})$ (both x_p and co_q are characterised by m attributes), the ED between x_p and co_q is defined as:

$$ED(x_p, co_q) = \sqrt{\sum_{t=1}^{m}(x_{pt} - co_{qt})^2} \tag{2.4}$$

After assigning all the objects from X to a cluster, K-means algorithm then updates the attribute values of the center object for each of the k clusters, where the values of all the attributes of the center object are replaced by the related mean values. For instance, after the step 3 of Algorithm 2.2, the cluster $c_j \in C$ contains r objects $\{x_1^{c_j}, x_2^{c_j}, \ldots, x_r^{c_j}\}$, in step 4 the value of the first attribute co_{j1} of the center object co_j of cluster c_j is replaced by $\frac{1}{r}\sum_{v=1}^{r} x_{v1}^{c_j}$. Finally, K-means algorithm will check if the assignment becomes stable[2]. If it is so, the iteration stops, otherwise steps 3 and 4 repeat.

[2]This is checked by comparing the present distribution of the objects in the clusters with the distribution of the previous round. The assignment of objects is judged stable by the algorithm if the present distribution is consistent with the former distribution

2.1.2.2. Hierarchical Clustering Approach

In this subsection, we simply introduce the basic thought of hierarchical clustering approach (because the trace clustering techniques proposed in Chapter 4 are built on such thought). For the specific techniques, the interested reader could refer to [113–117]. The hierarchical clustering approach tries to build a cluster hierarchy that is also referred to as cluster tree, where each node of the tree stands for a cluster of data objects. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of data objects waiting to be clustered, $C = \{c_1, c_2, \ldots, c_m\}$ be the set of clusters output by implementing a hierarchical clustering algorithm on X where $m \leq n$. $\forall c_i, c_j \in C$, $i < j \Rightarrow c_i \sqsubset c_j \parallel c_i \cap c_j = \varnothing$.

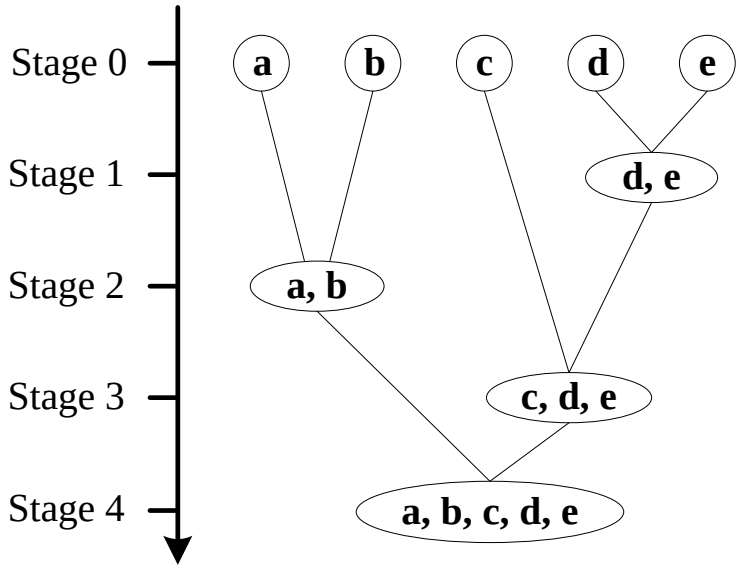


Figure 2.2.: Illustration of the agglomerative clustering method.

There are two variants of hierarchical clustering approach: agglomerative method (bottom-up) [113–115] and divisive method (top-down) [116, 117]. As shown in Figure 2.2, the agglomerative method adopts a bottom-up strategy. At stage 0, each data object forms a single cluster. Then, for each following stage two clusters are merged into a larger cluster (according to the similarity measure [93]) until all the objects are in the same cluster or a predefined stopping condition is met. The divisive method that is described in Figure 2.3 uses the top-down strategy. All the data objects are first put in the same cluster (root cluster). Afterwards, the root cluster is divided into several smaller sub-clusters based on some splitting rule [97]. The division process is recursively executed on each generated sub-cluster until the sub-cluster contains only one object or the objects within the cluster are sufficiently alike with each other. The two trace clustering techniques TDTC and CTC put forward in Chapter 4 employ the divisive (top-down) method for clustering the traces from an event log.

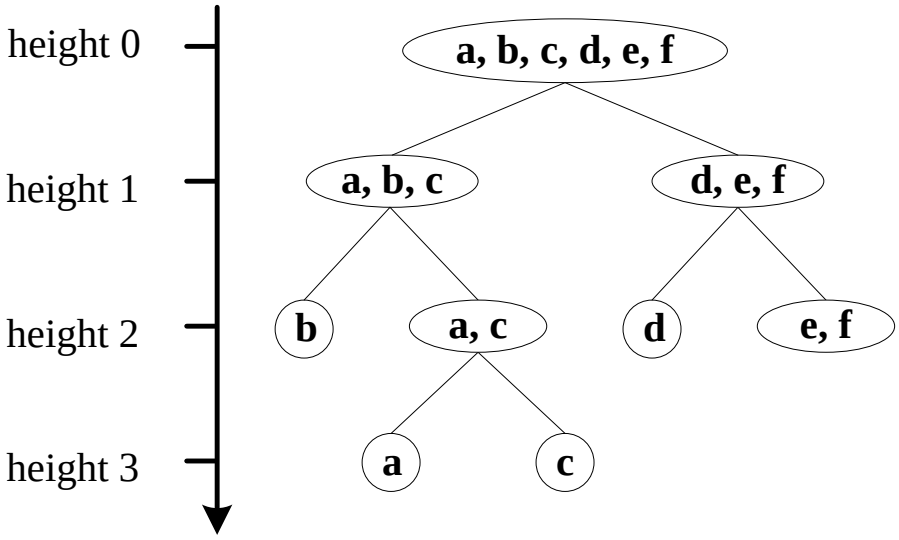


Figure 2.3.: Illustration of the divisive clustering method.

2.1.2.3. Graph Clustering

Before introducing the basic idea of graph clustering, the elementary notions about graph will be first reviewed. A graph $G = (V, E)$ is composed by two kinds of elements: vertice and edge [118, 119]. The set V stands for the set of vertices and E stands for the set of edges, where each edge from E is a pair of vertices (v_i, v_j) ($v_i, v_j \in V$ are also called the endpoints of the edge). $|V|$ represents the total number of vertices and $|E|$ represents the total number of edges in G. A graph $G = (V, E)$ can be categorised into directed graph and undirected graph [118]. If G is a directed graph then each of its edges (v_i, v_j) is an ordered pair which means that the edge starts from v_i and ends at v_j. If G is an undirected graph then each edge in G is an unordered pair of vertices [120]. Graph $G = (V, E)$ is a weighted graph if there exists a weight function $\widehat{W} : E \to \mathbb{R}$ that assigns a weight to each edge of G [119]. The density of a directed graph $G = (V, E)$ is defined as: $\mathcal{D}(G) = \frac{|E|}{|V| \times |V-1|}$, where $\mathcal{D}(G) = 0$ if $|V| \in \{0, 1\}$. For a node $v_i \in V$, the degree of v_i is the number (i.e., $|\{e \in E \mid v_i \in e\}|$) of edges incident to v_i and denoted as $deg(v_i)$. Figure 2.4 shows an example directed graph $G_1 = (V_1, E_1)$ which has 6 vertices (i.e., $|V_1| = 6$) and 8 edges (i.e., $|E_1| = 8$). The density of this graph is $\mathcal{D}(G_1) = \frac{|8|}{|6| \times |6-1|} = 0.2667$. Each edge in graph G_1 is assigned a weight, for instance, $\widehat{W}(v_3, v_2) = 3$. The degree of vertice v_3 is equal to 2 (i.e., $deg(v_3) = 2$) because there are two edges (v_4, v_3) and (v_3, v_4) in graph G_1 that have v_3 as endpoint. Likewise, the degree of vertice v_6 is 3 (i.e., $deg(v_6) = 3$).

Let $V_1, V_2 \in V$ be two subsets of vertices from graph $G = (V, E)$. If $V_1 \cap V_2 = \varnothing$ and $V_1 \cup V_2 = V$, the tuple (V_1, V_2) is called a cut [121] of graph G and denoted as $cut(V_1, V_2)$. The cut size is the total number of edges that connect vertices from

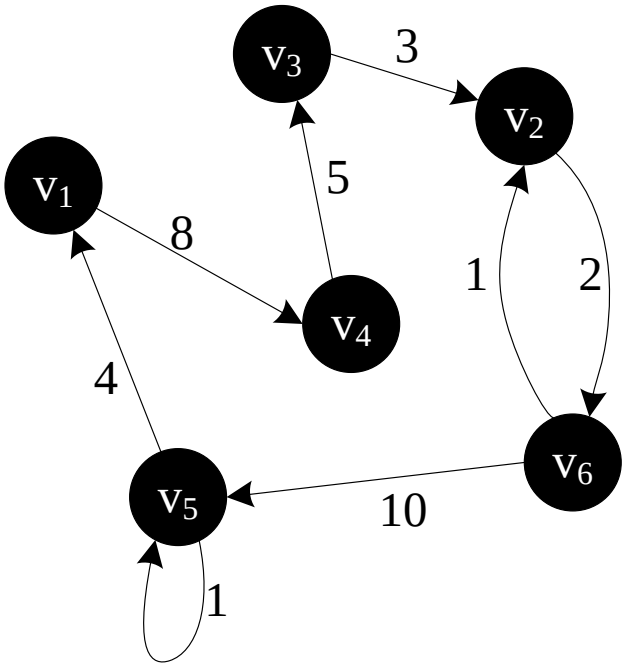


Figure 2.4.: An example graph G_1 utilised for explaining the fundamental concepts about graph.

V_1 to vertices from V_2. For the example graph G_1 (see Figure 2.4), it has a cut $cut(V_1 = \{v_1, v_4, v_5, v_6\}, V_2 = \{v_2, v_3\})$ and the size of this cut is 3 because there are three edges which are edge (v_4, v_3), (v_6, v_2) and (v_2, v_6) that connect subset V_1 and V_2.

Given a graph $G = (V, E)$, the task of graph clustering technique is to split the vertices from V into different groups so that the vertices with close relations are put in the same group. In the literature, there are mainly two approaches for calculating the degree of intimacy of graph vertices [44]. The first one computes the similarities between graph vertices and a lot of effective similarity metrics such as the adjacency-based metrics [122] and the connectivity-based metrics [123,124] have been proposed in the last a few years. The second approach is based on the cluster fitness measures [125] which then can be classified into two variants: density-based metrics [126–128] and cut-based metrics [129–132]. In this subsection, we only concentrate on introducing the cut-based metrics because the graph clustering algorithm (i.e., the LinLog algorithm [133–135]) that is employed by the mined model abstraction technique GTCA proposed in Chapter 5 is built on such kind of metrics.

The graph clustering algorithms that use cut-based metrics as clustering criteria tend to partition the vertices of a certain graph into clusters in the way that the number of edges within each cluster should be kept as many as possible while the number of edges between clusters should be kept as few as possible [44]. For

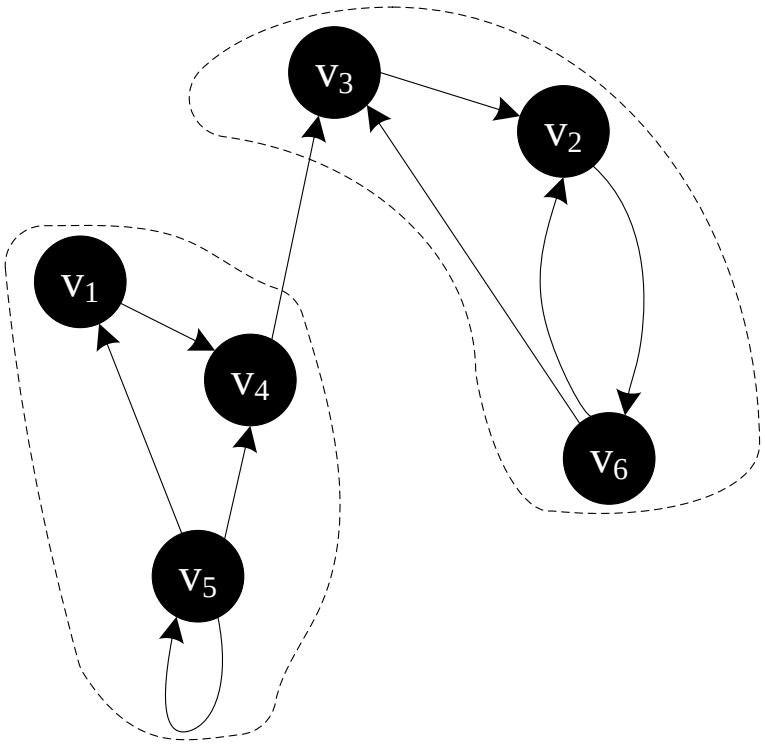


Figure 2.5.: An example graph G_2 utilised for explaining the basic idea of graph clustering methods that utilise cut-based clustering criteria.

instance, as shown in Figure 2.5, the vertices of graph G_2 (presume that the value of weight of all the edges in G_2 is equal to 1) can be divided into two clusters according to the minimum cut criterion[3] [133], where the first cluster contains vertices v_1, v_4 and v_5 and the second cluster contains vertices v_2, v_3 and v_6. However, the minimum cut criterion has its own weakness because it often generates two clusters of vertices where one cluster contains many vertices and the other one contains few vertices [134]. To overcome the limitation of the minimum cut criterion, many advanced cut-based criteria have been proposed in the literature, for example, the node-normalised cut [136], the edge-normalised cut [137], Shi and Malik's normalised cut [137], the expansion and conductance [129] and the Newman's modularity [138] (which is a measure of partition for k disjoint subsets of vertices).

The LinLog Graph Clustering (LGC) algorithm put forward in [133] will be utilised for clustering the vertices of graphs in the mined model abstraction approach GTCA (proposed in Chapter 5). This algorithm builds on two energy models named node-repulsion LinLog and edge-repulsion LinLog which are able to help group the vertices in a graph according to the clustering criterion: Newman's modularity. The reason for using LGC is that it not only offers an unbiased

[3]The minimum cut criterion can help find a bipartition of the vertices of a given graph with the minimum cut size.

mechanism for clustering graph vertices with high degrees, but can automatically generate a suitable number of clusters of graph vertices (deciding the number of clusters is always a difficult but crucial issue in a normal clustering task). The interested reader can refer to [133, 134] for learning more details about LGC.

2.1.3. Sequential Pattern Mining Technique

Sequential Pattern Mining [43, 139, 140] techniques solve the problem of finding all frequent subsequences from a given set of sequences, where each sequence contains a list of ordered itemsets. A minimum support threshold is manually given for judging if the occurrence of a subsequence is frequent or not [50].

Let $I = \{I_1, I_2, \ldots, I_n\}$ be a set of data items, $S = \{s_1, s_2, \ldots, s_m\}$ be the set of sequences over I. Each sequence $s_k =< is_{k1}, is_{k2}, \ldots, is_{kq} >$ from S is comprised by a list of ordered itemsets (every itemset happens earlier than the other itemsets that appear on its right side in the same sequence), where each itemset $is_{kp} \in s_k$ contains a set of items from I and denoted as $(I_{kp_1}, I_{kp_2}, \ldots, I_{kp_l})$. However, it should be noticed that there exists no specific order between items in an itemset. Table 2.3 shows an example sequence database S_1 that consists of six items (i.e., item a, b, c, d, e and f) and five sequences. For instance, the *sequence* 1 contains five itemsets which are (a), (b), (b), (a, c) and (d, f).

Table 2.3.: An example sequence database S_1.

Sequence ID	Sequence
1	$<a, b, b, (a, c), (d, f)>$
2	$<(a, c, e), b, (a, f), d, f, f>$
3	$<(a, e), b, a, (e, f), (c, d)>$
4	$<(e, f), c, (a, e), d, f, (a, d, f)>$
5	$<a, e, f, d, (c, f)>$

Let $s_1 =< x_1, x_2, \ldots, x_i >$ and $s_2 =< y_1, y_2, \ldots, y_j >$ be two sequences as defined above. Sequence s_1 is a subsequence of s_2 denoted as $s_1 \sqsubseteq s_2$ if $1 \leq u_1 < u_2 < \cdots < u_i \leq j$ such that $x_1 \subseteq y_{u_1}, x_2 \subseteq y_{u_2}, \ldots, x_i \subseteq y_{u_i}$. For example, the sequence $<a, b, a, (d, f)>$ is a subsequence of *sequence* 1 from S_1 (Table 2.3). Given a set of sequences S and a minimum support min_sup $(0 < min\_sup < 1)$, a sequence λ is called a sequential pattern if $support(\lambda) \geq min\_sup \cdot |S|$, where $support(\lambda)$ is the number of sequences in S which contain λ as a subsequence and $|S|$ represents the total number of sequences in S. The set of sequential patterns, SP, contains all of the subsequences from S whose support values are no less than $min\_sup \cdot |S|$. For instance, set $min\_sup = 0.4$, the subsequence $\eta =< (a, e), b, f >$ is a sequential pattern for the set of sequences S_1 (Table 2.3) because $support(\eta) = min\_sup \times 5$ (η appears as a subsequence for *sequence* 2 and *sequence* 3 in S_1). The set of Closed

Sequential Pattern (CSP) is defined as $SCSP = \{\alpha | \alpha \in SP \, and \, \nexists \beta \in SP \, such \, that \, \alpha \sqsubseteq \beta \, and \, support(\alpha) = support(\beta)\}$. For example, sequence η is a sequential pattern but not a closed sequential pattern for S_1 because there exists another sequential pattern $\omega =< (a,e), b, f, d >$ for S_1 where $\eta \sqsubseteq \omega$ (ω is a CSP of S_1). CSP effectively decreases the total number of sequential patterns generated but in the meantime preserves the complete information about all the sequential patterns. Given a set of sequences S and a minimum threshold min_sup, the sequential pattern mining techniques are able to locate the complete set of sequential patterns for S while the closed sequential pattern mining techniques can discover the complete set of CSP for S.

In the literature, sequential pattern mining techniques are typically classified into three types, the apriori-based mining approach [141–145], the pattern-growth-based approach [146, 147] and the early-pruning approach [148]. In this thesis the closed sequential pattern mining algorithm CM-ClaSP [148] that belongs to the early-pruning approach is utilised because it has a good performance (low memory consumption) for the candidate pruning process (fewer candidates will be generated by CM-ClaSP compared to other similar approaches).

2.2. Business Process Mining Techniques

Business process mining techniques aim at discovering, monitoring and improving real processes by extracting relevant knowledge from event logs recorded by enterprise information systems [9]. In general, current process mining techniques mainly consider three perspectives: workflow discovery [149], conformance checking [150–156] and process extension [8]. The starting point of these analyses is usually an event log. Therefore, we first give a formal definition about event log in Section 2.2.1. Due to the techniques proposed in this thesis are devised to help mine accurate and simple process models from real-life event logs, several classical process model discovery algorithms (i.e., the α algorithm, Heuristics Miner [37], Inductive Miner [25–27] and ILP Miner [24]) are then reviewed in Section 2.2.2. The conformance checking techniques can be utilised for evaluating the accuracy of mined process models, so in Section 2.2.2 the basic knowledge about conformance checking is surveyed. Accuracy only reflects one aspect of the quality of the mined process models and the other aspect is depicted by the process model complexity metrics which is also introduced in Section 2.2.2. In Section 2.2.3, the ideas of existing assistant methods for helping generate high quality process models from real-life event logs are elaborated.

2.2.1. Event Log

Historical process execution related data can be logged by the information systems that implement the relevant business processes. Before carrying out a process mining task, these historical data should be extracted from the systems and organised into suitable format so that the utilised process mining techniques can be easily executed on these data. A file that contains appropriately structured process execution related data is called event log [20, 23] which forms the input of process mining techniques. Let A be a set of activities, an event log L over A is defined by the following concepts [9]:

Definition 2.1. *(Event)*
An event $e = (N_e, V_e, \mathcal{A})$ is an instance of a particular activity from A, where $N_e = \{id, n_2, \ldots, n_i\}$ is the set of attribute names for event e, V_e is the set of values of the attributes for event e, function $\mathcal{A} : N_e \rightarrow V_e$ maps an attribute name to its value. Each event e has at least one attribute named *id* which uniquely identifies e.

Definition 2.2. *(Trace)*
A trace $t =< e_1, e_2, \ldots, e_k >$ is a finite sequence of ordered events. For instance, event e_1 is the starting event and e_k is the ending event of trace t, event e_2 happens immediately after event e_1.

Definition 2.3. *(Case)*
A case $c = (N_c, V_c, \mathcal{B})$ is an instance of a certain business process, where $N_c = \{id, trace_c, n_3, \ldots, n_j\}$ is the set of attribute names for case c, V_c is the set of values of the attributes for c, function $\mathcal{B} : N_c \rightarrow V_c$ correlates an attribute name to its value. Each case c has at least two attributes named *id* which uniquely identifies c and $trace_c$ which holds the trace of c.

Definition 2.4. *(Event Log)*
Let C be the universal set of cases, the event log $L \in C$ is a set of cases.

Figure 2.6 shows an example event log with the contents from the hospital log from Business Process Intelligence Challenge (BPIC) 2011. The example log contains 6 cases where each case represents a treatment process accepted by a certain patient. As shown in Figure 2.6, the case with id '1' contains a trace with four events and nine other attributes such like the age of the patient and the starting time of the case. The four events in the trace of case 1 are ordered according to their timestamp. For instance, the second event happens immediately after the first event but before the third event. Figure 2.7 shows the attribute information of the second event in the trace of case 1.

Definition 2.5. *(Simple Event Log)*
Let C_s be the universal set of cases. An event log $L \in C_s$ is called a simple event log, if for each case $c_s \in C_s$ such that $N_{c_s} = \{id, trace_{c_s}\}$ and for each event $e \in c_s$ such that $N_e = \{id\}$.

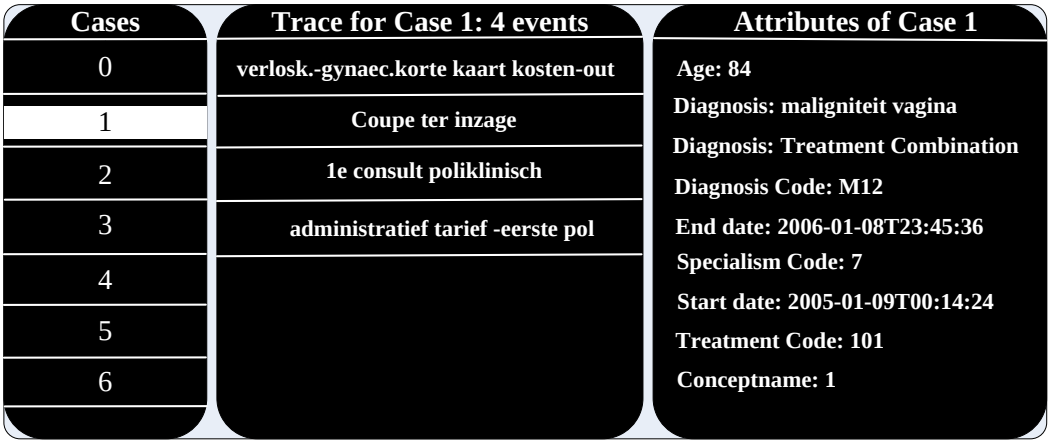


Figure 2.6.: An example event log with the contents from the hospital event log from BPIC 2011.

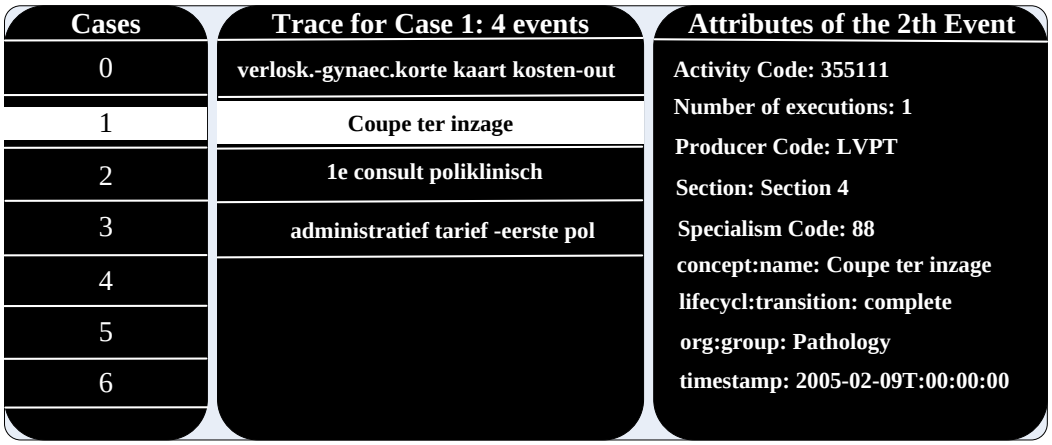


Figure 2.7.: The attribute information for the second event of Case 1.

Definition 2.5 gives a formal description about simple event log. It can be seen that in a simple event log the cases only contain two attributes which are id and trace and all the events in these cases only contain the attribute id. In this thesis, a simple event log will be displayed as a multi-set of traces. For example, let's make the assumption that a simple event log L_1 consists of four cases, where the first two cases contain the same kind of trace $<e_1, e_2, e_3>$ and the last two cases contain the same kind of trace $< e_1, e_4, e_3 >$, then $L_1 = \{< e_1, e_2, e_3 >^2, < e_1, e_4, e_3 >^2\}$. Most of existing process model discovery algorithms only require an input of simple event log [9].

2.2.2. Business Process Model Discovery (BPMD)

BPMD is one of the most important learning tasks in process mining research area which is able to automatically generate the enterprise process models by mining the event logs that record the execution information of enterprise business processes [23]. Figure 2.8 illustrates the basic idea of BPMD.

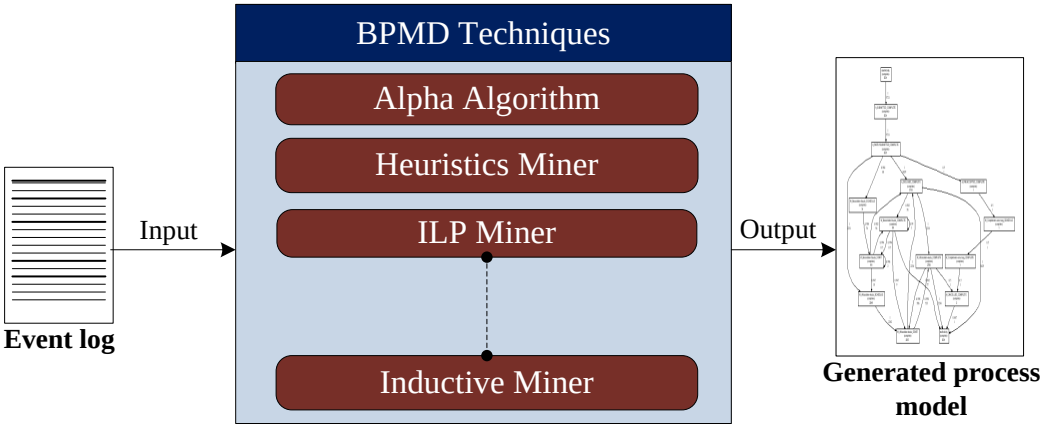


Figure 2.8.: Basic idea of business process model discovery.

A lot of excellent BPMD techniques such as Alpha algorithm [23], Heuristics Miner (HM) [37] and Inductive Miner (IM) [25] have been put forward in the literature. These techniques are capable of extracting the casual relations [23] between activities by analysing the direct relations of these activities that are recorded in the event logs and then expressing the obtained casual activity relations in the so called process models. In Section 2.2.2.1, the basic knowledge of HM is introduced because it has been widely utilised for analysing real-life event logs and the process mining techniques proposed in this thesis are based on HM. In Section 2.2.2.2, the KPIs for evaluating the quality of a mined process model are reviewed. In Section 2.2.2.3, we make a detailed comparison between classical BPMD techniques (including Alpha algorithm, HM, IM, ILP Miner (ILPM) [24], Fuzzy Miner (FM) [28] and Genetic Miner (GM) [29, 157–159]) and explains why we utilise HM for our process mining techniques.

2.2.2.1. Heuristics Miner (HM)

Figure 2.9 shows four fundamental workflow patterns [160, 161] which are XOR-choice pattern, OR-choice pattern, Parallel pattern and Loop pattern [160]. Most existing advanced BPMD techniques such like HM are capable of mining the four basic process patterns. The XOR-choice pattern as shown in Figure 2.9 means that only one of activity B and C can be chosen immediately after the execution of activity A. Therefore a simple event log consisting of two traces $< A, B, D >$ and

$< A, C, D >$ can be generated after carrying out the XOR-choice pattern. The OR-choice pattern from Figure 2.9 specifies that an arbitrary number (larger than or equal to 1) of activities among activity B and C can be chosen after executing activity A. As a result, four kinds of traces $< A, B, D >$, $< A, C, D >$, $< A, B, C, D >$ and $< A, C, B, D >$ can be generated by this OR-choice pattern. The Parallel pattern in Figure 2.9 indicates that activity B and C concurrently happen after implementing activity A and there is not a dependency relation between B and C. So two types of traces $< A, B, C, D >$ and $< A, C, B, D >$ can be output by running this parallel pattern. At last, the Loop pattern from Figure 2.9 states that the substructure composed by activity A, B and C can repeatedly happen. The HM is able to analyse the direct relations (see Definition 2.6) between activities recorded in event logs and then output the four basic workflow patterns mentioned above.

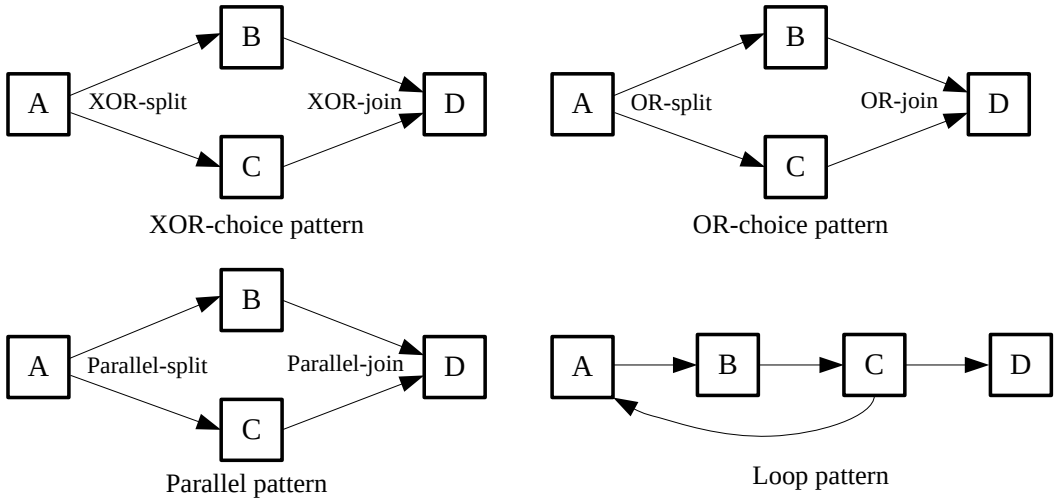


Figure 2.9.: Illustration of four kinds of basic workflow patterns.

HM is one of the most practical process model discovery techniques which considers the frequencies of relations between activities while mining process models from given logs [37, 162]. The infrequent activity relations will be neglected by HM from the generated process model. The strong expressive ability for process behaviours and the usage of frequency make HM a very robust BPMD technique. The process model output by HM is a casual net (C-net) which is defined by the following definitions [37]:

Definition 2.6. *(Direct and Casual Activity Relations)*
Let A be the set of activities of an event log L. Symbol $>_L$ represents the direct relation between two activities from A and symbol $\geq_L$ represents the causal relation between two activities from A. Let $a, b \in A$ be two activities, $\phi \in [-1.0, 1.0]$ be a threshold, $a >_L b = true$ if $|a >_L b| > 0$, where $|a >_L b|$ is the number of times that a is directly followed by b in L. $a \geq_L b = true$ if $|a \geq_L b| \geq \phi$, where $|a \geq_L b| \in [-1.0, 1.0]$ is the strength of casual relation between a and b.

Definition 2.7. *(Dependency Graph)*
Let A be a set of activities, $\Upsilon(A)$ denotes the set of casual activity graphs over

A. A causal activity graph $DG \in \Upsilon(A)$ is a tuple (V, E) where $V \in A$ is the set of vertices and $E \in (V \times V)$ is the set of edges. Each edge in DG represents a casual relation between two activities.

Definition 2.8. *(C-net)*
A C-net $cn = (A, \mathcal{I}, \mathcal{O})$ is a tuple, where A is the set of activities, $\mathcal{I} : A \to \mathcal{P}(\mathcal{P}(A))$ is a function which generates the input pattern for a certain activity, $\mathcal{O} : A \to \mathcal{P}(\mathcal{P}(A))$ is a function for generating the output pattern for a given activity, $\mathcal{P}(A)$ represents the powerset of A.

HM consists of two steps for generating a C-net for a given event log L. In the first step, HM builds a Dependency Graph (DG) which describes the casual relations of activities from L. Equation 2.5 displays how HM calculates the degree of dependency between any two activities a and b from L. In the second step, HM learns the splits and joins in the built DG so as to generate the target C-net for log L.

$$|a \succeq_L b| = \begin{cases} \frac{|a>_L b| - |b>_L a|}{|a>_L b| + |b>_L a| + 1} & \text{if } a \neq b \\ \\ \frac{|a>_L a|}{|a>_L a| + 1} & \text{if } a = b \end{cases} \tag{2.5}$$

Let's take the simple event log $L_1 = \{<A, B, D, E, F, G>^{100}, <A, C, D, F, E, G>^{100}, <A, B, D, F, E, G>^{100}, <A, C, D, E, F, G>^{100}\}$ as an example for explaining the execution process of HM. For building the DG for L_1, HM first computes the degree of casual relations for each pair of activities from L_1 (the computation results are shown in Table 2.4). Afterwards, a DG dg_1 is built by HM with a dependency threshold $\phi = 0.9$ which is shown in Figure 2.10.

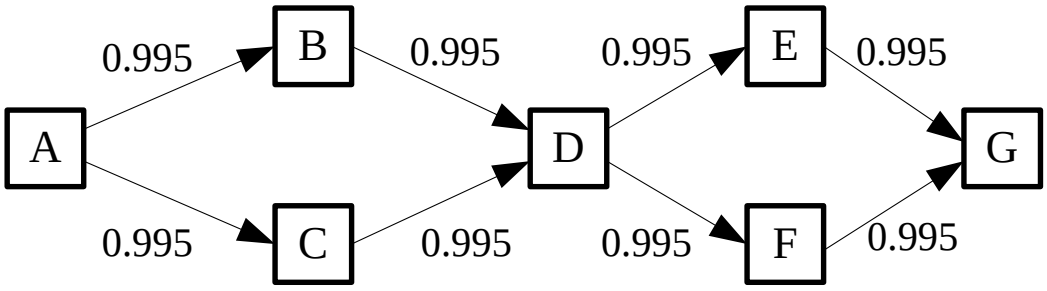


Figure 2.10.: Dependency graph dg_1 for the activities from event log L_1.

Afterwards, HM tries to learn the splits and joins for dg_1 and outputs a C-net cn_1 which is shown in Figure 2.11. Let's take activity A and D from L_1 as an example to interpret how HM decides the type of a split or a join. According to dg_1 (Figure 2.10), activity A has two output activities which are activity B and C,

Table 2.4.: Degree of casual relations between activities from event log L_1.

$\succeq_{L_1}$	A	B	C	D	E	F	G
A	0	0.995	0.995	0	0	0	0
B	-0.995	0	0	0.995	0	0	0
C	-0.995	0	0	0.995	0	0	0
D	0	-0.995	-0.995	0	0.995	0.995	0
E	0	0	0	-0.995	0	0	0.995
F	0	0	0	-0.995	0	0	0.995
G	0	0	0	0	-0.995	-0.995	0

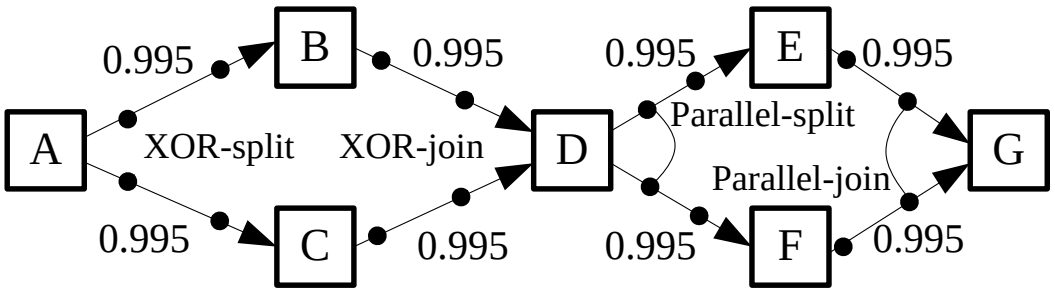


Figure 2.11.: The C-net cn_1 mined by HM for event log L_1.

denoted as $A\bullet = \{B, C\}$. Similarly, $\bullet B = \{A\}$ and $\bullet C = \{A\}$ (which means that both activity B and C only have activity A as their input activity). By checking the first kind of trace $< A, B, D, E, F, G >^{100}$ from log L_1, HM discovers that the nearest activity that can activate B is activity A in the trace and thereby records the found output pattern for A (i.e., $\mathcal{O}(A) = \{\{B\}^{100}\}$). By checking the second kind of trace $< A, C, D, F, E, G >^{100}$ from L_1, HM finds out that the nearest activity that can activate C is activity A in the trace and then records the discovered output pattern for A (i.e., $\mathcal{O}(A) = \{\{B\}^{100}, \{C\}^{100}\}$). After checking all the traces in L_1, the ultimate output pattern $\mathcal{O}(A) = \{\{B\}^{200}, \{C\}^{200}\}$ of activity A is obtained. Then HM judges that the output pattern of A represents a XOR-split (as shown in Figure 2.11). Activity D also has two output activities (i.e., activity E and F) according to cn_1. Through checking the first kind of trace of L_1, HM discovers that both activity E and F are activated by the same activity D and then records the found output pattern for D (i.e., $\mathcal{O}(D) = \{\{E, F\}^{100}\}$). After checking all the traces in log L_1, the final output pattern $\mathcal{O}(D) = \{\{E, F\}^{400}\}$ for D is obtained which is judged to stand for a Parallel-split (as shown in Figure 2.11). The same strategy can be utilised for mining the joins in dg_1 by going backwards through the traces

in log L_1. More technical details about HM can be found in [37, 162–165].

2.2.2.2. Mined Model Accuracy and Complexity Evaluation Methods

The quality of a mined process model for a certain event log should be evaluated before it can be utilised for a process-related analysis task. As indicated in [9], every BPMD technique has its own expressive bias and may generate inaccurate process models when mining event logs with process behaviours (expressed by activities and their precedence relations recorded in event logs) that it cannot express. Normally, an inaccurately mined process model will be less useful. Fitness and precision [9] are two primary metrics for estimating the accuracy of a mined process model. Fitness represents the percentage of the behaviour from an event log that can be covered by the process model mined from this log. Precision quantifies the ability of a BPMD technique to mine a process model which doesn't induce too many process behaviours that are not registered in the relevant event log. The ICS fitness proposed in [159] is utilised in this thesis for measuring the fitness of the process models mined by HM and the ETConformance checker put forward in [166] is employed for measuring the precision of the mined models. In this subsection, we mainly focus on introducing the basic knowledge about ICS fitness because fitness is the most significant KPI for evaluating the accuracy of a mined process model.

First, the definition about *token* from the Petri net [167–170] research area will be explained. A token is like a "virtual key" with which a specific activity can be activated. For the XOR-choice pattern from Figure 2.12, after executing activity A one token will be generated by A (see part a of Figure 2.12). With this token, both activity B and C are enabled. However, only one out of B and C can be activated according to the definition of XOR-choice pattern introduced in Section 2.2.2.1. In this example we presume that activity B acquires this token and is executed. Then, a new token is generated by B so that activity D is enabled (the XOR-join indicates that activity D only needs one token for execution). For the OR-choice pattern in Figure 2.13, by executing activity A one or two tokens can be generated. If one token is output by A then either B or C can be executed and both B and C can be activated if two tokens are output by A. Presume that two tokens are output by A (see part a of Figure 2.13), then each of B and C will consume a token, be executed and generate a new token which will be later consumed by D[4]. For the Parallel pattern shown in Figure 2.14, through activating activity A two tokens will be output by A where one token is for B and the other is for C. Next, each of B and C will generate a new token after being executed. Activity D needs the two tokens from both B and C to be implemented. The further knowledge

[4] If activity A outputs one token then activity D will only require one token for being implemented, or two tokens (one is from B and the other is from C) will be needed for implementing D

about token can be found in [9].

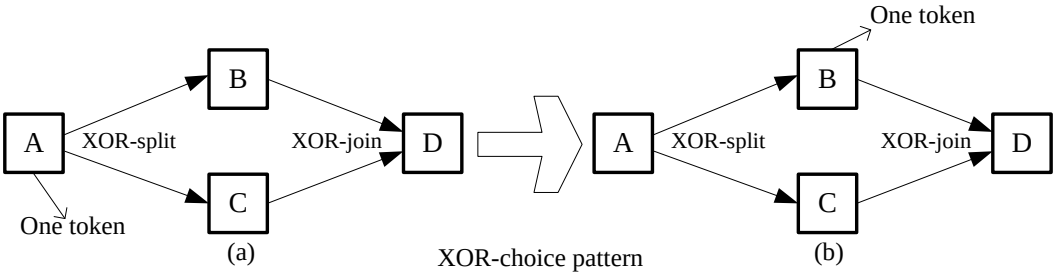


Figure 2.12.: The process of token generation and consumption for XOR-choice pattern.

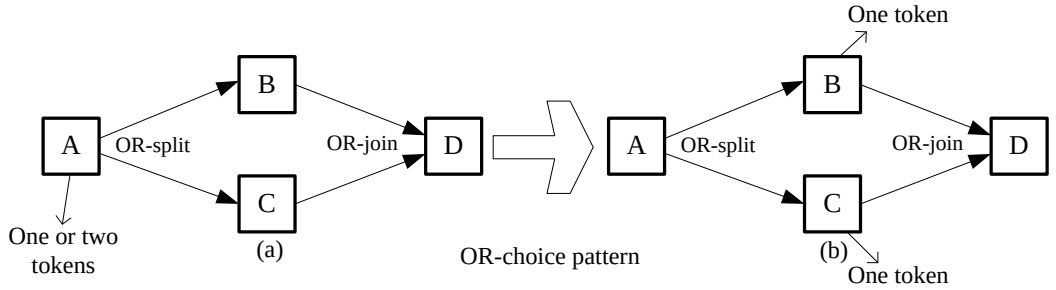


Figure 2.13.: The process of token generation and consumption for OR-choice pattern.

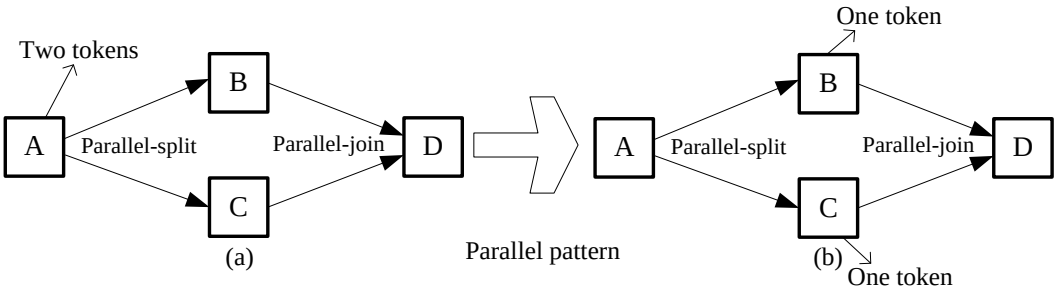


Figure 2.14.: The process of token generation and consumption for Parallel pattern.

The token replay technique [151–153, 171–173] is the essential technique for calculating the fitness of mined process models. For a given event log and a process model, the token replay technique tries to replay each trace from this event log by using the given model so as to find out which part of the trace can be represented by the model correctly. Here, an example trace $t =< A, B, C, D, E, G >$ and the process model shown in Figure 2.15 are employed to exhibit how the token replay technique works. Let $m = 0$ be the counter for missing tokens and $r = 0$ be the counter for remaining tokens (which are produced but not consumed by other activity), Figure 2.15 shows the token replay procedure for trace t. The activity A of trace t is first executed (an artificial token is given to A because it is the starting activity) and produces one token (because A leads to a XOR-split in

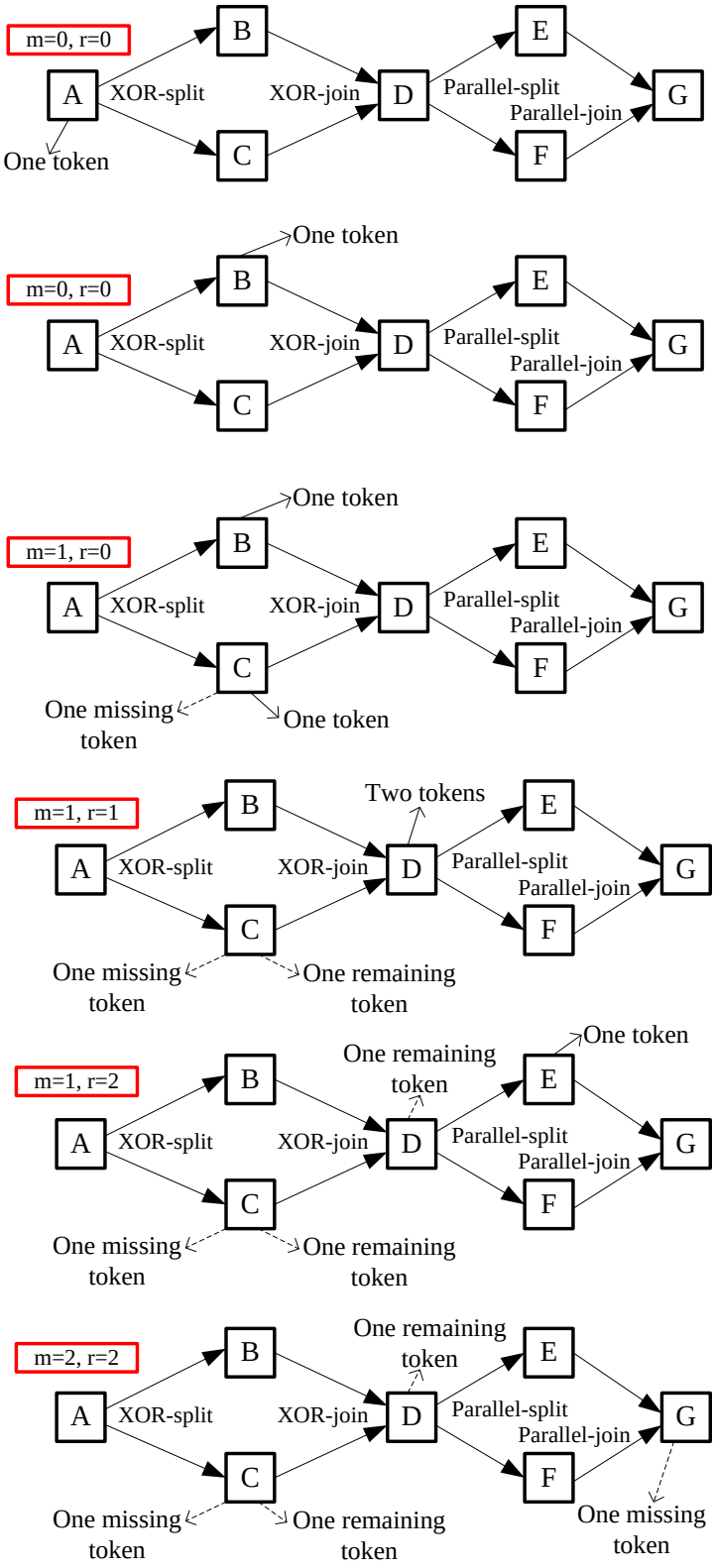


Figure 2.15.: Token replay for trace t over the example process model.

the given process model). At this stage A can be perfectly replayed (i.e., $m = 0$ and $r = 0$). At stage two, activity B from t is executed by consuming the token produced by A and then generates a new token. At the third stage, activity C from t cannot be activated because A only generates one token which is consumed by B. After marking the missing token for C, an artificial token is given to C with which C is implemented and produces a new token. At this stage the value of m is updated to one. At stage four, the fourth activity D of t is activated by consuming the token produced by B (D only needs one token because of the XOR-join structure in the model) and produces two tokens (according to the Parallel-split structure). As a result, the token produced by C is remained (i.e., the value of r is updated to one). At stage five, activity E from t is executed by consuming one token produced by D and then generates a new token. This gives rise to a remaining token at activity D because activity F doesn't appear in t. Therefore, the value of r is updated to two at this stage. Finally, due to the Parallel-join structure in the example model, the last activity G of t needs two tokens to be executed. However, there is only one token (produced by E) for G at this stage, as a result an artificial token (marked as a missing token) is given to G so that it can be implemented. At this stage, the value of m is updated to two.

Let L be a simple event log and $|L|$ be the total number of traces in L, A be the set of activities for L and $|A|$ be the number of activities in A, M be a process model mined from L by HM, m records the total number of missing tokens and r records the total number of remaining tokens obtained by replaying all the traces from L over model M, CA be the set of activities from A which can be replayed without any problem, MT be the set of traces from L in which tokens are missing during the replaying process and RT be the set of traces in which tokens are left behind during the replaying procedure. The ICS fitness within a range $(-\infty, 1]$ of M is defined as [159]:

$$ICS\text{-}fitness = \frac{|CA| - Punishment}{|A|} \tag{2.6}$$

$$Punishment = \frac{m}{|L| - |MT| + 1} + \frac{r}{|L| - |RT| + 1} \tag{2.7}$$

Another important perspective for evaluating the quality of a mined process model is complexity [174–180]. A process model with very high complexity is usually hard to be interpreted. Two kinds of process model complexity metrics are used in this thesis which are Place/Transition Connection Degree (PT-CD) [36] and Extended Cardoso Metric (E-Cardoso) [177], the greater the PT-CD and E-Cardoso are, the more complicated the models will be. However, both metrics are appropriate for the model expressed by Petri net but not by C-net generated

by HM. For this reason, the Heuristics net to Petri net plugin in ProM 6[5] is utilised for transforming the C-net mined by HM into a Petri net so that the PT-CD and E-Cardoso can be calculated. Let $|a|$ be the total number of arcs, $|P|$ be the number of places and $|T|$ be the number of transitions of a Petri net, the PT-CD of the Petri net can be calculated according to Equation 2.8 [36]. It can be seen that the metric PT-CD has the similar meaning as the density of a graph. It defines the complexity of a Petri net according to the density of relations between the elements (i.e., place and transition) in the Petri net. Dissimilarly, E-Cardoso tries to define the complexity of a Petri net according to its control flow behaviour [178]. It locates various splits (XOR-split, OR-split and Parallel-split) and gives each found split a certain penalty based on how many different substates the split induces after being implemented. Equation 2.9 shows the details of the computation for E-Cardoso of a specific Petri net $PN = (P, T, F)$, where P stands for the set of places, T stands for the set of transitions and F stands for the set of arcs for PN, for each $t \in T$ such that $t\bullet = \{p \mid (t,p) \in F\}$, for each $p \in P$ such that $p\bullet = \{t \mid (p,t) \in F\}$.

$$PT\text{-}CD = \frac{1}{2}\frac{|a|}{|P|} + \frac{1}{2}\frac{|a|}{|T|} \tag{2.8}$$

$$\text{E-Cardoso}(PN) = \sum_{p \in P} |\{t\bullet \mid t \in p\bullet\}| \tag{2.9}$$

2.2.2.3. Comparison between Classical BPMD Techniques

The Alpha algorithm [23, 181] can be regarded as the foundation stone in the BPMD research area based on which many more advanced algorithms such as HM, IM, ILPM, FM and GM are developed. However, the limitations on the expressive ability of Alpha algorithm make it hard to be the right choice when trying to handle the task of mining process models from real-life event logs. The expressive weakness of Alpha algorithm is normally reflected in modeling Non-free choice [23, 182, 183] (Figure 2.16) and short loops of length one and length two [37, 184] (as shown in Figure 2.17).

The IM has been designed to mine a sound and highly fitting block-structured process model expressed by Petri net from a certain real-life event log. As described in [27], IM adopts a divide-and-conquer strategy which partitions the

[5]ProM 6 is a very famous open-source tool published and maintained by the laboratory of Professor van der Aalst which integrates a lot of process mining techniques. More information about ProM 6 can be found in http://www.promtools.org.

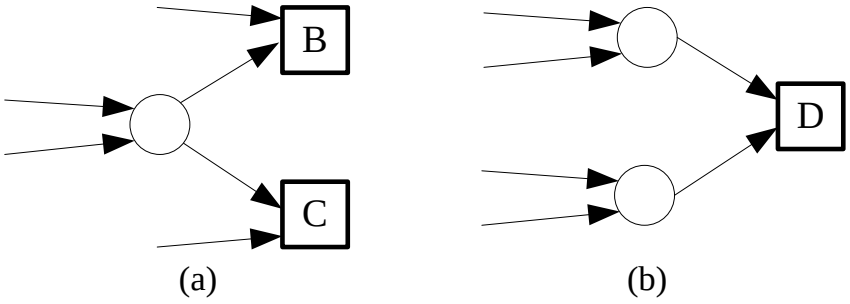


Figure 2.16.: Illustration of Non-free choice in process models.

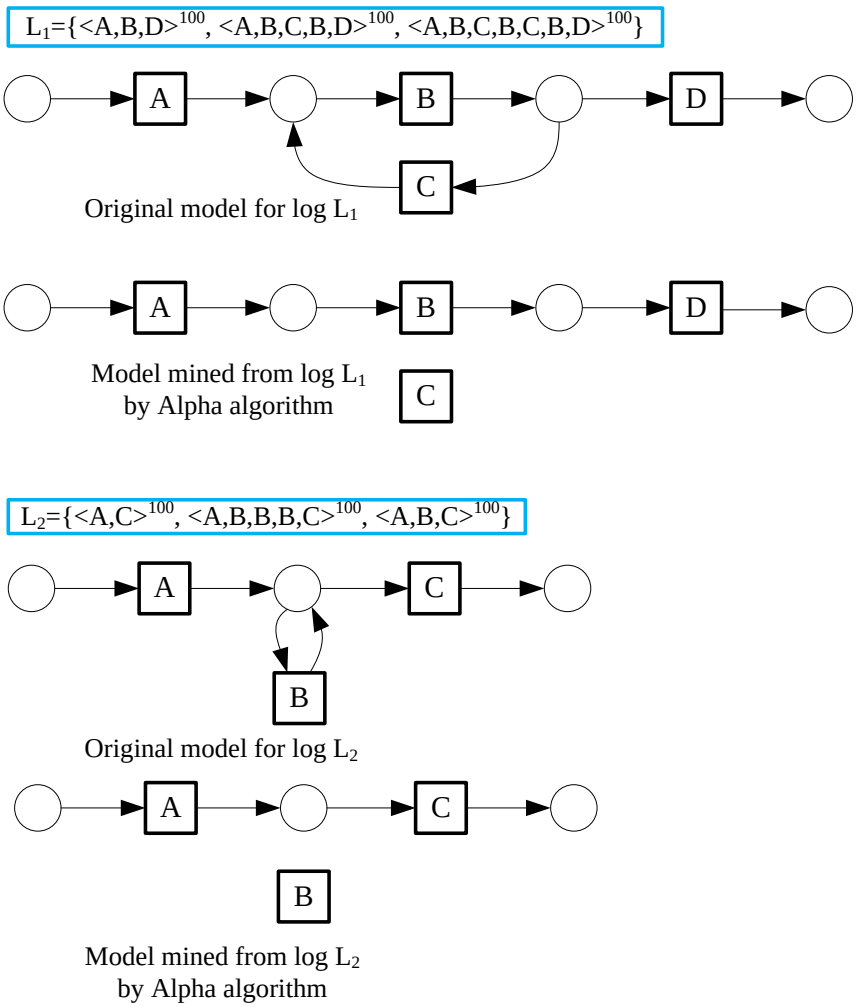


Figure 2.17.: Illustration of the problems met by Alpha algorithm for expressing loop of length one and length two.

original event log into sublogs according to four predefined cuts (in directly-follows graph) as shown in Figure 2.18, where these generated sublogs are organised by a process tree. Afterwards, the process model can be built according to the structure of the process tree. Figure 2.19 exhibits the running process of

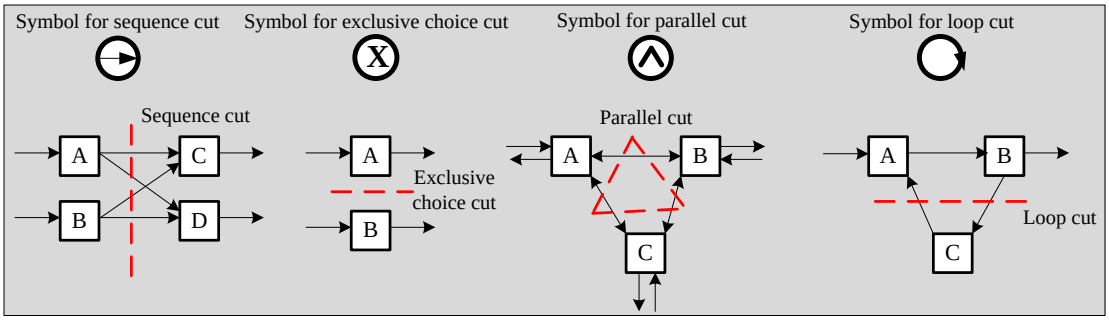


Figure 2.18.: Four kinds of cuts defined by Inductive Miner.

IM by using an example simple event log L_3. IM first builds the directly-follows graph for the activities from L_3. Then, the sequence cut is found in the directly-follows graph by IM. As a result, L_3 is divided into three sublogs which are SL_1 (contains activity X), SL_2 (contains activity A, B, C, D and E) and SL_3 (contains activity F). The three sublogs are connected by the symbol for sequence cut in the process tree built. Furthermore, the directly-follows graph for each generated sublog is also built by IM. Nevertheless, no more cuts can be further discovered in the three directly-follows sub-graphs, so the log division process stops and a process model (see Figure 2.20) is finally generated by IM based on the process tree obtained. Due to no cut is found in the directly-follows graph for SL_2, the relations between activity A, B, C, D and E from SL_2 are hidden in a flower-style model (Figure 2.20) and this case reflects one of the weaknesses of IM. The process model mined by HM for L_3 is shown in Figure 2.21 which can perfectly express all the activity relations recorded in L_3.

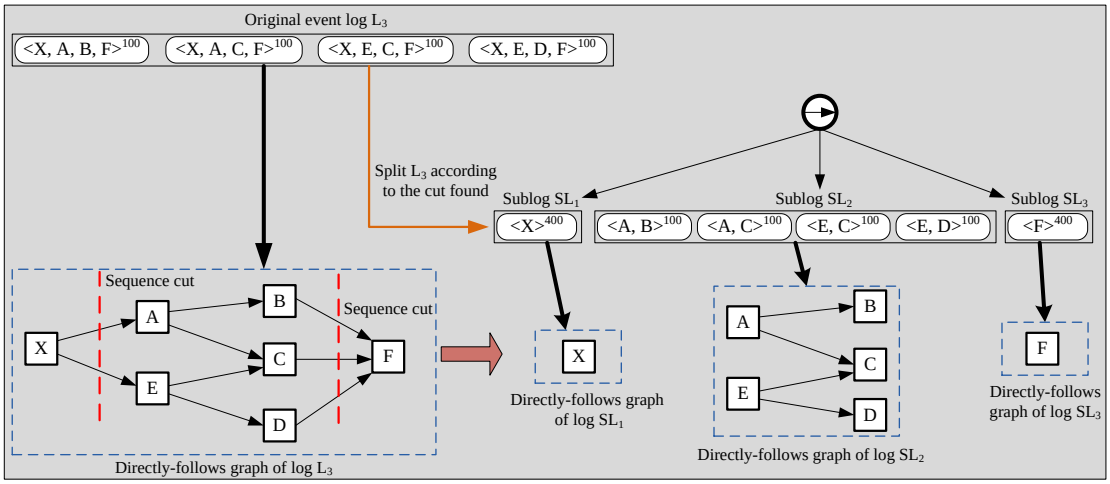


Figure 2.19.: Running process of IM over the simple event log L_3.

Another example simple event log L_4 is shown in Figure 2.22. By detecting the directly-follows graph for L_4, two types of cuts (i.e., the sequence cut and the loop cut) are found by IM. At last, a process model as shown in Figure 2.23 for L_4 is output by IM. According to L_4, activity E has a parallel relation with activity

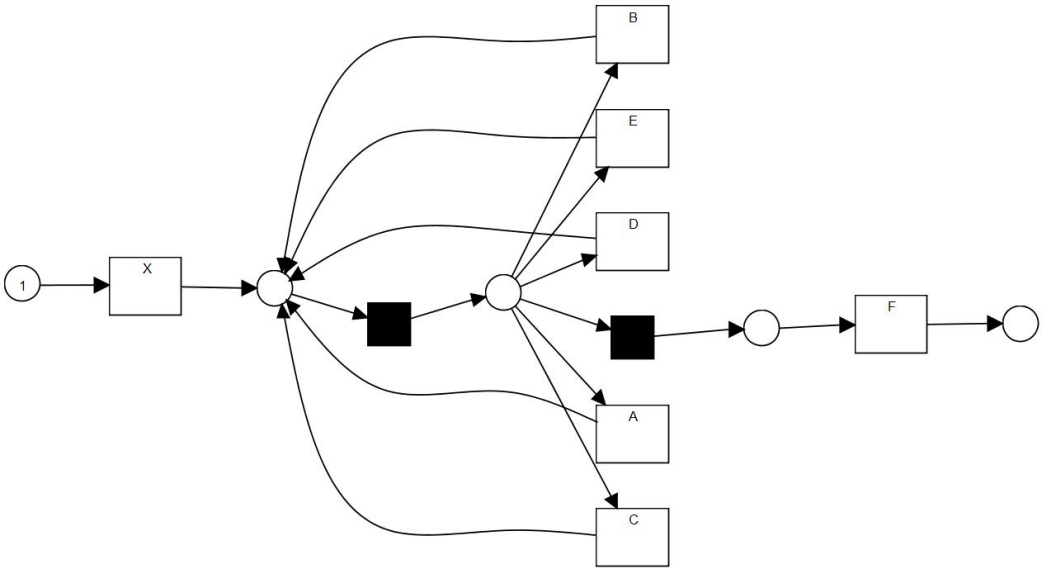


Figure 2.20.: The process model mined by IM for the simple event log L_3.

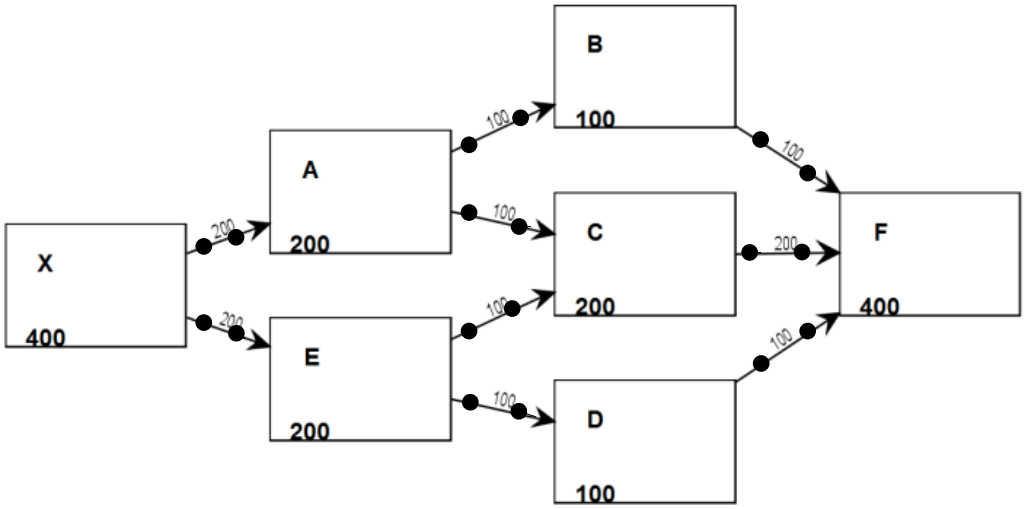


Figure 2.21.: The process model mined by HM for the simple event log L_3.

C and D which is not expressed in the model in Figure 2.23 because activity B, C, D and E are judged to belong to a loop structure by IM as shown in Figure 2.22. This case reveals another weakness of IM which is that IM has a relatively weak definition about loop cut. Figure 2.24 shows the process model mined by HM from L_4 which can express the concurrency relation between activity E and activity C and D well.

The ILPM is a BPMD technique which guarantees high fitness of the mined process model and is based on the idea that places can restrict the possible firing sequences of a Petri net. The developers of ILPM define an optimality criterion

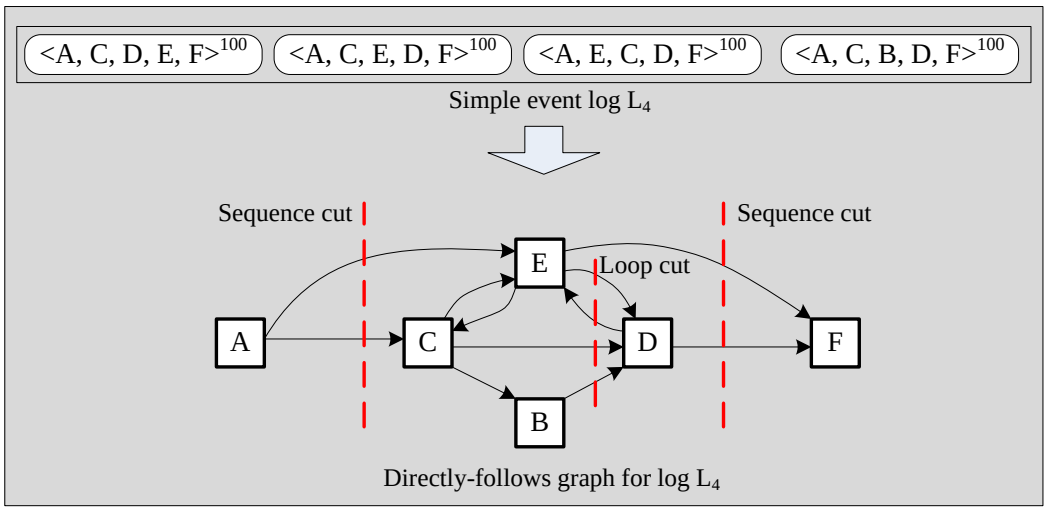


Figure 2.22.: Cuts discovered by IM from the directly-follows graph for event log L_4.

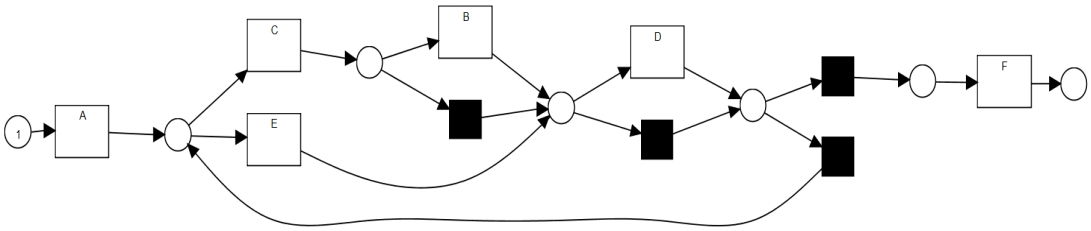


Figure 2.23.: The process model mined by IM for the simple event log L_4.

utilised to transform the inequation system into an ILP (integer linear programming) [185] which then is resolved under different conditions so as to build the places of a Petri net capable of replaying the log. Though ILPM can generate highly fitting process models when mining real-life event logs, the built models might contain too many arcs which greatly influences the comprehensibility of the models (not to speak that the activity relations in real-life logs are originally very complex) and this exerts bad effects on the practicality of ILPM in real-life

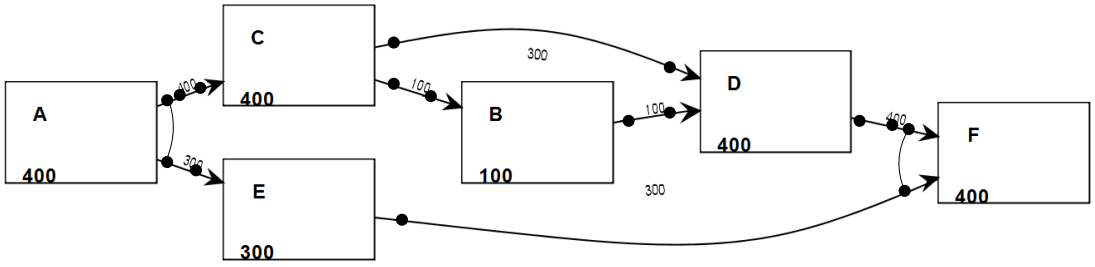


Figure 2.24.: The process model mined by HM for the simple event log L_4.

cases. Figure 2.25 shows the process model mined from the example log L_4 (Figure 2.22) by ILPM.

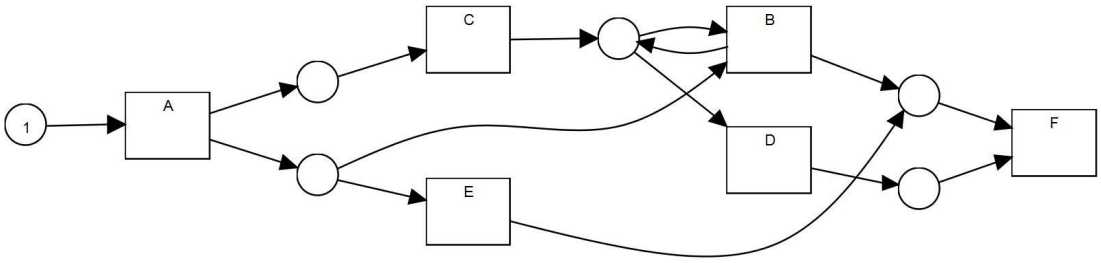


Figure 2.25.: The process model mined by ILPM for the simple event log L_4.

The FM [28] has been developed to deal with the problem of less-structured process models mined from real-life event logs. It utilises the same heuristics for generating the casual relations between activities as HM. However, the found casual relations of activities are not preserved in the model output by FM (i.e., the basic workflow patterns like XOR-split and Parallel-split cannot be distinguished). The concepts such as aggregation, abstraction, emphasis and customisation from the road map have been employed for simplifying and visualising complex and less-structured process models [186–188]. The GM uses the basic idea of genetic algorithm [189–192] from the evolutionary algorithm research area for mining highly accurate process models from event logs. The conformance checking technique is used for calculating the fitness of individuals in the populations and three genetic operators: selection, crossover and mutation [192] are utilised to explore the correct casual relations between the activities from the relevant event logs. Though GM can output an accurate process model with appropriate iteration times, the mining process usually takes a very long time to be finished especially when encountering event logs containing a large number of complex activity relations.

The IM and HM are both practical for mining real-life event logs. Each of them has its own advantages and disadvantages. However, the model mining mechanism of HM makes its expressive power easier to be extended. This is the main reason for us to choose HM for testing and working with our process mining techniques.

2.2.3. Assistant Techniques for Mining Better Process Models

Except for the pure BPMD techniques, several other kinds of assistant technologies have been developed in the process mining research area for helping mine better process models from real-life event logs, in which the trace clustering [31] technique and mined model abstraction technique [33] are most utility.

2.2.3.1. Trace Clustering

Some pioneering approaches have been developed to solve the problem of inaccurate and complex process models mined from real-life event logs. One efficient technique is trace clustering [31, 35, 36, 193–196] which mines the structural behaviours[1] of traces (trace behaviours) in an event log and then groups the traces with similar behaviours into the same sublog. Afterwards, by applying workflow discovery algorithms on each simpler sublog, more accurate and comprehensible process models can be obtained. Figure 2.26 shows the basic procedure for trace clustering.

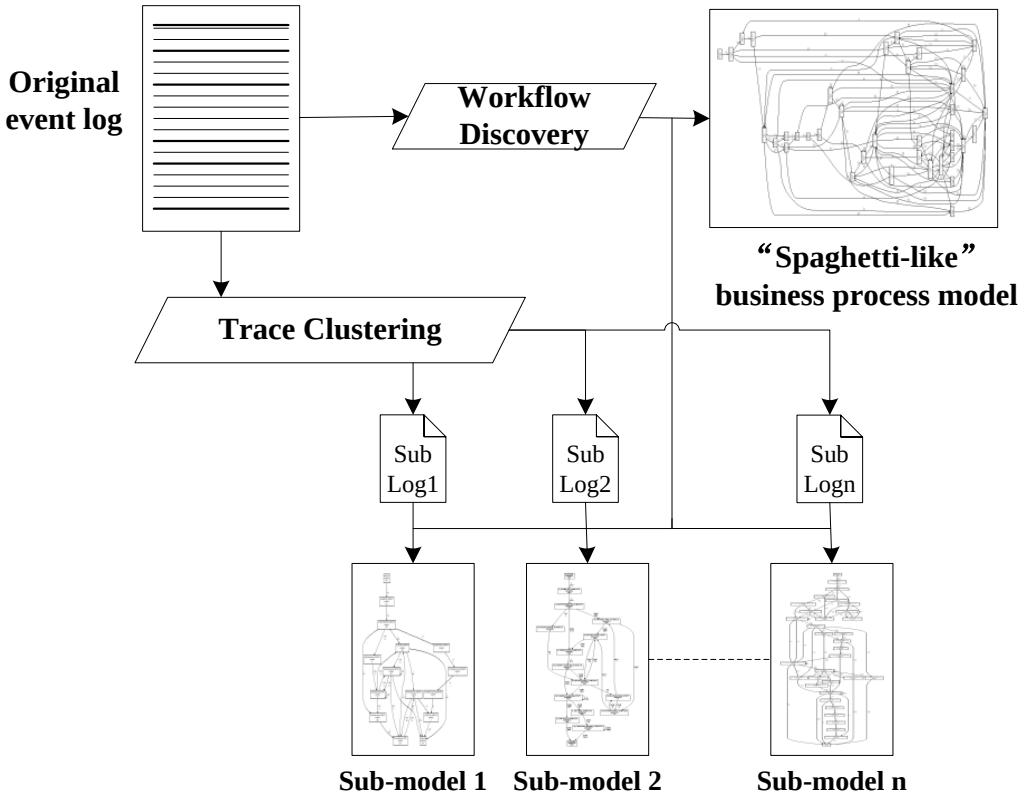


Figure 2.26.: Illustration of the basic trace clustering procedure in process mining.

In the literature, different trace clustering approaches have been put forward to overcome the negative impacts from high variety of behaviours stored in event logs. In [31] the authors propose an approach for expressing the traces by profiles so that a suitable environment can be built for clustering the traces. Each profile is a set of items that describe the trace from a specific perspective. Five profiles are recommended in [31] which are activity profile, transition profile, case attributes profile, event attributes profile and performance profile. By converting the profiles defined into an aggregate vector the distance between any two traces can

[1] Most trace clustering techniques only consider the structural behaviours of traces, while some consider the behaviours from both traces and other case attributes.

be measured. One advantage of this technique is that it provides a full range of metrics for clustering traces. In [193] the authors point out that a fully complete model (with high fitness) discovered may support a high variety of behaviours that are not registered in event log, as a result some significant structural features may be concealed in the mined model. Such a problem can be dealt with by considering the metric soundness [9] which measures the percentage of behaviours of the mined model that are recorded in the log among all of the behaviours supported by the model. An efficient technique is proposed in [193] which divides the whole process into a set of distinct sub-processes based on a greedy strategy which makes sure the further division of a process will lead to another increasingly sound sub-process. This method can also help solve the problem of high complexity of the initial model. Context-aware trace clustering techniques are proposed in [195] and [35]. In [195] the authors indicate that the feature sets based on subsequences of different lengths are context-aware for the vector space model and can reveal some set of common functions. Two traces that have a lot of conserved features in common should be gathered in the same cluster. In [35] the authors present an edit distance-based approach for distributing traces into clusters such that each cluster consists of traces with similar structural behaviours. The cost of edit operations is associated with the contexts of activities so that the calculated edit distance between traces is more accurate. The sequence clustering technique based on first-order Markov chains is presented in [196]. This technique learns a potential first-order Markov model for each cluster through an expectation-maximization algorithm. A sequence is assigned to a cluster which is able to generate it with higher probability. The technique proposed in this paper also inherits the idea from sequence clustering, the difference is our technique represents each cluster with a set of separate sequences (significant trace behaviours). In [36] a novel technique named Active Trace Clustering (ATC) is presented which directly optimises the fitness of each cluster's underlying process model during the run time. This method doesn't consider the vector space model for trace clustering, it simply discovers the suitable traces for each cluster so that the combined accuracy of the related models for these clusters is maximised. This method sufficiently resolves the gap between the clustering bias and the evaluation bias.

However, most currently available trace clustering techniques treat all of the trace behaviours captured in the event log equally. As a result, the impacts of some important trace behaviours are reduced. Moreover, these techniques focus mainly on the discovery of various kinds of trace behaviours while the quality of the underlying process model for each cluster learnt is not taken into account [36]. Hence, high-quality sub-process models from these trace clustering techniques cannot be guaranteed. The ATC that is put forward in [36] directly optimises the accuracy of each cluster's underlying process model. However, ATC only considers model accuracy metrics while the complexity of process models is neglected during trace clustering. The complexity of process models is also a very important metric and should not be ignored for trace clustering. Because a highly

accurate process model can still be very complicated.

2.2.3.2. Mined Model Abstraction

The limitation of current trace clustering techniques will be revealed while dealing with event logs containing massive trace behaviours. For instance, the event log from BPIC 2011 contains 624 activities among which a large number of relations are exhibited (the average out-degree for each activity is 6.2564) and most of the classical trace clustering methods cannot bring a significant improvement on the mining result for this hospital log (as shown in Section 6.4 of Chapter 6). Process model abstraction-based approaches make the assumption that the raw models mined from real-life logs contain low level sub-processes which should be discovered in the form of sub-traces in the original event logs and abstracted into high level activities so that the insignificant low level process behaviours can be hidden by being replaced via high level activities. Thus, more accurate and simpler high level process models can be obtained.

Process model abstraction [41, 42, 197–201] approach is also very effective for dealing with "spaghetti-like" business process models mined. In [33] the authors develop a two-step approach for mining hierarchical business process models. This approach searches for the sub-traces that repeatedly happen in event logs. Two kinds of such sub-traces are defined which are tandem arrays and maximal repeats. This approach first searches for all the tandem arrays and the maximal repeats in the event logs and then replaces them in the original event logs by using high level activities (each high level activity is an abstraction of a tandem array or a maximal repeat found) so that the high level event logs can be generated. Finally, the high level models (more accurate and simpler) could be mined by using existing workflow discovery algorithms executed on the high level logs. The authors in [41, 42, 199] indicate that the low level events recorded in event logs may be too granular and should be mapped to high level activities predefined in the enterprise process specifications. Hence, they put forward a mapping method that combines the domain knowledge captured from these specifications. With the high level activities generated better models on the higher abstraction level can be built. The authors in [38] present an automated technique for mining the BPMN models with sub-processes. This technique analyses the dependencies among the data attributes attached to events. The events that are judged to have high dependencies will be put in the same sub-processes.

Nevertheless, most of the classical process model abstraction approaches presented focus mainly on searching for the sub-processes and cannot assure the quality of the built high level models. It is possible that the high level activities in the underlying abstracted models may still have a large amount of relations among each other.

3

A Novel Heuristic Method for Improving the Fitness of Mined Business Process Models

3.1. Introduction and Motivation

As one of the most important learning tasks in the business process mining area, the present BPMD techniques encounter great challenges when trying to mine models from real-life event logs. Such logs that usually stem from the business processes executed in highly flexible environments often contain complex trace behaviours (expressed by the activities and their precedence relations in the trace) which might be far beyond the expression ability of existing BPMD approaches. As a result, low-fitness process models might be generated. For instance, Figure 3.1 shows an example event log L_1 together with the process model generated by carrying out HM [37] on this log. It can be seen that the mined model for L_1 has a relatively low fitness value which is 0.7752 and this is largely due to the existence of inexpressible process behaviours for HM in L_1.

One effective auxiliary strategy for solving this problem has been proposed in the literature: the mining algorithm enhancement-based strategy (MEBS) which aims at improving the expressive ability of existing BPMD algorithms or developing new algorithms that are able to model more complex workflow patterns. This chapter will focus on this strategy. Several state-of-the-art application instances of MEBS such as the Language-Based Region Miner [39], the State-Based Region Miner [202], the ILP Miner [24], the Inductive Miner [25] and the model repair technique [203] have been put forward in academia. These proposed application instances are able to help mine high-fitness models expressed by Petri net [170]. However, few efforts have been made to help HM generate better mining results. According to [8], HM is one of the most popular BPMD tools in the ProM framework [9] for dealing with real-life event logs and developing an auxiliary method to help it mine high-fitness process models is far from trivial.

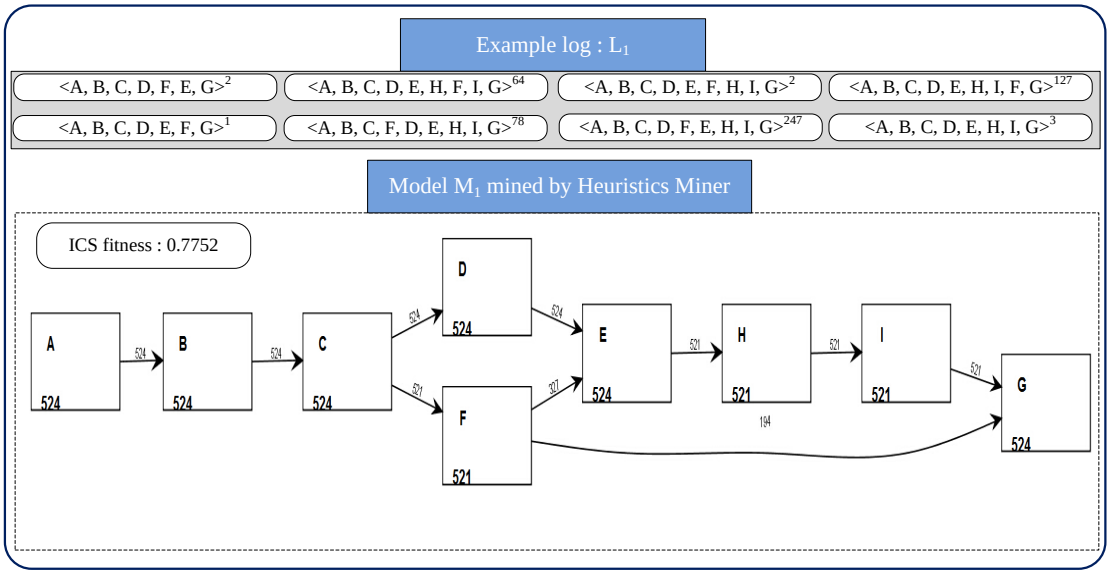


Figure 3.1.: An example event log L_1 and the process model mined by executing Heuristics Miner on L_1.

In this chapter, we put forward a novel heuristic method for helping HM mine high-fitness process models. In addition, the principle of our method has a universal significance and can also be utilised for assisting other kinds of BPMD algorithms in mining better process models. But in this chapter, we only focus on HM and adapting our technique to other BPMD algorithms will be a future job. The structure of the main contents of this chapter is organised as:

- The problem addressed by this chapter is discussed in detail in Section 3.2.

- In Section 3.3, a method for extracting and organising the process behaviours from real-life event logs is proposed.

- In Section 3.4, an activity ranking algorithm is developed which is able to sort the activities from a particular event log according to the potential impact of each activity on the fitness of the process model mined from this log.

- In Section 3.5, a heuristic algorithm named HIF that inherits the idea of MEBS for improving the fitness of models mined by HM is presented. This proposed technique is able to detect and convert the inexpressible process behaviours related to a particular activity.

- In Section 3.6, we carry out HIF on an example event log for explicitly exhibiting the operating principle of HIF so that the readers can better understand it. In the meantime, the correctness of HIF can also be testified through this preliminary test.

3.2. Problem Description

Every BPMD algorithm has its own feature reflected by its representational bias, i.e., the category of process behaviours that can be expressed by the algorithm [9]. Creating an approach to represent all kinds of workflow patterns therefore cannot be accomplished. Occasionally, the business processes executed under flexible environments, e.g., healthcare, customer relationship management (CRM) and product development [36], may exhibit a lot of complex behaviours which are far beyond the expressive ability of the utilised BPMD techniques. As a result, mining the event logs that record the behaviours of these business processes might generate non-fitting process models.

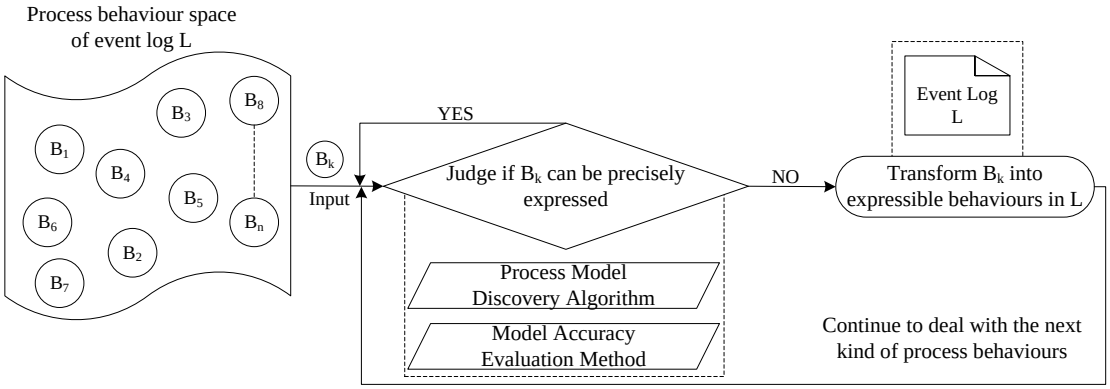


Figure 3.2.: The process for dealing with inexpressible process behaviours recorded in real-life event logs.

In this chapter, we transform the fitness improvement problem for the non-fitting models mined by HM into the problem of locating the inexpressible process behaviours for HM in event logs and converting these found behaviours into expressible behaviours. As shown in Figure 3.2, an element B_k in the process behaviour space (PBS) represents a kind of process behaviour extracted from a specific event log L. Afterwards, it is assessed whether B_k can be expressed by HM. If B_k cannot be expressed then all the process behaviours that pertain to B_k in L will be converted into expressible behaviours. Given an event log, how to build the PBS relevant to this log and how to locate the inexpressible behaviours in the PBS and then transform them into expressible behaviours for HM are the main problems that this chapter is going to solve.

In this chapter, we propose a novel heuristic method which is able to solve the problems mentioned above efficiently. The proposed method utilises a novel way for building the PBS of a specific event log based on which the procedure for detecting and converting the inexpressible process behaviours for HM can be implemented smoothly.

3.3. Build Process Behaviour Space

The business process behaviours recorded in real-life event logs are displayed by the activities and their precedence relations from traces. How to effectively extract and organise these behaviours is the first challenge encountered by our approach. In this section, we present a new perspective to view the business process behaviours recorded in real-life event logs based on two concepts: Behaviour-Related Activity and Behaviour-Related Sub-Trace.

Definition 3.1. *(Behaviour-Related Activity)*
Let SA_L be the set of activities for event log L. Symbol $\Rightarrow_L$ represents a behaviour-based relation between any two activities from SA_L. For two activities $a, b \in SA_L$, $a \Rightarrow_L b = true$ if $a >_L b = true$ or $b >_L a = true$ and b is called a Behaviour-Related Activity (BRA) of a.

According to Definition 3.1, activity $b \in SA_L$ is a BRA of activity $a \in SA_L$ if there is a direct relation (defined in Chapter 2) between a and b in event log L (no matter how the direct precedence order between a and b is).

Definition 3.2. *(Behaviour-Related Sub-Trace)*
Let SA_L be the set of activities for event log L. Let t be a trace from L, $st \sqsubseteq t$ be a sub-trace of t and SA_{st} be the set of activities for st. Given an activity $a \in SA_L$, st is a Behaviour-Related Sub-Trace (BRST) of a if $\forall b \in SA_{st} \wedge b \neq a$ such that $a \Rightarrow_L b$ and $a \in SA_{st}$. And st is a Maximal Behaviour-Related Sub-Trace (MRST) of a if $\nexists st'$ such that st' is a BRST of a and $st \sqsubset st'$.

Let's take the simple event log L_1 depicted in Figure 3.1 as an example. According to Definition 3.1, activity F has six BRAs which are activity D, E, H, I, G and C. Seven kinds of MRSTs for activity F can be discovered from L_1 (as shown in Figure 3.3) according to Definition 3.2. It can be seen that every MRST of F contains activity F and all the other activities in the MRST are BRAs of F.

Definition 3.3. *(Set of MRSTs and PBS)*
Let L be an event log, SM is called a set of MRSTs of an activity $a \in L$ if SM contains all the MRSTs that can be discovered in L for activity a. The PBS for log L comprises the sets of MRSTs of all activities in L.

Let Ω be a process model discovery algorithm, Σ_f be a fitness calculation mechanism and α be a threshold of fitness of mined process models. The inexpressible process behaviours are defined as follows:

Definition 3.4. *(Inexpressible Process Behaviours)*
Let L be an event log, PBS_L be the PBS related to L, $SM_a \in PBS_L$ be the set of MRSTs of activity a from L. The activity relations stored in SM_a are called inexpressible process behaviours if $\Sigma_f(\Omega(SM_a), SM_a) < \alpha$.

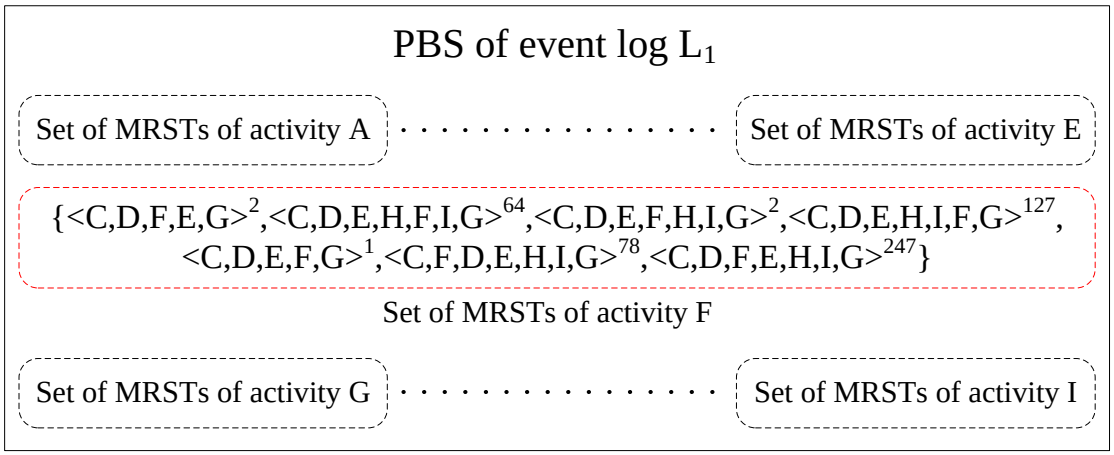


Figure 3.3.: The PBS built for the example event log L_1.

In our technique, the process behaviours recorded in an event log are divided into several groups where each group is relevant to a single activity from this log and the process behaviours for a group are stored in the MRSTs of its related activity. For instance, the PBS for log L_1 consists of nine sets of MRSTs where each set of MRSTs is relevant to a specific activity from L_1 (as shown in Figure 3.3). Our technique is devised to detect each set of MRSTs stored in PBS iteratively for finding and converting inexpressible process behaviours for the utilised BPMD technique. Algorithm 3.1 gives a brief description of the PBS building approach.

Algorithm 3.1 Construct the PBS for a specific event log L (CPBS)

Input: an event log L, the set of activities SA_L for L

1: **Let** PBS_L be a set of sets of MRSTs for log L.
2: $PBS_L \leftarrow null$
3: **for** each activity $a \in SA_L$ **do**
4: generate the set of MRSTs BST_a for a
5: $PBS_L \leftarrow PBS_L \cup BST_a$
6: **end for**

Output: a set of sets of MRSTs PBS_L for the input log L

3.3.1. Direct Activity Relations vs Casual Activity Relations

The goal of our approach is to locate the complex process behaviours (i.e., those that cannot be expressed by HM) recorded in event logs and convert them into simple behaviours so that the mined models possess high fitness. The concept casual activity relation mentioned in Chapter 2 has been widely utilised in existing BPMD techniques. Nevertheless, the casual activity relation is not applicable in all situations because it itself has a limited expressive ability. While dealing

with relatively complex relations among activities this concept might not be sufficient enough. However, the concept direct activity relation is able to help extract all possible process behaviours recorded in event logs and this is also the main reason for our approach to employ this concept.

3.4. Activity Ranking

In this section, an activity ranking method is put forward in which the process behaviours related to the higher-ranked activities will be handled before the process behaviours relevant to the lower-ranked activities. The proposed activity ranking method is based on two concepts: Behaviour-Related Activity Weight (BAW) and Activity Ranking Weight (ARW). Given an activity a from event log L, the BAW of a is defined as:

$$BAW_a = |\bullet >_L a| + |a >_L \bullet|. \tag{3.1}$$

In Equation 3.1, $|\bullet >_L a|$ represents the total number of activities from L that are directly followed by a at least once and $|a >_L \bullet|$ represents the total number of activities which directly follow a in L at least once. The value of BAW for each activity in log L_1 (see Figure 3.1) is shown in Table 3.1. For instance, $|\bullet >_{L_1} F| = 5$ because activity F directly follows five kinds of activities in L_1 which are activity D, H, E, I and C. $|F >_{L_1} \bullet| = 5$ because activity F is directly followed by five kinds of activities which are activity E, I, H, G and D. As a result, the value of BAW for activity F is 10 (i.e., $BAW_F = 10$).

Table 3.1.: The activity ranking result for the example event log L_1.

Ranking	Activity	BAW	Frequency of Occurrence	ARW
1	F	**10**	521	0.9943
2	E	5	**524**	0.5
3	D	4	524	0.4
4	H	4	521	0.3977
5	I	4	521	0.3977
6	G	3	524	0.3
7	C	3	524	0.3
8	B	2	524	0.2
9	A	1	524	0.1

Axiom 3.1.
The larger the BAW of an activity from an event log L is, the more possible this

activity will be the main factor that leads to the inexpressible process behaviours in log L.

According to Axiom 3.1, the BAW is employed by our technique to quantify the complexity induced by an activity on its related process behaviours (i.e., the MRSTs of this activity) recorded in the relevant event log. However, Axiom 3.1 might not be applicable in all situations, e.g., an activity that only joins a concurrent behaviour may also have a large BAW but it will not cause any inexpressible process behaviour as long as the utilised BPMD algorithm can model concurrency. These additional situations are also considered in our approach proposed in the next section.

Let a be an activity from event log L, the ARW of a is defined as:

$$ARW_a = \frac{BAW_a}{BAW_{max}} \times \frac{|a|}{OF_{max}}. \tag{3.2}$$

where BAW_{max} stands for the BAW of a particular activity from L which has the largest BAW, $|a|$ stands for the occurrence frequency of activity a in log L and OF_{max} represents the occurrence frequency of an activity from L which has the largest frequency of occurrence. According to Equation 3.2, the ARW of an activity consists of two parts. The fist part is based on the BAW of this activity while the second part considers the influence level of activity on the fitness of the final mined model[1]. Table 3.1 shows the ranking of activities from L_1. It can be seen that activity F which has the largest ARW ranks first among all the activities in L_1 which means that our technique will first check the MRSTs of F recorded in the PBS built for L_1 (see Figure 3.3). The details of the activity ranking method are depicted in Algorithm 3.2.

3.5. A Heuristic Method: HIF

The core algorithm of our technique is introduced in this section. In Section 3.5.1, we present a method (named DCIB) which is able to detect and convert the inexpressible process behaviours related to one specific activity. Afterwards, we propose a heuristic method (named HIF) based on the discussions in the former sections for improving the fitness of the mined process models in Section 3.5.2.

[1] An infrequent activity may not exert great influence on the fitness of the final generated process model, even though it leads to complex process behaviours.

Algorithm 3.2 Activity ranking (AR)

Input: the set of activities SA_L for an event log L.

Let i, j be two variables of type Integer.
Let a be an activity.
$i \leftarrow 0$, $j \leftarrow 0$, $a \leftarrow null$
for $(i; i < |SA_L|; i{+}{+})$ **do**
  for $(j \leftarrow i+1; j < |SA_L|; j{+}{+})$ **do**
    if $ARW_{SA_L[i]} < ARW_{SA_L[j]}$ **then**
      $a \leftarrow SA_L[i]$
      $SA_L[i] \leftarrow SA_L[j]$
      $SA_L[j] \leftarrow a$
    end if
  end for
end for

Output: a set of activities SA_L, the larger the ARW of an activity is, the lower the index of this activity in SA_L is.

3.5.1. Detection and Conversion of Inexpressible Process Behaviours

Before presenting the method DCIB, we first introduce a new concept called Environment Item.

Definition 3.5. *(Environment Item)*
Let SA_L be the set of activities from event log L, activity a, b and c are three activities from SA_L, the tuple (b, c) is an Environment Item (EI) of activity a if $\exists t \in L$ such that $< b, < a \ldots >, c > \sqsubseteq t$, where t stands for a trace from L and $< a \ldots >$ represents a sub-trace that only consists of activity a (one or more).

According to Definition 3.5, the activity F in the example log L_1 has six EIs which are EI (D,E), (H,I), (E,H), (I,G), (E,G) and (C,D). Activity A has one EI which is (null,B) where the value *null* indicates that activity A appears as a starting activity.

Axiom 3.2.
Converting an activity into a new activity under appropriate environment item will help reduce the complex process behaviours induced by this activity.

Figure 3.4 shows an event log L_2 generated by converting activity F under environment (D,E) into a new activity F1 and converting F under environment (H,I) into a new activity F0 in log L_1. As illustrated in Figure 3.4, the process model mined from the newly created log L_2 has a much higher fitness than the model mined from L_1. The main reason for such an improvement on fitness is that

the conversion of activity F under environment (D,E) and (H,I) transforms the inexpressible (complex) process behaviours (related to F) for HM into expressible (simple) process behaviours.

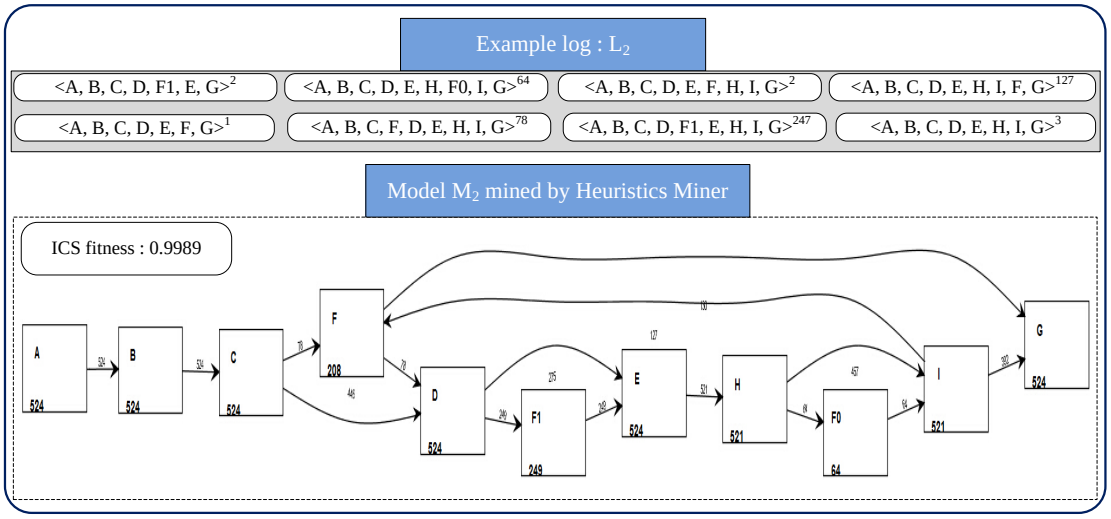


Figure 3.4.: The process model mined from newly generated log L_2.

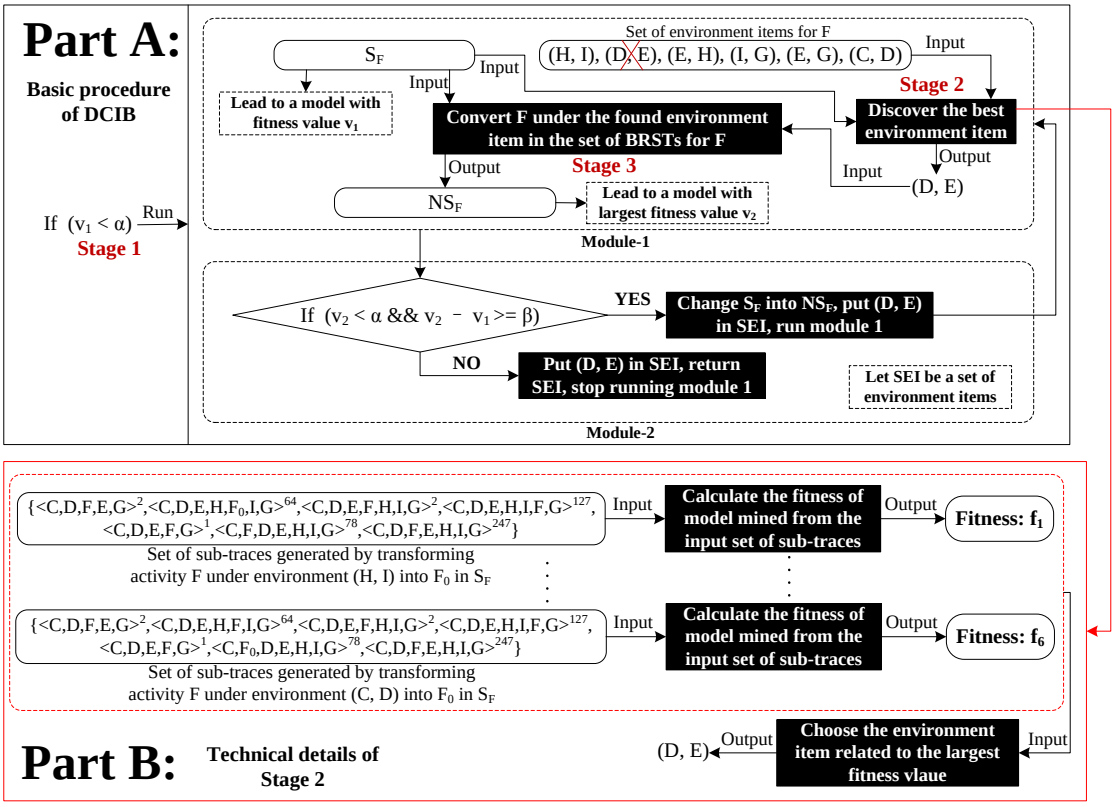


Figure 3.5.: The basic procedure for technique DCIB.

The algorithm DCIB (that will be) proposed in this subsection is able to assist in detecting the suitable EIs for a specific activity under which transforming the

Algorithm 3.3 Detection and conversion of inexpressible behaviours (DCIB)

Input: an activity a from event log L, the process behaviour space PBS_L for log L, a target fitness α, a model fitness improvement threshold β

1: **Let** BST_a be a set of maximal behaviour-related sub-traces for activity a.
2: **Let** SEI be a set of environment items.
3: **Let** vb be a variable of type Boolean.
4: $BST_a \leftarrow null$, $SEI \leftarrow null$, $vb \leftarrow null$
5: discover the set of environment items SEI_a for activity a
6: $BST_a \leftarrow PBS_L[a]$ # extract the set of behaviour-related sub-traces for a
7: **repeat**
8: Module−1(BST_a, SEI_a) # See Figure 3.5 for Module−1 and Module−2
9: **until** (($vb \leftarrow$ Module−2(SEI, α, β)) == $false$)

Output: a set of discovered environment items SEI for activity a.

activity into new activities can help simplify the complex process behaviours led by this activity. The storage structure of process behaviours in PBS provides a basis for DCIB to fulfil such a function. Specifically speaking, for each time DCIB discovers the qualified EIs for a certain activity by detecting its MRSTs stored in the relevant PBS[2]. Let's take the activity F from log L_1 as an example to explain the primary procedure for DCIB. Let S_F stand for the set of MRSTs for activity F (the details of S_F are exhibited in Figure 3.3), v_1 represents the fitness value of the process model mined from S_F, SEI be a set of EIs, α be a target fitness and β be a minimum fitness improvement threshold. As illustrated in part A of Figure 3.5, DCIB contains three stages and two modules. In stage 1, it judges whether v_1 is less than the target fitness α. If it is not, DCIB stops because the negative influences aroused by the inexpressible process behaviours related to activity F is acceptable. In stage 2 and stage 3 (that belong to Module−1), DCIB searches for the best EI ((D,E) in our example) of activity F among all its EIs (mentioned above) under which converting F into a new activity will generate a new set of MRSTs NS_F for F where the fitness of the model mined from NS_F has the largest value (i.e., v_2) compared with the models mined from other set of MRSTs generated by transforming F under other EIs of F. Then the found EI (D,E) is removed from the original set of EIs for activity F. The part B of Figure 3.5 shows the details for realising the stage 2 of DCIB. In Module−2, DCIB judges if $v_2 < \alpha$ and $v_2 - v_1 \geq \beta$. If it is, put the found EI (D,E) in SEI, replace S_F by using NS_F and continue running Module−1. If $v_2 \geq \alpha$, DCIB stops because the EIs found so far are enough to help reduce the negative influence led by the complex process behaviours aroused by F to a certain extent (indicated by α). DCIB will also stop running if $v_2 - v_1 < \beta$. Because adding new activities will help improve the accuracy of the potential model but may also increase the complexity of the model at the same time. It is not worth to add new activities if the model fitness cannot

[2]Detecting the qualified EIs in the MRSTs of an activity instead of in the whole process behaviours recorded in the event log will greatly reduce the detection time (i.e., the algorithm efficiency is improved).

be improved to a certain extent. The main procedure of technique DCIB is briefly depicted in Algorithm 3.3.

3.5.2. A Heuristic Method for Improving the Fitness of Mined Process Models

In this subsection, we propose a heuristic method (named HIF) based on the discussions in the former sections for improving the fitness of mined models. The details about HIF are shown in Algorithm 3.4.

Algorithm 3.4 HIF

Input: an event log L, the set of activities SA_L of log L, a target fitness α, a model fitness improvement threshold β, a threshold μ for the number of newly added activities.

1: **Let** x be a variable of type Integer.
2: **Let** PBS_L be a set of sets of maximal behaviour-related sub-traces.
3: **Let** SEI be a set of environment items.
4: **Let** LRA be a list of ranked activities.
5: **Let** AR be the activity ranking method introduced in Section 3.4.
6: $SEI \leftarrow null$, $x \leftarrow |SA_L|$ # x records the total number of activities in log L
7: $PBS_L \leftarrow CPBS(L, SA_L)$ # create the process behaviour space for event log L
8: $LRA \leftarrow AR(SA_L)$ # rank activities from SA_L
9: **repeat**
10: get the activity a that has the highest ranking out of LRA
11: $SEI \leftarrow \text{DCIB}(a, PBS_L, \alpha, \beta)$
12: **repeat**
13: get an environment item ei out of SEI
14: convert activity a under ei into a new activity in log L
15: put the newly generated activity for a in SA_L
16: **until** $((|SA_L| - x) == (\mu \times x) \,||\, \Sigma_f(\Omega(L), L) \geq \alpha \,||\, |SEI| == 0)$
17: **until** $((|SA_L| - x) == (\mu \times x) \,||\, \Sigma_f(\Omega(L), L) \geq \alpha)$

Output: a business process model with higher accuracy $M = \Omega(L)$.

Firstly, the number of activities in the given log L is stored in variable x (step 6). Then, algorithm HIF creates the PBS for log L (step 7) and also a ranking list LRA for the activities in L (step 8) according to the method proposed in Section 3.4. Next, HIF chooses an activity a which has the highest ranking in LRA and removes a from LRA (step 10). The inexpressible process behaviours aroused by a will be first handled which means that HIF always give priority to the main contradiction. Then, HIF searches for the qualified EIs for activity a through technique DCIB (introduced in Section 3.5.1) and the found EIs are put in set SEI (step 11). Afterwards, for each environment item $ei \in SEI$, HIF changes the activity a

into a new activity under environment ei in log L (this action will help improve the fitness of the model mined from L as demonstrated in Section 3.5.1), removes ei from SEI and put the newly generated activity in the set of activities SA_L for log L (steps 12–16). In HIF, a threshold μ is used to limit the number of the newly added activities because adding too many new activities might increase the complexity of the final model. If the number of the newly added activities is larger than $\mu \times x$ then HIF stops (step 16 and 17). The activity ranking procedure described in step 8 makes sure that the accuracy of the mined model could be improved as much as possible under the limitation given by μ. Furthermore, if the fitness of the model mined from L is larger than or equal to the given target fitness α then HIF also stops (step 16 and 17). Finally, a process model M with higher fitness value is output by HIF. For the example event log L_1 shown in Figure 3.1, our technique HIF discovers that the activity F under environment (D,E) and (H,I) is a factor for HM to generate low-fitness model. Then, through replacing activity F under environment (H,I) by a new activity F0 and replacing F under (D,E) by F1 in L_1 a more fitting process model is generated (as shown in Figure 3.4).

3.6. Preliminary Verification for HIF

In the verification process, we utilise the HM from ProM 6 for mining process models from event logs. Furthermore, we use the ICS fitness [29] for assessing the accuracy of the mined models because it has a computationally efficient calculative process. We tested the correctness and effectiveness of our technique by utilising an example event log L_3 as shown in Figure 3.6. It can be seen that this example log has 400 traces. The model M_3 mined from L_3 has a fitness value 0.6.

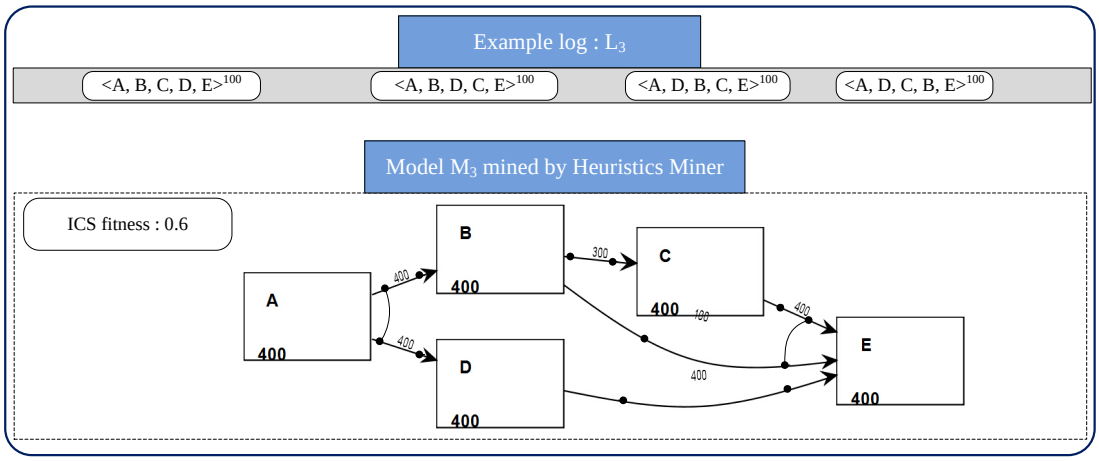


Figure 3.6.: The example event log L_3 and its relevant process model minded by HM.

Table 3.2.: The activity ranking result for the example event log L_3.

Ranking	Activity	BAW	Frequency of Occurrence	ARW
1	B	**6**	**400**	1.0
2	D	6	400	1.0
3	C	5	400	0.8333
4	E	3	400	0.5
5	A	2	400	0.3333

In this preliminary test for HIF on the example log, the target fitness α is set to 1, the model fitness improvement threshold β is set to 0[3] and the threshold for the number of newly added activities μ is set to 1.0 (we utilise such a combination of parameters because the example event log is very simple). Table 3.2 shows the activity ranking result from HIF for log L_3. Figure 3.7 exhibits the event log and process model output by HIF executed on L_3. For event log L_3 HIF generates a new log L_4 by converting activity D under EI (A,C) into D0 and converting activity B under EI (C,E) into B0 in L_3. The process model M_4 mined from L_4 has a fitness value 1.0.

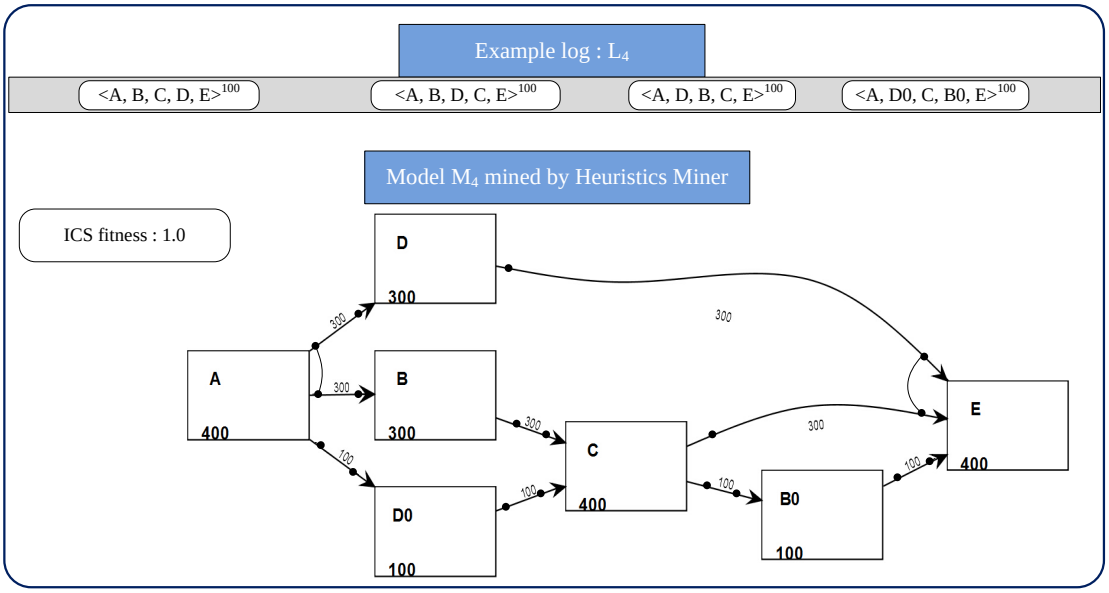


Figure 3.7.: The process model output by HIF for the example log L_3.

In our experiment, we found that HIF is usually able to discover the most efficient way for acquiring high fitness for the mined process models. Normally, there exist multiple options for improving the fitness of the underlying process model for a certain event log. Semantically speaking, the most efficient way might not always be the best way sometimes. In Chapter 6, we will display more

[3]The sign "≤" in the rhombus of Module-2 for technique DCIB (shown in Figure 3.5) should be transformed into "<" if the model fitness improvement threshold β is set to 0.

detailed experiment results and analyses about HIF in which the problem mentioned above is also discussed in details. In the meantime, a comparison between our method and some classical BPMD techniques proposed in the literature is carried out in Chapter 6 as well.

4

Trace Clustering and Classification Techniques

4.1. Introduction and Motivation

As introduced in Chapter 3, the current BPMD techniques may generate inaccurate process models while mining real-life event logs. The technique HIF proposed in Chapter 3 is able to help solve this problem efficiently. However, complex rate is also a significant metric for evaluating the quality of a process model which should not be neglected [179]. While dealing with event logs that record the execution information of complex business processes, the relevant models mined through HIF may not be helpful, even though these generated models are highly fitting. Because a mined process model that is too complex to be comprehended is almost useless in reality (no matter how accurate this model is). All too often, though, "spaghetti-like" business process models might be generated by existing BPMD techniques with an input of real-life event log [187]. Such models are often both inaccurate and too complex to be well interpreted. For instance, Figure 4.1 shows a "spaghetti-like" process model with a fitness (ICS fitness) value 0.7878 obtained by mining the event log of the loan and overdraft approvals process (LOA) from BPIC 2012 with HM.

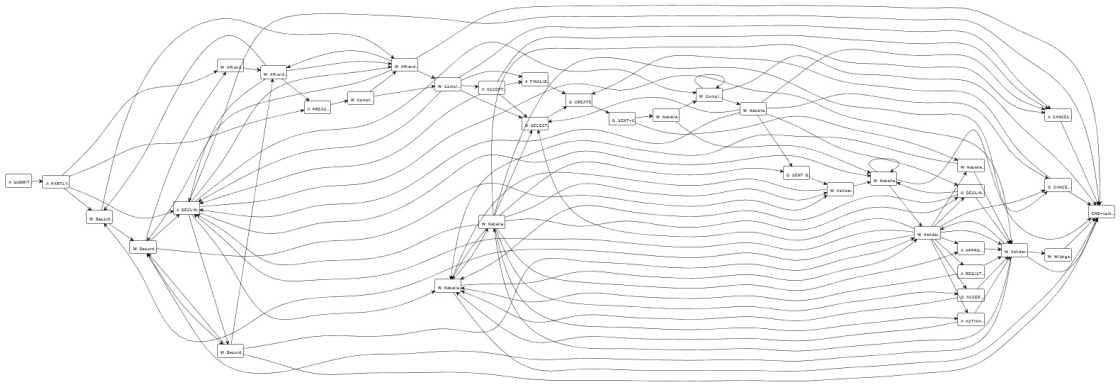


Figure 4.1.: Business process model of the loan and overdraft approvals process.

Accordingly, some pioneering approaches have been developed to solve the problem of inaccurate and complex process models mined from real-life event logs. One efficient technique is trace clustering [31] which is an application instance of MDS. Trace clustering approach tries to mine and analyse the structural behaviours of traces (trace behaviours) in an event log and then group the traces with similar behaviours into the same sublog. Afterwards, by applying BPMD algorithms on each simpler sublog, more accurate and comprehensible process models can be obtained. Figure 2.26 (Chapter 2) shows the basic procedure for trace clustering.

Nevertheless, most currently available trace clustering techniques treat all of the trace behaviours captured in the event log equally. As a result, the impacts of some important trace behaviours are reduced. Moreover, these techniques focus mainly on the discovery of various kinds of trace behaviours while the quality of the underlying process model for each sublog is not taken into account [36]. Hence, high-quality sub-process models from these trace clustering techniques cannot be guaranteed. A promising method called ATC is put forward in [36] which directly optimises the accuracy of each sublog's underlying process model. However, ATC only considers model accuracy metric while the complexity of process models is neglected during trace clustering. The complexity of process models is also a very important metric and should not be ignored for trace clustering. Because a highly accurate process model can still be very complicated.

Furthermore, most existing trace clustering techniques cannot deal with complex trace behaviours that are beyond the expressive ability of the utilised BPMD techniques from the similar traces (i.e., the traces that have similar sequence of activities) because these similar traces are more possible to be grouped into the same sublog. As a result, some sub-models output by these trace clustering techniques may still suffer from low fitness. Table 4.1 (T-number is a short name for trace number) exhibits the evaluation results for six classical trace clustering techniques which are 3-gram [31], MR and MRA [195], ATC [36], GED [35] and sequence clustering (SCT) [196] executed on event log LOA. For each technique the number of generated sublogs is set to five. It can be seen that all the six trace clustering techniques will generate one or more sub-process models with low fitness. For instance, for technique SCT, the fitness of the model mined from sublog 5 is only 0.7828, even though the Weighted Average Fitness[1] (WAF) of the models mined from the five sublogs generated by SCT is 0.8671.

In this chapter, we introduce two trace clustering techniques which are Top-Down Trace Clustering Technique (TDTC) and Compound Trace Clustering Technique (CTC). The two developed techniques are able to optimise the accuracy and

[1] Let j be the number of trace clusters, n_i denotes the number of traces in cluster i , where $1 \leq i \leq j$. Let F_i represents the fitness of the process model mined for cluster i, the weighted average fitness is defined as: $WAF = \frac{\sum_{i=1}^{j} n_i \times F_i}{\sum_{i=1}^{j} n_i}$.

Table 4.1.: The information about the sub-process models mined from the sublogs of LOA generated by six classical trace clustering techniques.

Method	Metrics	Model of sublog 1	Model of sublog 2	Model of sublog 3	Model of sublog 4	Model of sublog 5
3-gram	Fitness	0.9918	0.9681	0.879	-0.4672	0.2284
	T-number	118	763	10322	594	1290
MR	Fitness	0.9909	0.9669	0.8681	0.0803	-0.3583
	T-number	112	938	10768	454	815
MRA	Fitness	0.8663	0.204	0.979	-0.3643	0.9677
	T-number	10727	1083	67	372	838
ATC	Fitness	1	-0.1515	-0.1034	-0.04	0.4566
	T-number	7512	27	25	19	5504
GED	Fitness	0.9718	0.9959	0.8049	0.5193	0.6197
	T-number	1509	1607	8073	784	1114
SCT	Fitness	0.9095	0.8436	0.9636	0.932	0.7828
	T-number	2091	1839	1740	2765	4652

complexity of the potential sub-process models for the generated sublogs. Moreover, the technique CTC is able to assure the fitness of the sub-model mined from each sublog output by it.

Trace clustering can help find a lot of hidden behaviours among the traces. However, it is an unsupervised learning technique and lack domain knowledge. As a result, it is unable to indicate which behaviours are wanted by customers or process analysts for dividing the traces. Sometimes, the trace behaviours found by trace clustering techniques may not assist in generating a meaningful division of traces (by considering the demand of end users).

Classification (supervised learning technique) which is able to combine the domain knowledge from enterprise business experts can be a useful tool for dividing the traces in a meaningful way. In this chapter, we also put forward a classification technique-based method for classifying cases[2]. The structure of the main contents in this chapter is organised as:

- The trace clustering technique TDTC is proposed in Section 4.2.

- In Section 4.3, we put forward the technique CTC which inherits the basic ideas of TDTC and the mined model fitness improvement technique HIF (introduced in Chapter 3).

[2]Trace is a special attribute of case. In this chapter, we utilise the term "case classification" instead of "trace classification" because the other case attributes may also be important factors for classifying the traces. The proposed classification technique in this chapter also considers these extra case attributes except for trace.

- In Section 4.4, we first demonstrate and formalise the problem of multi-label case classification. Then, we develop a systematic method Multi-Label Case Classification (MLCC) based on sequential pattern mining technique which is able to utilise the case attribute trace for classifying cases.

- In Section 4.5, we carry out preliminary evaluations for TDTC, CTC and the proposed multi-label case classification technique MLCC.

4.2. A Novel Top-Down Trace Clustering Technique

In this section, the trace clustering problem is surveyed from a new perspective and redefined as an issue of searching for a global optimal solution in a solution space. The proposed technique employs a greedy strategy for searching for the optimal way to cluster the traces in an event log based on a specific model evaluation mechanism that considers both the accuracy and complexity of the potential sub-process models during the run time. In Section 4.2.1, the basic idea of TDTC is depicted. In Section 4.2.2, the details about TDTC are elaborated. In Section 4.2.3, the assumptions for TDTC are presented.

4.2.1. Outline for Technique TDTC

Under certain conditions, an inaccurate and complex business process can be divided into several simpler and more accurate sub-processes where each sub-process performs some unique functions reflected by certain specific sub-process constructional behaviours. These behaviours can be recorded in event log after the execution of the sub-process and expressed through the structural behaviours of traces (trace behaviours). In this subsection, the trace behaviours that adhere to a more accurate and simpler sub-process model compared with the original model (generated by using the original event log) are called significant behaviours (defined in Section 4.2.2). Discovering these significant trace behaviours from event log will assist in mining better sub-process models by clustering the traces based on these behaviours. However, due to the lack of domain knowledge about the significant trace behaviours, capturing them directly from event log seems to be a difficult task.

In this subsection, we transform the traditional trace clustering problem into the problem of finding the optimal way for clustering the traces among all possible solutions. As shown in Figure 4.2, each element in the solution space represents one strategy for clustering the traces from an event log into several subsets of traces. A best solution is defined as a solution which is able to divide the

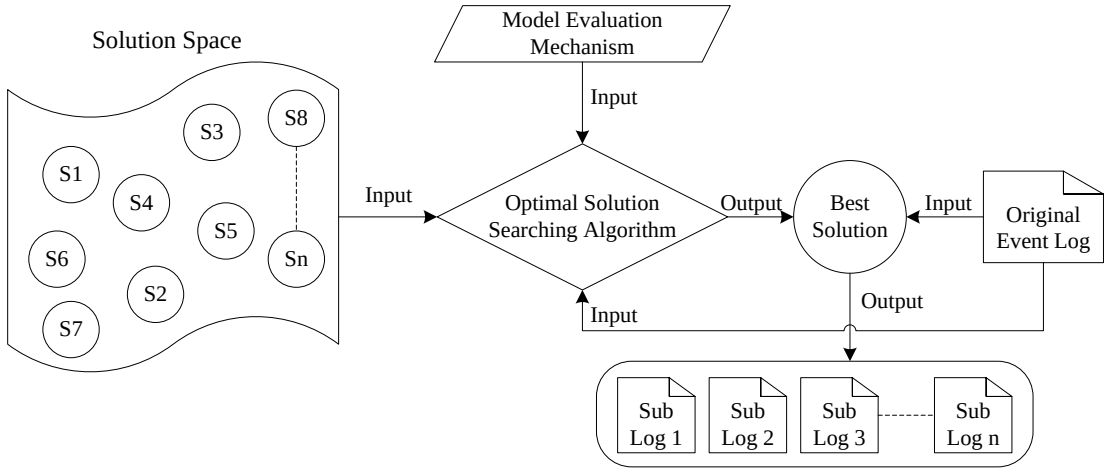


Figure 4.2.: Illustration of the basic idea of the proposed technique TDTC.

traces in the original event log into several sublogs where the overall quality of the underlying sub-models for these sublogs is optimal. Given a process model evaluation mechanism, how to find the optimal solution for clustering the traces from an event log is the main problem we are going to solve.

In this subsection, we propose the technique TDTC which inherits the basic ideas of traditional trace clustering techniques and ATC for discovering the optimal way of clustering traces. This technique considers both the behaviours of traces and the accuracy and complexity of each potential sub-process model during the mining procedure for the optimal solution.

4.2.2. Approach Design

In this subsection, we introduce the details of TDTC which differs from existing correlative techniques because it searches for an optimal way for clustering the traces among all of the possible solutions. Four kinds of trace behaviours defined in this subsection provide a basis for this technique to carry out the searching process.

4.2.2.1. Concepts Related to Trace Behaviours

Traces are generated performing a specific category of functions determined by business process-based domain criterion. Such criteria can be very diverse, e.g., presence or absence of activities, presence or absence of combinations of activities [36]. These underlying criteria are recorded in event log and reflected by cer-

tain compositional behaviours of traces. In the first step, our technique searches for the structural behaviours of traces in an event log. Then, according to the identified trace behaviours the optimal solution searching process is carried out.

Given an event log L, the set of trace behaviours TB mined from L is defined as:

Definition 4.1. *(Set of Trace Behaviours)*
Let $\widehat{\Gamma}$ be a closed sequential pattern mining algorithm and min_sup be a minimum support, the set of trace behaviours TB for L is $TB = \{tb | tb \in \widehat{\Gamma}(L, min\_sup)\}$.

According to Definition 4.1, a trace behaviour tb is equivalent to a frequent pattern mined from event log L. In our opinion, certain frequently appeared subsequences among traces in an event log are able to reveal some particularly important criteria of business processes and can help distinguish sub-process models with different functions hidden in the event log. Another benefit of utilising sequential patterns is that they cannot only represent consecutive structural behaviours of traces, but inconsecutive trace behaviours as well. For instance, given a simple event log $L_s = \{<a_1, b_1, c_1, d_1>^1, <a_1, b_1>^1, <a_1, d_1>^1\}$ and a minimum support $min\_sup = 0.4$, the set of trace behaviours $TB_{L_s} = \{< a_1 >, < a_1, d_1 >, < a_1, b_1 >\}$ can be acquired, where the sequential pattern $< a_1, b_1 >$ is a consecutive trace behaviour because activity b_1 always occur right next to a_1 and $< a_1, d_1 >$ is an inconsecutive trace behaviour because activity a_1 and d_1 might occur in a trace discretely. However, most existing pattern-based trace clustering techniques are only able to capture consecutive trace behaviours in an event log. Moreover, employing frequent patterns is also in accordance with the main idea of most advanced process discovery techniques: only the frequent structures should be considered in the process mining procedure [37].

Additionally, we classify the behaviours of traces from a real-life event log into significant behaviours and insignificant behaviours. Let Ω be a BPMD algorithm and Σ be a process model evaluation mechanism. Let L be an event log, tb be a trace behaviour discovered from L, $L_1 \subseteq L$ be a sublog of L which consists of all the traces with a subsequence tb, $L_2 \subseteq L$ be a sublog of L which contains all the traces without a subsequence tb, $V_L = \Sigma(\Omega(L))$, $V_{L_1} = \Sigma(\Omega(L_1))$ and $V_{L_2} = \Sigma(\Omega(L_2))$ be the assessed values obtained by performing the process model evaluation mechanism Σ on the process models for L, L_1 and L_2. The significant behaviour is conveyed by the following definition:

Definition 4.2. *(Significant Trace Behaviour)*
For a given minimum threshold μ, the trace behaviour $tb \in TB$ is called Significant Trace Behaviour (STB) if $((V_{L_1} + V_{L_2})/2 - V_L)/V_L \geqslant \mu$, otherwise tb is called an insignificant behaviour.

As stated in Definition 4.2, a STB is able to divide the original set of traces into two subsets that lead to two process models of which the average quality should

be increased by at least μ (a minimum threshold) compared with the quality of the model generated by utilising the original set of traces.

According to Definition 4.2, the starting point for identifying a STB is a process model evaluation mechanism Σ. As mentioned in Section 4.1, while evaluating a process model both the accuracy and complexity should be taken into account. Accordingly, the model evaluation mechanism Σ should contain two parts: the fitness computing mechanism Σ_f and the complexity evaluation mechanism Σ_c. Let L be an event log, a trace behaviour tb discovered from L separates L into two sublogs L_1 and L_2, the sub-model improvement SMI is defined as:

$$SMI(L_1, L_2, L) = \alpha \times SMI_F(L_1, L_2, L) + \beta \times SMI_C(L_1, L_2, L) \tag{4.1}$$

$$SMI_F(L_1, L_2, L) = \frac{\frac{1}{2}(\Sigma_f(\Omega(L_1)) + \Sigma_f(\Omega(L_2))) - \Sigma_f(\Omega(L))}{\Sigma_f(\Omega(L))} \tag{4.2}$$

$$SMI_C(L_1, L_2, L) = \frac{\Sigma_c(\Omega(L)) - \frac{1}{2}(\Sigma_c(\Omega(L_1)) + \Sigma_c(\Omega(L_2)))}{\Sigma_c(\Omega(L))} \tag{4.3}$$

According to Equation 4.1, the SMI is composed by two parts. The first part is related to the model accuracy and the second part is related to the model complexity. Our technique utilises the ICS fitness [29] and PT-CD [36] for the evaluation of process models. The main reason for using ICS fitness is that it has a computationally efficient calculative process. In Equation 4.1, α and β represent the weights for the two parts and meet the condition of $\alpha+\beta = 1$. The values of α and β should be set upon the conditions of accuracy and complexity of the original model. For instance, if the original model has a good accuracy but suffers from a bad complexity then the value of β should be set higher than α and vice versa. According to Definition 4.2, given a minimum threshold μ, the trace behaviour tb is a STB if $SMI(L_1, L_2, L) \geq \mu$.

The sub-model improvement criterion SMI considers both the fitness and complexity of the process models at the same time. However, in reality the fitness and complexity of a model are not associated with each other. The increment of fitness is not always accompanied by a decrement of model complexity and vice versa. For example, let tb be a trace behaviour from event log L which divides L into L_1 and L_2, pretend that $\mu = 0.15$, $\alpha = 0.5$, $\beta = 0.5$, $SMI_F(L_1, L_2, L) = -0.1$, $SMI_C(L_1, L_2, L) = 0.4$, according to Equation 4.1 and Definition 4.2, $SMI(L_1, L_2, L) = 0.15$ is equal to the value of μ so tb is judged to be a STB. Even though the average fitness of the sub-models for L_1 and L_2 is decreased, the value of $SMI(L_1, L_2, L)$ augments because the average complexity of the sub-models is greatly reduced.

To avoid this situation, a stricter definition for STB needs to be developed. Let stb be a STB mined from a log L which divides L into L_1 and L_2, the strict significant trace behaviour is defined as follows:

Definition 4.3. *(Strict Significant Trace Behaviour)*
The stb is called a Strict Significant Trace Behaviour (SSTB) if $SMI_F(L_1, L_2, L) \geq \mu_f$ and $SMI_C(L_1, L_2, L) \geq \mu_c$, where μ_f is a minimum threshold for the average fitness increment of the models for L_1 and L_2 compared with the original model and μ_c is a minimum threshold for the average complexity decrement of the models for L_1 and L_2.

Based on Definition 4.3, a SSTB satisfies all the conditions for STB, in the meantime some additional conditions should be fulfilled: both the average fitness and average complexity of the related sub-models need to be improved to a certain extent.

Given a minimum threshold φ_f and a maximum threshold φ_c:

Definition 4.4. *(Fitness-Based Conditional Strict STB)*
The trace behaviour tb is called a Fitness-Based Conditional Strict Significant Trace Behaviour (FCSTB) if $(\Sigma_f(\Omega(L_1)) + \Sigma_f(\Omega(L_2)))/2 \geq \varphi_f$, $(SMI_C(L_1, L_2, L) \geq \mu_c \vee (\Sigma_c(\Omega(L_1)) + \Sigma_c(\Omega(L_2)))/2 \leq \varphi_c)$ and $SMI(L_1, L_2, L) \geq \mu$.

Definition 4.5. *(Complexity-Based Conditional Strict STB)*
The trace behaviour tb is called a Complexity-Based Conditional Strict Significant Trace Behaviour (CCSTB) if $(\Sigma_c(\Omega(L_1)) + \Sigma_c(\Omega(L_2)))/2 \leq \varphi_c$, $(SMI_F(L_1, L_2, L) \geq \mu_f \vee (\Sigma_f(\Omega(L_1)) + \Sigma_f(\Omega(L_2)))/2 \geq \varphi_f)$ and $SMI(L_1, L_2, L) \geq \mu$.

The FCSTB is defined to deal with an event log of which the potential model has a high fitness but an inferior complexity. For instance, let tb be a trace behaviour from the event log L which divides L into L_1 and L_2, pretend that $\mu = 0.15$, $\alpha = 0.5$, $\beta = 0.5$, $SMI_F(L_1, L_2, L) = -0.1$, $SMI_C(L_1, L_2, L) = 0.4$, $\varphi_f = 0.9$, $(\Sigma_f(\Omega(L_1)) + \Sigma_f(\Omega(L_2)))/2 = 0.93$, according to Definition 4.2 and Definition 4.3, tb is a STB but not a SSTB. However, even though the average fitness of the sub-models decreases compared to the original model, it still remains a large value and greater than φ_f. In such a situation, the effect of tb should not be neglected. A corresponding definition to FCSTB is CCSTB which is defined in Definition 4.5. It should also be noticed that a trace behaviour can be both the FCSTB and the CCSTB at the same time.

4.2.2.2. A Top-Down Algorithm for Clustering Traces (TDTC)

In this subsection, an algorithm named TDTC is put forward for finding the optimal way to cluster the traces in an event log based on the definitions elaborated

in Section 4.2.2.1. This algorithm applies a greedy strategy which discovers the best trace behaviour (that is either a SSTB or a FCSTB or a CCSTB) for splitting the original event log for each stage according to the value of SMI. Before introducing TDTC, we first introduce a trace behaviour type judging algorithm $\widehat{\Phi}$, an unqualified trace behaviour removing algorithm Π and a best trace behaviour discovering algorithm Φ.

Let $\widehat{\Phi}$ be an algorithm (depicted in Algorithm 4.1) that helps judge whether a given trace behaviour tb mined from log L is either a SSTB or a FCSTB or a CCSTB. Let L_1 and L_2 be two sublogs generated by splitting L with tb. According to Algorithm 4.1, $\widehat{\Phi}$ first calculates the elements for judging SSTB, FCSTB and CCSTB according to Definition 4.3, 4.4 and 4.5 (steps 5–9). Then, if tb is judged to be a SSTB or a FCSTB or a CCSTB, the variable j with a value $true$ will be returned by $\widehat{\Phi}$ (steps 10–19). Or the variable j with a value $false$ is returned by $\widehat{\Phi}$.

Algorithm 4.1 Judge the type of a specific trace behaviour ($\widehat{\Phi}$)

Input: an event log L, sublogs L_1 and L_2 of L, the fitness weight α and complexity weight β for SMI, the minimum threshold μ for STB, the minimum thresholds μ_f and μ_c for SSTB, the minimum threshold φ_f for FCSTB, the maximum threshold φ_c for CCSTB.

1: **Let** smi, smi_f, smi_c, i_1 and i_2 be five variables of type Double.
2: **Let** j be a variable of type Boolean.
3: $smi \leftarrow 0$, $smi_f \leftarrow 0$, $smi_c \leftarrow 0$, $i_1 \leftarrow 0$, $i_2 \leftarrow 0$
4: $j \leftarrow false$
5: $smi_f \leftarrow ((\Sigma_f(\Omega(L_1)) + \Sigma_f(\Omega(L_2)))/2 - \Sigma_f(\Omega(L)))/\Sigma_f(\Omega(L)))$
6: $smi_c \leftarrow (\Sigma_c(\Omega(L)) - (\Sigma_c(\Omega(L_1)) + \Sigma_c(\Omega(L_2)))/2)/\Sigma_c(\Omega(L))$
7: $smi \leftarrow \alpha \times smi_f + \beta \times smi_c$
8: $i_1 \leftarrow (\Sigma_f(\Omega(L_1)) + \Sigma_f(\Omega(L_2)))/2$
9: $i_2 \leftarrow (\Sigma_c(\Omega(L_1)) + \Sigma_c(\Omega(L_2)))/2$
10: **if** $smi \geq \mu$ && $smi_f \geq \mu_f$ && $smi_c \geq \mu_c$ **then**
11: $j \leftarrow true$
12: $return\ \ j$
13: **else if** $i_1 \geq \varphi_f$ && $(smi_c \geq \mu_c \,||\, i_2 \leq \varphi_c)$ && $smi \geq \mu$ **then**
14: $j \leftarrow true$
15: $return\ \ j$
16: **else if** $i_2 \leq \varphi_c$ && $(smi_f \geq \mu_f \,||\, i_1 \geq \varphi_f)$ && $smi \geq \mu$ **then**
17: $j \leftarrow true$
18: $return\ \ j$
19: **end if**
20: $return\ j$

Output: a variable j of type Boolean

Let Π be an unqualified trace behaviour removing algorithm that is described in Algorithm 4.2. Let TB represent a set of trace behaviours mined from event log

L, a trace behaviour $tb \in TB$ is able to divide L into two sublogs: L_1 (contains the traces with a subsequence tb) and L_2 (contains the traces without a subsequence tb) (steps 5–9). If $|L_1| < \theta$ or $|L_2| < \theta$ then tb is judged to be an unqualified trace behaviour and removed from TB by algorithm Π (steps 11–14). In our technique, θ stands for a minimum number of traces for each sublog. A trace behaviour that leads to a sublog with a number of traces less than θ will not be considered.

Algorithm 4.2 Unqualified trace behaviours removing method (Π)

Input: an event log L, the set of trace behaviour TB mined from L, the minimum size θ for each sublog.

1: **Let** L_1 and L_2 be two sublogs.
2: $L_1 \leftarrow null$, $L_2 \leftarrow null$
3: **for** each trace behaviour $tb \in TB$ **do**
4: **for** each trace $t \in L$ **do**
5: **if** $tb \sqsubseteq t$ **then**
6: $L_1 = L_1 \cup \{t\}$
7: **else**
8: $L_2 = L_2 \cup \{t\}$
9: **end if**
10: **end for**
11: **if** $|L_1| < \theta \,||\, |L_2| < \theta$ **then**
12: *remove tb from TB*
13: $L_1 \leftarrow null$
14: $L_2 \leftarrow null$
15: **end if**
16: **end for**

Output: a set of trace behaviour TB which does not contain unqualified behaviours

Let Φ be an algorithm which searches for the best trace behaviour for dividing a given event log into two sublogs, where the SMI stemming from such a division is maximal (Algorithm 4.3). According to Algorithm 4.3, for a trace behaviour $tb \in TB$, Φ first generates two sublogs L_1 and L_2 (steps 6–12) where L_1 contains all the traces with tb as a subsequence from L and L_2 consists of all the traces without tb as a subsequence. Then, Φ checks if tb is a type of the three trace behaviours: SSTB, FCSTB and CCSTB (step 13). If it is not, tb is not further considered and the algorithm Φ continues to deal with the next trace behaviour. The main reason to employ the SSTB, FCSTB and CCSTB is: if the average quality of the sub-process models mined from the generated sublogs (L_1 and L_2) cannot be increased to a certain extent based on the division process compared with the quality of the model mined from the original event log (L), then it is not worth making the division (this requirement derives from the consideration for the balance between the integrity and the quality of the process model). If tb is judged to belong to one of the three types of trace behaviours (i.e., SSTB, FCSTB and CCSTB), the value of SMI relevant to tb is then compared with the maximal SMI found by Φ by

checking the former trace behaviours (step 14). If the value of SMI related to tb is larger than the present maximal SMI (stored in $p[smi]$), the value of $p[smi]$ is updated to the value of SMI for tb, the sublogs L_1 and L_2 are stored in $p[log_1]$ and $p[log_2]$ respectively, the trace behaviour tb is stored in $p[trace\_behavior]$ (steps 15–18). Finally, Φ outputs an array p which contains the information related to the best trace behaviour found (if no trace behaviour is discovered for splitting log L, then Φ outputs the value $null$).

Algorithm 4.3 Search for the best trace behaviour (Φ)

Input: a set of trace behaviours TB, an event log L, the fitness weight α and complexity weight β for SMI, the minimum threshold μ for STB, the minimum thresholds μ_f and μ_c for SSTB, the minimum threshold φ_f for FCSTB, the maximum threshold φ_c for CCSTB.

1: **Let** L_1, L_2 be two sublogs.
2: **Let** p be an array and $length(p) = 4$.
3: L_1, $L_2 \leftarrow null$
4: $p[trace\_behaviour] \leftarrow null$; $p[smi] \leftarrow -\infty$; $p[sublog_1]$, $p[sublog_2] \leftarrow null$
5: **for** each trace behaviour $tb \in TB$ **do**
6: **for** each trace $t \in L$ **do**
7: **if** $tb \sqsubseteq t$ **then**
8: $L_1 = L_1 \cup \{t\}$ # tb is a subsequence of t
9: **else**
10: $L_2 = L_2 \cup \{t\}$ # tb is not a subsequence of t
11: **end if**
12: **end for**
13: **if** $\widehat{\Phi}(L_1, L_2, L, \alpha, \beta, \mu, \mu_f, \mu_c, \varphi_f, \varphi_c)$ **then**
14: **if** $p[smi] < SMI(L_1, L_2, L, \alpha, \beta)$ **then**
15: $p[smi] \leftarrow SMI(L_1, L_2, L, \alpha, \beta)$
16: $p[sublog_1] \leftarrow L_1$
17: $p[sublog_2] \leftarrow L_2$
18: $p[trace\_behaviour] \leftarrow tb$
19: **end if**
20: **end if**
21: **end for**

Output: an array p which contains the information related to the found trace behaviour tb

Given a workflow discovery algorithm Ω, a closed sequential pattern mining algorithm $\widehat{\Gamma}$, a process model fitness evaluation mechanism Σ_f and a process model complexity evaluation mechanism Σ_c, the details of our method TDTC is described in Algorithm 4.4.

To prevent the tendency of our technique to generate the sublogs containing too few traces (too few traces means a very simple model), a minimum size θ of each potential sublog is requested to be set before starting the algorithm. Steps

Algorithm 4.4 A top-down trace clustering technique (TDTC)

Input: an event log L, the set of closed sequential patterns $CSP \leftarrow \widehat{\Gamma}(L, min\_sup)$ mined from L with a minimum threshold min_sup, the fitness weight α and complexity weight β for SMI, the minimum threshold μ for STB, the minimum thresholds μ_f and μ_c for SSTB, the minimum threshold φ_f for FCSTB, the maximum threshold φ_c for CCSTB, the minimum size θ for each cluster.

1: **Let** N, N_{left} and N_{right} be the nodes for a binary tree.
2: **Let** TB be a set of trace behaviours.
3: **Let** l be an array and $length(l) = 4$.
4: $N \leftarrow null$ # create a node N
5: $N_{left} \leftarrow null$ # N_{left} is the left child node of N
6: $N_{right} \leftarrow null$ # N_{right} is the right child node of N
7: $l \leftarrow null$
8: **if** $|L| \geq 2\theta$ **then**
9: $TB = TB \cup \Pi(CSP, L, \theta)$
10: **else**
11: return $N \leftarrow (L, \Omega(L), \Sigma_f(\Omega(L)), \Sigma_c(\Omega(L))$
12: **end if**
13: $l \leftarrow \Phi(TB, L, \alpha, \beta, \mu, \mu_f, \mu_c, \varphi_f, \varphi_c)$
14: **if** $l[trace\_behaviour] == Null$ **then**
15: return $N \leftarrow (L, \Omega(L), \Sigma_f(\Omega(L)), \Sigma_c(\Omega(L)))$
16: **else**
17: $N_{left} \leftarrow TDTC(l[sublog_1], CSP, \alpha, \beta, \mu, \mu_f, \mu_c, \varphi_f, \varphi_c, \theta)$
18: $N_{right} \leftarrow TDTC(l[sublog_2], CSP, \alpha, \beta, \mu, \mu_f, \mu_c, \varphi_f, \varphi_c, \theta)$
19: **end if**
20: return $N \leftarrow (L, \Omega(L), \Sigma_f(\Omega(L)), \Sigma_c(\Omega(L)))$

Output: a binary tree bt with a root node N

8–12 in Algorithm 4.4 check the number of traces in the original event log and if there is no way to divide the log so that the sizes of both the sublogs generated are larger than or equal to θ then the algorithm stops. Afterwards, the trace behaviours discovered are filtered and all the trace behaviours that cannot lead to a valid division of the original event log according to the minimum size rule are removed (step 9). Step 13 searches for the best trace behaviour for dividing log L among all of the behaviours found in step 9 through the algorithm Φ depicted in Algorithm 4.3. A best trace behaviour is defined as a behaviour (either a SSTB or a FCSTB or a CCSTB) which can help generate a maximum sub-model improvement SMI as shown in the steps 13–20 in Algorithm 4.3. As described in Algorithm 4.4, TDTC takes a greedy strategy for clustering the traces step by step, the same procedure continues on the sublogs generated by the present stage (steps 14–19). Finally, a binary tree bt is output by TDTC where each leaf node in bt represents a found sublog.

4.2.3. Assumptions

In this section, we assume that the inaccurate and complex business process subjected to our method is able to be divided into several simpler and more accurate sub-processes where each sub-process carries out some specific functions. These functions are identified by certain behaviours of traces recorded in the event log.

4.3. A Compound Trace Clustering Technique

In our opinion, traditional trace clustering techniques suffer from a common downside: most of them cannot deal with complex trace behaviours that are beyond the expressive ability of the utilised BPMD techniques deriving from similar traces (i.e., the traces that have similar sequence of events). Because the similar traces are more possible to be grouped into the same sublog by traditional trace clustering techniques. In this section, we put forward a new type of trace clustering technique named CTC which considers the accuracy and complexity of the underlying sub-process models separately during the clustering procedure. This means that CTC assures optimised accuracy and complexity for the sub-process model for each sublog generated by it. In addition, CTC overcomes the restrictions led by the complex trace behaviours from similar traces while optimising the accuracy of the underlying sub-process models.

Basically, the proposed technique CTC consists of two stages (as shown in Figure 4.3). Stage 1 which builds on the basic idea of TDTC (presented in Section 4.2) focuses on mining sub-process models with optimised average complex rate. Stage 2 then improves the accuracy of inaccurate sub-models stemming from stage 1 by employing the technique HIF developed in Chapter 3. In Section 4.3.1, a Complexity-Related Top-Down Trace Clustering Technique (C-TDTC) for realising stage 1 in Figure 4.3 is put forward. In Section 4.3.2, the mined model fitness improvement technique HIF that is developed in Chapter 3 is reviewed. In Section 4.3.3, the details about CTC are depicted.

4.3.1. A Complexity-Related Top-Down Trace Clustering Approach

To realise the first stage depicted in Figure 4.3, we have developed a new top-down trace clustering method (C-TDTC) in this subsection which is able to solve the problem of generating sub-process models with optimised average complex

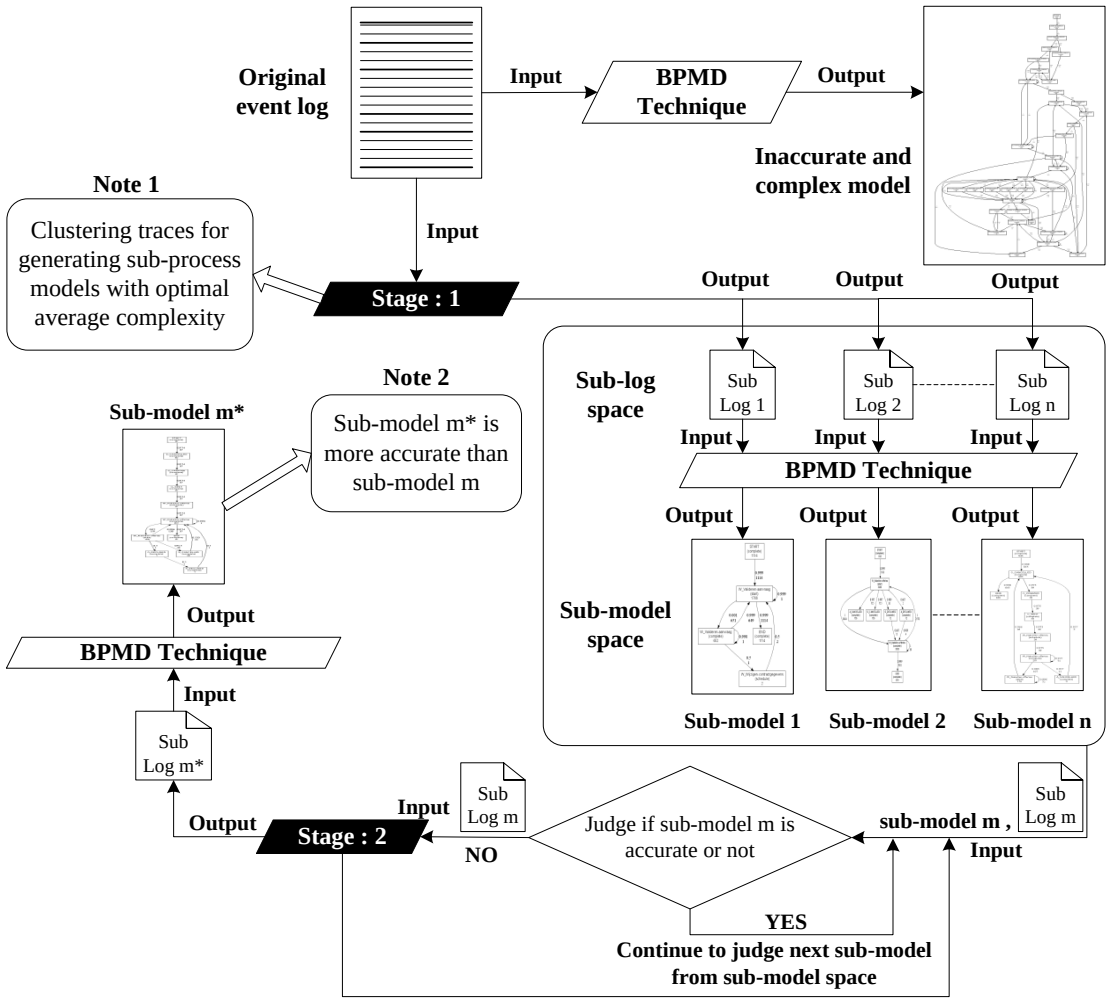


Figure 4.3.: Illustration of the basic idea for the proposed compound trace clustering technique.

rate. The proposed technique C-TDTC adopts a novel strategy which converts the trace clustering problem mentioned above into the problem of discovering the optimal way for clustering the traces among all possible solutions.

Let $\widehat{S} = \{\widehat{s}_1, \widehat{s}_2, \ldots, \widehat{s}_n\}$ be a solution space and each solution $\widehat{s}_m \in \widehat{S}$ stands for a unique way to divide the original event log into a fixed number of sublogs. C-TDTC employs a greedy strategy for searching for the optimal solution $\widehat{s}_{op}$ from $\widehat{S}$ for splitting the given event log, where the weighted average complexity of the potential sub-process models for the generated sublogs should be optimal. As shown in Figure 4.4, for a log L and a target number (three in this example) of sublogs, C-TDTC first searches for the best way to divide L into two sublogs L_1 and L_2. Then, C-TDTC continues to detect the best way to split L_2 (which is assumed to lead to a sub-model with highest complexity) into L_3 and L_4. The optimal solution searching procedure of C-TDTC is based on the following concepts that are relevant to trace behaviours.

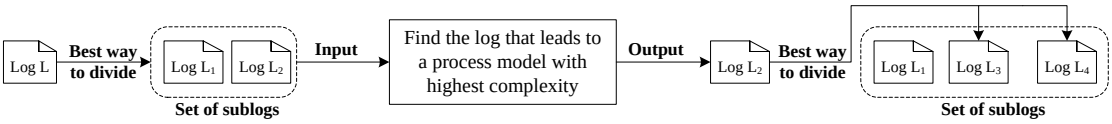


Figure 4.4.: Illustration of the basic idea for technique C-TDTC.

Complexity-Related Significant Trace Behaviours

As demonstrated in Section 4.2, a complex business process can often be divided into several simpler sub-processes where each sub-process performs unique functions reflected by certain specific constructional behaviours of the sub-process. These behaviours will be recorded in the relevant event log in the form of trace behaviours after the implementation of the sub-process. In this subsection, the trace behaviours that adhere to a simpler sub-process model compared with the original model (obtained by mining the raw event log) are called Complexity-Related Significant Behaviours (CRSB). Detecting such significant behaviours from real-life event logs for clustering traces can help mine sub-process models with low complexity.

In this subsection, we still utilise the same definition about trace behaviour as defined in Definition 4.1. Furthermore, we classify the trace behaviours from a real-life event log into CRSB and Complexity-Related Insignificant Behaviours (CRIB). Let L be a simple event log, tb be a trace behaviour from the set of trace behaviours TB derived from L, $L_1 \subseteq L$ be a sublog of L which contains all the traces with a subsequence tb from L, $L_2 \subseteq L$ be a sublog of L which consists of all the traces from L without tb as a subsequence, $m_1 = |L_1|$ and $m_2 = |L_2|$ be the total number of traces in L_1 and L_2 respectively. Let $v_L = \Sigma_c(\Omega(L))$, $v_{L_1} = \Sigma_c(\Omega(L_1))$ and $v_{L_2} = \Sigma_c(\Omega(L_2))$ be three assessed values generated by implementing the process model complexity evaluation mechanism Σ_c on the process models for L, L_1 and L_2. The Average Sub-Model Complexity Reduction Value (ASCRV), the CRSB and CRIB are conveyed by the following definitions:

$$ASCRV = \frac{v_L - \frac{m_1 \cdot v_{L_1} + m_2 \cdot v_{L_2}}{m_1 + m_2}}{v_L}. \tag{4.4}$$

Definition 4.6. *(CRSB and CRIB)*
A trace behaviour $tb \in TB$ is called a complexity-related significant behaviour (CRSB) if $ASCRV > 0$, otherwise tb is called a complexity-related insignificant behaviour (CRIB).

According to Definition 4.6, a trace behaviour tb is judged to be a CRSB if it is able to divide the original event log L into two sublogs where the weighted

average complexity of the sub-models mined from the generated sublogs can be decreased (compared with the complexity of the original process model).

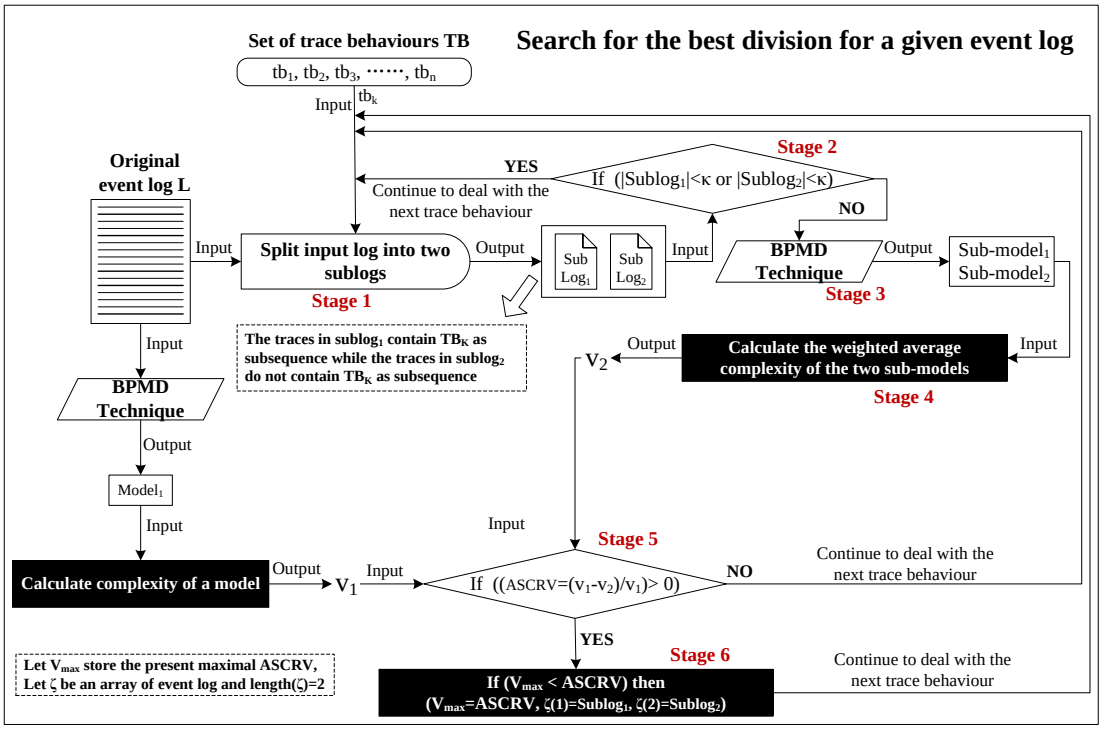


Figure 4.5.: Illustration of the basic idea for searching for the best CRSB.

The Technique C-TDTC

Before introducing C-TDTC, we first introduce an algorithm $\widehat{\Upsilon}$ (described in Figure 4.5 and Algorithm 4.5) which is able to search for the best CRSB from a set of trace behaviours for splitting a given event log into two sublogs where the weighted average complexity reduction of the sub-models mined from the sublogs is maximal. According to Algorithm 4.5 and Figure 4.5, given an event log L and a set of trace behaviours TB mined from L, for each trace behaviour $tb_k \in TB$, $\widehat{\Upsilon}$ first creates two sublogs $sublog_1$ and $sublog_2$ (steps 7–13 of Algorithm 4.5 and stage 1 of Figure 4.5) where $sublog_1$ contains all the traces with tb_k as a subsequence from L and $sublog_2$ consists of all the traces without tb_k as a subsequence. Then, the number of traces in $sublog_1$ and $sublog_2$ is examined (steps 14–17 of Algorithm 4.5 and stage 2 of Figure 4.5). The trace behaviour tb_k will not be considered and $\widehat{\Upsilon}$ continues to check the next trace behaviour from TB if the number of traces for either of the two sublogs is less than a minimum threshold κ. The input parameter κ is mainly utilised to prevent algorithm $\widehat{\Upsilon}$ from generating sublogs with too few traces (resulting in an overly simplified model). If tb_k passes the examination, $\widehat{\Upsilon}$ continues to check if tb_k is a CRSB according to Definition 4.6 (steps 18–20 of Algorithm 4.5 and stage 3, 4 and 5 of Figure 4.5). If it is not, then

Algorithm 4.5 Search for the best division for a given log ($\widehat{\Upsilon}$)

Input: an event log L, a set of trace behaviours TB, the minimum size κ for each generated sublog.

1: **Let** $sublog_1$, $sublog_2$ be two event logs.
2: **Let** $ASCRV$, v_L, v_1, v_2 and V_{max} be five variables of type Double.
3: **Let** ξ be an array of event log and $length(\xi) = 2$.
4: $sublog_1 \leftarrow null$, $sublog_2 \leftarrow null$, $\xi \leftarrow null$
5: $ASCRV \leftarrow 0$, $v_L \leftarrow 0$, $v_1 \leftarrow 0$, $v_2 \leftarrow 0$, $V_{max} \leftarrow -\infty$
6: **for** each trace behaviour $tb_k \in TB$ **do**
7: **for** each trace $t \in L$ **do**
8: **if** $tb_k \sqsubseteq t$ **then**
9: $sublog_1 \leftarrow sublog_1 \cup \{t\}$
10: **else**
11: $sublog_2 \leftarrow sublog_2 \cup \{t\}$
12: **end if**
13: **end for**
14: **if** $|sublog_1| < \kappa \parallel |sublog_2| < \kappa$ **then**
15: $sublog_1 \leftarrow null$, $sublog_2 \leftarrow null$
16: $continue$
17: **end if**
18: $v_1 \leftarrow \Sigma_c(\Omega(sublog_1))$, $v_2 \leftarrow \Sigma_c(\Omega(sublog_2))$, $v_L \leftarrow \Sigma_c(\Omega(L))$
19: $ASCRV \leftarrow (v_L - (|sublog_1| \cdot v_1 + |sublog_2| \cdot v_2)/(|sublog_1| + |sublog_2|))/v_L$
20: **if** $ASCRV > 0$ **then**
21: **if** $ASCRV > V_{max}$ **then**
22: $V_{max} \leftarrow ASCRV$, $\xi[0] \leftarrow sublog_1$, $\xi[1] \leftarrow sublog_2$
23: **end if**
24: **end if**
25: $sublog_1 \leftarrow null$, $sublog_2 \leftarrow null$
26: **end for**

Output: an array of event log ξ.

tb_k is not further considered by $\widehat{\Upsilon}$. The main reason to employ CRSB can be stated as follows: if the weighted average complexity of the sub-models mined from the generated sublogs (i.e., $sublog_1$ and $sublog_2$) cannot be decreased based on the division procedure compared to complexity of the original model (mined from log L), then it should not make the division. If tb_k is a CRSB, the ASCRV for tb_k is then compared with the present maximal ASCRV (stored in V_{max}) found by $\widehat{\Upsilon}$ through checking the former trace behaviours (steps 21–24 of Algorithm 4.5 and stage 6 of Figure 4.5). If $ASCRV > V_{max}$, the value of V_{max} is updated to the value of ASCRV and $sublog_1$ and $sublog_2$ are stored in array ξ which forms the output of $\widehat{\Upsilon}$.

Let Ψ be a log ranking function which gets an array of event logs as input and

creates an array of ranked logs as output. For an input array of log $\xi' = \{L_1', L_2', L_3'\}$ (i.e., $\xi'[0] = L_1'$, $\xi'[1] = L_2'$ and $\xi'[2] = L_3'$), Ψ first calculates the complexity of the model mined from each log in ξ'. Afterwards, the logs are ranked in ξ' by Ψ, where the log that leads to a model with higher complexity will be assigned a lower index in ξ'. For instance, presume that the model for L_3' has the highest complexity while the model for L_1' has the lowest complexity, then $\Psi(\xi') = \{L_3', L_2', L_1'\}$ (i.e., $\xi'[0] = L_3'$, $\xi'[1] = L_2'$ and $\xi'[2] = L_1'$).

Algorithm 4.6 C-TDTC

Input: an event log L, a minimum support min_sup for mining closed sequential patterns, the minimum size κ for each generated sublog, the target number of generated sublogs μ.

1: **Let** TB be a set of trace behaviours.
2: **Let** SL be an array of event logs.
3: **Let** m, i be two variables of type Integer.
4: **Let** ξ be an array of event logs and $length(\xi) = 2$.
5: $SL \leftarrow null$, $\xi \leftarrow null$, $TB \leftarrow null$
6: $m \leftarrow 0$, $i \leftarrow 0$
7: $TB \leftarrow \widehat{\Gamma}(L, min\_sup)$ # generate trace behaviours for log L
8: $SL[0] \leftarrow L$
9: **while** ($|SL| \leq \mu$) **do**
10: $m \leftarrow |SL|$
11: $\Psi(SL)$
12: **for** ($i \leftarrow 0; i < |SL|; i$++) **do**
13: $\xi \leftarrow \widehat{\Upsilon}(SL[i], TB, \kappa)$
14: **if** $\xi \neq null$ **then**
15: remove $SL[i]$ from SL
16: $SL[|SL|] \leftarrow \xi[0]$, $SL[|SL|+1] \leftarrow \xi[1]$
17: *break*
18: **end if**
19: **end for**
20: **if** $m == |SL|$ **then**
21: *break*
22: **end if**
23: **end while**

Output: an array of event log SL.

The details about C-TDTC is described in Algorithm 4.6. Basically, C-TDTC applies a greedy strategy which detects the best CRSB for splitting the raw event log for each stage by considering the ASCRV. Let $\widehat{\Gamma}$ be a closed sequential pattern mining algorithm. According to Algorithm 4.6, for an input event log L, C-TDTC first acquires the set of trace behaviours TB for L (step 7). Then, L is put in an array of logs SL (step 8). Afterwards, C-TDTC tries to iteratively divide the raw event log L into several sublogs until the total number of generated sublogs reaches μ

(steps 9–23). For each iteration of the *while-loop*, the original number of logs in array *SL* is first assigned to variable *m* (step 10). Then, the logs in *SL* are ranked by utilising function Ψ (mentioned above) so that the logs leading to higher complex models will be handled earlier (step 11). The *for-loop* (steps 12–19) tries to split one log in array *SL* into two sublogs. The log $SL[0]$ (which leads to a process model with maximal complex rate) is first processed by algorithm $\widehat{\Upsilon}$ (see Algorithm 4.5). If $\widehat{\Upsilon}$ cannot find a qualified CRSB from *TB* to split $SL[0]$ (i.e., the value *null* is returned by $\widehat{\Upsilon}$ and assigned to array ξ), our algorithm continues to deal with the next log in *SL* (i.e., log $SL[1]$), or log $SL[0]$ is removed from *SL* (step 15), the two sublogs (i.e., $\xi[0]$ and $\xi[1]$) generated through dividing $SL[0]$ by $\widehat{\Upsilon}$ are stored in *SL* (step 16) and C-TDTC continues the next iteration of the *while-loop* (step 17). Furthermore, the *while-loop* will also stop when no log in array *SL* can be further divided (steps 20–22). Finally, an array of sublogs *SL* is returned by C-TDTC, where the weighted average complexity of the sub-process models mined from the sublogs in *SL* is optimised.

4.3.2. A Mined Process Model Fitness Improvement Method

Fitness is an important metric for calculating the accuracy of a mined process model which represents the ratio of traces in the event log that can be expressed by the generated model [9]. In this section, the proposed technique CTC is devised to optimise the fitness of the non-fitting sub-process models stemming from the stage 1 in Figure 4.3 (through stage 2). The fitness improvement technique HIF (proposed in Chapter 3) for the inaccurately mined process models is utilised to help CTC achieve this purpose.

Basically, the technique HIF is able to locate the process behaviours recorded in the event log which cannot be expressed by the utilised BPMD algorithm and then convert them into expressible behaviours so that a more fitting process model can be mined. The advantage to employ HIF for optimising the accuracy of the inaccurate sub-process models derived from technique C-TDTC (Section 4.3.1) is that the problem of complex process behaviours appearing in the similar traces can be effectively dealt with.

4.3.3. The Compound Trace Clustering Method

Let $\Sigma_f : (M^+, L^+) \rightarrow FV^+$ be a process model fitness evaluation mechanism, where M^+ is the set of process models, L^+ is the set of event logs and FV^+ is the set of all possible fitness values that can be output by Σ_f. The details of CTC is described in Algorithm 4.7.

Algorithm 4.7 The compound trace clustering technique: CTC

Input: an event log L, a minimum support min_sup for mining closed sequential patterns, the minimum size κ for each generated sublog, the target number of generated sublogs μ, a target fitness ϵ for HIF, a model fitness improvement threshold ρ for HIF, a threshold ν for the number of newly added activities for HIF.

1: **Let** SL be an array of event log.
2: **Let** MO be a set of sub-process models.
3: $SL \leftarrow null$, $MO \leftarrow null$
4: ***Stage 1:*** *cluster traces for generating sub-process models with optimised complexity*
5: $SL \leftarrow C\text{–}TDTC(L, min\_sup, \kappa, \mu)$
6: ***Stage 2:*** *generate high-fitness sub-process models*
7: **for** each sublog $sl \in SL$ **do**
8: **if** $\Sigma_f(\Omega(sl), sl) < \epsilon$ **then**
9: $sl \leftarrow HIF(sl, \epsilon, \rho, \nu)$
10: $MO \leftarrow MO \cup \Omega(sl)$
11: **else**
12: $MO \leftarrow MO \cup \Omega(sl)$
13: **end if**
14: **end for**

Output: a set of sub-process models MO, an array of event log SL.

As described above, CTC contains two stages (as shown in Figure 4.3). In the stage 1, the technique C-TDTC (introduced in Algorithm 4.6) is employed to divide the original event log L into a fixed number (indicated by parameter μ) of sublogs that are then stored in array SL (step 5 of Algorithm 4.7), where the average complex rate of the sub-models mined from the generated sublogs is optimised. In the stage 2, if a sublog sl from SL leads to a sub-model which has a fitness value less than a given target value ϵ (step 8), then HIF is used to detect the inexpressible process behaviours and transform these found behaviours into expressible behaviours for the utilised BPMD technique Ω in log sl until the sub-process model mined from sl gets a fitness no less than ϵ (step 9). Finally, the sub-process models with optimised fitness values and average complex rate are stored in MO (step 10 and 12) which forms the output of CTC (the generated sublogs are also output by CTC).

4.4. Multi-Label Case Classification

In this section, we first elaborate the research topic of multi-label case classification and the challenge encountered for classifying the cases from real-life event logs (Section 4.4.1). Afterwards, we propose a systematic method named MLCC

based on sequential pattern mining technique to deal with the challenge so that the case classification process can be executed smoothly (Section 4.4.2, 4.4.3 and 4.4.4).

4.4.1. Problem Description

Figure 4.6 depicts a detailed model for case classification in the scenario of business process mining. Basically, after being classified by case-classifier, the primitive cases are linked to labels. Each label represents one category and cases connected to the same label share some common behaviours. Finally by working with the already classified cases, the process mining techniques are able to analyse the enterprise business process in different points of view and generate more readable and meaningful analysis results.

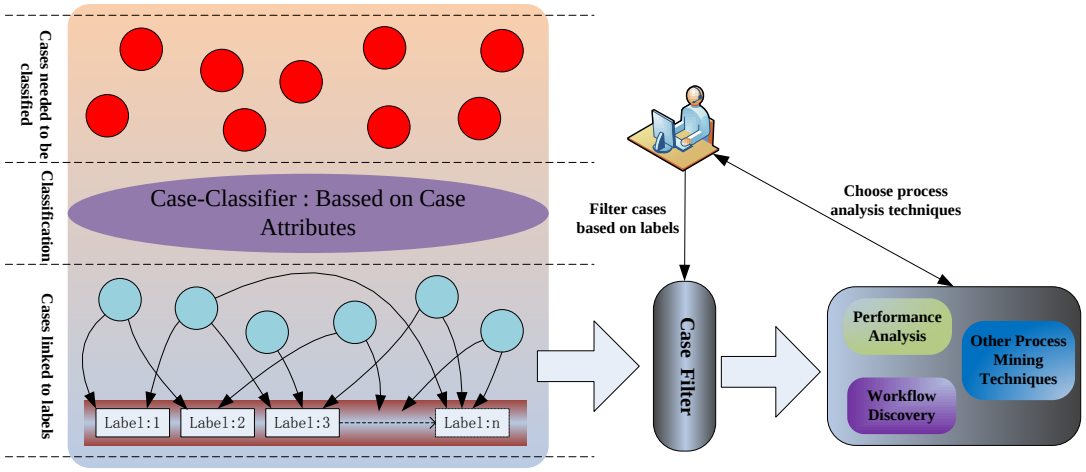


Figure 4.6.: Model of case classification in the scenario of process mining.

Traditional data classification approaches proceed in two steps [56]:

– In the **first step**, training data is analysed by a classification algorithm so that a function $y = f(X)$ can be learned. The generated function (classifier) is able to predict the associated category label y of a given tuple $X = (x_1, x_2, \ldots, x_n)$, where X represents the set of attributes of a specific item.

– In the **second step**, predictive accuracy of the classifier built is first estimated by utilising a test set made up of tuples and their relevant category labels. Then the classifier can be used to predict the class labels of future data tuples if its accuracy is considered acceptable.

A training event log should be generated first for a case classification prob-

lem. Labels are added manually to part of the cases in an event log by domain experts according to their domain knowledge and the behaviours of cases. Then these labeled cases are extracted to generate the training event log and test event log. A training event log is used for building the classifier and a test event log for estimating the performance of the classifier. By looking into some real-life event logs we discovered that a case may have more than one label which makes this case classification problem a multi-label classification problem (as shown in Figure 4.6).

Table 4.2.: An example event log.

Case id	Event id	Properties		
		Activity	Resource	Cost
1	421	A	Pete	40
	422	B	Sun	200
	423	C	Simon	300
	424	D	Chris	100
	425	E	Pete	200
2	452	A	Mike	30
	453	C	Simon	300
	454	F	Chris	200
	455	D	Sun	100
	456	G	Mike	500
...	...	...	...	...

The behaviours of cases are reflected by the values of case attributes recorded in the event log. Most case attributes that have discrete values or numeric values can be easily utilised for building the classifier or judging which labels a case belongs to. But the trace of a case cannot be directly used. A trace is an important attribute of a case which is a finite sequence of ordered events. For instance, in Table 4.2, the trace of case 1 is $< A, B, C, D, E >$. The trace may be a major element for deciding which labels a case pertains to (while the labels are related to the structural feature of trace).

For the cases generated by structured business processes, it is easy to transform the traces into a suitable form that can be utilised as an attribute for case classification because the compositions of traces are limited by a structured process model[3]. However, for the cases from a real-life event log, such a transformation should not be directly carried out because it is possible that cases with similar features may seem very different from each other. Additionally, there might be multiple structural features (presence or absence of an activity, presence or absence of combinations of activities and so on) of traces related to one label. So

[3]If a structured business process has a loop structure it can also generate a large amount of isomerous traces.

how to capture the possible structural behaviours of traces relevant to labels is the key factor for solving the multi-label case classification problem.

In this section, we put forward a sequential pattern mining technique-based method for mining all of the possible label-related structural features of traces and then transforming these found features into suitable forms of case attributes to help the later case classification.

4.4.2. Basic Concepts Relevant to Multi-Label Case Classification

A multi-label classification technique solves the problem of predicting to which set of classes (also represented by labels) a new instance belongs by exploiting a training set of data. The training data is a set $T = \{t_1, t_2, \ldots, t_n\}$ of already classified samples where each sample t_i is constructed by a k-dimensional vector $X_{t_i} = (x_{i_1}, x_{i_2}, \ldots, x_{i_k})$. The dimensions in X_{t_i} represent attributes of the sample, as well as the categories to which t_i pertains.

The existing multi-label classification methods are mainly divided into two types, one is algorithm independent and the other one is algorithm dependent [58]. In this section, we will utilise the algorithm independent approach for solving the problem of multi-label case classification. In the algorithm independent approach, a multi-label classification problem can be converted into several single-label problems. For each label (or category) a classifier is built so that the classification problem related to this label can be dealt with. For the multi-label case classification problem in this subsection, the training data is conveyed by the following definitions:

Definition 4.7. *(Set of Training Cases)*
Let C_t be a set of training cases. A case $c \in C_t$ is defined as a tuple $c = (N_c, LA_c, \Theta_c)$, where $N_c = \{n_1, n_2, \ldots, n_k\}$ is the set of names of case attributes, LA_c is a set of labels, $\Theta_c : N_c \to A_c$ is an attribute-transition function which maps the name of an attribute into the value of this attribute, where A_c is the set of attribute values for case c. A label $la \in LA_c$ represents a manually given class to which case c belongs.

As already mentioned, a case in an event log may be assigned multiple labels in the real world, thus for all $c \in C_t$, we have $|LA_c| \geq 1$, where $|LA_c|$ stands for the number of labels.

Definition 4.8. *(Training Event Log)*
A training event log is defined as $L_t \subseteq C_t$, for any $c_1, c_2 \in L_t$ such that $c_1 \neq c_2$.

Let's presume that the example event log in Table 4.2 is a training event log, all cases in this log have an attribute *originator* and an attribute *labels*, case 1 has an

originator "*Mike*" and a set of assigned labels {"*very good*","*good*"}. According to the concepts defined above, $\Theta_1(originator)$ = "*Mike*" is the originator for case 1, $\Theta_1(trace) =< A, B, C, D, E >$ is the trace for case 1, $LA_1 = \{"very\ good","good"\}$ is the set of labels for case 1.

Definition 4.9. *(Multi-Label Case Classification)*
The multi-label case classification problem is defined as $Prob = (L_t, L, \widetilde{\Phi})$, where L_t is a training event log, L is an event log consisting of cases waiting to be classified, $\widetilde{\Phi}$ is a multi-label classification algorithm. $\widetilde{\Phi} : (L_t^+, L^+) \rightarrow CL^+$ represents the process for generating an event log consisting of classified cases with an input of training event log and an input of event log containing unclassified cases, where L_t^+ stands for the set of all possible training event logs, L^+ represents the set of event logs with cases waiting to be classified and CL^+ is the set of event logs with classified cases.

4.4.3. Definitions Relevant to Functions

The sequential pattern mining techniques solve the problem of finding all frequent subsequences from a given set of sequences, where each sequence contains a list of ordered events and each event consists of a set of items [62]. A minimum support threshold is manually given for judging if the occurrence of a subsequence is frequent or not.

As introduced in Chapter 2, the trace of a case is a sequence of ordered events. Thus, the set of traces collected from an event log can be deemed as a sequence database on which the sequential pattern mining algorithms can be directly implemented. We define a sequential pattern mined from a set of traces extracted from an event log as a function:

Definition 4.10. *(Function)*
Let L be an event log, $ST \subseteq L$ be a set of traces collected from L, $\widehat{\Gamma}$ be a sequential pattern mining algorithm. Let $F = \widehat{\Gamma}(ST, min\_sup)$ be the mined set of sequential patterns with a minimum support threshold min_sup. A sequential pattern $f_i \in F$ is defined as a function relevant to ST and F is called a set of functions.

Definition 4.11. *(Label-Related Function)*
Let L_t be a training event log, $ST_{label} \subseteq L_t$ be a set of traces from L_t, where each trace in ST_{label} is related to one common label. A label-related function set F_{label} is a set of functions mined from ST_{label}, and a function $f_k \in F_{label}$ is called a label-related function.

In our opinion, the label-related functions in F_{label} reveal the commonly and frequently appeared structures of the traces in ST_{label}. As mentioned in Section

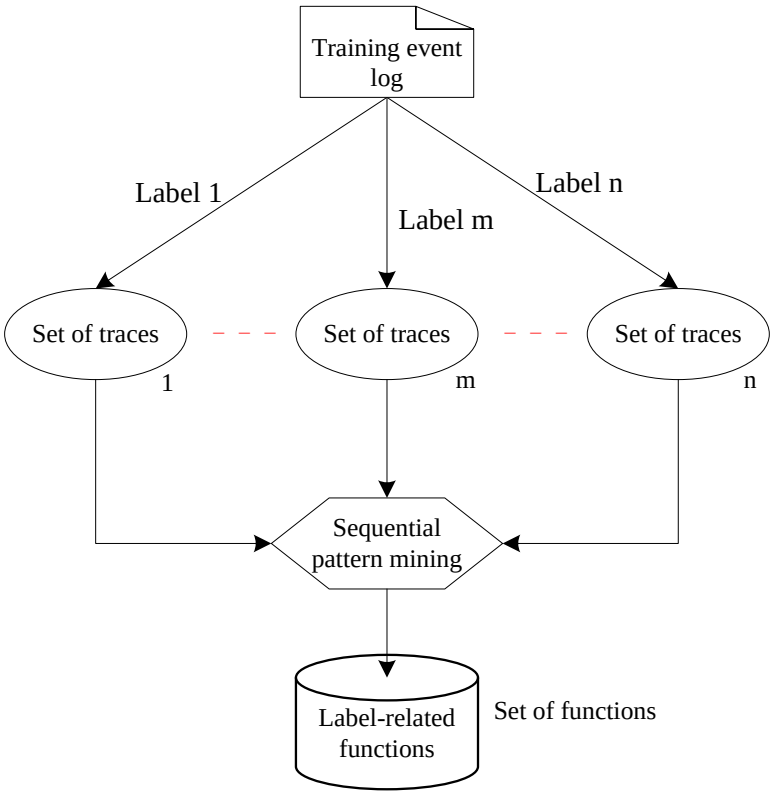


Figure 4.7.: The procedure of mining label-related functions from a training event log.

4.4.1, a label may be associated with the structural characteristics of traces. While this is true, the label-related functions can be exploited to judge weather a trace belongs to a specific label.

Figure 4.7 illustrates the process of mining all the possible label-related functions from a training event log. In the first step, all the traces included by a training event log are separated into different sets where each set is associated with one label. For example, in Figure 4.7 the traces of cases with label 1 are collected and sent to set 1. In the second step, a sequential pattern mining procedure is executed on each set of traces for discovering label-related functions. Finally, all of the found functions are grouped together in one set (i.e., label-related function set).

Let's presume that the event log shown in Table 4.2 is a training event log and each case in this log has a set of labels from $\{la_1, la_2, la_3, la_4\}$. For this training event log, a set of traces ST_{la_2} is obtained through extracting all traces with label la_2. Then by mining ST_{la_2} using a sequential pattern mining method a label-related function set F_{la_2} can be extracted. This label-related function mining procedure mentioned above is described in Algorithm 4.8.

Algorithm 4.8 Mine label-related functions from a given training event log

Input: a training event log L_t, the minimum support threshold min_sup

Let $G(label)$ be a set of traces related to a specific label in L_t.
Let F_{label} be a set of all possible label-related functions for L_t.
Let LA be the set of labels in L_t.
Let Θ be a case attribute-transition function as described in Definition 4.7.
Let $\widehat{\Gamma}$ be a sequential pattern mining algorithm.
$F_{label} \leftarrow null$
for each $G(label_i)$ such that $label_i \in LA$ **do**
    $G(label_i) \leftarrow null$
end for
for each case $c_j \in L_t$ **do**
    for each label $la_k \in LA_{c_j}$ **do**
        $G(la_k) \leftarrow G(la_k) \cup \Theta_{c_j}(trace)$
    end for
end for
for each label $la \in LA$ **do**
    $F_{label} \leftarrow F_{label} \cup \widehat{\Gamma}(G(la), min\_sup)$
end for

Output: the set of all label-related functions F_{label} mined from L_t

4.4.4. Transforming Label-Related Functions into Case Attributes

The raw label-related functions found cannot be exploited directly, they should be transformed into a suitable form of case attributes so that they can be checked by a classification algorithm directly. In the following parts we will propose a method for this purpose.

Through algorithm 4.8, the traces in a training event log are grouped into different sets where each set is related to one label and the functions for each set are discovered. Before building a classifier for each label, the found functions need to be converted into usable case attributes. To explain our method clearly, we will first set up a few variants: L_t is a training event log, a set $G = \{G(label_1), G(label_2), \ldots, G(label_n)\}$ where each $G(label_i)$ is a set of all the traces relevant to $label_i$ in L_t, a set $F = \{F(label_1), F(label_2), \ldots, F(label_n)\}$ where each $F(label_j)$ is a function set for $G(label_j)$, $F^* = \{function_1, function_2, \ldots, function_m\}$ is a set which contains all of the functions in F. An association table AT is then established which connects each function in F^* with a global unique identifier. In the association table shown in Table 4.3, for instance, $function_1$ has an id A_1.

Let $c_p \in L_t$ be a case from training event log L_t, add all of the function ids in the association table as attribute names to the attribute list of c_p, and their

Table 4.3.: An example association table for the functions in F^*.

Function ID	Function
A_1	$function_1$
A_2	$function_2$
$\vdots$	$\vdots$
A_m	$function_m$

initial values are set to 0. The next step is to calculate the value of each newly added attribute. Take function A_1 from Table 4.3 as an example, a subsequence-detection process is carried out with $\Theta_{c_p}(trace)$ (the trace of c_p) and $function_1$ (matched with A_1 in Table 4.3) as inputs. If $function_1$ is judged to be a subsequence of $\Theta_{c_p}(trace)$, then $\Theta_{c_p}(A_1)$ is reset to be 1. The procedure mentioned above should also be applied to both the test event logs and the normal event logs which contain cases needed to be classified. Let $\widehat{\Pi}(F^*, ST) \rightarrow \{True, False\}$ be a subsequence detection function, where F^* is the set of functions and ST stands for the set of traces. $\widehat{\Pi}(f_1, t_1) = True$ if function f_1 is a subsequence of trace t_1 and $\widehat{\Pi}(f_1, t_1) = False$ if function f_1 is not a subsequence of trace t_1. Algorithm 4.9 describes the details of the function-to-case-attribute transformation method introduced above.

Algorithm 4.9 Transform label-related functions into case attributes

Input: an event log L, a set of label-related functions F_{label}

Let AT be an association table with two fields: $Function\_ID$ and $Function$.
Let Θ be a case attribute-transition function as described in Definition 4.7.
$AT \leftarrow null$
for each function $f_i \in F_{label}$ **do**
    $AT \leftarrow AT \cup (ID_{f_i}, f_i)$
end for
for each case $c_j \in L$ **do**
    for each item $(ID_{f_m}, f_m) \in AT$ **do**
        if $\widehat{\Pi}(f_m, \Theta_{c_j}(trace)) == True$ **then**
            $\Theta_{c_j}(ID_{f_m}) = 1$
        else
            $\Theta_{c_j}(ID_{f_m}) = 0$
        end if
    end for
end for

Output: an event log L with newly added attributes

4.5. Preliminary Verification for Techniques TDTC, CTC and MLCC

In this section, the proposed techniques TDTC, CTC and MLCC are preliminarily verified. The verification process for TDTC and CTC described in Section 4.5.1 is based on the repair log from [9]. The technique MLCC is validated by employing the hospital event log from BPIC 2011 in Section 4.5.2.

4.5.1. Verification for TDTC and CTC

In the verification process, the HM from ProM 6 is used for mining process models, the ICS fitness is used for evaluating the accuracy of mined models (because it has a computationally efficient calculative process) and the PT-CD (introduced in Chapter 2) is utilised for evaluating the complexity of mined process models. The CSP mining algorithm CMClaSP [148] is employed to mine CSPs for both TDTC and CTC. The *Heuristics net to Petri net* plugin in ProM 6 is used for transforming the Heuristics net mined by HM into Petri net so that the PT-CD can be calculated. The repair log utilised consists of 1000 traces, 10827 events and twelve activities. Each of the twelve activities is assigned a unique ID as shown in Table 4.4 (in this subsection the activity ID is used to represent its associated activity). The process model mined from the repair log by HM has an ICS fitness value 0.6768 and a PT-CD value 2.3656 (see Figure 4.8).

Table 4.4.: The association table for the activities from the repair log.

Activity ID	Activity Name
A_1	*Register complete*
A_2	*Analyze Defect start*
A_3	*Analyze Defect complete*
A_4	*Repair (Complex) start*
A_5	*Repair (Complex) complete*
A_6	*Archive Repair complete*
A_7	*Inform User complete*
A_8	*Test Repair start*
A_9	*Test Repair complete*
A_{10}	*Repair (Simple) start*
A_{11}	*Repair (Simple) complete*
A_{12}	*Restart Repair complete*

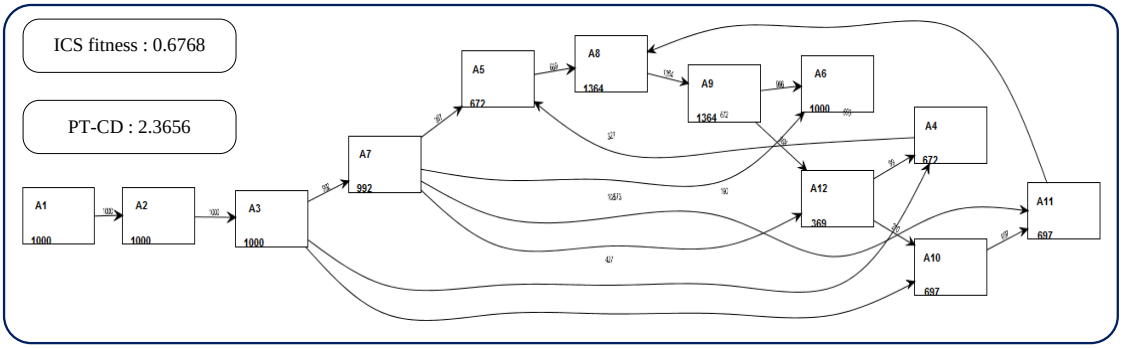


Figure 4.8.: The process model mined from the repair log.

For mining CSPs from the repair log for technique TDTC and CTC, the minimum support min_sup for CMClaSP is set to 0.25. As a result, 26 CSPs are discovered of which the details are shown in Table 4.5. Due to the model mined from the repair log has a low complexity and a relatively low fitness, the fitness weight α is set to 0.8 and the complexity weight β is set to 0.2 for calculating the SMI for TDTC. Furthermore, the minimum threshold μ for SMI is set to 0.02, both the minimum thresholds μ_f and μ_c for SSTB are set to 0 (which means the minimum average improvement on both fitness and complexity should be larger than 0), the minimum threshold φ_f for FCSTB is set to 0.8, the maximum threshold φ_c for CCSTB is set to 2.5 and the minimum size θ for each underlying sublog is set to 50 for technique TDTC. Finally, a binary tree as shown in Figure 4.9 is output by TDTC (with the parameters mentioned above) for the repair log. According to Figure 4.9, four sublogs are output by TDTC which are sublog 1.1 (contains 102 traces), sublog 1.2 (contains 388 traces), sublog 2.1 (contains 263 traces) and sublog 2.2 (contains 247 traces). The CSPs discovered for splitting the original repair log are also displayed in the built binary tree. For example, the CSP with an ID 19 in Table 4.5 is utilised by TDTC to divide the raw repair log into sublog 1 and sublog 2, where sublog 1 contains all the traces with CSP 19 as sub-trace from the repair log and sublog 2 consists of all the traces without CSP 19 as sub-trace from the repair log. Afterwards, the generated sublog 1 is separated into sublog 1.1 and sublog 1.2 by TDTC utilising CSP 9. The sublog 2 is divided into sublog 2.1 and sublog 2.2 by TDTC using CSP 26. At last, TDTC stops because it cannot find any qualified CSPs from Table 4.5 to further split the sublog 1.1, sublog 1.2, sublog 2.1 and sublog 2.2. Figure 4.11 shows the detailed information about the process models mined from the four sublogs output by TDTC.

For the technique CTC, the minimum size κ for each potential sublog is also set to 50, the target number μ of generated sublogs is set to 5. For the technique HIF employed in CTC, the target fitness ϵ is set to 1, the model fitness improvement threshold ρ is set to 0 and the threshold ν for the number of newly added activities is set to $+\infty$ (we utilise such a combination of parameters for HIF because the generated sublogs by C-TDTC are very simple). Figure 4.10 shows the division details for the repair log processed by technique C-TDTC (Algorithm

Table 4.5.: The CSPs mined from the repair log by using CMClaSP for technique TDTC and CTC.

Pattern ID	Frequency	Closed Sequential Pattern (CSP)
1	427	$<A1, A2, A3, A10, A11, A6>$
2	786	$<A1, A2, A3, A7, A9, A6>$
3	276	$<A1, A2, A3, A7, A11, A8, A9, A6>$
4	666	$<A1, A2, A3, A7, A8, A9, A6>$
5	353	$<A1, A2, A3, A8, A9, A7, A6>$
6	402	$<A1, A2, A3, A7, A5, A6>$
7	996	$<A1, A2, A3, A8, A9, A6>$
8	426	$<A1, A2, A3, A10, A11, A8, A9, A6>$
9	605	$<A1, A2, A3, A4, A5, A8, A9, A6>$
10	1000	$<A1, A2, A3, A6>$
11	608	$<A1, A2, A3, A4, A5, A6>$
12	298	$<A1, A2, A3, A4, A7, A8, A9, A6>$
13	294	$<A1, A2, A3, A10, A7, A9, A6>$
14	401	$<A1, A2, A3, A7, A5, A8, A9, A6>$
15	371	$<A1, A2, A3, A10, A7, A6>$
16	265	$<A1, A2, A3, A8, A9, A12>$
17	490	$<A1, A2, A3, A8, A7, A6>$
18	296	$<A1, A2, A3, A4, A7, A5, A6>$
19	497	$<A1, A2, A3, A4, A7, A6>$
20	275	$<A1, A2, A3, A10, A11, A7, A6>$
21	272	$<A1, A2, A3, A10, A11, A8, A7, A6>$
22	287	$<A1, A2, A3, A8, A7, A9, A6>$
23	277	$<A1, A2, A3, A7, A11, A6>$
24	992	$<A1, A2, A3, A7, A6>$
25	368	$<A1, A2, A3, A4, A7, A9, A6>$
26	295	$<A1, A2, A3, A4, A7, A5, A8, A9, A6>$

4.6). According to Figure 4.10, the traces that have the CSP 1, CSP 20 and CSP 2 as sub-traces from the repair log form the first sublog 1.1.1. The traces with the CSP 1 as a sub-trace and without the CSP 20 and CSP 2 as sub-traces construct the second sublog 1.1.2. The traces with the CSP 1 as a sub-trace and without the CSP 20 as a sub-trace compose the third sublog 1.2. The traces without the CSP 1 as a sub-trace and with the CSP 2 as a sub-trace constitute the fourth sublog 2.1. Finally, the traces without the CSP 1 and CSP 2 as sub-traces make up the fifth

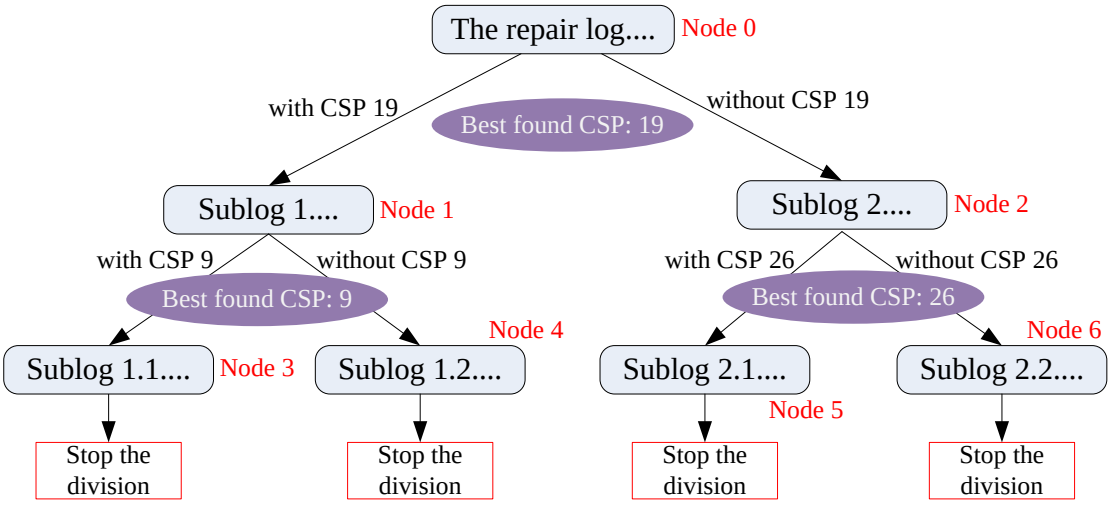


Figure 4.9.: The binary tree output by TDTC executed on the repair log.

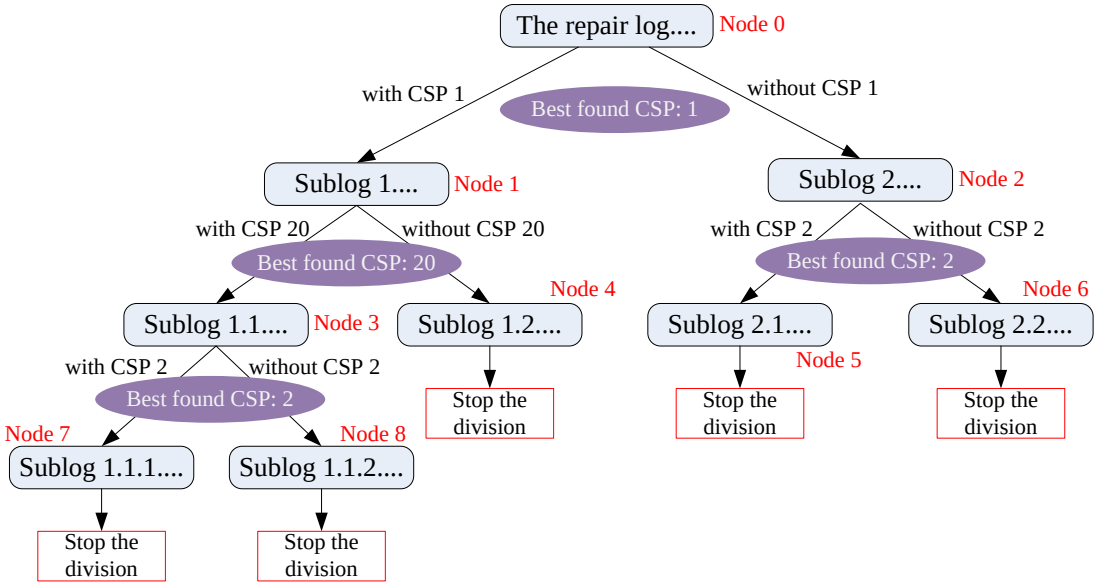


Figure 4.10.: The information of the division process for the repair log executed by technique C-TDTC.

sublog 2.2. Then, the technique HIF is utilised by CTC to help improve the fitness of the sub-models (have fitness values less than ϵ) mined from the corresponding sublogs. Figure 4.12 shows the details of the finally generated sub-process models output by CTC.

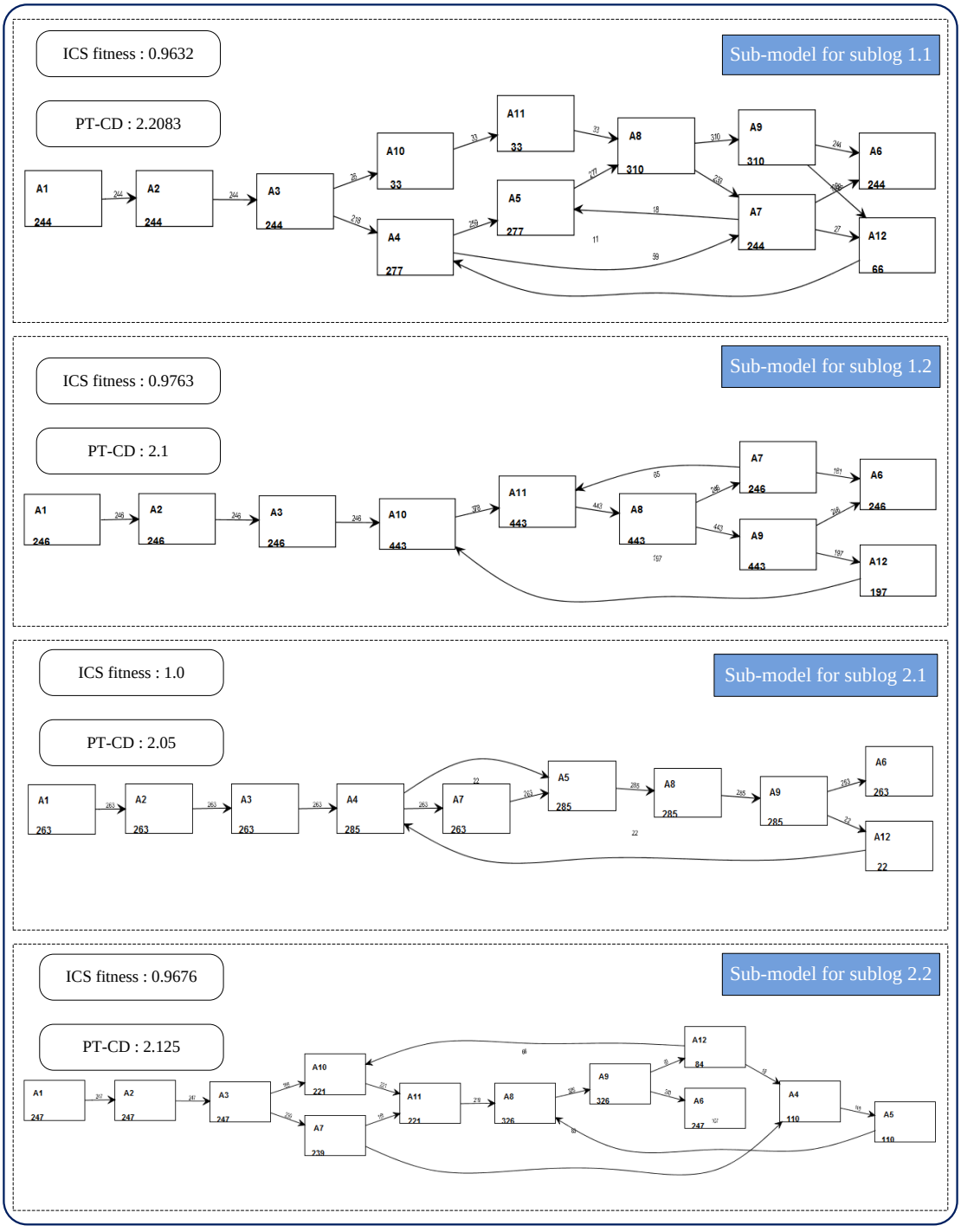


Figure 4.11.: The four sub-process models generated by technique TDTC for the repair log.

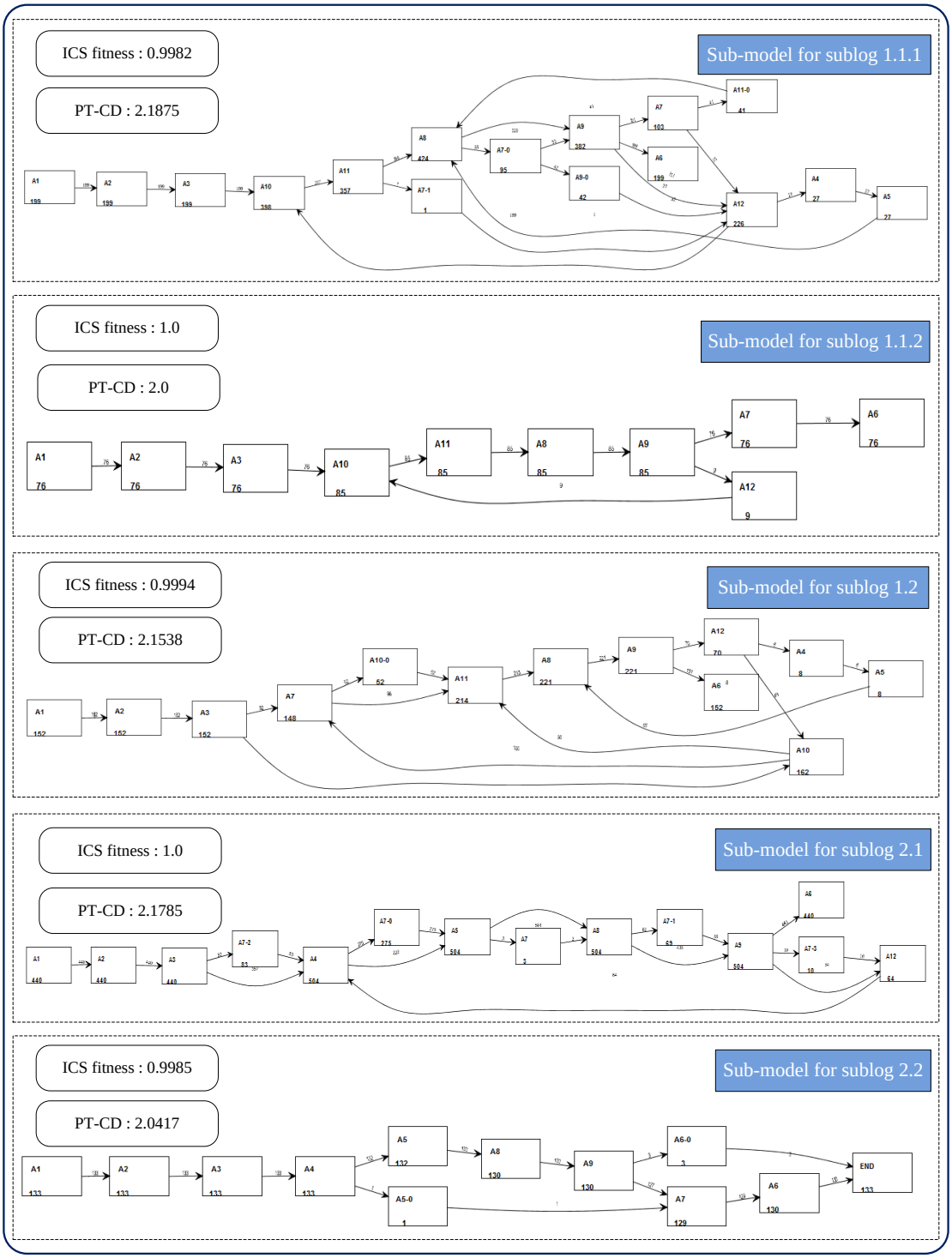


Figure 4.12.: The five sub-process models generated by technique CTC for the repair log.

4.5.2. Verification for MLCC

We tested the effectiveness of our technique MLCC on the hospital event log from BPIC 2011. This log contains 624 activities and 1143 cases where each case stands

for a treatment process for a patient from the gynaecology department. A lot of attributes relevant to the cases have been recorded in this log, such as the ages of patients and the final diagnosis for the patients.

In the experiment, the treatments performed on a patient are regarded as labels (categories). One reason is that in the healthcare industry treatment is often used as a label for classifying cases, for instance, the SAP Business Suite for Patient Management exploits the treatment as one category for case classification[4]. There are overall 48 kinds of treatment (coded by number) in this log from which we have chosen seven frequently happened treatments (namely 13, 23, 61, 101, 113, 603 and 3101) for analysis. Each case may belong to more than one treatments and each treatment may be characterized by multiple behaviours of traces.

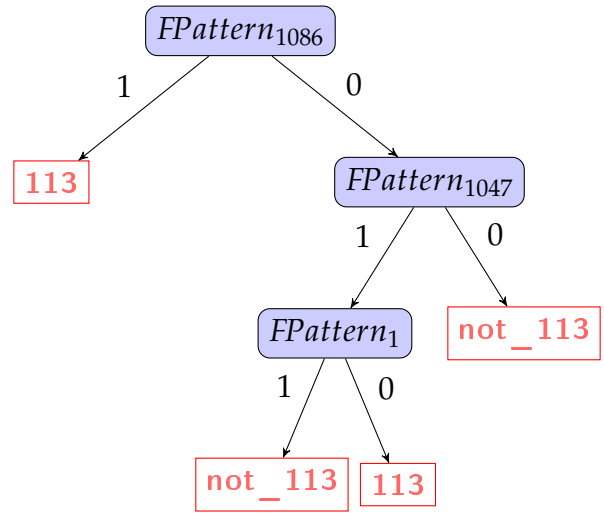
For generating the training event log, a simple strategy mentioned in [57] is employed which discards every multi-label case in the hospital event log. For instance, all the cases that have only a single treatment belonging to the treatment set $TS = \{13, 23, 61, 101, 113, 603, 3101\}$ are extracted and form the training event log. All of the multi-label cases are organized as the test event log. As a result, we obtain a training event log with 279 cases and a test event log with 621 cases.

For mining the label-related functions in Algorithm 4.8, the closed sequential pattern mining algorithm CMClaSP [148] is used. By implementing Algorithm 4.8 with the generated training event log and a minimum support $min\_sup = 0.3$ as input parameters, a label-related function set F_{label} which contains 3159 functions can be discovered. Then all of the found functions are transformed into case attributes for both the training event log and the test event log through Algorithm 4.9. In the association table AT generated in Algorithm 4.9, the id of a function is in the form of $FPattern_k$ where k represents the position of this function in AT. For example, $A_1 = FPattern\_1$ in Table 4.3 because $function_1$ is the first item and $A_m = FPattern\_m$ because $function_m$ is the m^{th} item in this association table.

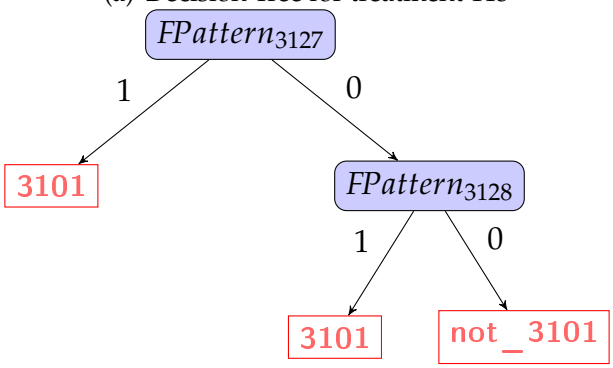
In this chapter, we utilise an algorithm independent approach for solving the problem of multi-label case classification which converts the learning problem into traditional single-label classification. For each element in the treatment set TS (mentioned above) a classifier is learned by using a training event log. For example, for treatment 13, a classifier is built which is able to judge if a case falls in treatment 13 or not. In our experiment, seven binary classifiers (because seven kinds of treatment are analysed) are established.

To testify the standpoint that the labels pertained by a case may be related to the structural feature of its trace, only the case attributes generated by the transformation of discovered label-related functions are considered in our experiment.

[4]http://help.sap.com



(a) Decision Tree for treatment 113



(b) Decision Tree for treatment 3101

Figure 4.13.: Decision Trees built for treatment 113 and treatment 3101.

For instance, in this hospital event log, the other case attributes such like *Age* and *Diagnisis* are not put to use.

Firstly, we utilise the Decision Tree-based algorithm C4.5 [86] in our experiment. For each treatment in *TS* a decision tree is built by exploiting the attributes of cases. For example, we get a decision tree for treatment 113 as shown in Figure 4.13(a), and Figure 4.13(b) shows the decision tree for treatment 3101. According to the decision tree for treatment 113, a case will be inferred to belong to treatment 113 if it has an attribute *FPattern*_1086 = 1, and not if it has the attributes *FPattern*_1086 = 0 and *FPattern*_1047 = 0 at the same time.

Table 4.6 shows the performances for the seven classifiers evaluated by using the test event log generated. Several metrics are calculated in the evaluation step. The Area Under the ROC Curve (AUC) which has a value between 0 and 1 reflects the performance of the classification model. An ideal classifier has an AUC value close to 1. Correctly Classified Instances Ratio reflects the total classification accuracy of a specific classifier. The Kappa Statistic (KS) measures the diversity

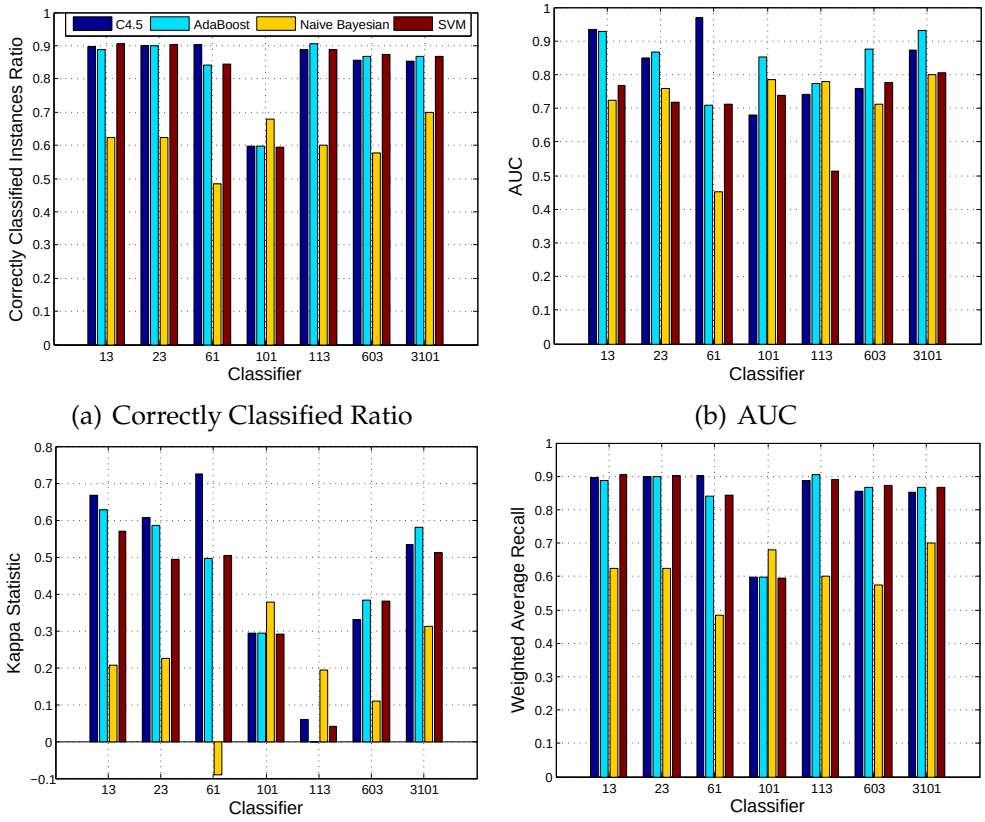
factor between the classification results from a classifier built and a classification by chance. According to [204], $KS \in (0.75, 1)$ implies that the effect of the classifier is very good, $KS \in (0.4, 0.75)$ is characterized as fair to good and $KS \in (0, 0.4)$ as poor. The metric Recall measures the proportion of correctly classified instances among all the instances with the same label or without a specific label. According to Table 4.6, the performances of the classifiers for treatment 113 and 101 are not good. The main reason is that the emerging frequencies of some trace structural behaviours relevant to these two treatments are very low in the training event log, as a result they cannot be captured in the sequential pattern mining procedure. One feasible way to solve this problem is to increase the training cases for treatment 101 and 113 in the training event log.

Table 4.6.: Performances of the classifiers built for each treatment in TS.

Treatment	Correctly classified instances ratio	AUC	Kappa statistic	Recall
13	0.896940	0.935	0.6662	0.988/0.882
23	0.900161	0.850	0.6060	0.782/0.917
61	0.901771	0.968	0.7241	0.729/0.959
101	0.597424	0.679	0.2935	0.481/1.000
113	0.887279	0.740	0.0590	0.068/0.973
603	0.855072	0.758	0.3304	0.638/0.873
3101	0.853462	0.873	0.5335	0.900/0.847

We also compared different classification techniques on the case classification problem. These techniques are AdaBoost [79] that uses Decision Stump [205] as weak learners, Naive Bayesian [78] and SVM (Support Vector Machine) [81]. From the comparison results shown in Figure 4.14 we can see that C4.5, AdaBoost and SVM perform better with function-based case attributes than the Naive Bayesian classification method on this data set.

Additionally, to testify the practicability of our technique in business process mining area, we then evaluate the effectiveness of the classification results (obtained by using Decision Tree algorithm) on the process model discovery task (the most crucial learning task in process mining domain). Let L_{test} be the test event log generated in our experiment, $SL = \{L_{13}, L_{23}, L_{61}, L_{101}, L_{113}, L_{603}, L_{3101}\}$ be the set of sublogs where each sublog contains the cases correlating to one treatment from the treatment set $TS = \{13, 23, 61, 101, 113, 603, 3101\}$. For instance, L_{13} contains all of the cases with treatment 13 from L_{test}. Let $PL = \{PL_{13}, PL_{23}, PL_{61}, PL_{101}, PL_{113}, PL_{603}, PL_{3101}\}$ be the set of sublogs where each sublog consists of the cases predicted to have one same treatment from TS. For example, sublog PL_{13} contains all of the cases which are predicted to have treatment 13 by the classifier built for treatment 13. Then process models for the entire



(a) Correctly Classified Ratio

(b) AUC

(c) Kappa Statistic

(d) Weighted Average Recall

Figure 4.14.: Comparison among different classification techniques on case classification

test log L_{test} and for each sublog in SL and PL are learned by using process discovery techniques. In our experiment, the BPMD technique HM as described in [37] is utilised. Afterwards, the ICS fitness [29] (fitness measures the proportion of behaviour in the event log possible according to the model) for each model is calculated and the results are shown in Figure 4.15. The ICS fitness related to each sublog in SL is called Test-Fitness and the ICS fitness for the sublogs in PL is called Predicted-Fitness. The Original-Fitness (as a base line in Figure 4.15) is calculated by using the entire test event log L_{test} and the model for it. In Figure 4.15, we can see that the Predicted-Fitnesses and Test-Fitnesses for most of the sublogs in SL and PL are much higher than the Original-Fitness due to the sublogs in SL and PL are simpler and contain less complex process behaviours than the original event log L_{test}. As a result, the models generated for these sublogs become more accurate. By comparing the Predicted-Fitness with the Test-Fitness in Figure 4.15, we discovered that most of the Predicted-Fitnesses are very close to their relevant Test-Fitnesses, for instance, the value of Predicted-Fitness for PL_{3101} is 0.9267 and the value of Test-Fitness for L_{3101} is 0.9382 which is very close to the Predicted-Fitness for PL_{3101}. However, the difference between the Predicted-Fitness and the Test-Fitness related to treatment 101 is a little large. The main reason is that the classifier for treatment 101 doesn't have a good performance. From the analy-

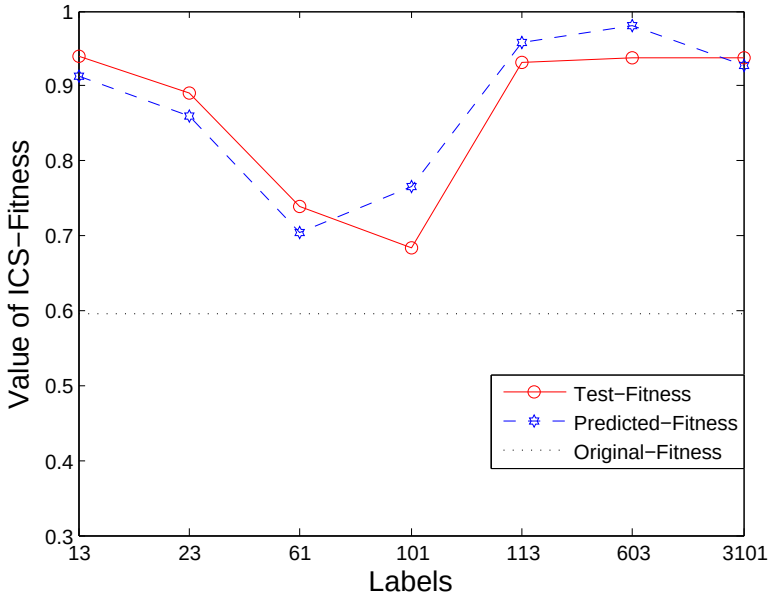


Figure 4.15.: ICS fitness for the models generated by utilising the sublogs from SL and PL.

ses mentioned above it can be deduced that the results from the multi-label case classification technique have some practical values in the business process mining area. Process discovery is only one perspective of business process mining techniques and the multi-label case classification method can also benefit other techniques in this area (e.g., business process performance analysis) to help generate more meaningful and accurate analysis results.

5

A Graph and Trace Clustering-Based Approach for Abstracting Mined Business Process Models

5.1. Introduction and Motivation

As introduced in Chapter 4, trace clustering techniques try to divide the raw event log into sublogs where each sublog contains traces with similar behaviours and helps generate a more accurate and comprehensible sub-model. Generally, these techniques perform well for handling the logs with a moderate amount of trace behaviours. Nevertheless, the limitation of current trace clustering techniques will be revealed while dealing with event logs containing massive trace behaviours. For instance, the event log of a Dutch academic hospital from BPIC 2011 consists of 624 activities among which a large number of relations are exhibited (the average out-degree for each activity is 6.2564) and most of the classical trace clustering methods cannot bring a significant improvement on the mining result for this hospital log (as shown in Section 6.4 of Chapter 6).

Process model abstraction-based approaches make the assumption that raw models mined from real-life logs contain low level sub-processes which should be discovered in the form of sub-traces in the original event logs and abstracted into high level activities so that the insignificant low level process behaviours can be hidden in the high level activities. Thus, more accurate and simpler high level process models can be obtained. However, most of the present process model abstraction-based techniques focus mainly on the discovery of sub-processes and cannot ensure the accuracy of the high level process models generated.

In this chapter, we put forward a new method named GTCA (graph and trace clustering-based approach) which inherits the characteristics of trace clustering

techniques and the process model abstraction-based approaches for solving the problem of "spaghetti-like" process models. The proposed technique is able to optimise the quality of the potential high level process model through a new abstraction strategy based on graph clustering technique [134]. As a result, a high-quality abstraction model can be built. Furthermore, the quality of the sub-models discovered for showing the details of their related high level activities (used for building the final high level model) is also considered by our approach. The structure of the main contents in this chapter is organised as:

- A new strategy for abstracting the raw models mined from real-life event logs is discussed in Section 5.2.

- In Section 5.3, a three-stage model abstraction method based on the strategy proposed in Section 5.2 is elaborated.

5.2. Basic Idea

In the real world, seemingly "spaghetti-like" business process models mined from event logs might still have some rules to follow. Sometimes, the main reason for the structurelessness of these mined models is that they contain several extremely complex sub-structures. However, the relations among these sub-structures may be straightforward. While turning to a specific event log, such kind of phenomenon mentioned above can be reflected by the existence of several clusters of activities from an event log where the activities in the same cluster are densely connected and the activities in different clusters are sparsely connected (this is also the assumption for our method). For instance, in Figure 5.1 an event log L contains 22 activities and a causal activity graph G can be established by employing the activities from L as vertices and the casual relations [37] among these activities as edges. According to Figure 5.1, the vertices in G can be grouped into three clusters by considering the edge structure in such a way that there should be many edges within each cluster and relatively few edges among the clusters.

With the assumption mentioned above, we put forward a new strategy for solving the problem of complex and inaccurate process models mined from real-life event logs. The basic idea is to generate the clusters of activities first by following the same rule utilised in the example shown in Figure 5.1. Afterwards, for each cluster one or several sub-models are generated where each sub-model only contains the activities from its relevant activity cluster. In the example from Figure 5.1, the sub-models for cluster A are built by using the activities from cluster A. Then, for a complex and inaccurate sub-model, trace clustering technique is employed to split it into several simple and accurate sub-sub-models so that the sub-model can be well comprehended. Finally, these sub-models (not includ-

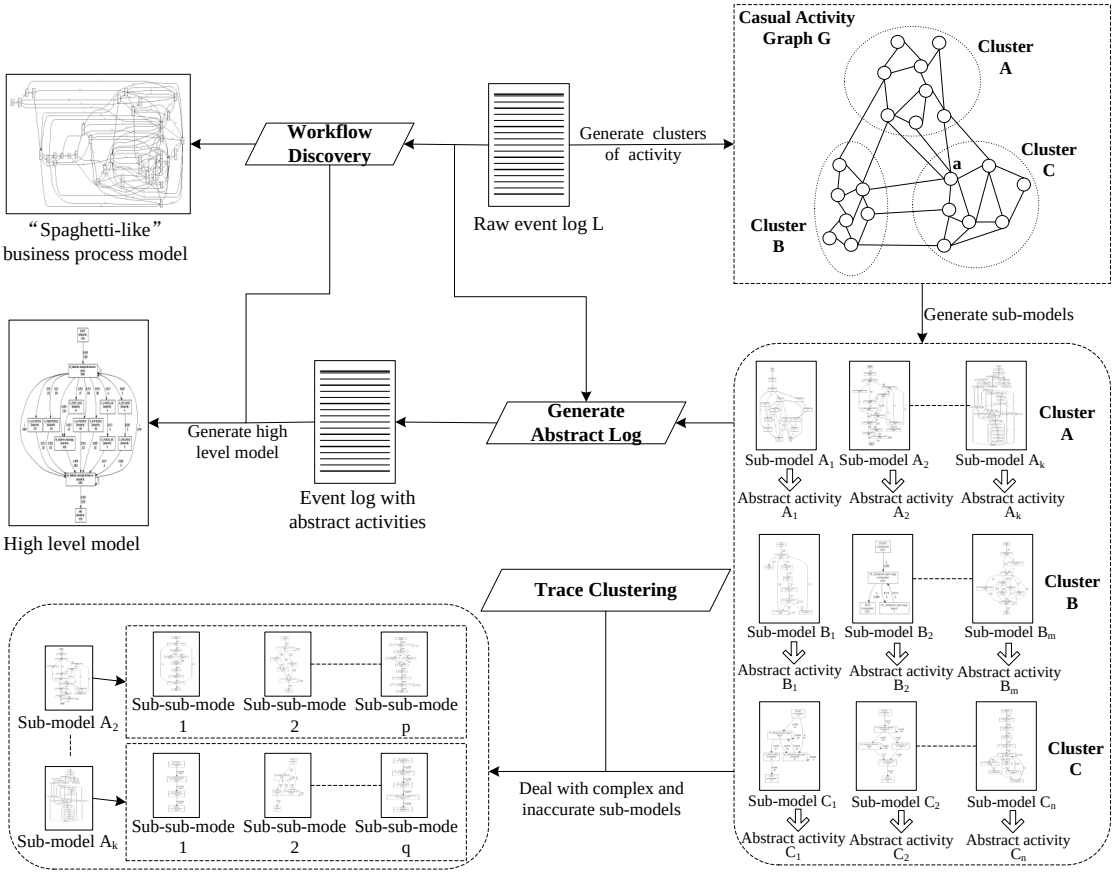


Figure 5.1.: Illustration of the basic ideas of the proposed approach GTCA.

ing the sub-sub-models) generated are abstracted into high level activities with which a simple and accurate ultima high level process model is formed. In this chapter, the high level process model together with the sub-models (each sub-model is related to one high level activity in the high level model built) are used to show the details of the whole business process recorded in event log.

Basically, two major benefits could be acquired from the strategy proposed above. On one hand, the original tough problem (deal with the entire model) met by current trace clustering techniques is transformed into small sub-problems (deal with the sub-models). Specifically, the raw mined model from event log may contain too many behaviours which might be far beyond the abilities of existing trace clustering techniques. However, by distributing the huge amount of behaviours from the original mined model to several small sub-models (each sub-model contains less behaviours but still might be complex and inaccurate) the trace clustering techniques can provide better results while being applied on these sub-models. On the other hand, the number of activity relations among the clusters is kept as small as possible (which means the relations among the high level activities created are kept as few as possible). As a result, the quality of the potential high level process model is optimised to a large extent because it

contains a limited number of behaviours among its activities.

5.3. A Three-Step Algorithm

In this section, we propose a new process model abstraction algorithm that utilises the strategy introduced in Section 5.2 for solving the problem of "spaghetti-like" process models mined from real-life event logs. The proposed algorithm consists of three main stages. Let $\ddot{\Lambda} : (L^+, V_r^+) \longrightarrow G^+$ be the casual activity graph building method introduced in Chapter 2, where L^+ is the set of event logs, V_r^+ is the set of values of thresholds for judging casual relations among activities and G^+ is the set of casual activity graphs, $\ddot{\Gamma} : G^+ \longrightarrow C_{ac}^+$ be the graph clustering algorithm from [134, 135][1], where C_{ac}^+ is the set of all possible sets of activity clusters. The details of our method is described in Algorithm 5.1.

Algorithm 5.1 Abstract the raw mined model

Input: an event log L, the threshold ϕ for judging the causal relations among activities, a threshold α for judging if a high level activity generated should be removed or not, a threshold β for searching for merging modes, a sub-model complexity threshold τ and a sub-model accuracy threshold χ, a trace number threshold κ, the number of clusters n.

1: **Let** G be a casual activity graph.
2: **Let** C_{ac} be a set of activity clusters.
3: $G \leftarrow null$
4: $C_{ac} \leftarrow null$
5: $G \leftarrow \ddot{\Lambda}(L, \phi)$ # build the casual activity graph
6: $C_{ac} \leftarrow \ddot{\Gamma}(G)$ # mine the activity clusters
7: ***Stage 1:*** Find multi-cluster activities and extract sublogs.
8: ***Input:*** L, C_{ac}.
9: ***Output:*** a new set of activity clusters C_{ac}^M, a set of sublogs SSL.
10: ***Stage 2:*** Generate high level activities and high level model.
11: ***Input:*** SSL, L, α, β.
12: ***Output:*** a high level model M_h, a set of high level activities SA_H, a set of sublogs SSL^H.
13: ***Stage 3:*** Deal with complex and inaccurate sub-models derived from SSL^H.
14: ***Input:*** SSL^H, τ, χ, κ, n.
15: ***Output:*** a set of sub-models SUM.

Output: a high level model M_h, a set of sub-models SUM.

[1]The main reason to select this graph clustering technique is that it is able to automatically generate a suitable number of clusters of vertices according to the edge structure of a graph and also has a good performance

5.3.1. Find Multi-Cluster Activities and Extract Sub-Logs

In this subsection, we make the assumption that a set of activity clusters $C_{ac} = \{c_1, c_2, \ldots, c_m\}$ for event log L has been acquired by Algorithm 5.1. Sometimes, an activity $a \in c_k \in C_{ac}$ may also have a lot of casual relations with the activities from other clusters. For instance, in the casual activity graph G from Figure 5.1, the activity a that pertains to cluster C is also connected to many activities in cluster A. In the graph clustering research area most of the classical methods developed presume that a vertice of a graph only belongs to one specific cluster. The graph clustering algorithm utilised in our approach also has the same assumption. However, it is a normal situation that some activities in a casual activity graph should pertain to more than one clusters according to the edge structure of the graph. Based on this fact, we develop a new concept named Multi-Cluster Activity (MCA) which is defined as:

Definition 5.1. *(Multi-Cluster Activity)*
Let $\ddot{\Phi} : G^+ \longrightarrow V_d^+$ be a graph density calculation mechanism, where G^+ is the set of casual activity graphs and V_d^+ is the set of values of graph density. Given a set of activity clusters $C_{ac} = \{c_1, c_2, \ldots, c_n\}$, an activity $a \in c_k \in C_{ac}$ is a MCA if $\exists c_m \in C_{ac}$ such that $\ddot{\Phi}(G_m^{'}) \geq \ddot{\Phi}(G_m)$, where $G_m = (V_m, E_m)$ represents the casual activity graph built by using the activities from activity cluster c_m and $G_m^{'} = (V_m \cup a, E_m^{'})$ is a new graph generated by adding activity a in G_m.

Given a graph $G = (V, E)$, $\ddot{\Phi}(G) = |E|/(|V| \times (|V| - 1))$, where $|E|$ and $|V|$ stand for the total number of edges and the total number of vertices in graph G respectively. The main reason to use graph density for judging a MCA is that densely connected activities are more likely to cause complex process behaviours that cannot be expressed by the utilised BPMD algorithms (GTCA leaves these potential complex behaviours to trace clustering technique). GTCA detects all of the MCAs in C_{ac} and then distributes each of them to the eligible activity clusters in C_{ac} so that a new set of activity clusters C_{ac}^M with MCAs can be generated. For example, let $C_{ac}^{'} = \{c_1, c_2, c_3\}$ be a set of activity clusters mined from event log $L^{'}$, $c_1 = \{a, b, c\}$, $c_2 = \{d, e\}$ and $c_3 = \{f, g, h\}$, pretend that $\ddot{\Phi}(G_{c_2}) = 0.5$, $\ddot{\Phi}(G_{c_3}) = 0.8$, $\ddot{\Phi}(G_{c_2}^-) = 0.63$ and $\ddot{\Phi}(G_{c_3}^-) = 0.7$, where G_{c_2} is the casual graph for cluster c_2, G_{c_3} for cluster c_3, $G_{c_2}^-$ is the casual graph generated by adding activity $a \in c_1$ in G_{c_2} and $G_{c_3}^-$ generated by adding activity a in G_{c_3}. According to Definition 5.1, a is a MCA because $\ddot{\Phi}(G_{c_2}^-) > \ddot{\Phi}(G_{c_2})$. Afterwards, a new activity cluster $c_2^{'} = \{a, d, e\}$ is generated by adding a in c_2. Activity a should not be added in c_3 because $\ddot{\Phi}(G_{c_3}^-) < \ddot{\Phi}(G_{c_3})$. Let's presume that a is the only MCA found, then the new set of activity clusters $C_{ac}^{M'} = \{c_1, c_2^{'}, c_3\}$ can be generated.

An intuitive proof about the benefit for locating MCAs is shown in the example in Figure 5.1. We assume that the activity a in cluster C is a MCA corresponding to cluster A. By adding a to cluster A the original casual graph G can

be transformed into $G^{'}$ as shown in Figure 5.2. In $G^{'}$, the interrelations between cluster A and C are further decomposed which helps improve the quality of the potential high level model.

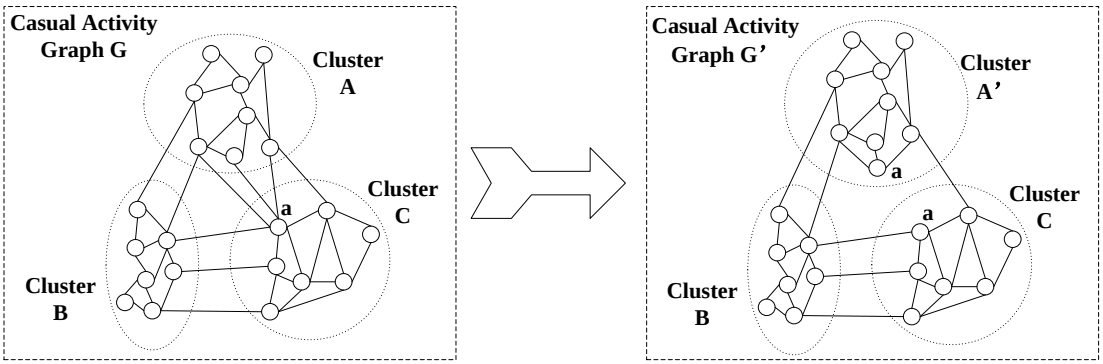


Figure 5.2.: Further decompose the interrelations between cluster A and C.

Definition 5.2. *(Activity Cluster-Related Sub-Trace)*
Let L be an event log, $C_{ac}^{M} = \{c_1, c_2, \ldots, c_n\}$ be the set of activity clusters with MCA mined from L, a sub-trace $st \sqsubseteq t \in L$ is called c_m-related sub-trace if $\forall a \in A_{st}$ such that $a \in c_m$, where A_{st} is the set of activities for st. Furthermore, st is called a maximal c_m-related sub-trace of t if $\nexists a \in c_m$ such that $<a, st> \sqsubseteq t$ or $<st, a> \sqsubseteq t$.

Definition 5.3. *(Activity Cluster-Related Sublog)*
Let L be an event log, $C_{ac}^{M} = \{c_1, c_2, \ldots, c_n\}$ be the set of activity clusters with MCA mined from L, a sublog SL is called c_m-related sublog if SL contains all the maximal c_m-related sub-traces that can be discovered in L.

Whereafter, the stage 1 of Algorithm 5.1 creates a cluster-related sublog (Definition 5.3) for each activity cluster in $C_{ac}^{M} = \{c_1, c_2, \ldots, c_n\}$. For example, for the activity cluster $c_k \in C_{ac}^{M}$ a new sublog SL_{c_k} is built which contains all of the maximal c_k-related sub-traces extracted from the original event log L. For instance, let $C_{ac}^{M'} = \{\{a,b,v,c,d\},\{u,v,x,z\}\}$ be a set of activity clusters generated by stage 1 of Algorithm 5.1 executed on an event log $L^{'} = \{<a,b,c,d,v,x,z>^{80}, <a,c,d,u,v,x,z>^{150}, <a,b,v,c,d,u,v,z>^{200}\}$ (pretend that v is a MCA). For the first activity cluster $\{a,b,v,c,d\} \in C_{ac}^{M'}$ a new sublog $SL_1^{'} = \{<a,b,c,d,v>^{80}, <a,c,d>^{150}, <a,b,v,c,d>^{200}\}$ can be created. Similarly, the sublog $SL_2^{'} = \{<v,x,z>^{80}, <u,v,x,z>^{150}, <u,v,z>^{200}\}$ can be generated for the second activity cluster $\{u,v,x,z\}$.

5.3.2. Generate High Level Activities and High Level Process Model

We presume that the set of activity cluster-related sublogs $SSL = \{SL_1, SL_2, \ldots, SL_n\}$ has been output by the stage 1 of Algorithm 5.1 for log L. Let $\ddot{\Psi} : L^{+} \longrightarrow SL^{+}$ be

a function which splits an event log into several sublogs where each sublog contains traces with the same start activity and end activity, where L^+ represents the set of event logs and SL^+ represents the set of all possible sets of sublogs. Take the simple event log $L^{'} = \{<a,b,c>^{15}, <a,d,c>^{15}, <a,f>^{3}, <a,e,d>^{5}\}$ as an example, $\ddot{\Psi}(L^{'}) = \{SL_1^{'}, SL_2^{'}, SL_3^{'}\}$, where $SL_1^{'} = \{<a,b,c>^{15}, <a,d,c>^{15}\}$, $SL_2^{'} = \{<a,f>^{3}\}$ and $SL_3^{'} = \{<a,e,d>^{5}\}$.

Definition 5.4. *(High Level Activity)*
Let $SSL = \{SL_1, SL_2, \ldots, SL_n\}$ be the set of activity cluster-related sublogs generated for log L. For each sublog $SL_m \in SSL$, $\ddot{\Psi}(SL_m) = \{SL_1^H, SL_2^H, \ldots, SL_p^H\}$. A high level activity a_q^H is an artificially created activity which is utilised to represent all the sub-traces from sublog SL_q^H.

Definition 5.5. *(High Level Process Model)*
Let L be an event log, SA_L^H be the set of high level activities discovered from L, L^H be the event log generated by using the high level activities from SA_L^H to replace all their related sub-traces in L. A high level process model M_h is defined as a model mined from L^H by employing a certain process model discovery algorithm.

The high level activity generation method for the stage 2 of Algorithm 5.1 is depicted in Algorithm 5.2. To explain Algorithm 5.2 explicitly, an example is employed here (which is also utilised for the rest part of this subsection). Let $C_{ac}^{M'} = \{\{a,b,c,d\}, \{u,v,x,z\}\}$ be a set of activity clusters generated by stage 1 of Algorithm 5.1 executed on an event log $L^{'} = \{<a,b,d,u,x,z>^{100}, <a,b,c,d,v,x,z>^{80}, <a,c,d,u,v,x,z>^{150}, <a,b,v,c,d,u,x,z>^{8}\}$, $SSL^{'} = \{SL_1^{'}, SL_2^{'}\}$ be a set of sublogs generated by stage 1 of Algorithm 5.1 with inputs $C_{ac}^{M'}$ and $L^{'}$, where sublog $SL_1^{'} = \{<a,b,d>^{100}, <a,b,c,d>^{80}, <a,c,d>^{150}, <a,b>^{8}, <c,d>^{8}\}$, sublog $SL_2^{'} = \{<u,x,z>^{108}, <v,x,z>^{80}, <u,v,x,z>^{150}, <v>^{8}\}$. A set of sublogs $SSL^{H'} = \{\{<a,b,d>^{100}, <a,b,c,d>^{80}, <a,c,d>^{150}\}_0, \{<a,b>^{8}\}_1, \{<c,d>^{8}\}_2, \{<u,x,z>^{108}, <u,v,x,z>^{150}\}_3, \{<v,x,z>^{80}\}_4, \{<v>^{8}\}_5\}$ can be generated if $SSL^{'}$ is directly dealt with by steps 11–13 of Algorithm 5.2 (replace the set SSL^M in step 11 by using $SSL^{'}$). Afterwards, according to steps 14–16 of Algorithm 5.2 a set of high level activities $SA^{H'} = \{H\text{–}Activity(0)^{330}, H\text{–}Activity(1)^{8}, H\text{–}Activity(2)^{8}, H\text{–}Activity(3)^{258}, H\text{–}Activity(4)^{80}, H\text{–}Activity(5)^{8}\}$ is generated where each high level activity is related to a specific sublog in $SSL^{H'}$. In our method, a high level activity will replace all the sub-traces that exist in its relevant sublog in $SSL^{H'}$ in the original event log $L^{'}$. For instance, the high level activity $H\text{–}Activity(0)$ will replace all the sub-traces from sublog $\{<a,b,d>^{100}, <a,b,c,d>^{80}, <a,c,d>^{150}\}_0$ in $L^{'}$. Ultimately, a high level event log $L_h^{'} = \{<H\text{–}Activity(0), H\text{–}Activity(3)>^{100}, <H\text{–}Activity(0), H\text{–}Activity(4)>^{80}, <H\text{–}Activity(0), H\text{–}Activity(3)>^{150}, <H\text{–}Activity(1), H\text{–}Activity(5), H\text{–}Activity(2), H\text{–}Activity(3)>^{8}\}$ is acquired. The steps 17–22 of Algorithm 5.2 remove all the infrequent high level activities generated

Algorithm 5.2 Generate high level activities

Input: the set of sub-logs SSL, the original log L, a threshold α, a threshold β.

Let $\ddot{\Upsilon}$ be a trace merging technique which is described in Algorithm 5.3.
Let SSL^H be a set of sub-logs where each sub-log $SL_q \in SSL^H$ is relevant to one potential high level activity.
Let SA^H be a set of high level activities.
Let SSL^M be a set of event logs with merged traces.
$SSL^H \leftarrow null$
$SA^H \leftarrow null$
$SSL^M \leftarrow null$
for each log $SL_k \in SSL$ **do**
    $SSL^M \leftarrow SSL^M \cup \ddot{\Upsilon}(SL_k, SSL, L, \beta)$
end for
for each log $SL_p^M \in SSL^M$ **do**
    $SSL^H \leftarrow SSL^H \cup \ddot{\Psi}(SL_p^M)$
end for
for each log $SL_q^H \in SSL^H$ **do**
    $SA^H \leftarrow SA^H \cup H\text{–}Activity(q)$
end for
for each $H\text{–}Activity(l) \in SA^H$ **do**
    if $|H\text{–}Activity(l)| < \alpha$ **then**
        remove $H\text{–}Activity(l)$ from SA^H
        remove SL_l^H from SSL^H
    end if
end for

Output: the set of high level activities SA^H, the set of sub-logs SSL^H.

together with their relevant sublogs in $SSL^{H'}$. Removing infrequent activities which is in accordance with the main idea of most advanced process model mining techniques can make the potential model mined concentrate on exhibiting the most frequent process behaviours. In our example, given a threshold α = 20, the high level activity $H\text{–}Activity(1)$, $H\text{–}Activity(2)$ and $H\text{–}Activity(5)$ are removed from $SA^{H'}$ and $L_h^{'}$ because the value of their frequency is eight which is smaller than α. At the same time, the sub-logs $\{< a, b >^8\}$, $\{< c, d >^8\}$ and $\{< v >^8\}$ are removed from $SSL^{H'}$. Afterwards, a high level model can be built by mining the generated high level event log $L_h^{'}$ with an existing process model discovery algorithm (this is the way for GTCA to generate a high level model). Each sublog in $SSL^{H'}$ will be used to build a sub-model for indicating the details of its relevant high level activity.

Such a design for generating high level activities will help maintain the precision [9] (precision quantifies the ratio of behaviours that can be generated by the mined models which are also recorded in the event logs) of the potential high

level model together with the sub-models generated compared to the precision of the model mined by using the original log $L^{'}$. Furthermore, GTCA might generate a huge amount of high level activities while encountering event logs that have casual graphs with uniform structures. So we make the assumption that the casual graphs of event logs processed by our method have structures with natural clusters.

Three infrequent high level activities ($H\text{–}Activity(1)$, $H\text{–}Activity(2)$ and $H\text{–}Activity(5)$) are generated in the example introduced above. This is because activity v happens between activity b and c in some traces in $L^{'}$ infrequently and v belongs to a different activity cluster from b and c. As a result, three kinds of infrequent sub-trace $<a, b>$, $<v>$ and $<c, d>$ in $SSL^{H'}$ are generated by GTCA. The Algorithm 5.2 will remove all infrequent high level activities and also the sublogs related to these activities. A lot more activities like activity v might lead to the situation that a huge amount of process behaviours in the original event logs will get lost because of being distributed into many infrequent sublogs in SSL^{H} which then will be removed. In this subsection, we propose a sub-trace merging approach Ÿ (which appears in the step 9 of Algorithm 5.2 and helps preserve the process behaviours recorded in the original logs as many as possible) for fixing this problem by employing the following definitions:

Definition 5.6. *(Merging Mode)*
Let $SSL = \{SL_1, SL_2, \ldots, SL_n\}$ be a set of sublogs output by stage 1 of Algorithm 5.1 executed on an event log L. Let st_1 and st_2 be two sub-traces from $SL_k \in SSL$, sa_1 and ea_1 be the starting and ending activity of st_1 respectively, sa_2 and ea_2 be the starting and ending activity of st_2 respectively. The pair (st_1, st_2) is called a merging mode for SL_k if (1) the number of traces in SL_k which have sa_1 as starting activity and ea_1 as ending activity at the same time is smaller than $\beta \times |SL_k|$, (2) the number of traces in SL_k which have sa_2 as starting activity and ea_2 as ending activity at the same time is smaller than $\beta \times |SL_k|$, (3) st_1 and st_2 appear in the same trace from L in the way $<st_1, \ldots, st_2>$, (4) the number of traces in SL_k which have sa_1 as starting activity and ea_2 as ending activity at the same time is larger than or equal to $\beta \times |SL_k|$.

Definition 5.7. *(Minimum Merging Mode)*
Let (st_1, st_2) be a merging mode for a sublog $SL_k \in SSL$, sa_1 be the starting activity of st_1 and ea_2 be the ending activity of st_2, $<st_1, \ldots, st_2>$ be a sub-trace from the original log L. The merging mode (st_1, st_2) is called a minimum merging mode if there exists no other merging modes in the sub-trace $< st_1, \ldots |st_2 >$ or in the sub-trace $< st_1 | \ldots, st_2 >$, where $< st_1, \ldots |st_2 >$ represents a sub-trace generated by removing st_2 from $< st_1, \ldots, st_2 >$ and $< st_1 | \ldots, st_2 >$ by removing st_1 from $<st_1, \ldots, st_2>$.

For the example mentioned above, given a threshold $\beta = 0.05$, the pair $(<a, b>, < c, d >)$ from $SL_1^{'}$ is a merging mode (there are eight of such merging modes)

because there are 330 traces in $SL_1^{'}$ that have activity a as starting activity and activity d as ending activity which is larger than $\beta \times |SL_1^{'}| = 17.3$. In the meantime, $| < a, b > | = 8 < 17.3$ and $| < c, d > | = 8 < 17.3$. Furthermore, the way for the sub-traces $< a, b >$ and $< c, d >$ to appear in the trace $< a, b, v, c, d, u, x, z >$ from $L^{'}$ also satisfies the condition proposed in Definition 5.6. Furthermore, the merging mode $(< a, b >, < c, d >)$ is also a minimum merging mode according to Definition 5.7.

Algorithm 5.3 Merging sub-traces (Ÿ)

Input: the set of sublogs SSL, a sub-log $SL_k \in SSL$, a threshold β.

1: **Let** SMD be a set of merging modes.
2: $SMD \leftarrow null$
3: **for** each sub-trace $st_p \in SL_k$ **do**
4: **if** st_p doesn't pertain to any merging mode in SMD **then**
5: **if** there is a sub-trace $st_q \in SL_k$ and (st_p, st_q) is a merging mode **then**
6: put (st_p, st_q) in SMD
7: put the related sub-trace $< st_p, \ldots, st_q >$ from L in SL_k
8: remove st_p and st_q from SL_k
9: remove the sub-traces that appear between st_p and st_q in
10: $< st_p, \ldots, st_q >$ from their original places in SSL
11: **end if**
12: **else**
13: *continue*
14: **end if**
15: **end for**

Output: the sublog SL_k with merged traces.

With the two definitions created above, the details of the sub-trace merging technique Ÿ is described in Algorithm 5.3. Here we still use the last example to explain how Ÿ works. As is shown that three infrequent high level activities are generated by running the Algorithm 5.2 directly from step 11 in our example. One intuitive method to solve this problem is to find all minimum merging modes in $SSL^{'}$ and then merge the sub-traces in the same merging mode (reflected by the steps 3–15 of Algorithm 5.3 and the steps 8–13 of Algorithm 5.2). For example, eight merging modes $(< a, b >, < c, d >)^8$ for $SL_1^{'}$ can be constituted (given a threshold $\beta = 0.05$) and each pair of the sub-traces should be merged into a single sub-trace $< a, b, v, c, d >$ (eight of such merged sub-traces can be generated). Then, a new set of sub-logs $SSL^{M'} = \{SL_1^{M'}, SL_2^{M'}\}$ can be formed, where $SL_1^{M'} = \{< a, b, d >^{100}, < a, b, c, d >^{80}, < a, c, d >^{150}, < a, b, v, c, d >^{8}\}$ and $SL_2^{M'} = \{< u, x, z >^{108}, < v, x, z >^{80}, < u, v, x, z >^{150}\}$ ($SL_2^{M'}$ doesn't contain the kind of sub-trace $< v >$ any more because all of them are merged into the kind of sub-trace $< a, b, v, c, d >$ in $SL_1^{M'}$). Afterwards, by using the steps 11–22 of Algorithm 5.2 to deal with the $SSL^{M'}$ a new set of sub-logs $SSL^{H'} = \{\{< a, b, d >^{100}, < a, b, c, d >^{80}, < a, c, d >^{150}, < a, b, v, c, d >^{8}\}_0, \{< u, x, z >^{108}, < u, v, x, z >^{150}\}_1, \{< v, x, z >^{80}\}_2\}$ and a new set of high

level activities $SA^{H'} = \{H\text{–}Activity(0)^{338}, H\text{–}Activity(1)^{258}, H\text{–}Activity(2)^{80}\}$ can be generated. Now no infrequent high level activities exist in $SA^{H'}$ any longer.

5.3.3. Deal With Complex and Inaccurate Sub-Models

In this subsection, we presume that a set of sub-logs SSL^H has been output by the stage 2 of Algorithm 5.1. For each sub-log in SSL^H a sub-model is mined with existing BPMD technique to depict the details of the sub-log's relevant high level activity. In our approach, the business process recorded in an event log is expressed by the generated high level model and the sub-models together. However, the strategy (mentioned in Section 5.2) used by GTCA tries to decrease the number of behaviours in the potential high level model by hiding most of the original process behaviours inside the high level activities generated. As a result, the sub-models for the high level activities might still be complex and inaccurate. Trace clustering technique is utilised for solving this problem.

Let $\Omega : L^+ \rightarrow M^+$ be a BPMD algorithm, where M^+ is the set of process models and L^+ is the set of event logs, $\Sigma_f : (L^+, M^+) \rightarrow V_f^+$ be a process model accuracy evaluation method, where V_f^+ is the set of accuracy values for the mined process models, $\Sigma_c : M^+ \rightarrow V_c^+$ be a process model complexity evaluation method, where V_c^+ is the set of complexity values for the mined process models. Let $\widehat{T} : (L^+, V_n^+) \rightarrow SSL^+$ be a trace clustering algorithm, where SSL^+ is the set of all sets of sublogs and V_n^+ is the set of numbers for the generated clusters.

Definition 5.8. *(Low-Quality Process Model)*
Let L be an event log, τ be a model complexity threshold, χ be a model accuracy threshold, $M_L = \Omega(L)$ be the process model mined from L. M_L is called a low-quality process model if $\Sigma_c(M_L) > \tau$ or $\Sigma_f(M_L, L) < \chi$.

The main procedure for dealing with low-quality sub-models mined is depicted in Algorithm 5.4 according to which, a sublog SL from SSL^H that leads to a low-quality sub-model M (the quality is judged by using the sub-model accuracy and complexity thresholds χ and τ in the step 8 of Algorithm 5.4) will be divided into n sub-sub-logs by using the trace clustering technique if the number of the traces inside SL is larger than or equal to a threshold κ. Afterwards, for each sub-sub-log, a sub-sub-model is built (in the step 11 of Algorithm 5.4). If the weighted average accuracy of the sub-sub-models generated is larger than or equal to the accuracy of the original sub-model then these sub-sub-models are added to the set of sub-models SSM which will be finally output by Algorithm 5.4 (Algorithm 5.4 will not use the sub-sub-models if their weighed average accuracy is lower than the accuracy of their related original sub-model). If a sublog SL' from SSL^H leads to a good-quality sub-model M' then add M' in SSM (step 25 of Algorithm 5.4).

Algorithm 5.4 Deal with low-quality sub-models

Input: a set of sub-logs SSL^H, a sub-model complexity threshold τ and a sub-model accuracy threshold χ, a trace number threshold κ, cluster number n.

Let SSM, SSM_c be two sets of sub-models.
Let SSL be a set of sublogs.
Let m_1, m_2 be two variants of float type.
Let m_3 be a variant of int type.
$SSM \leftarrow null$, $SSM_c \leftarrow null$, $SSL \leftarrow null$
$m_1 \leftarrow 0$, $m_2 \leftarrow 0$, $m_3 \leftarrow 0$
for each sublog $SL \in SSL^H$ **do**
    if $\Sigma_f(\Omega(SL), SL) < \chi \ \ || \ \Sigma_c(\Omega(SL)) > \tau$ && $|SL| \geq \kappa$ **then**
        $SSL = \widehat{T}(SL, n)$
        for each sub-log $SL_1 \in SSL$ **do**
            $SSM_c \leftarrow SSM_c \cup \Omega(SL_1)$
            $m_1 \leftarrow m_1 + \Sigma_f(\Omega(SL_1), SL_1) \times |SL_1|$
            $m_3 \leftarrow m_3 + |SL_1|$
        end for
        $m_2 \leftarrow m_1/m_3$
        if $m_2 \geq \Sigma_f(\Omega(SL), SL)$ **then**
            for each sub-model $SM_c \in SSM_c$ **do**
                $SSM \leftarrow SSM \cup SM_c$
            end for
        else
            $SSM \leftarrow SSM \cup \Omega(SL)$
        end if
        $m_1 \leftarrow 0$, $m_3 \leftarrow 0$, $SSM_c \leftarrow null$
    else
        $SSM \leftarrow SSM \cup \Omega(SL)$
    end if
end for

Output: the set of sub-models SSM.

In Chapter 6, a detailed evaluation on the proposed technique GTCA will be executed by employing three event logs: the repair log from [9], the log of the loan and overdraft approvals process (LOA) from BPIC 2012 and the hospital log from BPIC 2011. Especially, through the evaluation result on the hospital event log, the effectiveness of GTCA for dealing with totally unstructured business processes will be fully exhibited.

6

Evaluation

6.1. Introduction

In the last three chapters we have introduced five techniques (i.e., HIF, TDTC, CTC, MLCC and GTCA) which are devised to help mine more accurate and simpler process models from real-life event logs. We have made short verifications for technique HIF (see Chapter 3), TDTC and CTC (see Chapter 4) by using some simple event logs. For technique MLCC, we have given a relatively detailed experiment to test its effectiveness by employing the hospital log from BPIC 2011 in Chapter 4. In this chapter, more elaborate experiments are designed so as to reveal various aspects about these techniques (including technique HIF, TDTC, CTC and GTCA). These experiments mainly focus on three emphases: the impact on the results brought by parameter settings of these techniques, the performance of these techniques in real-life cases and the comparison with other similar techniques. In Section 6.2, the details of the experiment for technique HIF is introduced. The experiments for technique TDTC and CTC are presented in Section 6.3. Section 6.4 elaborates the experiment process and results for technique GTCA.

6.2. Evaluation on Technique HIF

In Chapter 3, we have put forward a heuristics method named HIF for improving the fitness of process models mined from real-life event logs by HM. The technique HIF inherits the basic idea of MEBS. To testify the correctness of the principle of HIF, we have also performed a short verification experiment in Chapter 3 in which the technique HIF is executed on one example event log. In this section, more elaborate experiment results about HIF are given which reflect various aspects of the characteristics of HIF. In Section 6.2.1, we compare HIF with four classical BPMD techniques by running them on three example event logs. In Section 6.2.2, the technique HIF is executed on four real-life event logs for proving

its practicability in handling real-world problems. In the meantime, we have also tested the impact from the three parameters (the target fitness α, the model fitness improvement threshold β and the threshold for newly added activities μ) for HIF on the quality of the process models output by HIF and the running time of HIF.

6.2.1. Comparison

In this subsection, we compare HIF to four classical BPMD techniques which are HM, IM [25–27], ILPM [24] and Alpha Algorithm (AA) [23] by making use of three example event logs (i.e., L_1, L_2 and L_3) as shown in Figure 6.1. Particularly, the traces in L_3 are extracted from the real-life event log of the loan and overdraft approvals process (LOA) from BPIC 2012. Due to the three example logs are very simple, the target fitness α for HIF is set to 1, the model fitness improvement threshold β for HIF is set to 0[1] and the threshold for the number of newly added activities μ for HIF is set to $+\infty$.

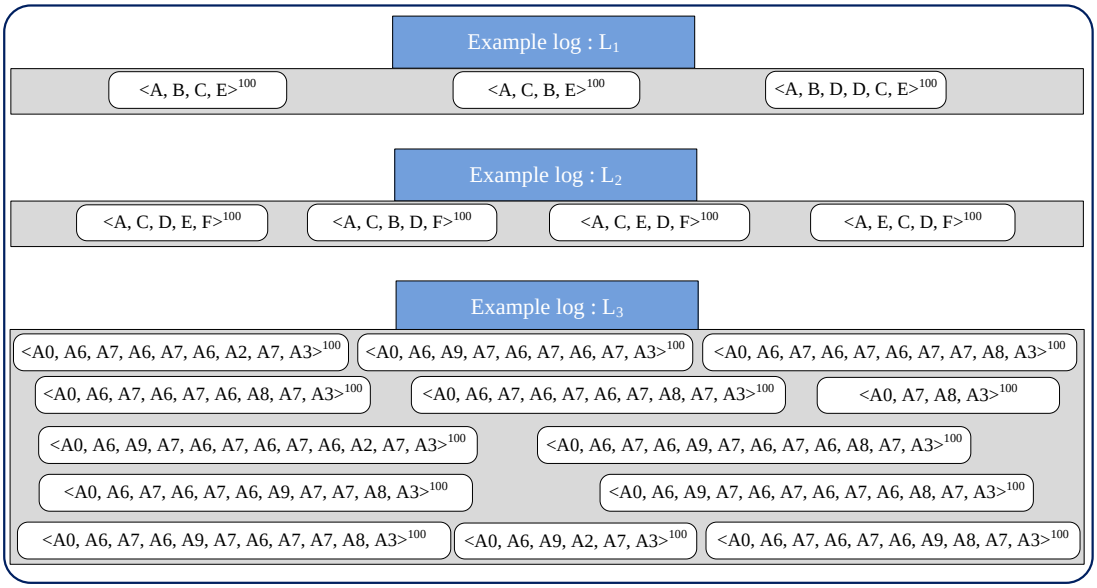


Figure 6.1.: The three example event logs utilised for the comparison between HIF and other BPMD techniques.

The process models built for log L_1 are shown in Figure 6.2. It can be seen that only the models mined by HIF and ILPM are able to replay all the traces recorded in L_1. Furthermore, the model mined by ILPM is better than the model output by HIF because the model built by HIF contains more workflow patterns which are not recorded in L_1. The process models built for log L_2 are exhibited in Figure

[1]The sign "≤" in the rhombus of Module-2 for technique DCIB (shown in Figure 3.5 of Chapter 3) should be transformed into "<" if the model fitness improvement threshold β is set to 0.

6.3 according to which the models constructed by HIF, HM, IM and ILPM are capable of replaying all the traces in L_2. However, the models built by HIF, HM and ILPM are better than the model mined by IM which is able to generate many traces that don't exist in log L_2. Meanwhile, the models from HIF, HM and ILPM can exhibit the parallel relation between activity E and activity C and D. The traces in log L_3 are extracted from the real-life event log LOA (mentioned above). The models (Figure 6.4) mined by HIF, IM and ILPM can express all the traces from L_3 correctly. However, in the process model generated by IM, the relations between activity $A9$, $A8$, $A6$, $A2$ and $A7$ are only implicitly described through a flower model. Additionally, the model mined by ILPM shows a high complexity (compared with the model output by HIF and IM).

As indicated in Chapter 3, we discovered in the experiment that HIF is often able to find the most efficient way to help improve the fitness of the mined models and this is mainly due to the internal operation mechanism of HIF. Nevertheless, there often exist multiple options for improving the fitness of the underlying process model for a certain event log. Semantically speaking, the most efficient way may not be the best way. For example, to help HM mine a high fitness model from log L_1, another way maybe to change activity B under EI (A, D) into $B0$ and activity C under EI (D, E) into $C0$ in log L_1. But HIF chooses to change activity C under EI (B, E) into $C0$ as shown in Figure 6.2. Both options can help mine a highly fitting model. Though the way selected by HIF is the most efficient (only add one new activity) one, it is not the best one because it destroys the parallel relation between activity B and activity C (see Figure 6.2) for acquiring high model fitness and also decreases the precision of the generated model (the mined model generates some traces which are not logged in L_1). The other option descried above is better because it can assist in generating a precise model which preserves the parallel relations between activity B and C. Helping HIF to solve this problem will be one of our primary future jobs.

6.2.2. Experiment on Real-Life Event logs

We tested the effectiveness of HIF on five real-life event logs: the repair log (Repair) from [9], the log of the loan and overdraft approvals process (LOA) from BPIC 2012, the log of Volvo IT incident and problem management (VIPM) from BPIC 2013, the log of CRM process (MCRM) from [36] and the log (LOA1) that is generated by randomly choosing approximate 10% of the traces from log LOA. The basic information about the five logs is shown in Table 6.1.

In our experiment, we employ the HM [37] from ProM 6 for generating the process models and the ICS fitness [29] is used for evaluating the accuracy of the mined models. The *ETConformance Checker* from ProM 6 is utilised for evaluating the influence of our method on the precision [9] of the mined models. Two

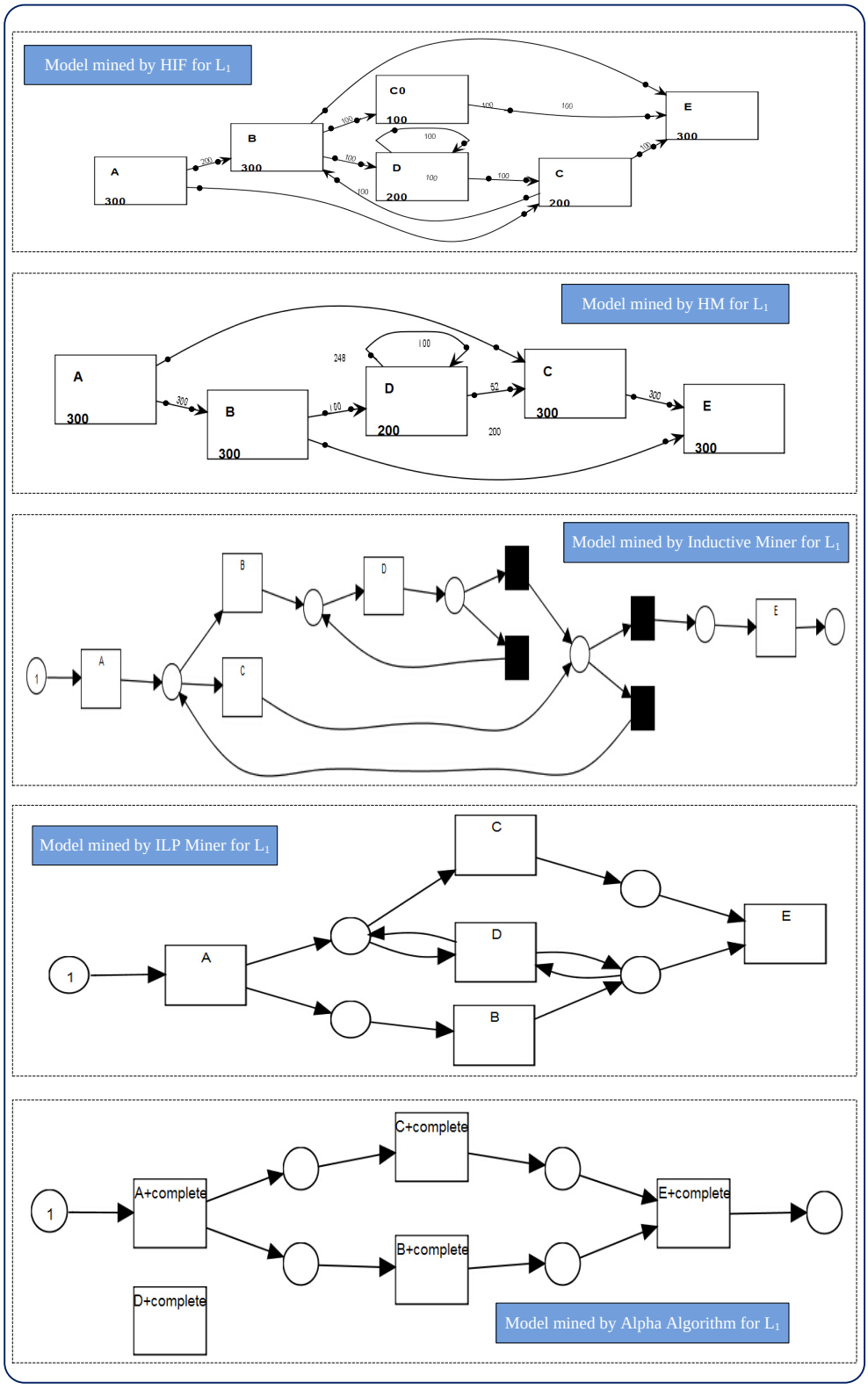


Figure 6.2.: Process models mined from example log L_1.

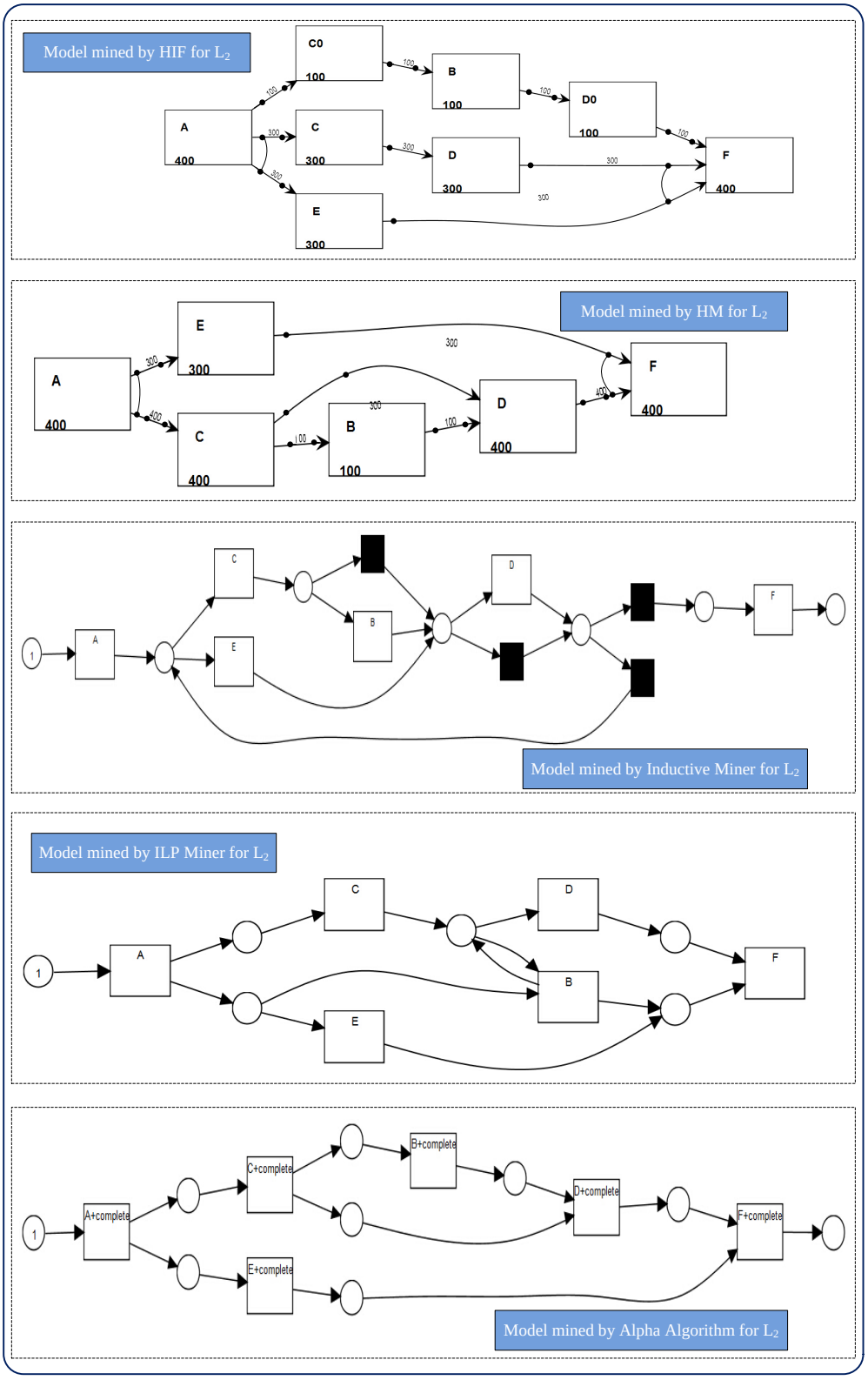


Figure 6.3.: Process models mined from example log L_2.

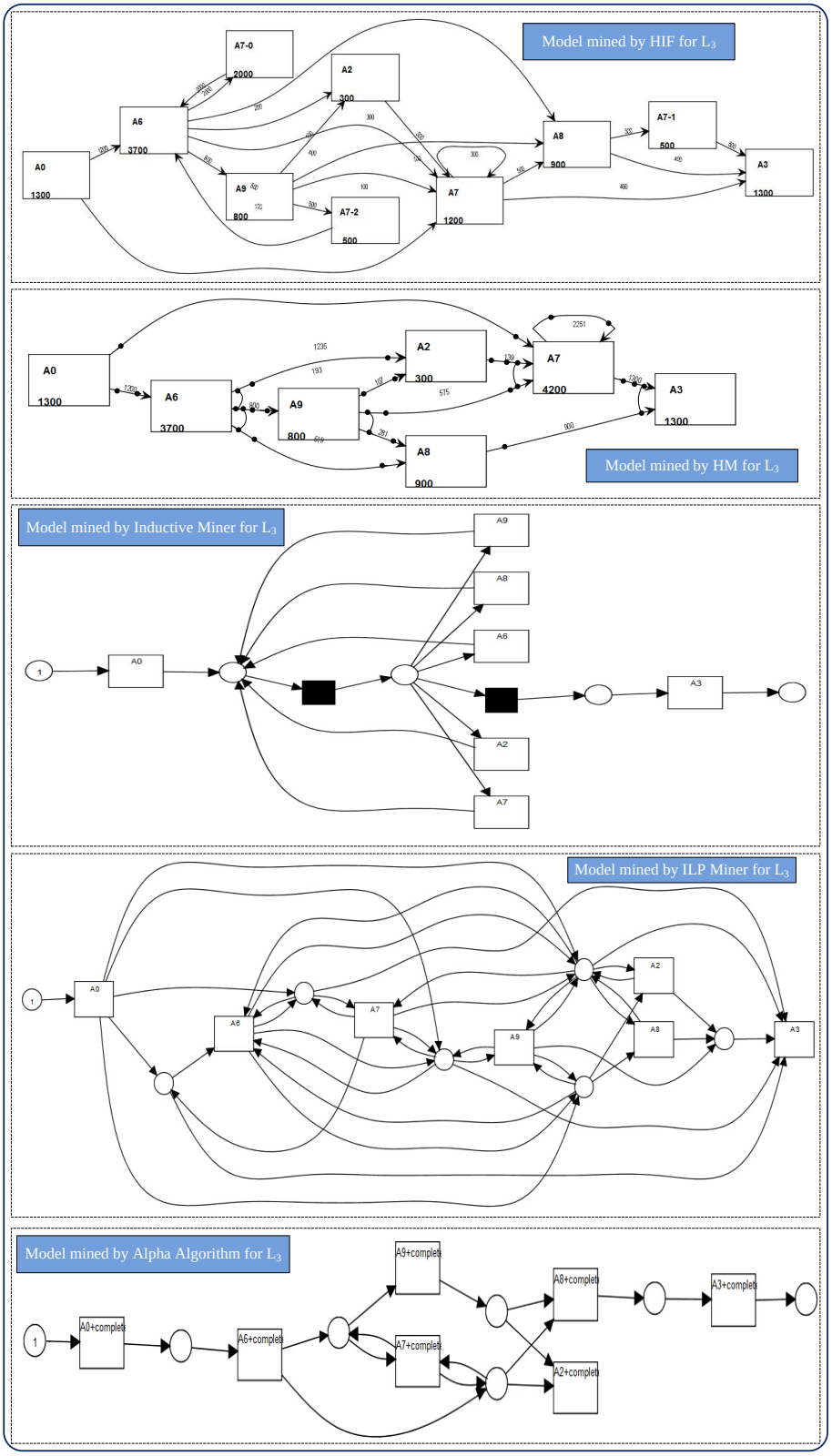


Figure 6.4.: Process models mined from example log L_3.

Table 6.1.: Basic information of the evaluated logs.

Log	Traces	Events	Event types
Repair	1000	10827	12
LOA	13087	262200	36
LOA1	1303	35978	35
MCRM	956	11218	22
VIPM	7554	65533	13

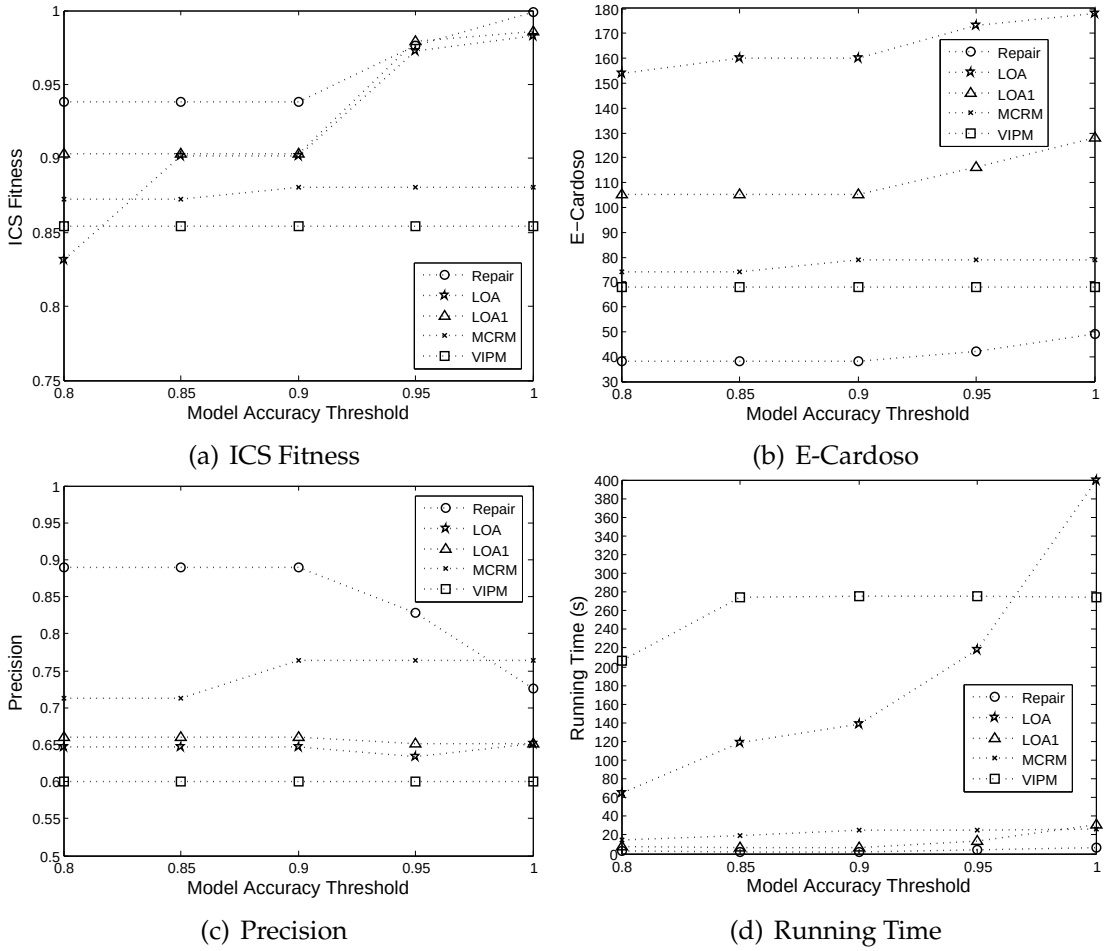
process model complexity evaluation metrics E-Cardoso [177] and PT-CD [36] are employed for evaluating the impact of our method on the complexity of the mined models (the greater the E-Cardoso and PT-CD are, the more complicated the models will be). The *Heuristics Net to Petri Net plugin* in ProM 6 is used for transforming the heuristics net obtained into a Petri net so that the E-Cardoso, the PT-CD and the precision for the generated model can be calculated.

In the preliminary test for HIF on the five logs, the target model fitness α is set to 1, the model fitness improvement threshold β is set to 0.03 and the threshold for the newly added activities μ is set to 0.3. Table 6.2 shows the evaluation results. In Table 6.2, *M–Repair* represents the model generated by directly mining the original event log Repair and $M\text{–}Repair_N$ stands for the model output by HIF (the same applies to the other models). It can be seen that the technique HIF can improve the fitness of the mined models to a large extent, while for most of the models output by HIF their precision and complexity are kept within an acceptable range compared with their original models. The model $M\text{–}Repair_N$ only has four more activities than the model *M–Repair* but the fitness for $M\text{–}Repair_N$ has been greatly improved (the same to the model $M\text{–}VIPM_N$). This benefits from the activity ranking method presented in Section 3.4 of Chapter 3.

Table 6.2.: Evaluation results in the preliminary test on HIF.

Model	ICS-fitness	Precision	E-Cardoso	PT-CD	Event types	Time(s)
M–Repair	0.6768	0.426	31	2.3656	12	
$M\text{–}Repair_N$	**0.9989**	**0.7252**	**49**	**2.3611**	**16**	**6.39**
M–LOA	0.7878	0.6717	148	3.1478	36	
$M\text{–}LOA_N$	**0.9826**	**0.6511**	**178**	**2.9149**	**47**	**399.71**
*M–LOA*1	0.628	0.6605	101	2.4404	35	
$M\text{–}LOA1_N$	**0.9859**	**0.652**	**128**	**2.41**	**45**	**30.878**
M–MCRM	-0.1379	0.7454	64	2.4545	22	
$M\text{–}MCRM_N$	**0.8802**	**0.7641**	**79**	**2.3621**	**29**	**25.953**
M–VIPM	0.3594	0.86	54	2.8848	13	
$M\text{–}VIPM_N$	**0.8539**	**0.6**	**68**	**2.979**	**17**	**274.51**

Furthermore, we also tested the impact from the target model fitness α, the model fitness improvement threshold β and the threshold for the newly added activities μ on the quality of the models output by HIF and the running time



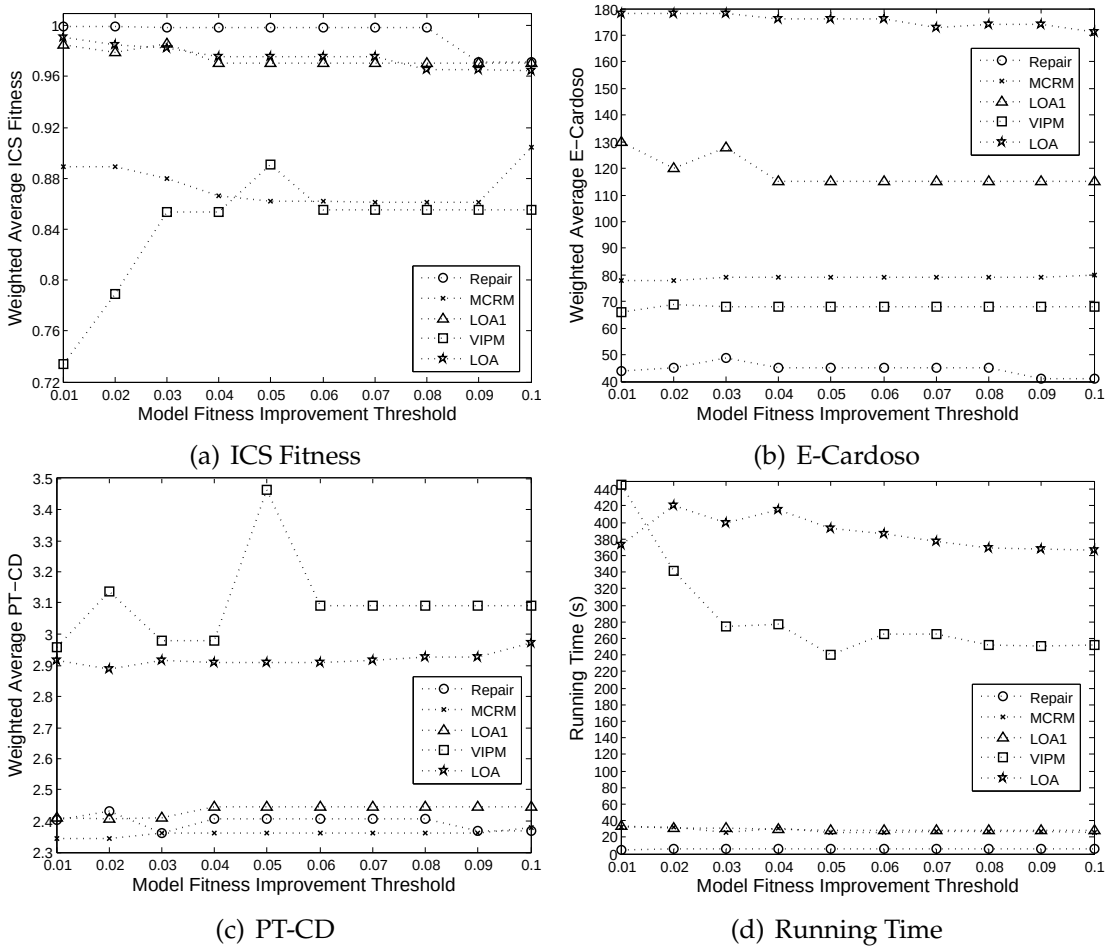
(a) ICS Fitness (b) E-Cardoso

(c) Precision (d) Running Time

Figure 6.5.: Comparison among different values for parameter α for HIF

of HIF. For evaluating the effect from the target model fitness α, five groups of parameters are assigned to HIF for each log which are $(\alpha = 0.8, \beta = 0.03, \mu = 0.3)$, $(\alpha = 0.85, \beta = 0.03, \mu = 0.3)$, $(\alpha = 0.9, \beta = 0.03, \mu = 0.3)$, $(\alpha = 0.95, \beta = 0.03, \mu = 0.3)$ and $(\alpha = 1, \beta = 0.03, \mu = 0.3)$. The evaluation results are shown in Figure 6.5. In most cases, as α increases the complexity of the models (evaluated by E-Cardoso) output by HIF also increases because more new activities are added in the output models. The precision curves remain flat relatively as the value of α increases. Additionally, the running time for HIF grows as the value of α increases and the amplification becomes maximal between the values 0.95 and 1 for most of the cases. According to the analysis results we recommend that the value of α should be set to less than 1 so that a model with a lower complex rate and a next best fitness can be generated by HIF within a shorter time.

Figure 6.6 shows the effects on the mining results of HIF from the model fitness improvement threshold β while Figure 6.7 exhibits the impacts from the threshold μ of newly added activities. By considering the accuracy and complexity of the process models mined by HIF together with the running time of



(a) ICS Fitness (b) E-Cardoso

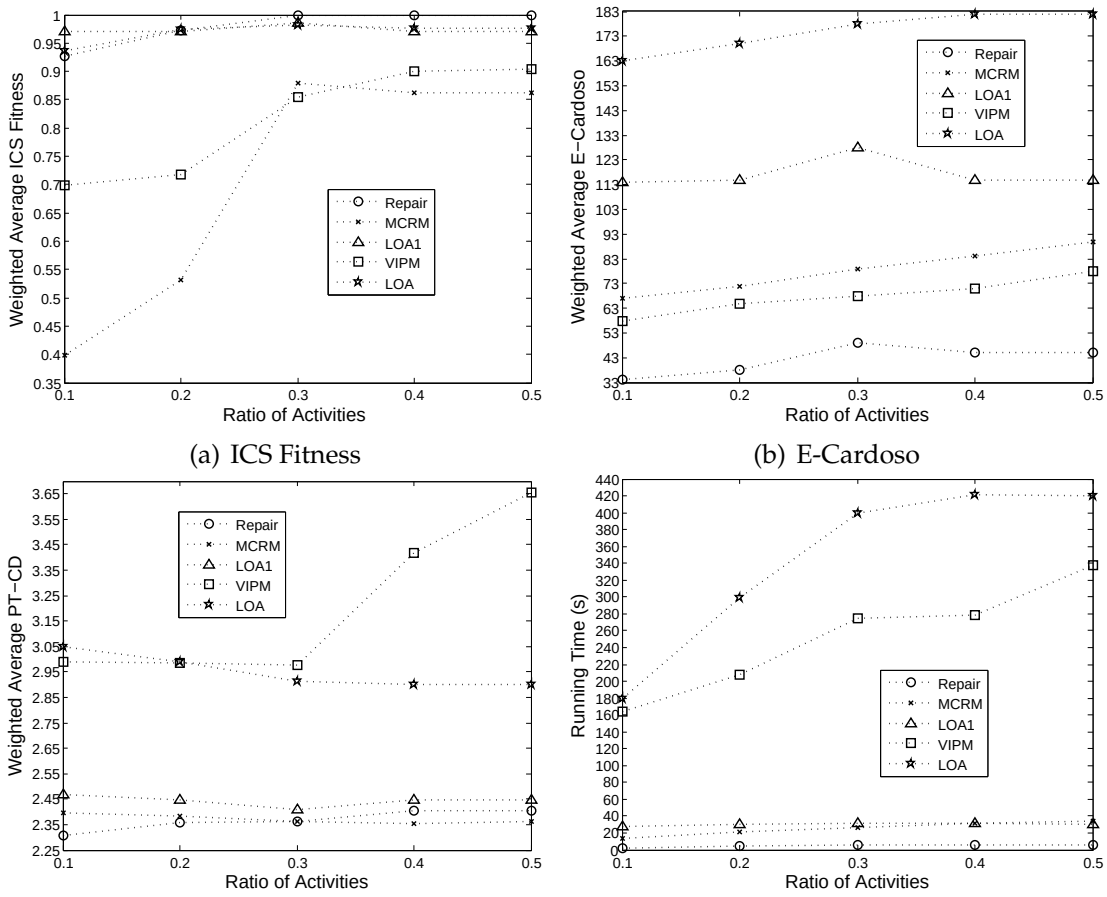
(c) PT-CD (d) Running Time

Figure 6.6.: Comparison among different values for parameter β for HIF

HIF under different parameter settings, the default value 0.03 is suggested for β and the default value for μ is suggested to be 0.3 for HIF.

6.3. Evaluation on Trace Clustering Technique TDTC and CTC

In Chapter 4, two trace clustering techniques TDTC and CTC that inherit the idea from MDS are proposed and then a short verification for these two techniques is given with the help of a repair log. In this section, the performance of TDTC and CTC is further evaluated by employing four real-life event logs. In Section 6.3.1, different parameter settings are tested for both TDTC and CTC on the given event logs so as to help understand the influence of different parameter settings (for TDTC and CTC) on the mining results. Furthermore, a comparison between our techniques and other six classical trace clustering techniques is carried out in



(a) ICS Fitness (b) E-Cardoso

(c) PT-CD (d) Running Time

Figure 6.7.: Comparison among different values for parameter μ for HIF

Section 6.3.2.

6.3.1. Assessment on the Parameter settings for TDTC and CTC

We tested the performance of our technique TDTC and CTC in a full-scale manner on four real-life event logs: the log of Volvo IT incident and problem management (VIPM) from BPIC 2013, the log of loan and overdraft approvals process (LOA) from BPIC 2012, the log of ICT service process (KIM) and the log of CRM process (MCRM) from [36]. The basic information about the four logs is shown in Table 6.3. Moreover, Table 6.4 displays the information about the accuracy and complexity of the process models mined directly from VIPM, LOA, KIM and MCRM by HM.

Firstly, the effectiveness of TDTC and CTC on assisting in mining simple and accurate sub-process models from the four given real-life event logs is certified. The minimum support *min_sup* for the CSPs discovered from log VIPM, KIM

Table 6.3.: Basic information of the evaluated real-life event logs.

Log	Traces	Events	Event types
VIPM	7554	65533	13
LOA	13087	262200	36
KIM	24770	124217	18
MCRM	956	11218	22

Table 6.4.: Evaluation results for the process models mined from the given logs.

Event Log	ICS Fitness	E-Cardoso	PT-CD
VIPM	0.3594	54	2.8848
LOA	0.7878	148	3.1478
KIM	0.7904	79	3.4797
MCRM	-0.1379	64	2.4545

and MCRM is set to 0.1 and the minimum support for the CSPs mined from log LOA is set to 0.25 because the first three logs contain much fewer process variants than LOA.

For TDTC, the fitness weight α = 0.6 and the complexity weight β = 0.4 are set for calculating the SMI for log LOA and VIPM, α = 0.4 and β = 0.6 for log KIM, α = 0.8 and β = 0.2 for log MCRM (according to the accuracy and complexity of the original models mined from the given logs). Furthermore, the minimum threshold μ for SMI is set to 0.03, both the minimum thresholds μ_f and μ_c for SSTB are set to 0 (which means the minimum average improvement on both fitness and complexity should be larger than 0), the minimum threshold φ_f for FCSTB is set to 0.8, the maximum threshold φ_c for CCSTB is set to 2.5 and the minimum size θ for each underlying sublog is set to 50 for TDTC for dealing with the four event logs. Table 6.5 exhibits the evaluation results for TDTC.

For technqiue CTC, the minimum size κ for each potential sublog is also set to 50, the target number μ of generated sublogs (clusters) is set to 5. For the technique HIF employed in CTC, the target fitness α is set to 1, the model fitness improvement threshold β is set to 0.03 and the threshold μ_{HIF} for the number of newly added activities is set to 0.3 (as recommended in Section 6.2). The results of assessment for CTC are displayed in Table 6.6.

It can be seen that the weighted average ICS fitness[2] for the sub-models generated by TDTC and CTC for the four event logs are much higher than the ICS

[2]Let j be the number of sublogs, n_i denotes the number of traces in sublog i where $1 \leq i \leq j$. Let F_i represent the ICS fitness of the process model for sublog i, the weighted average ICS fitness is defined as: $WICS\_f = \frac{\sum_{i=1}^{j} n_i \times F_i}{\sum_{i=1}^{j} n_i}$.

fitness of the original process models (see Table 6.4) directly mined from the four logs. Meanwhile, the average complexity (measured by E-Cardoso and PT-CD) of the sub-models generated by both TDTC and CTC is lower than the original models for the four event logs. Additionally, the average fitness of the sub-models output by CTC is higher than the average fitness of the sub-models output by TDTC because CTC utilises the technique HIF for optimising the accuracy of the generated sub-models. However, average complexity of the sub-models generated by TDTC is lower than the complexity of the sub-models generated by CTC because the technique HIF used by CTC will bring new activities to the sub-models (as introduced in Chapter 3).

Table 6.5.: Evaluation results for the sub-models generated by TDTC.

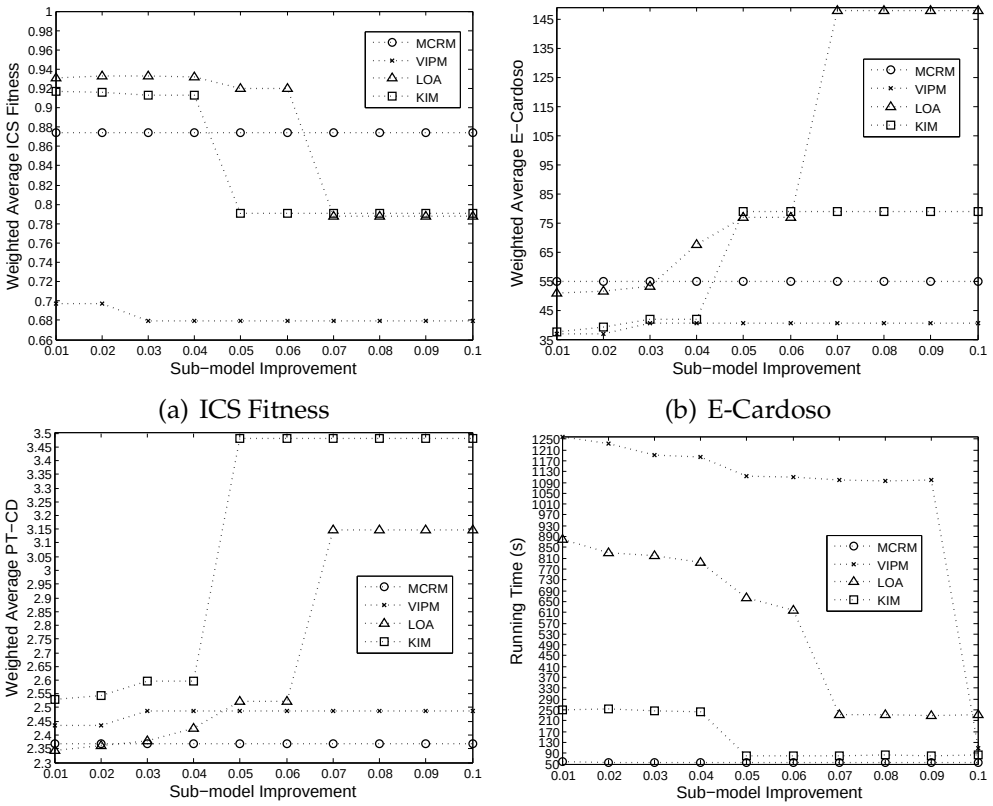
Event Log	Weighted Average ICS Fitness	Weighted Average PT-CD	Weighted Average E-Cardoso
VIPM	0.6791	2.4881	40.5081
LOA	0.9318	2.3803	53.0131
KIM	0.9127	2.5967	41.7958
MCRM	0.8745	2.3677	54.8159

Table 6.6.: Evaluation results for the sub-models generated by CTC.

Event Log	Weighted Average ICS Fitness	Weighted Average PT-CD	Weighted Average E-Cardoso
VIPM	0.9159	2.3577	47.3313
LOA	0.9909	2.4845	110.7463
KIM	0.9461	2.8626	63.2614
MCRM	0.9512	2.2818	51.7364

Next, the impact of parameter settings for TDTC and CTC on their mining results is assessed. For technique TDTC, we mainly focus on there parameters: the minimum threshold μ for SMI (Figure 6.8), the minimum support min_sup (Figure 6.9) for CSPs and the minimum size θ (Figure 6.10) for each underlying sublog. When checking one specific parameter, the values of the other parameters are kept the same as have been set in the last paragraph (the same strategy applies to CTC). For technique CTC, we concentrate on two parameters which are the target number μ_{CTC} of generated sublogs (clusters) and the minimum support min_sup for CSPs.

According to Figure 6.8, as the value of parameter μ for TDTC increases, the average fitness of the obtained sub-models tends to decrease, the average complexity of the sub-models tends to increase and the running time of TDTC tends to decrease. TDTC seems to have a relatively good overall performance when the value of its parameter μ is equal to 0.03. According to Figure 6.9, it takes two long to mine the CSPs from the log LOA when the parameter min_sup for



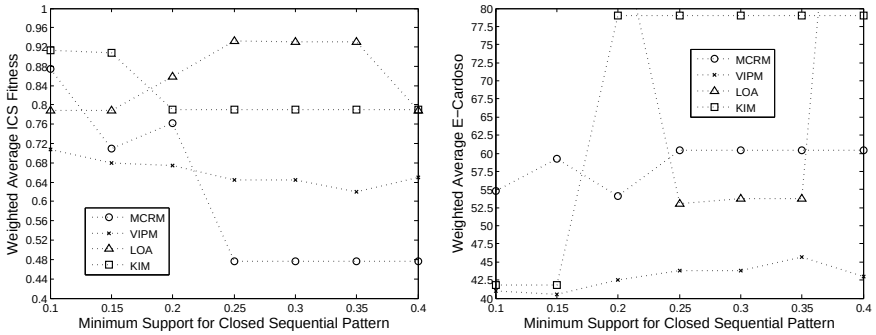
(a) ICS Fitness (b) E-Cardoso

(c) PT-CD (d) Running Time

Figure 6.8.: Comparison among different values for parameter μ for TDTC

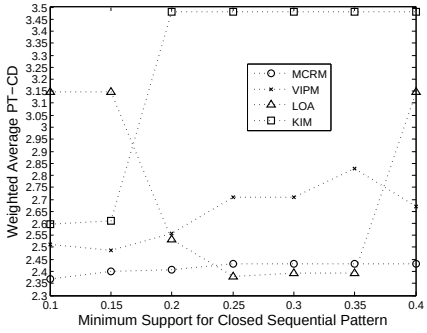
TDTC is set to 0.1 and 0.15. As a result TDTC just returns the original model mined from LOA as the output model. By not considering this exceptional case, the overall trend is that as the value of min_sup increases the average fitness of the generated sub-models decreases, the average complexity of the sub-models increases and the running time of TDTC decreases. Range $[0.1, 0.3]$ seems to be an ideal range of values for parameter *min_sup* of TDTC. Through Figure 6.10, the average fitness of the generated sub-models tends to decrease, the average complexity of the sub-models increases and the running time of TDTC decreases as the value of θ increases. A good choice for the value of parameter θ seems to be 50.

Regarding technique CTC, according to Figure 6.11, as the number of cluster increases, the average fitness of the generated sub-models tends to increase, the average model complexity tends to decrease and the running time for CTC tends to increase. Finally, through Figure 6.12, it can be seen that as the value of *min_sup* for CTC increases, the average fitness of the sub-models generated tends to decrease (by not considering the exceptional case that happens to log LOA as mentioned above), the average model complexity tends to increase and the running time of CTC tends to decrease.

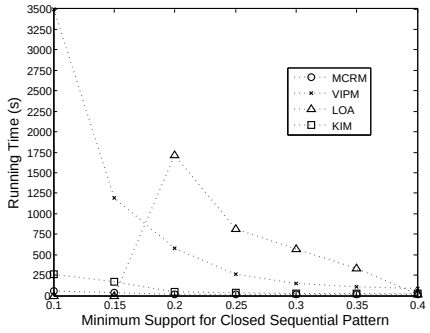


(a) ICS Fitness

(b) E-Cardoso

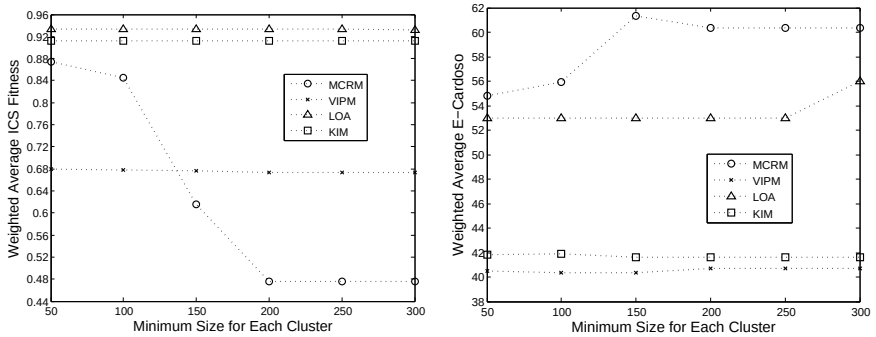


(c) PT-CD



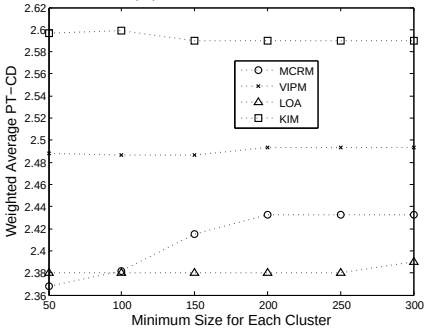
(d) Running Time

Figure 6.9.: Comparison among different values for parameter *min_sup* for TDTC

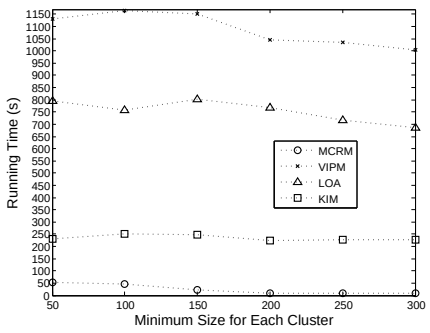


(a) ICS Fitness

(b) E-Cardoso

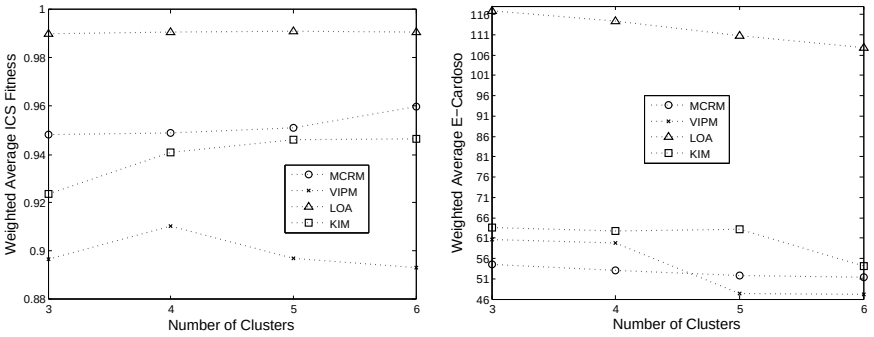


(c) PT-CD



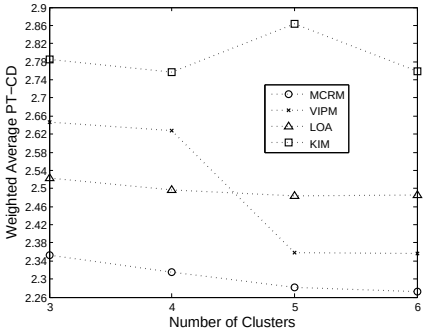
(d) Running Time

Figure 6.10.: Comparison among different values for parameter θ for TDTC

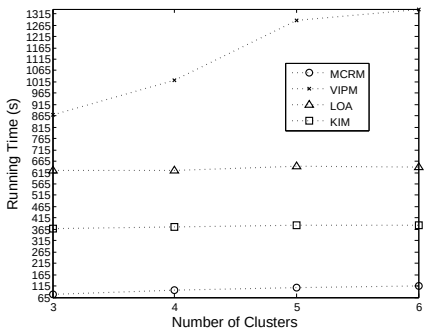


(a) ICS Fitness

(b) E-Cardoso

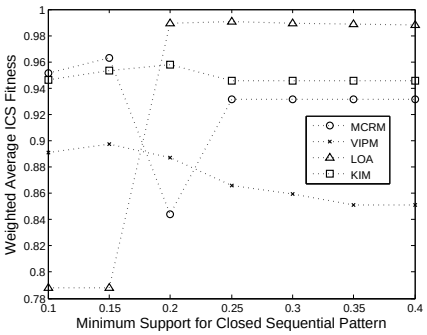


(c) PT-CD

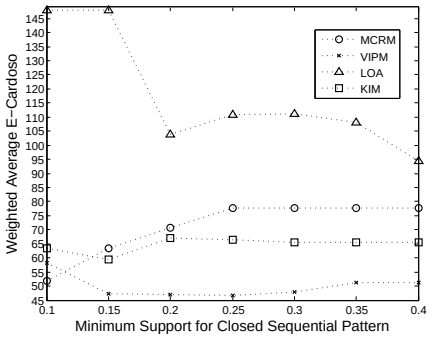


(d) Running Time

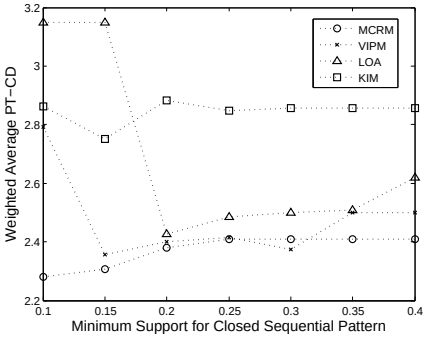
Figure 6.11.: Comparison among different values for parameter μ for CTC



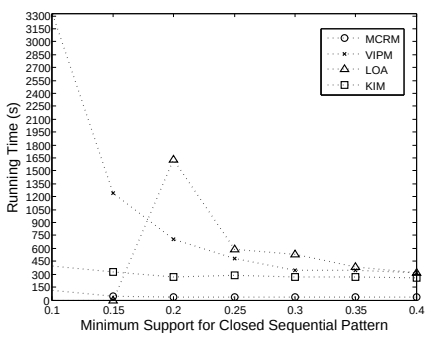
(a) ICS Fitness



(b) E-Cardoso



(c) PT-CD



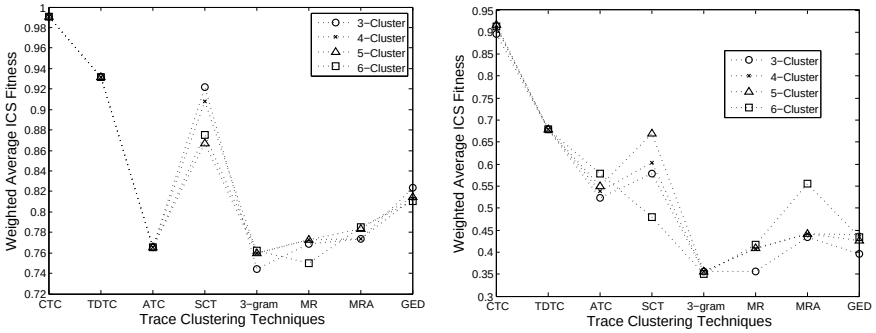
(d) Running Time

Figure 6.12.: Comparison among different values for parameter *min_sup* for CTC

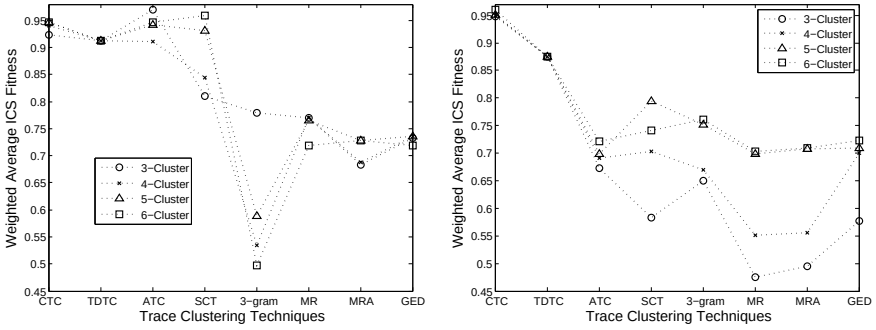
6.3.2. Comparison

We also compared our technique TDTC and CTC to other six classical trace clustering techniques which are 3-gram [31], ATC [36], GED [35], MR and MRA [195] and sequence clustering (SCT) [196] by employing the real-life event log VIPM, LOA, KIM and MCRM. The minimum support min_sup for mining CSPs for both TDTC and CTC is set to 0.25 for log LOA and is set to 0.1 for log VIPM, KIM and MCRM. For TDTC, The fitness weight α = 0.6 and the complexity weight β = 0.4 are set for calculating the SMI for log LOA and VIPM, α = 0.4 and β = 0.6 for log KIM, α = 0.8 and β = 0.2 for log MCRM. Furthermore, the minimum threshold μ for SMI is set to 0.03, both the minimum thresholds μ_f and μ_c for SSTB are set to 0, the minimum threshold φ_f for FCSTB is set to 0.8, the maximum threshold φ_c for CCSTB is set to 2.5 and the minimum size θ for each underlying sublog (cluster) is set to 50. For technqiue CTC, the minimum size κ for each potential sublog (cluster) is set to 50. For the technique HIF utilised by CTC, its target fitness α_{HIF} is set to 1, the model fitness improvement threshold β_{HIF} is set to 0.03 and the threshold μ_{HIF} for the number of newly added activities is set to 0.3.

Additionally, the accuracy of the generated sub-process models is also measured in terms of *F-score* which stands for the harmonic mean of recall (fitness) and precision (appropriateness) [200] (the ETConformance Checker from ProM 6 is utilised for evaluating the precision of the mined sub-models). For each log, we evaluate the trace clustering technique with different numbers of clusters (from 3, 4, 5 and 6). Figure 6.13 shows the comparison results from the perspective of fitness. The results illustrate that CTC and TDTC perform better on event logs LOA, VIPM and MCRM than the other six trace clustering methods. For the log KIM, ATC has better overall performance than CTC and TDTC because ATC also has a fitness improvement mechanism that is applied to the sub-process models. However, the mechanism provided by CTC and TDTC seems more stable on the four real-world event logs. Figure 6.14 shows the comparison results on F-score. It can be seen that CTC and TDTC also perform better than the traditional trace clustering techniques on most of the tested logs. Figure 6.15 highlights the comparison results from the angle of PT-CD. Here, CTC, TDTC and SCT outperform the other techniques. Figure 6.16 depicts the comparison results on E-Cardoso, based on which the performance of CTC mediates. The main reason is that the fitness improvement method HIF utilised by CTC may decrease the performance of CTC on optimising the complexity (evaluated by E-Cardoso) of the potential sub-models. The performance of TDTC on E-Cardoso remains a very high level. At last, we conclude that under a comprehensive assessment, CTC and TDTC improve beyond the state-of-the-art in trace clustering in the context of process model discovery.

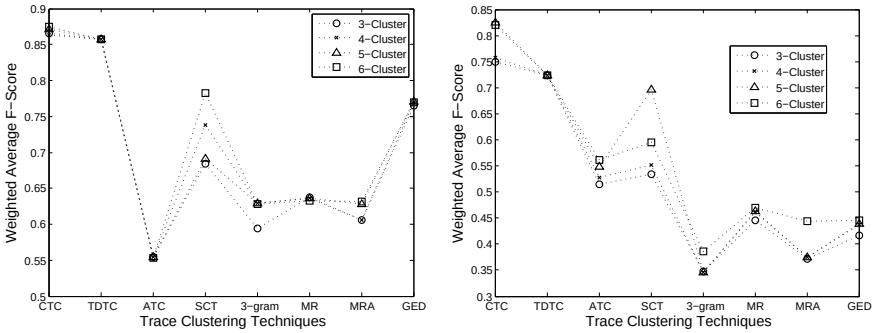


(a) LOA (b) VIPM

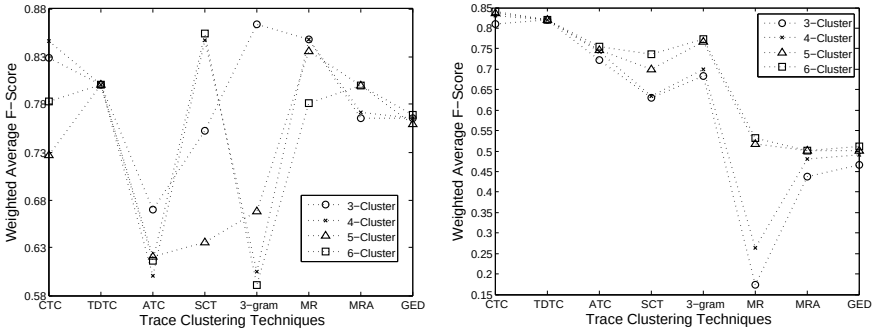


(c) KIM (d) MCRM

Figure 6.13.: Comparison on weighed average fitness.

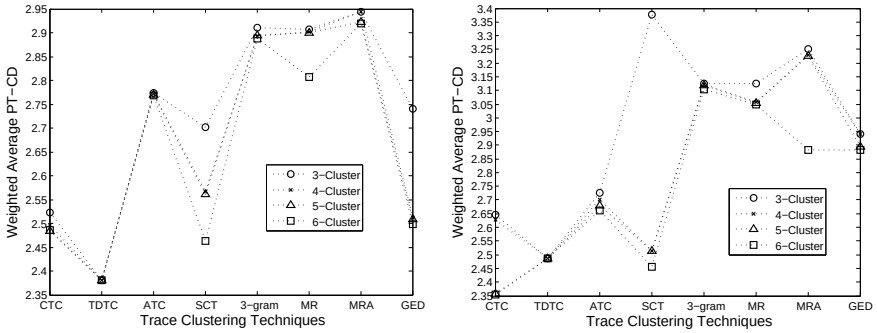


(a) LOA (b) VIPM

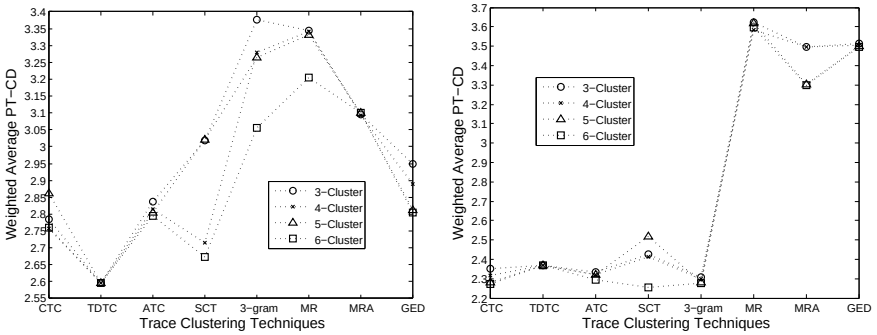


(c) KIM (d) MCRM

Figure 6.14.: Comparison on weighted average F-score.

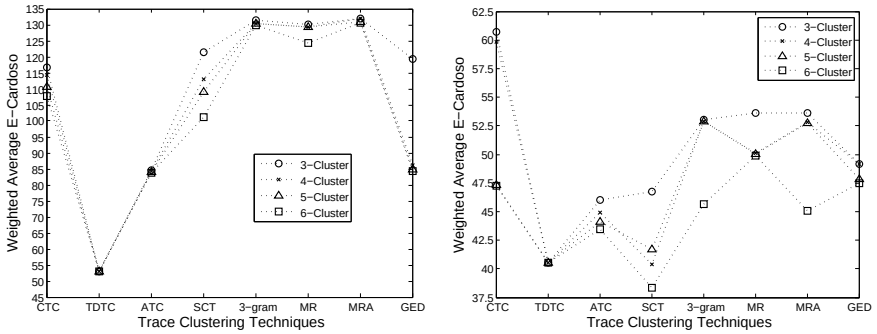


(a) LOA (b) VIPM

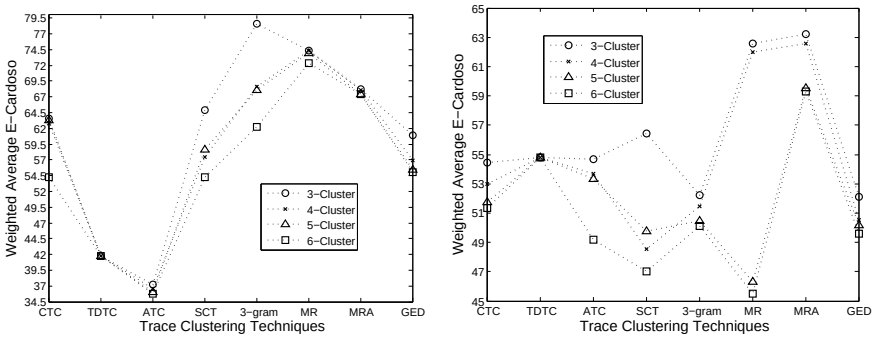


(c) KIM (d) MCRM

Figure 6.15.: Comparison on weighted average PT-CD.



(a) LOA (b) VIPM



(c) KIM (d) MCRM

Figure 6.16.: Comparison on weighted average E-Cardoso.

6.4. Evaluation on Technique GTCA

In Chapter 5, it is indicated that trace clustering technique might meet its limitation while dealing with totally unstructured business processes. Afterwards, a method named GTCA (graph and trace clustering-based approach) is put forward in Chapter 5 which inherits the basic idea of MAS for handling extremely complex process models. The technique GTCA that combines the characteristics from the trace clustering technique and the process model abstraction-based approach is able to optimise the quality of the potential high level process model using a new abstraction strategy based on graph clustering technique. As a result, a high-quality abstraction model can be assured. Moreover, the quality of the sub-models built for displaying the details of their related high level activities is also optimised through trace clustering technique. In Section 6.4.1, we first execute six classical trace clustering techniques (i.e., 3-gram, ATC, GED, SCT, MR and MRA) on two event logs which are the log of loan and overdraft approvals process (LOA) from BPIC 2012 and the hospital event log (Hospital) from BPIC 2011 to reveal the limitations of trace clustering technique. Afterwards, the effectiveness of technique GTCA is measured by using the three logs mentioned above in Section 6.4.2.

6.4.1. The Limitations of Trace Clustering Technique

The repair log (Repair) [9], the hospital log (Hospital) and the log of the loan and overdraft approvals process (LOA) are employed in our experiment. The basic information about the three logs is shown in Table 6.7. The quality information of the models mined from the three logs by using HM is listed in Table 6.8. The ICS fitness is used to measure the accuracy of the mined process models. The process model complexity metrics PT-CD and E-Cardoso are used for evaluating the complexity of the mined models.

Table 6.7.: Basic information of the evaluated logs.

Log	Traces	Events	Event types
Repair	1000	10827	12
LOA	13087	262200	36
Hospital	1143	150291	624

Firstly, six classical trace clustering techniques are executed on the event logs Hospital and LOA[3] which are 3-gram [31], MR and MRA [195], ATC [36], GED

[3]The log Repair is a very simple event log which contains simple process behaviours and most of the six trace clustering techniques utilised can perform well for dealing with this log.

Table 6.8.: Evaluation results for the models mined by using the log Repair, Loan and Hospital.

Log	*ICS Fitness*	*E – Cardoso*	*PT – CD*
Repair	0.6768	31	2.3656
LOA	0.7878	148	3.1478
Hospital	0.6058	2108	2.703

[35] and SCT [196]. For each trace clustering approach five sublogs are generated for every of the two logs utilised. The assessment results on these techniques are shown in Table 6.9. The metric $W_t - ICS$ stands for the weighted average ICS fitness based on the number of traces and $W_e - ICS$ represents the weighted average ICS fitness based on the number of events. For example, let $S_L = \{sl_1, sl_2, sl_3, sl_4, sl_5\}$ be a set of sublogs output by a trace clustering technique carried out on event log L. For a sublog $sl_k \in S_L$, $|sl_k|_t$ represents the total number of traces in sl_k, $|sl_k|_e$ represents the total number of events in sl_k and ICS_{sl_k} represents the value of ICS fitness for the sub-model mined from sublog sl_k. Then, the $W_t - ICS$ for the sublogs in S_L is equal to $(\sum_{k=1}^{5} |sl_k|_t \times ICS_{sl_k}) / \sum_{k=1}^{5} |sl_k|_t$ and the $W_e - ICS$ is equal to $(\sum_{k=1}^{5} |sl_k|_e \times ICS_{sl_k}) / \sum_{k=1}^{5} |sl_k|_e$. According to the evaluation results shown in Table 6.9, for the logs Loan and Hospital which consist of many complex trace behaviours, most trace clustering techniques employed could not bring a significant improvement on the accuracy of the mined models (especially for the log Hospital which records the trace behaviours from a totally unstructured business process).

Table 6.9.: Evaluation results for the six classical trace clustering techniques executed on the log LOA and Hospital.

Log	**Method**	$W_t - ICS$	$W_e - ICS$
LOA	3-gram	0.76	0.4661
	MR	0.7725	0.6923
	MRA	0.7836	0.7005
	ATC	0.7655	0.5665
	GED	0.8147	0.8384
	SCT	0.8671	0.8883
Hospital	3-gram	0.5406	0.5916
	MR	0.6139	0.6846
	MRA	0.5593	0.6784
	ATC	0.5844	0.5949
	GED	0.5867	0.6848
	SCT	0.5977	0.6111

6.4.2. Measurement of the Performance of GTCA on Three Event Logs

Whereafter, the approach GTCA proposed in this thesis is evaluated by using the three logs mentioned above. The threshold ϕ for judging the casual relations is set to 0 (such a setting will help find more complete activity clusters), the threshold α for judging whether a high level activity generated should be removed or not is set to 20, the threshold β for searching for the merging modes is set to 0.05, the sub-model complexity threshold τ (for PT-CD) is set to 2.5, the sub-model accuracy threshold χ (for ICS fitness) is set to 0.8, the trace number threshold κ is set to 100 and the number of clusters for the trace clustering technique *GED* [35] (GED is utilised because it has a better overall performance when dealing with unstructured business processes) is set to 6. The quality information of the sub-models generated is shown in Table 6.10, the quality information of the three high level models (for the log Repair, LOA and Hospital) output by GTCA is shown in Table 6.12 and the basic information of the three high level logs created by GTCA is shown in Table 6.11.

Table 6.10.: The weighted average quality of the sub-models generated by GTCA.

Log	$W_t - ICS$	$W_e - ICS$	$W_t - E\text{–}Cardoso$	$W_e - E\text{–}Cardoso$	$W_t - PT\text{–}CD$	$W_e - PT\text{–}CD$
Repair	0.9738	0.9687	11.57	12.46	2.0688	2.0929
LOA	0.9514	0.9297	21.934	26.4995	2.1729	2.2238
Hospital	0.8891	0.902	467.84	465.2	3.1257	3.0956

Table 6.11.: Basic information of the generated high level logs.

H-Log	**Traces**	**Events**	**Event types**
H-Repair	1000	2700	10
H-LOA	13087	40783	44
H-Hospital	1143	37740	65

According to Table 6.11, the generated high level event logs H-Repair and H-Hospital contain fewer activities than their related raw event logs Repair and Hospital. The main reason is that the activities in the original repair log and hospital log can form high quality activity clusters (more activity relations inside the cluster and fewer among the clusters). In the experiment about 1% events from log Hospital and 0.5% events from log Loan are removed together with the infrequent high level activities generated and for the log Repair no events are removed (very few events are removed because of the effects of the trace merging technique proposed in Section 5.3 of Chapter 5). However, the high level log H-LOA contains eight more activities than log LOA. This is because the casual graph formed by the activities from LOA does not have a structure with clear clusters. According to Table 6.12, all of the three high level models generated have high accuracy which benefits from the abstraction strategy proposed for GTCA.

Table 6.12.: The quality information of the high level models generated for each log by GTCA.

Log	*ICS Fitness*	*E – Cardoso*	*PT–CD*
Repair	0.978	33	2.483
LOA	0.9671	137	3.378
Hospital	0.95	192	2.4328

Figure 6.17, 6.18 and 6.19 show the raw models mined from log Repair, LOA and Hospital respectively. Figure 6.20, 6.21 and 6.22 exhibit the high level process models output by technique GTCA for log Repair, LOA and Hospital. For the high level activities in the three built high level models, the average accuracy of their relevant sub-models is also generally good. The complexity (expressed by E-Cardoso and PT-CD) of the high level models for log Repair and LOA is a little higher than the complexity of the original process models mined directly from Repair and LOA but the complexity of the high level model for log Hospital is much lower than the original model for Hospital. This is because the activities in the log Hospital can form better activity clusters than the activities in the log LOA and Repair. However, due to the fact that the process behaviours between activities in the high level models for log LOA and Repair have been greatly simplified by GTCA, as a result the high level models for log LOA and Repair are still easier to be comprehended than their related original models (as proved by Figure 6.17, 6.18, 6.20 and 6.21). The raw model for log Hospital is extremely complex and is hard to be interpreted according to Figure 6.19. Nevertheless, the high level model for log Hospital (see Figure 6.22) is much simpler than its related raw model which proves the effectiveness of GTCA.

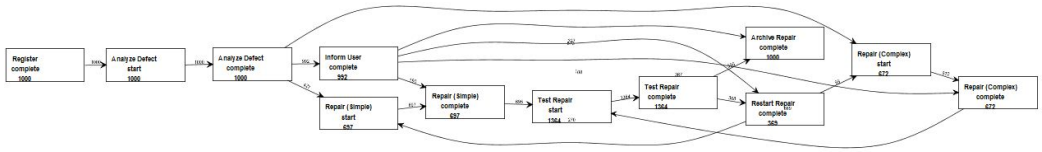


Figure 6.17.: Process model mined from log Repair.

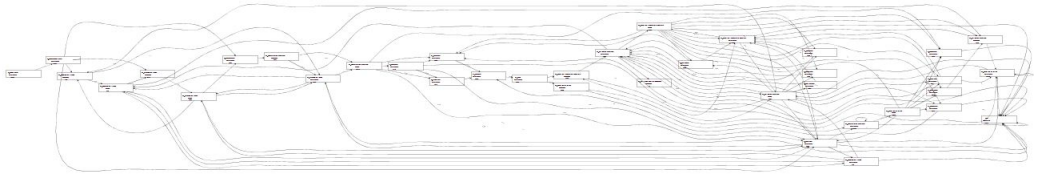


Figure 6.18.: Process model mined from log LOA.

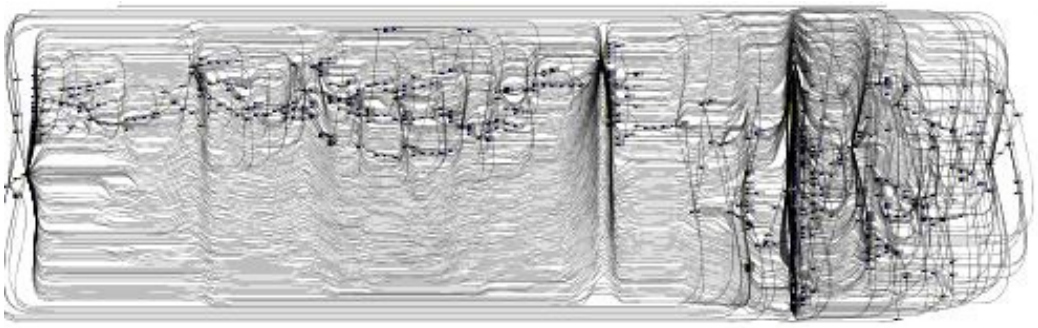


Figure 6.19.: Process model mined from log Hospital.

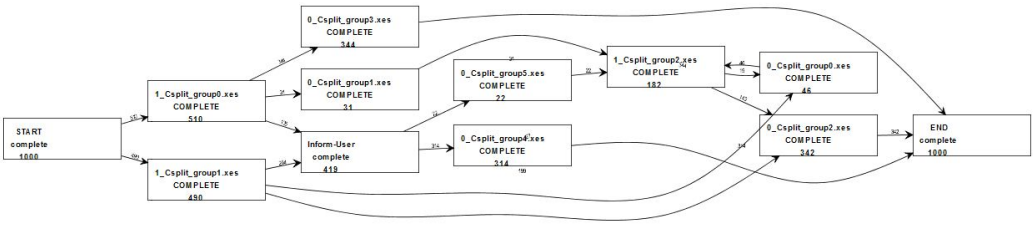


Figure 6.20.: High level model generated for log Repair.

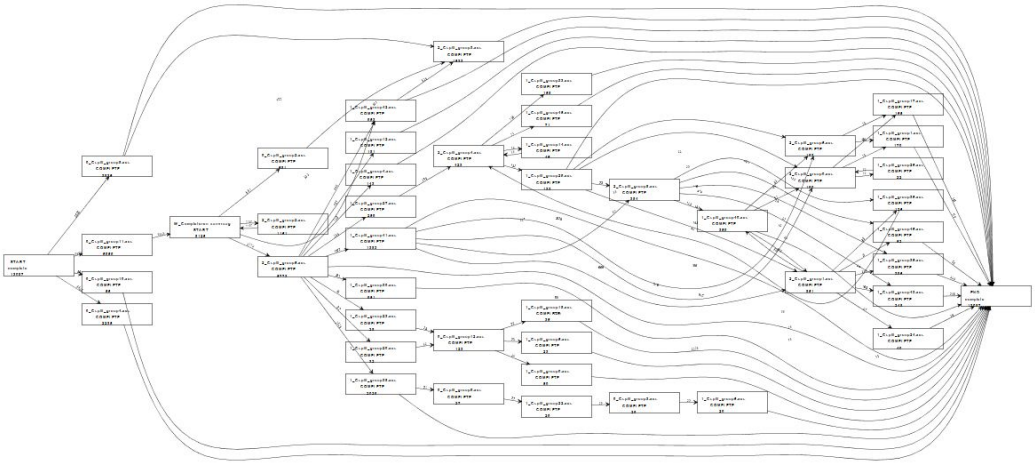


Figure 6.21.: High level model generated for log LOA.

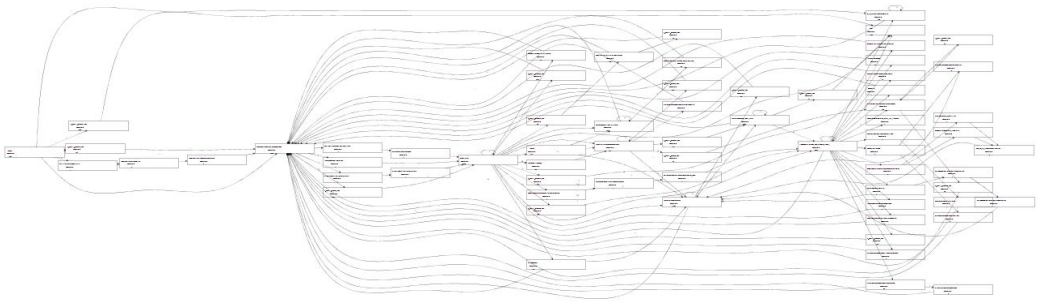


Figure 6.22.: High level model generated for log Hospital.

7

Conclusions

7.1. Summary

The present BPMD techniques often output inaccurate and complex process models when mining real-life event logs. In this thesis we investigated approaches and techniques to assist existing BPMD techniques in mining accurate and comprehensible process models from real-life event logs. The techniques proposed in this thesis are mainly based on three classical strategies for dealing with the problem of inaccurate and complex process models mined from real-life event logs: the MEBS, the MDS and the MAS. Each strategy has its own advantage and limitation. Given a specific context, an appropriate strategy should be selected to solve the relevant problem. To address the inefficiency of existing approaches that have realised the three strategies, we have developed several solutions.

Mined Model Fitness Improvement Technique: HIF

Real-life event logs that usually stem from the business processes executed in highly flexible environments often consist of extremely complex trace behaviours which might be far beyond the expressive ability of existing BPMD techniques. As a result, inaccurate process models are often obtained by the current BPMD techniques implemented on real-life logs. Thereby, we presented a new mined model fitness improvement technique named HIF. The basic idea of HIF is to transform the fitness improvement problem for the non-fitting models mined into the problem of locating the inexpressible process behaviours in the given event log and then converting them into expressible behaviours for the utilised BPMD techniques.

We have tested the effectiveness of the proposed technique HIF on several real-life event logs for the BPMD technique HM. The experiment results indicate that HIF greatly helps improve the accuracy of the process models mined by HM.

In addition, the principle of our method has a universal significance and can also be employed for helping other kinds of BPMD algorithms mine more accurate process models.

Trace Clustering Techniques: TDTC and CTC

The technique HIF cannot help decrease the complexity of the mined process models. Except for accuracy, complexity is also a significant KPI for evaluating the quality of a mined process model. A model that is too complex to be interpreted is almost useless. The MDS can be used to deal with the mined model which is both inaccurate and complex. The classical technique that inherits the idea of MDS is trace clustering. Nevertheless, most currently available trace clustering approaches cannot assure the accuracy and simplicity of the potential sub-models for each generated cluster. Therefore, we proposed two trace clustering techniques: TDTC and CTC. Both techniques are able to optimise the accuracy and complexity of the underlying sub-models for each trace cluster. Usually, CTC has a better performance than TDTC. However, TDTC is recommended when the number of activities in the given event logs is very large.

Through an elaborately designed experiment, we testified that TDTC and CTC generally have a higher performance than existing trace clustering techniques on the given real-life event logs.

Multi-label Case Classification Approach: MLCC

Trace Clustering can help find a lot of hidden trace behaviours among the cases. However, it is an unsupervised learning approach and cannot make use of domain knowledge during the clustering process. As a result, it is unable to indicate which behaviours located are significant for splitting the cases or which behaviours are desired by the users for dividing the cases. Classification that is a supervised learning technique is able to combine the domain knowledge from business experts and make a meaningful division of the cases in the given logs.

By realising this problem mentioned above, we put forward the multi-label case classification technique MLCC which is capable of employing the structural behaviours of traces and the case-based attributes for classifying cases. Afterwards, through executing MLCC on a hospital event log from BPIC 2011 the effectiveness of MLCC is proved.

Mined Model Abstraction Technique: GTCA

The trace clustering technique and case classification technique will meet their limitations when dealing with event logs consisting of massive trace behaviours. In such a situation, the MAS which tries to discover the sub-process models in the raw model and then abstract the found sub-models into high level activities in the raw model should be considered. However, most present model abstraction-based techniques focus mainly on the discovery of sub-process models and cannot ensure the accuracy of the high level process models generated. Hence, we proposed a new method named GTCA which inherits the characteristics of both trace clustering and the MAS for dealing with the problem of extremely complex process models. GTCA is able to optimise the quality of the potential high level process models through a new abstraction strategy based on graph clustering technique so that a high-quality abstraction process model can be built. Additionally, the quality of the potential sub-process models discovered is also improved by employing the trace clustering technique.

We tested GTCA using the hospital log from BPIC 2011 which contains the traces from a very complex treatment process that is very hard to be interpreted. The results output from the test shows that GTCA indeed provides a chance to help understand the treatment process recorded in this hospital log.

7.2. Discussion and Outlook

Though the proposed techniques HIF, TDTC, CTC, MLCC and GTCA have shown their promising potentials for helping generate high quality process models from real-life event logs, they still have limitations to overcome. In this section we will concentrate on the discussion of the shortages of these techniques and the future work that will be carried out to improve the performance of these techniques.

Mined Model Fitness Improvement Technique: HIF

As shown in Section 6.1, the technique HIF is usually capable of discovering the most efficient way for optimising the fitness of mined process models. This is because HIF is devised to optimise the model fitness to the largest extent with the least operations. Nevertheless, there are often different ways to improve the fitness of a mined process model and the most efficient way might not always be the best way (this is also proved in the experiment introduced in Section 6.1). How to increase the possibility for HIF to find the best way to improve the fitness of mined process models will be the most important topic in our future job.

However, another problem that can be foreseen is that the operation of finding the best way for fitness improvement will bring greater cost on the performance of HIF. Therefore, how to balance the degree for finding the best way for model fitness improvement and the performance of HIF will also be an important future job.

Mined Model Division-Based Techniques TDTC, CTC and MLCC

The common feature of the trace clustering technique TDTC and CTC and the multi-label case classification technique MLCC is that they employ the CSP for representing the process behaviours recorded in the event logs. However, for the event logs (such like the hospital event log from BPIC 2011) stemming from totally unstructured business processes, it often takes quite a long time for the existing CSP mining techniques to finish their mining job on these logs. In addition, a huge amount of CSPs might be generated. All these greatly influence the performance of TDTC, CTC and MLCC.

Thereby, our future job on TDTC, CTC and MLCC will be mainly focused on two directions for improving the performance of these techniques. The first direction is developing a new method to discover and eliminate the insignificant CSPs from the total set of CSPs that has been discovered by the utilised CSP mining techniques. The second direction will be finding a more efficient way to represent the process behaviours than using the CSPs.

A Recommendation Algorithm for HIF, TDTC, CTC, MLCC and GTCA

As introduced in Chapter 1, techniques HIF, TDTC, CTC, MLCC and GTCA are based on different strategies for helping generate high quality process models. Each of these techniques has its own application context decided by the feature of data that needs to be handled. Given a certain real-life event log, how to decide which technique should be selected for dealing with this log is an interesting problem. Furthermore, the technique GTCA has many parameters and deciding the ideal parameter setting for GTCA for a certain event log is also a difficult task.

Hence, another future job for us will be to develop a recommendation algorithm which is able to recommend a suitable technique to the users by analysing the characteristics of the data recorded in the event logs from the users. Addi-

tionally, if the technique GTCA is suggested, the recommendation algorithm can also help select a suitable parameter setting for GTCA.

Bibliography

[1] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M. Shan, "Business process intelligence," *Computers in Industry*, vol. 53, no. 3, pp. 321–343, 2004.

[2] D. R. Cooper, P. S. Schindler, and J. Sun, "Business research methods," 2003.

[3] C. Vercellis, *Business Intelligence: Data Mining and Optimization for Decision Making*. John Wiley Sons, 2011.

[4] M. Castellanos, A. A. De Medeiros, J. Mendling, B. Weber, and A. Weijters, "Business process intelligence," *Handbook of Research on Business Process Modeling*, pp. 456–480, 2009.

[5] M. Attaran, "Exploring the relationship between information technology and business process reengineering," *Information Management*, vol. 41, no. 5, pp. 585–596, 2004.

[6] J. F. Chang, *Business Process Management Systems: Strategy and Implementation*. CRC Press, 2016.

[7] K. Vergidis, A. Tiwari, and B. Majeed, "Business process analysis and optimization: Beyond reengineering," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 1, pp. 69–82, 2008.

[8] W. M. van der Aalst, *Process Mining: Data Science in Action*. Springer, 2016.

[9] W. M. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[10] J. Jeston and J. Nelis, *Business Process Management*. Routledge, 2014.

[11] W. M. van der Aalst, A. H. Ter Hofstede, and M. Weske, "Business process management: A survey," *International Conference on Business Process Management*, pp. 1–12, 2003.

[12] M. Dumas, W. M. van der Aalst, and A. H. Ter Hofstede, *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. John Wiley Sons, 2005.

[13] M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information*

Systems: Challenges, Methods, Technologies. Springer Science Business Media, 2012.

[14] W. M. van der Aalst, A. H. Ter Hofstede, and M. Weske, "Business process management: A survey," *International Conference on Business Process Management*, pp. 1–12, 2003.

[15] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. M. van der Aalst, "Process flexibility: A survey of contemporary approaches," *Advances in Enterprise Engineering I*, pp. 16–30, 2008.

[16] H. Schonenberg, B. Weber, B. Van Dongen, and W. M. van der Aalst, "Supporting flexible processes through recommendations based on history," *International Conference on Business Process Management*, pp. 51–66, 2008.

[17] R. S. Mans, W. M. van der Aalst, and R. J. Vanwersch, "Process mining," *Process Mining in Healthcare*, pp. 17–26, 2015.

[18] W. M. van der Aalst, "Process discovery: An introduction," *Process Mining*, pp. 163–194, 2016.

[19] W. M. van der Aalst and A. Weijters, "Process mining: A research agenda," *Computers in Industry*, vol. 53, no. 3, pp. 231–244, 2004.

[20] W. M. van der Aalst, "Process mining: Overview and opportunities," *ACM Transactions on Management Information Systems (TMIS)*, vol. 3, no. 2, p. 7, 2012.

[21] W. M. van der Aalst, "Process mining: Making knowledge discovery process centric," *ACM SIGKDD Explorations Newsletter*, vol. 13, no. 2, pp. 45–49, 2012.

[22] W. M. van der Aalst, "Process mining in the large: A tutorial," *Business Intelligence*, pp. 33–76, 2014.

[23] W. M. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[24] J. M. E. van der Werf, B. F. van Dongen, C. A. Hurkens, and A. Serebrenik, "Process discovery using integer linear programming," *International Conference on Applications and Theory of Petri Nets*, pp. 368–387, 2008.

[25] S. J. Leemans, D. Fahland, and W. M. van der Aalst, "Discovering block-structured process models from event logs containing infrequent be-

haviour," *International Conference on Business Process Management*, pp. 66–78, 2013.

[26] S. J. Leemans, D. Fahland, and W. M. van der Aalst, "Discovering block-structured process models from event logs-a constructive approach," *International Conference on Applications and Theory of Petri Nets and Concurrency*, pp. 311–329, 2013.

[27] S. J. Leemans, D. Fahland, and W. M. van der Aalst, "Discovering block-structured process models from incomplete event logs," *International Conference on Applications and Theory of Petri Nets and Concurrency*, pp. 91–110, 2014.

[28] C. W. Günther and W. M. van Der Aalst, "Fuzzy mining–adaptive process simplification based on multi-perspective metrics," *International Conference on Business Process Management*, pp. 328–343, 2007.

[29] A. K. de Medeiros, A. J. Weijters, and W. M. van der Aalst, "Genetic process mining: An experimental evaluation," *Data Mining and Knowledge Discovery*, vol. 14, no. 2, pp. 245–304, 2007.

[30] W. M. van der Aalst, H. A. Reijers, A. J. Weijters, B. F. van Dongen, A. A. De Medeiros, M. Song, and H. Verbeek, "Business process mining: An industrial application," *Information Systems*, vol. 32, no. 5, pp. 713–732, 2007.

[31] M. Song, C. W. Günther, and W. M. van der Aalst, "Trace clustering in process mining," *Business Process Management Workshops*, pp. 109–120, 2009.

[32] A. A. Kalenkova, I. A. Lomazova, and W. M. van der Aalst, "Process model discovery: A method based on transition system decomposition," *International Conference on Applications and Theory of Petri Nets and Concurrency*, pp. 71–90, 2014.

[33] R. J. C. Bose and W. M. van der Aalst, "Abstractions in process mining: A taxonomy of patterns," *International Conference on Business Process Management*, pp. 159–175, 2009.

[34] S. Smirnov, M. Weidlich, and J. Mendling, "Business process model abstraction based on behavioural profiles," *International Conference on Service-Oriented Computing*, pp. 1–16, 2010.

[35] R. J. C. Bose and W. M. van der Aalst, "Context aware trace clustering: Towards improving process mining results," *Proceedings of the 2009 SIAM International Conference on Data Mining*, pp. 401–412, 2009.

[36] J. De Weerdt, S. vanden Broucke, J. Vanthienen, and B. Baesens, "Active trace clustering for improved process discovery," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 12, pp. 2708–2720, 2013.

[37] A. Weijters and J. Ribeiro, "Flexible heuristics miner (fhm)," *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*, pp. 310–317, 2011.

[38] R. Conforti, M. Dumas, L. García-Bañuelos, and M. La Rosa, "Bpmn miner: Automated discovery of bpmn process models with hierarchical structure," *Information Systems*, vol. 56, pp. 284–303, 2016.

[39] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser, "Process mining based on regions of languages," *International Conference on Business Process Management*, pp. 375–383, 2007.

[40] W. M. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[41] T. Baier and J. Mendling, "Bridging abstraction layers in process mining by automated matching of events and activities," *Business Process Management*, pp. 17–32, 2013.

[42] T. Baier and J. Mendling, "Bridging abstraction layers in process mining: Event to activity mapping," *Enterprise, Business-Process and Information Systems Modeling*, pp. 109–123, 2013.

[43] F. Masseglia, M. Teisseire, and P. Poncelet, "Sequential pattern mining," in *Encyclopedia of Data Warehousing and Mining*, pp. 1028–1032, IGI Global, 2005.

[44] S. E. Schaeffer, "Graph clustering," *Computer science review*, vol. 1, no. 1, pp. 27–64, 2007.

[45] J. Becker, M. Rosemann, and C. Von Uthmann, "Guidelines of business process modeling," *Business Process Management*, pp. 30–49, 2000.

[46] M. Weske, *Business Process Management*. Springer, 2012.

[47] W. M. van der Aalst, "Business process management: A comprehensive survey," *ISRN Software Engineering*, vol. 2013, 2013.

[48] M. A. Ould and M. Ould, *Business Processes: Modelling and Analysis for Re-Engineering and Improvement*, vol. 598. Wiley Chichester, 1995.

[49] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining*, vol. 21. AAAI press Menlo Park, 1996.

[50] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*. Elsevier, 2011.

[51] D. J. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*. MIT press, 2001.

[52] P. Berkhin *et al.*, "A survey of clustering data mining techniques.," *Grouping multidimensional data*, vol. 25, p. 71, 2006.

[53] J. Grabmeier and A. Rudolph, "Techniques of cluster algorithms in data mining," *Data Mining and knowledge discovery*, vol. 6, no. 4, pp. 303–360, 2002.

[54] K. Hammouda and F. Karray, "A comparative study of data clustering techniques," *University of Waterloo, Ontario, Canada*, 2000.

[55] K. D. Bailey, *Typologies and Taxonomies: An Introduction to Classification Techniques*, vol. 102. Sage, 1994.

[56] T. N. Phyu, "Survey of classification techniques in data mining," *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, pp. 18–20, 2009.

[57] S. Kotsiantis, "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, pp. 249–268, 2007.

[58] A. de Carvalho and A. Freitas, "A tutorial on multi-label classification techniques," *Foundations of Computational Intelligence Volume 5*, pp. 177–195, 2009.

[59] A. Rozinat and W. M. van der Aalst, *Decision Mining in Business Processes*. Beta, Research School for Operations Management and Logistics, 2006.

[60] A. Rozinat and W. M. van der Aalst, "Decision mining in prom," *Business process management*, vol. 4102, pp. 420–425, 2006.

[61] N. R. Mabroukeh and C. I. Ezeife, "A taxonomy of sequential pattern mining algorithms," *ACM Computing Surveys (CSUR)*, vol. 43, no. 1, p. 3, 2010.

[62] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: Current

status and future directions," *Data Mining and Knowledge Discovery*, vol. 15, no. 1, pp. 55–86, 2007.

[63] O. Maimon and L. Rokach, *Data Mining and Knowledge Discovery Handbook*, vol. 2. Springer, 2005.

[64] E. Rahm and H. H. Do, "Data cleaning: Problems and current approaches," *IEEE Data Eng. Bull.*, vol. 23, no. 4, pp. 3–13, 2000.

[65] L. Geng and H. J. Hamilton, "Interestingness measures for data mining: A survey," *ACM Computing Surveys (CSUR)*, vol. 38, no. 3, p. 9, 2006.

[66] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2016.

[67] B. L. W. H. Y. Ma and W. Liu, "Integrating classification and association rule mining," *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, 1998.

[68] W. Li, J. Han, and J. Pei, "Cmar: Accurate and efficient classification based on multiple class-association rules," *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pp. 369–376, 2001.

[69] L. Rokach and O. Maimon, *Data Mining with Decision Trees: Theory and Applications*. World scientific, 2014.

[70] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[71] D. J. Hand, "Measuring classifier performance: A coherent alternative to the area under the roc curve," *Machine learning*, vol. 77, no. 1, pp. 103–123, 2009.

[72] T. Kautz, B. M. Eskofier, and C. F. Pasluosta, "Generic performance measure for multiclass-classifiers," *Pattern Recognition*, vol. 68, pp. 111–125, 2017.

[73] M. Carbonero-Ruz, F. J. Martínez-Estudillo, F. Fernández-Navarro, D. Becerra-Alonso, and A. C. Martínez-Estudillo, "A two dimensional accuracy-based measure for classification performance," *Information Sciences*, vol. 382, pp. 60–80, 2017.

[74] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006.

[75] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *European Conference on Computational Learning Theory*, pp. 23–37, 1995.

[76] P. Viola and M. Jones, "Fast and robust classification using asymmetric adaboost and a detector cascade," *Advances In Neural Information Processing Systems*, pp. 1311–1318, 2002.

[77] J. Zhu, H. Zou, S. Rosset, T. Hastie, *et al.*, "Multi-class adaboost," *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009.

[78] I. Rish, "An empirical study of the naive bayes classifier," *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, vol. 3, no. 22, pp. 41–46, 2001.

[79] C. Elkan, "Boosting and naive bayesian learning," *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 1997.

[80] R. R. Yager, "An extension of the naive bayesian classifier," *Information Sciences*, vol. 176, no. 5, pp. 577–588, 2006.

[81] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT press, 2002.

[82] I. Steinwart and A. Christmann, *Support Vector Machines*. Springer Science & Business Media, 2008.

[83] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, 1998.

[84] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999.

[85] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[86] J. R. Quinlan, *C4. 5: Programs for Machine Learning*. Elsevier, 2014.

[87] J. R. Quinlan, "Improved use of continuous attributes in c4. 5," *Journal of Artificial Intelligence Research*, vol. 4, pp. 77–90, 1996.

[88] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. University of Illinois Press, 1998.

[89] R. J. Roiger, *Data Mining: A Tutorial-Based Primer*. CRC Press, 2017.

[90] M. R. Anderberg, *Cluster Analysis for Applications: Probability and Mathematical Statistics: A Series of Monographs and Textbooks*, vol. 19. Academic Press, 2014.

[91] P. Berkhin *et al.*, "A survey of clustering data mining techniques.," *Grouping Multidimensional Data*, vol. 25, p. 71, 2006.

[92] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Computing Surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[93] J. A. Hartigan and J. Hartigan, *Clustering Algorithms*, vol. 209. Wiley New York, 1975.

[94] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: An overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, 2012.

[95] G. Karypis, E.-H. Han, and V. Kumar, "Chameleon: Hierarchical clustering using dynamic modeling," *Computer*, vol. 32, no. 8, pp. 68–75, 1999.

[96] Y. Zhao, G. Karypis, and U. Fayyad, "Hierarchical clustering algorithms for document datasets," *Data Mining and Knowledge Discovery*, vol. 10, no. 2, pp. 141–168, 2005.

[97] F. Murtagh, "A survey of recent advances in hierarchical clustering algorithms," *The Computer Journal*, vol. 26, no. 4, pp. 354–359, 1983.

[98] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231–240, 2011.

[99] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm gdbscan and its applications," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 169–194, 1998.

[100] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," *Kdd*, vol. 96, no. 34, pp. 226–231, 1996.

[101] Y. Lv, T. Ma, M. Tang, J. Cao, Y. Tian, A. Al-Dhelaan, and M. Al-Rodhaan, "An efficient and scalable density-based clustering algorithm for datasets with complex structures," *Neurocomputing*, vol. 171, pp. 9–22, 2016.

[102] A. Lulli, M. Dell'Amico, P. Michiardi, and L. Ricci, "Ng-dbscan: Scalable density-based clustering for arbitrary data," *Proceedings of the VLDB Endowment*, vol. 10, no. 3, pp. 157–168, 2016.

[103] N. H. Park and W. S. Lee, "Statistical grid-based clustering over data streams," *Acm Sigmod Record*, vol. 33, no. 1, pp. 32–37, 2004.

[104] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[105] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, *et al.*, "Constrained k-means clustering with background knowledge," *ICML*, vol. 1, pp. 577–584, 2001.

[106] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[107] D. Kavya and C. D. Desai, "Comparative analysis of k-means clustering sequentially and parallely," *nature*, vol. 3, no. 04, 2016.

[108] X. Jin and J. Han, "K-medoids clustering," *Encyclopedia of Machine Learning*, pp. 564–565, 2011.

[109] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert Systems with Applications*, vol. 36, no. 2, pp. 3336–3341, 2009.

[110] Q. Zhang and I. Couloigner, "A new and efficient k-medoid algorithm for spatial clustering," *Computational Science and Its Applications–ICCSA 2005*, pp. 207–224, 2005.

[111] J. V. De Oliveira and W. Pedrycz, *Advances in Fuzzy Clustering and its applications*. John Wiley & Sons, 2007.

[112] M. M. Deza and E. Deza, *Encyclopedia of Distances*, vol. 94. Springer, 2009.

[113] K. A. Heller and Z. Ghahramani, "Bayesian hierarchical clustering," *Proceedings of the 22nd International Conference on Machine Learning*, pp. 297–304, 2005.

[114] I. Davidson and S. Ravi, "Agglomerative hierarchical clustering with constraints: Theoretical and empirical results," pp. 59–70, 2005.

[115] P. Franti, O. Virmajoki, and V. Hautamaki, "Fast agglomerative clustering

using a k-nearest neighbour graph," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1875–1881, 2006.

[116] A. Guénoche, P. Hansen, and B. Jaumard, "Efficient algorithms for divisive hierarchical clustering with the diameter criterion," *Journal of Classification*, vol. 8, no. 1, pp. 5–30, 1991.

[117] D. Hofmeyr, N. Pavlidis, and I. Eckley, "Divisive clustering of high dimensional data streams," *Statistics and Computing*, vol. 26, no. 5, pp. 1101–1120, 2016.

[118] D. B. West *et al.*, *Introduction to Graph Theory*, vol. 2. Prentice hall Upper Saddle River, 2001.

[119] B. Bollobás, *Modern Graph Theory*, vol. 184. Springer Science & Business Media, 2013.

[120] J. L. Gross and J. Yellen, *Handbook of Graph Theory*. CRC press, 2004.

[121] B. Bollobás, B. Bollobás, B. Bollobás, and B. Bollobás, *Graph Theory: An Introductory Course*, vol. 63. Springer New York, 1979.

[122] A. Capocci, V. D. Servedio, G. Caldarelli, and F. Colaiori, "Detecting communities in large networks," *Physica A: Statistical Mechanics and its Applications*, vol. 352, no. 2, pp. 669–676, 2005.

[123] J. Edachery, A. Sen, and F. Brandenburg, "Graph clustering using distance-k cliques," in *Graph Drawing*, pp. 98–106, Springer, 1999.

[124] E. Hartuv and R. Shamir, "A clustering algorithm based on graph connectivity," *Information Processing Letters*, vol. 76, no. 4-6, pp. 175–181, 2000.

[125] H. Hu, X. Yan, Y. Huang, J. Han, and X. J. Zhou, "Mining coherent dense subgraphs across massive biological networks for functional discovery," *Bioinformatics*, vol. 21, no. suppl_1, pp. i213–i221, 2005.

[126] S. Kim, "Graph theoretic sequence clustering algorithms and their applications to genome comparison.," 2003.

[127] H. Matsuda, T. Ishihara, and A. Hashimoto, "Classifying molecular sequences using a linkage graph with their pairwise similarities," *Theoretical Computer Science*, vol. 210, no. 2, pp. 305–325, 1999.

[128] J. Šıma and S. E. Schaeffer, "On the np-completeness of some graph cluster measures," in *Proceedings of the Thirty-second International Conference on Cur-*

rent Trends in Theory and Practice of Computer Science (Sofsem 06), vol. 3831, pp. 530–537, Springer, 2006.

[129] R. Kannan, S. Vempala, and A. Vetta, "On clusterings: Good, bad and spectral," *Journal of the ACM (JACM)*, vol. 51, no. 3, pp. 497–515, 2004.

[130] X. He, H. Zha, C. H. Ding, and H. D. Simon, "Web document clustering using hyperlink structures," *Computational Statistics & Data Analysis*, vol. 41, no. 1, pp. 19–45, 2002.

[131] G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis, "Graph clustering and minimum cut trees," *Internet Mathematics*, vol. 1, no. 4, pp. 385–408, 2004.

[132] D. W. Matula and F. Shahrokhi, "Sparsest cuts and bottlenecks in graphs," *Discrete Applied Mathematics*, vol. 27, no. 1-2, pp. 113–123, 1990.

[133] A. Noack, "An energy model for visual graph clustering," in *Graph Drawing*, vol. 2912, pp. 425–436, Springer, 2003.

[134] A. Noack, "Energy models for graph clustering.," *J. Graph Algorithms Appl.*, vol. 11, no. 2, pp. 453–480, 2007.

[135] A. Noack, "Energy-based clustering of graphs with nonuniform degrees," *Lecture Notes in Computer Science*, vol. 3843, pp. 309–320, 2006.

[136] C. J. Alpert and A. B. Kahng, "Recent directions in netlist artitioning: A survey," *Integration, the VLSI Journal*, vol. 19, no. 1-2, pp. 1–81, 1995.

[137] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[138] M. E. Newman, "Analysis of weighted networks," *Physical review E*, vol. 70, no. 5, p. 056131, 2004.

[139] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, and U. Dayal, "Multi-dimensional sequential pattern mining," in *Proceedings of the Tenth International Conference on Information and Knowledge Management*, pp. 81–88, ACM, 2001.

[140] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proceedings of the Eleventh International Conference on Data Engineering*, pp. 3–14, IEEE, 1995.

[141] M. N. Garofalakis, R. Rastogi, and K. Shim, "Spirit: Sequential pattern mining with regular expression constraints," in *VLDB*, vol. 99, pp. 7–10, 1999.

[142] J. Pei, J. Han, and W. Wang, "Constraint-based sequential pattern mining: The pattern-growth methods," *Journal of Intelligent Information Systems*, vol. 28, no. 2, pp. 133–160, 2007.

[143] D.-Y. Chiu, Y.-H. Wu, and A. L. Chen, "An efficient algorithm for mining frequent sequences by a new strategy without support counting," in *Data Engineering, 2004. Proceedings. 20th International Conference on*, pp. 375–386, IEEE, 2004.

[144] X. Yan, J. Han, and R. Afshar, "Clospan: Mining: Closed sequential patterns in large datasets," in *Proceedings of the 2003 SIAM International Conference on Data Mining*, pp. 166–177, SIAM, 2003.

[145] C.-C. Yu and Y.-L. Chen, "Mining sequential patterns from multidimensional sequence data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 1, pp. 136–140, 2005.

[146] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu, "Freespan: Frequent pattern-projected sequential pattern mining," in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 355–359, ACM, 2000.

[147] J. Han, J. Pei, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *Proceedings of the 17th International Conference on Data Engineering*, pp. 215–224, 2001.

[148] P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas, "Fast vertical mining of sequential patterns using co-occurrence information," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 40–52, Springer, Cham, 2014.

[149] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining process models from workflow logs," *Advances in Database Technology—EDBT'98*, pp. 467–483, 1998.

[150] A. Rozinat and W. M. van der Aalst, "Conformance checking of processes based on monitoring real behaviour," *Information Systems*, vol. 33, no. 1, pp. 64–95, 2008.

[151] W. Van der Aalst, A. Adriansyah, and B. van Dongen, "Replaying history on process models for conformance checking and performance analysis,"

Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 2, no. 2, pp. 182–192, 2012.

[152] A. Adriansyah, B. F. van Dongen, and W. M. van der Aalst, "Conformance checking using cost-based fitness analysis," in *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*, pp. 55–64, IEEE, 2011.

[153] M. De Leoni, F. M. Maggi, and W. M. van der Aalst, "Aligning event logs and declarative process models for conformance checking.," in *BPM*, vol. 12, pp. 82–97, Springer, 2012.

[154] X. Lu, D. Fahland, and W. M. van der Aalst, "Conformance checking based on partially ordered event data.," in *Business process management workshops*, pp. 75–88, 2014.

[155] L. Garcia-Banuelos, N. van Beest, M. Dumas, M. La Rosa, and W. Mertens, "Complete and interpretable conformance checking of business processes," *IEEE Transactions on Software Engineering*, 2017.

[156] A. Rogge-Solti, A. Senderovich, M. Weidlich, J. Mendling, and A. Gal, "In log and model we trust? a generalized conformance checking framework," in *International Conference on Business Process Management*, pp. 179–196, Springer, 2016.

[157] A. A. De Medeiros and A. Weijters, "Genetic process mining," in *Applications and Theory of Petri Nets 2005, Volume 3536 of Lecture Notes in Computer Science*, Citeseer, 2005.

[158] W. M. van der Aalst, A. De Medeiros, and A. Weijters, "Genetic process mining," *Applications and Theory of Petri Nets*, pp. 985–985, 2005.

[159] A. K. Alves de Medeiros, "Genetic process mining," *CIP-Data Library Technische Universiteit Eindhoven*, 2006.

[160] W. M. van Der Aalst, A. H. Ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[161] W. M. van der Aalst, A. P. Barros, A. H. ter Hofstede, and B. Kiepuszewski, "Advanced workflow patterns," in *CoopIS*, vol. 1901, pp. 18–29, Springer, 2000.

[162] A. Weijters, W. M. van Der Aalst, and A. A. De Medeiros, "Process min-

ing with the heuristics miner-algorithm," *Technische Universiteit Eindhoven, Tech. Rep. WP*, vol. 166, pp. 1–34, 2006.

[163] A. Burattin and A. Sperduti, "Heuristics miner for time intervals.," in *ESANN*, 2010.

[164] S. De Cnudde, J. Claes, and G. Poels, "Improving the quality of the heuristics miner in prom 6.2," *Expert Systems with Applications*, vol. 41, no. 17, pp. 7678–7690, 2014.

[165] S. K. vanden Broucke and J. De Weerdt, "Fodina: A robust and flexible heuristic process discovery technique," *Decision Support Systems*, 2017.

[166] J. Munoz-Gama and J. Carmona, "Enhancing precision in process conformance: Stability, confidence and severity," in *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*, pp. 184–191, IEEE, 2011.

[167] M. Zhou and K. Venkatesh, *Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach*, vol. 6. World Scientific, 1999.

[168] K. Jensen and G. Rozenberg, *High-Level Petri Nets: Theory and Application*. Springer Science & Business Media, 2012.

[169] W. M. van der Aalst and K. Van Hee, "Business process redesign: A petri-net-based approach," *Computers in Industry*, vol. 29, no. 1, pp. 15–26, 1996.

[170] E. Best, R. Devillers, and M. Koutny, *Petri Net Algebra*. Springer Science and Business Media, 2013.

[171] W. M. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and E. Verbeek, "Conformance checking of service behaviour," *ACM Transactions on Internet Technology (TOIT)*, vol. 8, no. 3, p. 13, 2008.

[172] W. M. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and H. Verbeek, "Choreography conformance checking: An approach based on bpel and petri nets," 2005.

[173] R. Accorsi and T. Stocker, "On the exploitation of process mining for security audits: The conformance checking case," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pp. 1709–1716, ACM, 2012.

[174] J. Cardoso, J. Mendling, G. Neumann, and H. Reijers, "A discourse on complexity of process models," in *Business Process Management Workshops*, pp. 117–128, Springer, 2006.

[175] V. Gruhn and R. Laue, "Complexity metrics for business process models," in *9th International Conference on Business Information Systems (BIS 2006)*, vol. 85, pp. 1–12, 2006.

[176] G. Muketha, "A survey of business processes complexity metrics," 2010.

[177] K. B. Lassen and W. M. van der Aalst, "Complexity metrics for workflow nets," *Information and Software Technology*, vol. 51, no. 3, pp. 610–626, 2009.

[178] J. Cardoso, "Control-flow complexity measurement of processes and weyuker's properties," in *6th International Enformatika Conference*, vol. 8, pp. 213–218, 2005.

[179] J. Mendling, H. Reijers, and J. Cardoso, "What makes process models understandable?," *Business Process Management*, pp. 48–63, 2007.

[180] I. Vanderfeesten, J. Cardoso, J. Mendling, H. A. Reijers, and W. M. van der Aalst, "Quality metrics for business process models," *BPM and Workflow Handbook*, vol. 144, pp. 179–190, 2007.

[181] A. Alves de Medeiros, B. Van Dongen, W. van Der Aalst, and A. Weijters, "Process mining: Extending the α-algorithm to mine short loops," tech. rep., BETA Working Paper Series, 2004.

[182] L. Wen, W. M. van der Aalst, J. Wang, and J. Sun, "Mining process models with non-free-choice constructs," *Data Mining and Knowledge Discovery*, vol. 15, no. 2, pp. 145–180, 2007.

[183] J. Desel and J. Esparza, *Free Choice Petri Nets*, vol. 40. Cambridge University Press, 2005.

[184] B. Van Dongen, A. Alves de Medeiros, and L. Wen, "Process mining: Overview and outlook of petri net discovery algorithms," *Transactions on Petri Nets and Other Models of Concurrency II*, pp. 225–242, 2009.

[185] R. S. Garfinkel and G. L. Nemhauser, *Integer Programming*, vol. 4. Wiley New York, 1972.

[186] R. S. Mans, M. Schonenberg, M. Song, W. M. van der Aalst, and P. J. Bakker, "Application of process mining in healthcare–a case study in a dutch hospital," in *International Joint Conference on Biomedical Engineering Systems and Technologies*, pp. 425–438, Springer, 2008.

[187] W. M. van der Aalst and C. W. Gunther, "Finding structure in unstructured processes: The case for process mining," in *Application of Concurrency to*

System Design, 2007. ACSD 2007. Seventh International Conference on, pp. 3–12, IEEE, 2007.

[188] R. J. C. Bose and W. M. van der Aalst, "Trace alignment in process mining: Opportunities for process diagnostics.," in *BPM*, vol. 6336, pp. 227–242, Springer, 2010.

[189] L. Davis, "Handbook of genetic algorithms," 1991.

[190] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine Learning*, vol. 3, no. 2, pp. 95–99, 1988.

[191] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *Computer*, vol. 27, no. 6, pp. 17–26, 1994.

[192] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.

[193] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca, "Discovering expressive process models by clustering log traces," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 8, pp. 1010–1027, 2006.

[194] L. García-Bañuelos, M. Dumas, M. La Rosa, J. De Weerdt, and C. C. Ekanayake, "Controlled automated discovery of collections of business process models," *Information Systems*, vol. 46, pp. 85–101, 2014.

[195] R. Bose and W. M. van der Aalst, "Trace clustering based on conserved patterns: Towards achieving better process models," in *Business Process Management Workshops*, pp. 170–181, Springer, 2010.

[196] D. Ferreira, M. Zacarias, M. Malheiros, and P. Ferreira, "Approaching process mining with sequence clustering: Experiments and findings," in *International Conference on Business Process Management*, pp. 360–1374, Springer, 2007.

[197] S. Smirnov, H. A. Reijers, M. Weske, and T. Nugteren, "Business process model abstraction: A definition, catalog, and survey," *Distributed and Parallel Databases*, vol. 30, no. 1, pp. 63–99, 2012.

[198] S. Smirnov, H. Reijers, and M. Weske, "A semantic approach for business process model abstraction," in *Advanced Information Systems Engineering*, pp. 497–511, Springer, 2011.

[199] S. Smirnov, M. Weidlich, and J. Mendling, "Business process model abstraction based on synthesis from well-structured behavioural profiles," *Inter-*

national Journal of Cooperative Information Systems, vol. 21, no. 01, pp. 55–83, 2012.

[200] R. Conforti, M. Dumas, L. García-Bañuelos, and M. La Rosa, "Beyond tasks and gateways: Discovering bpmn models with subprocesses, boundary events and activity markers," in *International Conference on Business Process Management*, pp. 101–117, Springer, 2014.

[201] T. Baier, J. Mendling, and M. Weske, "Bridging abstraction layers in process mining," *Information Systems*, vol. 46, pp. 123–139, 2014.

[202] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, "Deriving petri nets from finite transition systems," *IEEE Transactions on Computers*, vol. 47, no. 8, pp. 859–882, 1998.

[203] D. Fahland and W. M. van der Aalst, "Repairing process models to reflect reality," in *International Conference on Business Process Management*, pp. 360–1374, Springer, 2007.

[204] J. Fleiss, *Statistical Methods for Rates and Proportions*. New York: John Wiley, 2003.

[205] W. Iba and P. Langley, "Introduction of one-level decision trees," in *International Conference on Machine Learning*, pp. 233–240, Morgan Kaufmann, 1992.

List of Figures

List of Tables

List of Algorithms

A

Acronyms

AA	Alpha Algorithm
ARW	Activity Ranking Weight
ASCRV	Average Sub-Model Complexity Reduction Value
ATC	Active Trace Clustering
AUC	Area Under the ROC Curve
BAW	Behaviour-Related Activity Weight
BPI	Business Process Improvement
BPIC	Business Process Intelligence Challenge
BPM	Business Process Management
BPMD	Business Process Model Discovery
BPMI	Business Process Mining
BRA	Behaviour-Related Activity
BRST	Behaviour-Related Sub-Trace
C-TDTC	Complexity-Related Top-Down Trace Clustering Technique
CCSTB	Complexity-Based Conditional Strict Significant Trace Behaviour
CRIB	Complexity-Related Insignificant Behaviours
CRM	Customer Relationship Management

CRSB Complexity-Related Significant Behaviours

CSP Closed Sequential Pattern

CTC Compound Trace Clustering Technique

DG Dependency Graph

DM Decision Mining

DT Decision Tree

E-Cardoso Extended Cardoso Metric

ED Euclidean Distance

EI Environment Item

ERP Enterprise Resource Planning

FCSTB Fitness-Based Conditional Strict Significant Trace Behaviour

FM Fuzzy Miner

GI Gini Index

GM Genetic Miner

GR Gain Ration

HM Heuristics Miner

IDC Inexpressible Behaviours Detection and Conversion

IG Information Gain

ILPM ILP Miner

IM Inductive Miner

LGC LinLog Graph Clustering

LRF Label-Related Function

MAE Mean Absolute Error

MAS Model Abstraction-Based Strategy

MCA Multi-Cluster Activity

MDS Model Division-Based Strategy

MEBS Mining Algorithm Enhancement-Based Strategy

MLCC Multi-Label Case Classification

MRST Maximal Behaviour-Related Sub-Trace

NB Naive Bayesian

PAIS Process-Aware Information Systems

PT-CD Place/Transition Connection Degree

SCM Supply Chain Management

SMI Sub-Model Improvement

SSTB Strict Significant Trace Behaviour

STB Significant Trace Behaviour

TDTC Top-Down Trace Clustering Technique

WAF Weighted Average Fitness

WFM Workflow Management