

TIME-BASED WORKFLOW MINING

A THESIS SUBMITTED TO  
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES  
OF  
MIDDLE EAST TECHNICAL UNIVERSITY

BY

DENİZ CANTÜRK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR  
THE DEGREE OF MASTER OF SCIENCE  
IN  
COMPUTER ENGINEERING

MAY 2005

Approval of the Graduate School of Natural and Applied Sciences

---

Prof. Dr. Canan Özgen  
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

---

Prof. Dr. Ayşe Kiper  
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science

---

Assoc. Prof. Dr. Nihan Kesim Çiçekli  
Supervisor

**Examining Committee Members**

Assoc. Prof Dr. Ferda Nur Alpaslan (METU,CENG) \_\_\_\_\_

Assoc. Prof. Dr. Nihan Kesim Çiçekli (METU,CENG) \_\_\_\_\_

Assoc. Prof. Dr. İlyas Çiçekli (Bilkent Unv.,CENG) \_\_\_\_\_

Assoc. Prof. Dr. İ.Hakkı Toroslu (METU,CENG) \_\_\_\_\_

Dr. Ayşenur Birtürk (METU,CENG) \_\_\_\_\_

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

Name, Last name : Deniz CANTÜRK

Signature :

# **ABSTRACT**

## **TIME-BASED WORKFLOW MINING**

Cantürk, Deniz

M. Sc., Computer Engineering

Supervisor : Assoc. Prof. Dr. Nihan Kesim Çiçekli

May 2005, 71 pages

Contemporary workflow management systems are driven by explicit process models, i.e., a completely specified workflow design is required in order to enact a given workflow process. Creating a workflow design is a complicated time-consuming process and typically there are discrepancies between the actual workflow processes and the processes as perceived by the management. Therefore, new techniques for discovering workflow models have been required. Starting point for such techniques are so-called “workflow logs” containing information about the workflow process as it is actually being executed. In this thesis, new mining technique based on time information is proposed. It is assumed that events in workflow logs bear timestamps. This information is used in to determine task orders and control flows between tasks. With this new algorithm, basic workflow structures, sequential, parallel, alternative and iterative (i.e., loops) routing, and advance workflow structure or-join can be mined. While mining the workflow structures, this algorithm also handles the noise problem.

Key words: Workflow management, data mining, workflow mining, process mining, workflow logs.

## ÖZ

### ZAMAN TEMELLİ İŞAKIŞI TAYİNİ

Cantürk, Deniz

Yüksek Lisans, Bilgisayar Mühendisliği

Tez Yöneticisi: Doç. Dr. Nihan Kesim Çiçekli

Mayıs 2005, 71 sayfa

Çağdaş iş akışı yönetim sistemleri açık işlem modelleri tarafından sürdürülmektedir. Örneğin, tanımlanmış iş akışı işlemlerinin sahnelenebilmesi için tüm ayrıntılarıyla belirtilmiş bir iş akışı tasarımına ihtiyaç duyulmaktadır. Bir iş akışı tasarımı oluşturmak karmaşık ve zaman alıcı bir işlemdir ve de asıl iş akışı işlemleri ile yönetimin algıladığı işlemler arasında farklılıklar bulunmaktadır. Bu nedenle iş akışı modeli keşfeden yeni yöntemlere ihtiyaç duyulmaktadır. Bu yöntemlerin başlangıç noktası ise "iş akışı günlüğü" diye adlandırılır ve işletilmiş iş akışı işlemleri bilgisi içerir. Bu tezde zaman bilgisine dayalı yeni bir iş akışı çıkarım yöntemi önerilmektedir. İş akışı günlüklerinin zaman bilgisi içerdiği kabul edilmektedir. Bu zaman bilgisi faaliyetler arası kontrol akışlarını ve faaliyet sıralarını belirlemede kullanılmaktadır. Bu önerilen yeni yöntem ile temel iş akışı yapıları olan ardışık, paralel, alternatif ve döngüsel yönelmeler ve ileri iş akışı yapısı olan veya-katılımı çıkarımları yapılabilmektedir. İş akışı yapılarının çıkarımları esnasında iş akışı günlüğündeki parazit problemi de ele alınmaktadır.

Anahtar Kelimeler: İş Akışı Yönetimi, Veri Keşfi, İş Akışı Keşfi, İşlem Keşfi, İş Akışı Günlüğü

To My Parents and Family

## **ACKNOWLEDGEMENTS**

I would like to thank my family, friends, supervisor, examining committee members and the society of the METU, Computer Engineering Department for their support, encouragement and invaluable ideas during my thesis study. Thank you all.

## TABLE OF CONTENTS

PLAGIARISM.....	iii
ABSTRACT.....	iv
ÖZ.....	v
ACKNOWLEDGEMENTS.....	vii
TABLE OF CONTENTS.....	viii
LIST OF TABLES.....	x
LIST OF FIGURES.....	xi
CHAPTER	
1. INTRODUCTION.....	1
2. RELATED WORK.....	8
2.1 Workflow Patterns.....	9
2.1.1 Basic Control Patterns.....	9
2.1.2 Advanced Branching and Synchronization Patterns.....	11
2.2 Process Mining.....	13
2.3 Workflow Mining.....	14
2.3.1 Rediscovering Workflow Models.....	16
2.3.2 Mining Workflows Having Duplicate Tasks.....	18
2.3.3 Mining Block-structured Workflows.....	21
3. MINING TIMED WORKFLOW LOGS.....	25
3.1 Event Logs with Two Types of Events.....	25
3.2 Ordering Relations.....	27
3.3 Measurement Used in Identifying the Relations.....	30



3.4	Identifying the Ordering Relations.....	33
3.5	The Algorithm.....	34
3.5.1	Constructing Average Values Tables.....	36
3.5.2	Determining the Edges Between Tasks.....	38
3.5.3	Timed Workflow Miner (TWM).....	44
4.	COMPARISON.....	48
4.1	Case Study 1.....	49
4.2	Case Study 2.....	51
4.3	Case Study 3.....	54
4.4	Other Issues.....	58
4.4.1	Noise.....	59
4.4.2	Complexity .....	59
4.4.3	Hidden Task.....	59
5.	CONCLUSION.....	61
	REFERENCES.....	63
	APPENDIX - A Sample Workflow Description File.....	66
	APPENDIX - B Sample Average Values Tables.....	67
	APPENDIX - C Larger version of Figure 21.....	71

## LIST OF TABLES

Table 1 An Event Log with START and COMPLETE events and occurrence times.....	7
Table 2 Average values table for some Task C .....	37
Table 3 Edge table after first pass .....	39
Table 4 Edge table after second pass .....	43
Table 5 Log $W_A$ summary.....	49
Table 6 Vertex–Edge matrix for figure 14.....	50
Table 7 Vertex–Edge matrix for figure 15.....	50
Table 8 Log $W_B$ summary .....	51
Table 9 Vertex–Edge matrix for Figure 16.....	53
Table 10 Vertex–Edge matrix for Figure 17.....	53
Table 11 Log $W_C$ summary.....	54
Table 12 Vertex–Edge matrix for Figure 20 .....	56
Table 13 Vertex–Edge matrix for Figure 21 .....	56

## LIST OF FIGURES

Figure 1 A process model corresponding to the workflow log.....	4
Figure 2 Rediscovery Problem .....	16
Figure 3 The Release Process .....	19
Figure 4 Split Operation .....	20
Figure 5 Succession relations.....	28
Figure 6 Following relations.....	29
Figure 7 Intersection relations.....	29
Figure 8 Formal description of the algorithm of the first pass .....	39
Figure 9 Induced workflow diagram after first pass.....	40
Figure 10 Formal description of the algorithm of the second pass.....	42
Figure 11 Induced workflow diagram of second pass.....	43
Figure 12 TWM showing Average Values Table of Task A .....	46
Figure 13 Mined workflow model .....	47
Figure 14 Workflow diagram of Log $W_A$ generated by TWM.....	50
Figure 15 Workflow diagram of Log $W_A$ generated by ProM.....	50
Figure 16 Workflow diagram of Log $W_B$ generated by TWM.....	52
Figure 17 Workflow diagram of Log $W_B$ generated by ProM.....	52
Figure 18 Loop of order 2.....	54
Figure 19 Workflow containing third order parallel tasks.....	55
Figure 20 Workflow diagram of Log $W_C$ generated by TWM.....	56
Figure 21 Workflow diagram of Log B obtained from ProM.....	56
Figure 22 States for incorrect implication.....	57
Figure 23 Average values table for activity H.....	58

# CHAPTER 1

## INTRODUCTION

Workflow concept is first used in manufacturing and the office. Process in manufacturing industry and the office is studied to improve the efficiency by dealing with the standard aspects of the work activities [10]. *Workflow* consists of activities that are executed in a coordinated way by different resources. In other words *workflow* is the coordination of a collection of activities to achieve a goal. An *activity (task)* is a definition of work to be done. It defines the work in different ways such as file with textual description, a form, an email or a computer program [15]. A resource that performs the task may be a person, a machine or a software system. A resource starts the workflow by performing the beginning task in the workflow. Then other tasks are performed in a coordinated way until the goal is achieved or a task in the route of final task is aborted. The tasks that are performed in the period from the start to stop constitutes a *workflow instance*.

Execution of the activities in a workflow can be coordinated by a human controller or a software system. Such a software system is called *workflow management system* [15]. Workflow management system has the ability to record the execution information of any task involved in the workflow, to *workflow log* files. Workflow log contains important information since this is recorded when the

task takes place. This recorded information should be the validation of the workflow design. To validate a designed workflow, another workflow can be generated by using the workflow log. The process of generating the workflow from a workflow log is called *workflow mining*. If the generated workflow and the designed workflow are not equivalent, differences can be studied for the improvement of the workflow.

During the last decade workflow management concepts and technology have been applied in many enterprise information systems. Workflow management systems such as Staffware, IBM MQSeries, COSA offer generic modeling and enactment capabilities for structured business processes [2].

By making graphical process definitions, i.e., models describing the life cycle of a typical case (workflow instance) in isolation, one can configure these systems to support business processes. Besides pure workflow management systems, many other software systems have adopted workflow technology, for example ERP (Enterprise Resource Planning) systems such as SAP, PeopleSoft, Baan and Oracle, CRM (Customer Relationship Management) software, etc. [2] Despite its promise, many problems are encountered when applying workflow technology. One of the problems is that these systems require a workflow design, i.e., a designer has to construct a detailed model accurately describing the routing of work. Modeling a workflow is far from trivial: It requires deep knowledge of the workflow language and lengthy discussions with the workers and management involved.

Instead of starting with a workflow design, we start by gathering information about the workflow processes as they take place. We assume that it is possible to record events satisfying the following:

- Each event refers to a task (i.e., a well-defined step in the workflow)
- Each event refers to a case (i.e., a workflow instance)
- Events are totally ordered.

Any information system using transactional systems such as ERP, CRM, or workflow management systems will offer this information in some form. Note that we do not assume the presence of a workflow management system. The only assumption we make, is that it is possible to collect workflow logs with event data. These workflow logs are used to construct a process specification, which adequately models the behavior registered. We use the term *workflow mining* for the method of distilling a structured process description from a set of real executions.

To illustrate the principle of workflow mining, we consider the workflow log shown in Table 1. This log contains information about four cases (i.e., workflow instances). The log shows that for the first case the tasks A, B, C, E, H, I, J and K have been executed. For case 2 the tasks A, D and J are executed but in this case complete event of Task D did not occur, possibly Task D is aborted, therefore complete event could not be logged. This kind of workflow instances are called *noise*, the algorithm proposed in this thesis handles the noisy traces too. For case 3, A, D, F, G, H, I, J and K have been executed. It is different from case 1 since in case 1, B, C and E have been executed but in case 3 D, F and G have been executed instead. For case 4, A, B, E, H, I, J and K have been executed. It is different from case 1 since in case 1 C has been executed but in case 4 it has not and also it is different from case 1 since in case 3 D, F and G have been executed but in case 4 B and E have been executed instead. Each case starts with the execution of A and ends with the execution of K.

Based on the information shown in Table 1 and by making some assumptions about the completeness of the log (i.e., assuming that the cases are representative and a sufficiently large subset of possible behaviors is observed), the process model shown in Figure 1 can be deduced. Workflow starts with task A and finishes with task K. After executing the task A, tasks B, C, D and J are executed in four different combinations: (i) B, C and J, (ii) B and J, (iii) C and J and (iv) D and J. After A one of the four combinations is executed. The split from A is a complex split; the executed tasks after A depend on data of A. After executing B or C, E is executed. After D is executed, F and G are executed. Split from D is called And-Split. After F and G are executed, H is executed. Join on H is called Or-Join since, from case 3, we know that H has been executed before F has completed but after G has completed. After J is completed, I is executed. After H and I are executed, K, the final task, is executed. Join on K is called And-Join, since in valid cases 1,3 and 4, K has been executed after both H and I have been completed.

By distinguishing between start events and end events for tasks it is possible to explicitly detect parallelism and Or-Splits and Or-Joins.

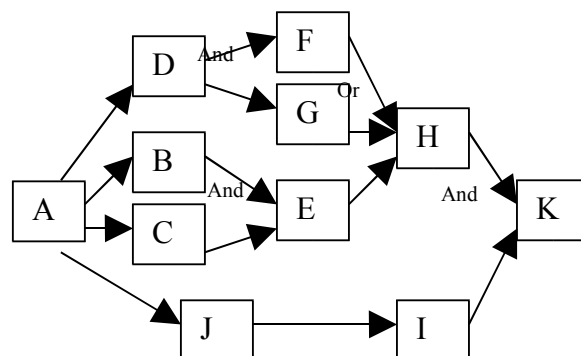


Figure 1 A process model corresponding to the workflow log.

The first two columns of Table 1 contain the minimal information (case and task information). By using the minimal information of a complete workflow log we assume to provide process models such as the one shown in Figure 1. In many applications the workflow log contains a timestamp for each event and this information can be used to extract information about the performance of the process, e.g., bottlenecks in the process. In this thesis, exploring ways to mine timed workflow logs is discussed. In the log shown in Table 1 both start and complete time can be seen. By the help of this timing information the proposed algorithm can explore workflow models despite incomplete workflow logs.

Workflow mining, in the most general case, is introduced as constructing a process model from event-based logged data. The first introduced approach, mined workflows assumed to be containing no loops and logs assumed noise-free [5]. Next approach enhanced the previous approach to mine basic loops [18]. Later, noise problem is studied. After mining workflow from noisy logs [2], timing information in the log is applied to mine workflows [3]. In the most recent work on workflow mining short loops have been discovered, by using start and complete events [19].

We have developed an algorithm for mining workflow graphs from a given log of events. We assume that the events are recorded with their start and complete times in the log. The algorithm uses this timing information to extract the dependencies between the events, and thus the tasks that they represent. In the algorithm, tasks are ordered by using the waiting time between activities and execution time of tasks. The metric used in ordering relation that is based on timing information also allows to compete with noise and to mine or-join and short loops. Current approaches depend on the counts of different configurations



therefore mined workflow does not contain any timing information. However, in this thesis we propose an algorithm to mine timed workflows. The mined workflow, contains all the timing information about the workflow such as the average execution time, average waiting time between tasks and the total workflow duration in average.

The rest of the thesis is organized as follows. First, the related work done in workflow mining is summarized in Chapter 2. We discuss the current workflow mining approaches and tools. In Chapter 3, first the definitions that are used in the proposed algorithm are presented. Next we present the algorithm with examples. Chapter 4 compares the result of proposed algorithm with the existing workflow mining tools. The conclusion and the future work are presented in chapter 5.

Table 1 An Event Log with START and COMPLETE events and occurrence times

Case	Task	Event	Date
Case 001	TASK A	START	12.09.2004 21:39:10
Case 001	TASK A	COMPLETE	12.09.2004 21:39:17
Case 001	TASK J	START	12.09.2004 21:39:25
Case 001	TASK B	START	12.09.2004 21:39:25
Case 001	TASK C	START	12.09.2004 21:39:26
Case 001	TASK C	COMPLETE	12.09.2004 21:39:30
Case 001	TASK J	COMPLETE	12.09.2004 21:39:34
Case 001	TASK B	COMPLETE	12.09.2004 21:39:34
Case 001	TASK I	START	12.09.2004 21:39:36
Case 001	TASK E	START	12.09.2004 21:39:38
Case 001	TASK I	COMPLETE	12.09.2004 21:39:39
Case 001	TASK E	COMPLETE	12.09.2004 21:39:42
Case 001	TASK H	START	12.09.2004 21:39:51
Case 001	TASK H	COMPLETE	12.09.2004 21:39:53
Case 001	TASK K	START	12.09.2004 21:39:59
Case 001	TASK K	COMPLETE	12.09.2004 21:40:03
Case 002	TASK A	START	12.09.2004 21:39:11
Case 002	TASK A	COMPLETE	12.09.2004 21:39:11
Case 002	TASK J	START	12.09.2004 21:39:11
Case 002	TASK D	START	12.09.2004 21:39:12
Case 002	TASK J	COMPLETE	12.09.2004 21:39:24
Case 003	TASK A	START	12.09.2004 21:39:11
Case 003	TASK A	COMPLETE	12.09.2004 21:39:11
Case 003	TASK J	START	12.09.2004 21:39:11
Case 003	TASK D	START	12.09.2004 21:39:12
Case 003	TASK D	COMPLETE	12.09.2004 21:39:21
Case 003	TASK F	START	12.09.2004 21:39:22
Case 003	TASK G	START	12.09.2004 21:39:22
Case 003	TASK G	COMPLETE	12.09.2004 21:39:23
Case 003	TASK H	START	12.09.2004 21:39:23
Case 003	TASK J	COMPLETE	12.09.2004 21:39:24
Case 003	TASK I	START	12.09.2004 21:39:24
Case 003	TASK F	COMPLETE	12.09.2004 21:39:26
Case 003	TASK I	COMPLETE	12.09.2004 21:39:30
Case 003	TASK H	COMPLETE	12.09.2004 21:39:30
Case 003	TASK K	START	12.09.2004 21:39:36
Case 003	TASK K	COMPLETE	12.09.2004 21:39:39
Case 004	TASK A	START	12.09.2004 21:39:12
Case 004	TASK A	COMPLETE	12.09.2004 21:39:13
Case 004	TASK B	START	12.09.2004 21:39:14
Case 004	TASK J	START	12.09.2004 21:39:14
Case 004	TASK B	COMPLETE	12.09.2004 21:39:17
Case 004	TASK J	COMPLETE	12.09.2004 21:39:17
Case 004	TASK I	START	12.09.2004 21:39:19
Case 004	TASK E	START	12.09.2004 21:39:25
Case 004	TASK E	COMPLETE	12.09.2004 21:39:28
Case 004	TASK I	COMPLETE	12.09.2004 21:39:30
Case 004	TASK H	START	12.09.2004 21:39:31
Case 004	TASK H	COMPLETE	12.09.2004 21:39:38
Case 004	TASK K	START	12.09.2004 21:39:45
Case 004	TASK K	COMPLETE	12.09.2004 21:39:47

## CHAPTER 2

### RELATED WORK

A workflow is a collection of cooperating, coordinated activities designed to accomplish a completely or partially automated process. An activity in a workflow is performed by an agent that can be a human, a device or a program. A workflow management system provides support for modeling, executing and monitoring the activities in a workflow.

The Workflow Management Coalition (WfMC) defines a reference model that describes the major components and interfaces within a workflow architecture. In a workflow, activities are related to one another via flow control conditions (transition information). According to this reference model we identify four routings that is *sequential*, *parallel*, *conditional* and *iteration*.

Among the most common frameworks for specifying workflows, control flow graphs are most appropriate for showing the execution dependencies of the activities in a workflow. It provides a good way to visualize the overall flow of control. A typical graph specifies the initial and the final activity in a workflow, the successor activities for each activity in the graph, and whether all of these successor activities must be executed concurrently, or it is sufficient to execute only one branch depending on a condition.

Several recent works have addressed the problem of discovering an unknown workflow model of a given process, by looking at the logs of a number of its executions. In several organizations, it has become increasingly popular to document and log the steps that make up a typical business process. In some situations, a workflow model of such processes is developed and it becomes important to know if such a model is actually being followed by analyzing the available activity logs. In other scenarios no model is available and with the purpose of evaluating cases or creating new production policies one is interested in learning a workflow representation of such activities. For these reasons empirically building process models from logs is of great interest and still relatively unexplored. Such a problem is called *workflow mining* because the usual representation of work processes is workflow graphs.

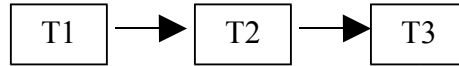
## **2.1 Workflow Patterns**

In this section we first review the well-defined patterns [20] that can be used in a workflow graph. We then present a survey of the existing workflow mining approaches to which our work will be compared later.

### **2.1.1 Basic Control Patterns**

#### **i) Sequence**

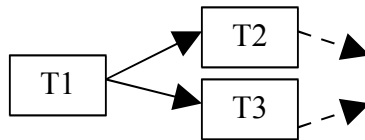
An activity in a workflow process is enabled after the completion of another activity in the same process. The sequence pattern is used to model consecutive steps in a workflow process and is directly supported by each of the workflow management systems available.



Possible execution: {T1, T2, T3, ...}

**ii) Parallel Split (AND - Split)**

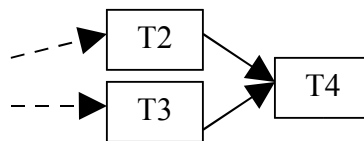
Parallel Split is a split of a single thread of control into multiple threads of control in the workflow process, which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order.



Possible executions: {T1, T2, T3, ...}, {T1, T3, T2, ...}

**iii) Synchronization (AND - Join)**

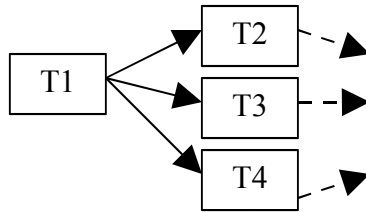
Synchronizer is a point in the workflow process where multiple parallel subprocesses/activities converge into one single thread of control. It is an assumption of this pattern that each incoming branch of a synchronizer is executed only once.



Possible executions: {..., T2, T3, T4}, {..., T3, T2, T4}

**iv) Exclusive Choice (XOR - Split)**

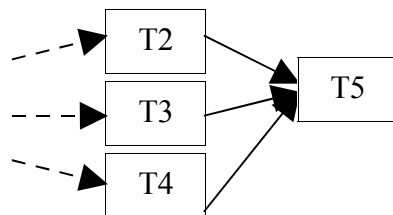
Exclusive choice is a pattern in the workflow process such that based on a decision or workflow control data, one of several branches is chosen.



Possible executions: {T1, T2, ...}, {T1, T3, ...}, {T1, T4, ...},  
 (But not {T1, T2, T3, ... })

**v) Simple Merge (XOR - Join)**

Simple merge is a pattern in the workflow process where two or more alternative branches come together without synchronization. It is an assumption of this pattern that none of the alternative branches is ever executed in parallel.

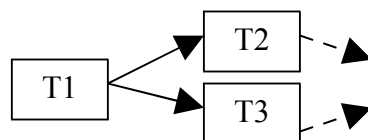


Possible executions: {..., T2, T5}, {..., T3, T5}, {..., T4, T5}

**2.1.2 Advanced Branching and Synchronization Patterns**

**i) Multi-choice (OR - Split)**

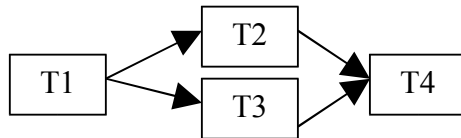
Multi-choice is a workflow pattern in the workflow process where, based on a decision or workflow control data, a number of branches are chosen.



Possible executions: {T1, T2, ...}, {T1, T3, ...}, {T1, T2, T3, ...},  
 {T1, T3, T2, ...}

## ii) Synchronizing Merge

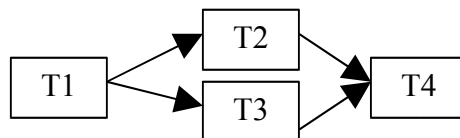
Synchronizing merge is a pattern in the workflow process where multiple paths converge into one single thread. If more than one path is taken, synchronization of the active threads needs to take place. If only one path is taken, the alternative branches should reconverge without synchronization. It is an assumption of this pattern that a branch that has already been activated, cannot be activated again while the merge is still waiting for other branches to complete. This pattern usually used for joining the threads that are created by multi-choice.



Possible executions: {T1, T2, T4}, {T1, T3, T4}, {T1, T2, T3, T4},  
{T1, T3, T2, T4}

## iii) Multi-merge

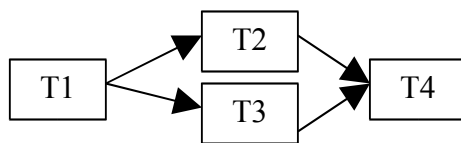
Multi-merge is a pattern in the workflow process where two or more branches reconverge without synchronization. If more than one branch gets activated, possibly concurrently, the activity following the merge is started for any activation of any incoming branch.



Possible executions: {T1, T2, T4, T3, T4}, {T1, T3, T4, T2, T4},  
{T1, T2, T3, T4, T4}, {T1, T2, T3, T4, T4}

#### iv) Discriminator (OR- Join)

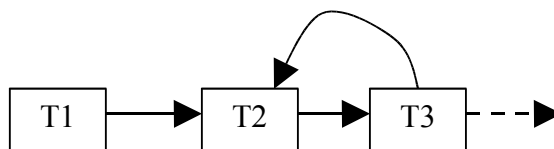
Discriminator is a pattern in the workflow process that waits for one of the incoming branches to complete before activating the subsequent activity. From that moment on it waits for all remaining branches to complete and "ignores" them. Once all incoming branches have been triggered, it resets itself so that it can be triggered again (which is important otherwise it could not really be used in the context of a loop). It is also known as Or-Join.



Possible executions: {T1, T2, T4}, {T1, T3, T4}, {T1, T2, T4, T3},  
{T1, T3, T4, T2}, {T1, T2, T3, T4}, {T1, T3, T2, T4}

#### v) Arbitrary Cycles

Arbitrary cycle is a pattern in the workflow process where one or more activities can be done repeatedly. This pattern is also called loop. Loops containing one or two tasks are called short-loop



Possible executions: {T1, T2, T3,...}, {T1, T2, T3, T2, T3,...}, ...

## 2.2 Process Mining

The idea of process mining is not new. In its evolution first we see Cook and Wolf. They have studied process discovery in the software engineering process.



They have developed three methods for process discovery that use different approaches [6]. First one uses the idea of a neural network. This one is the least promising one of all. Second one uses purely algorithmic approach that builds a finite state machine (FSM) where states are fused if their possible behaviors in next  $k$  step are identical. The third one uses Markovian approach, which is the mixture of the algorithmic and statistical methods. It can deal with noise.

Cook and Wolf improve their work to be able to deal with concurrent processes [7]. In the extension of their work they propose specific metrics, such as entropy, event type counts, periodicity, and causality, in order to discover models from event streams. However, they do not provide an approach to generate explicit process models. On the other hand, the final goal of workflow mining is to find an explicit workflow model. Nevertheless, in Cook and Wolf's approach only the set of dependency relations between events is discovered. Later, Cook and Wolf provide a measure to quantify differences between a process model and the behavior that is recorded in event-based data [8].

## **2.3 Workflow Mining**

The idea of applying process mining to workflow management was first introduced in [5]. This work is based on workflow graphs. In this paper, two problems are addressed. The first problem is to find a workflow graph from the given workflow log. The second problem is to identify the edge conditions. For the first problem an algorithm is provided that generates an explicit workflow model where the nature of joins and splits are not identified. This approach does not allow loops in the workflow graph. But it can deal with some iteration by enumerating the tasks and folding the graph.

Schimm [17] has introduced an approach to mine hierarchically structured workflow processes. However, this approach requires all the splits and joins for all tasks in the workflow model to be balanced. Herbst and Karagiannis address the problem of workflow mining from a different point of view. They use induction in their approach [11]. Their approach also allows concurrency. It uses stochastic activity graph (SAG) in the intermediate representation. Finally it generates a workflow model described in ADONIS modeling language. This approach uses induction to split and merge task nodes in order to find the process model. This approach has a major difference from other approaches since it can deal with the task that occurs multiple times in the workflow model. Graph generation is done in a similar way as in [5]. Splits and joins are identified as whether AND or XOR in the transformation step where the intermediate representation transformed into final representation of ADONIS format which is in block-structured model.

The work in [18] is characterized by the focus on workflow processes with concurrent behavior (rather than adding ad hoc mechanisms to capture parallelism). In [18] a heuristic approach using rather simple metrics is used to construct so-called “dependency/frequency tables” and “dependency/frequency graphs”. The preliminary results presented in [18] only provide heuristics and focus on issues such as noise.

The approach described in [2] introduces an algorithm called  $\alpha$ -algorithm. This approach differs from these approaches in the sense that for the  $\alpha$  algorithm it is proven that for certain subclasses it is possible to find the right workflow model. In [3] the  $\alpha$  algorithm is extended to incorporate timing information; nevertheless the timing information is not the ingredient of the algorithm. Timing information is applied on the graph of the process model constructed by  $\alpha$  algorithm.

### 2.3.1 Rediscovering Workflow Models

The approach examined in this section has a theoretical basis. In this approach only the subset of workflows, namely Petri Nets, are dealt with. Studies have been done on these approaches are stated in [2,3]. In this work there are two tools EMiT [3], and MiMo[2] implemented to verify the idea of the approach.

This approach needs the workflow logs to be complete and noise-free. With these ideal conditions, in this theoretical approach, studies mainly focus on rediscovery problem. They want to classify the workflow processes which can be reconstructed by only using the their log information. In fact it is really hard work since any workflow instance of a workflow model is a sequences of trace lines in a log without of the information of split and joins in the workflow model.

The rediscovery problem is illustrated in Fig 2. Assume, L be a log consists of the workflow instances of the process model defined by a class of Petri-Nets WF-Net  $WF_1$ . Using the information of L and the algorithm of this approach mined WF-Net is  $WF_2$ . In [3], for which class of WF-Nets the equation  $WF_1=WF_2$  held is explored.

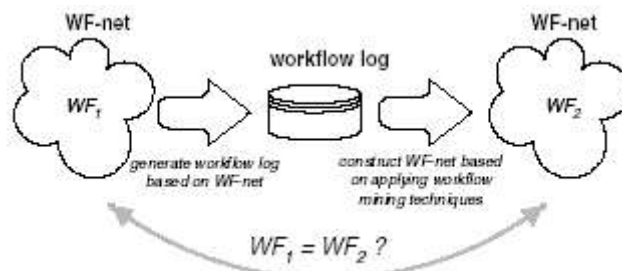


Figure 2 Rediscovery Problem

Another issue explained in [2,3] is that it is not possible to rediscover all classes at WF-Net, but with the  $\alpha$ -algorithm presented in [2,3] most of these classes can be rediscovered. To use  $\alpha$  algorithm, workflow logs have to be complete i.e. If

two events can follow each other, they will follow each other at least once in the log, all possible sequence containment is not the case.

$\alpha$  - algorithm establishes four ordering relations:  $>_w$ ,  $\rightarrow_w$ ,  $\parallel_w$  and  $\#_w$ , that are derived from the log.  $a >_w b$  represents  $b$  succeeds  $a$  in the log,  $a \rightarrow_w b$  represents the  $a$  is followed by  $b$  in the log possibly task occurrences between  $a$  and  $b$ .  $a \parallel_w b$  represents  $a$  and  $b$  executed in parallel.  $a \#_w b$  represents  $a$  and  $b$  disjoint tasks that is neither  $a$  follows  $b$  nor  $b$  follows  $a$ .

This theoretical approach requires perfect information, that is the workflow log is complete in the sense described before and everything recorded in the log is correct and every trace lines in the log is valid i.e. log is *noise-free*. In the real world, there is no log exists that is complete and noise-free. Therefore, in the real world, decision at the ordering of the events becomes harder. For example, if  $a$  is directly followed by  $b$  erroneously in the log,  $a$  will be directly followed by  $b$  in the constructed workflow and other errors might be done. Consequently, in order to overcome this severe problem, they developed a mining technique, which depends on heuristics. They called this heuristic approach as extended  $\alpha$ -algorithm [19]. The work done in the heuristics approach consists of three steps which are construction of D/F tables, derivation of ordering relations from D/F table and reconstruction of WF-Net.

While constructing the D/F tables, the overall occurrence count of *task a*, the count of *task a* directly preceded by *task b*, count of *task a* directly followed by *task b*, the count of *task a* directly or indirectly preceded by *task b* but before the previous appearance of *task b*, the count of *task a* directly or indirectly followed by *task b* but before the next appearance of *task a*, and finally a metric that indicates the strength of the causal relation between *task a* and another *task b* are extracted

from the workflow log. In the second step basic relations ( $>_w$ ,  $\rightarrow_w$ ,  $\parallel_w$  and  $\#_w$ ) in  $\alpha$ -algorithm are determined by using simple heuristics. At the last step workflow is constructed by using the relations in second step.

### **2.3.2 Mining Workflows Having Duplicate Tasks**

The approach presented in the preceding section uses graphical model to represent the workflow. Therefore every task in the model must have a different name although they have performed the same activity. Nevertheless, for some cases, it is not possible. The requirement for the unique names can be satisfied by enumerating the activities with same name, for example notify Eng occurs twice in Fig 3. To have different names they can be renamed as notify Eng-1, however this renaming requires the knowledge of workflow structure. This contradicts with the aim of workflow mining since the main goal is to find the workflow structure.

A solution for mining workflow models with duplicate tasks is presented in [12]. This solution contains two steps which are induction and transformation. In the induction step stochastic activity graphs (SAG) are used to simplify the graph generations algorithm that is put in to search procedure.

The search procedure takes some of its idea from machine learning and grammatical inference. This approach searches for mapping from the task instances in the workflow model. Search space for this process is the lattice of such mappings. Ordering used in the mapping is generality. Mapping is considered as more general than other or more specific than other one. This lattice is bounded from top by most general mapping in which every task with the same name in the workflow log mapped to only one task in the workflow model and also bounded from bottom by most specific mapping in which every task instance in the

workflow log is one-to-one and onto mapped to tasks in the workflow model. All of the mappings from top to bottom in the lattice are searched to find optimal mapping. In the split operation, task instances mapped to the same task are split into two groups and mapped to these distinct named tasks.

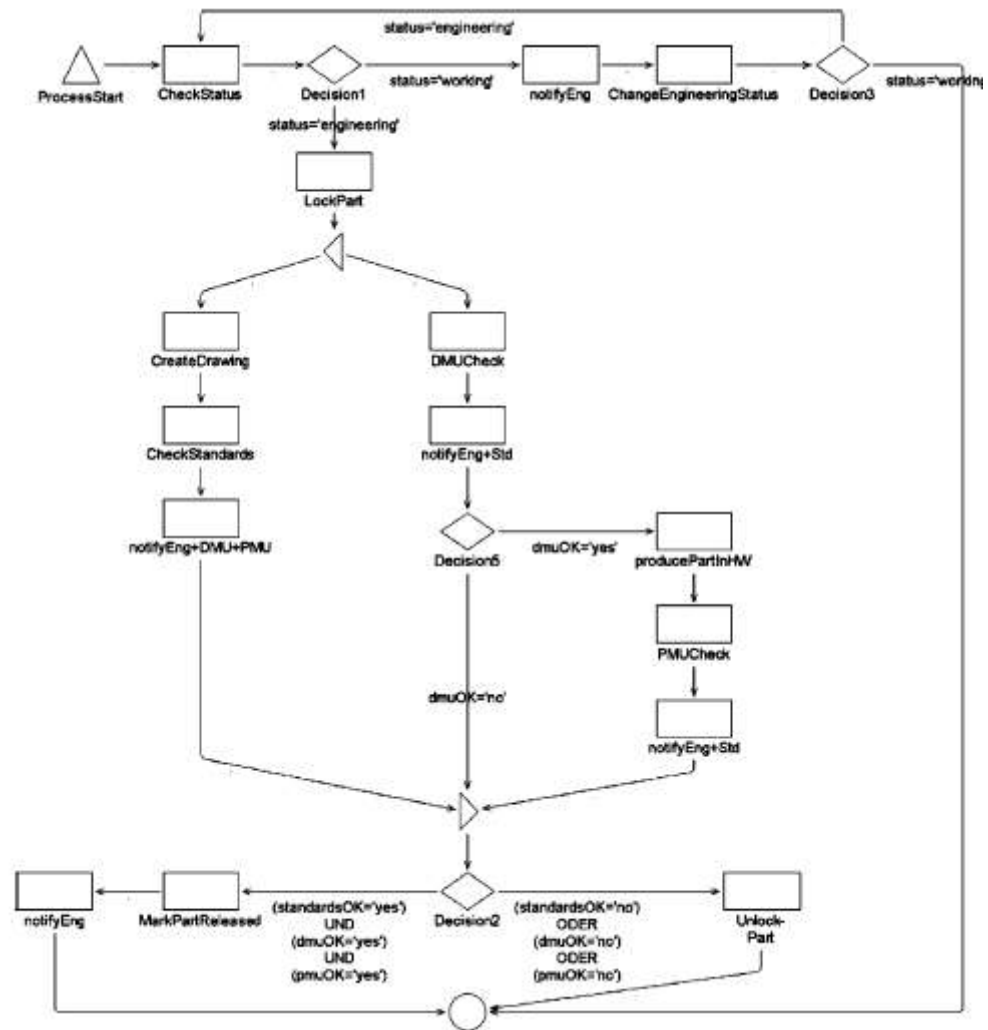


Figure 3 The Release Process

In fact, this approach, which is very similar to the approach presented in [5], has a different definition for dependency relation and two additional steps which are inserting copies of tasks and clustering tasks having same predecessors. Its

difference from the approach in [16] is determination of dependencies. In this approach every pair of task taken into consideration independent from the intermediate task in between them, but only the pairs of direct successors is considered to determine the dependency relation in [2,3]. This search is guided by SAG per sample and the algorithm uses beam-search. Calculation of SAG requires a stochastic sample that means that induction algorithm requires n different ordering case for ordering the n workflow instances of some task. In [2,3] only one ordering is enough for such cases. By using this information likelihood of the SAG and probability of the edges and task can be calculated, common and rare tasks can be determined.

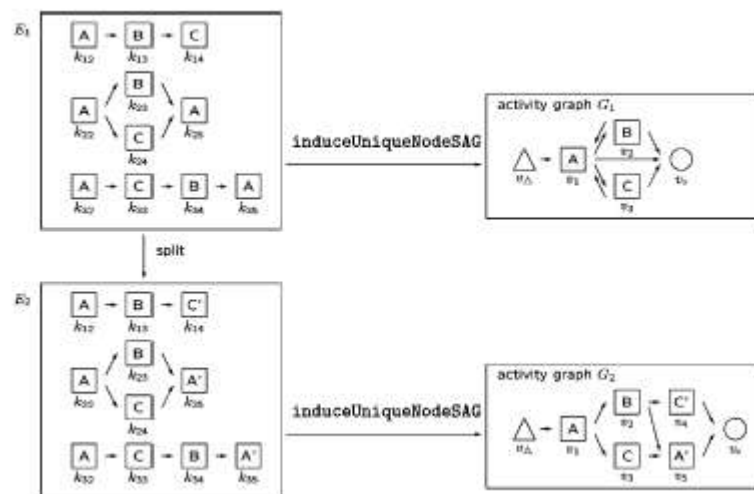


Figure 4 Split Operation

The other step, in this approach is the transformation step in which constructed SAG is transformed into block-structured workflow model in the ADONIS [13] format. Transformation is required since the SAG constructed in induction step cannot express the parallel and alternative routings separately. This step can be divided into three sub-steps. In the first sub-step synchronization structure of the workflow instances in the log is analyzed. Next, synchronization

structure of the workflow model is generated. At the Last sub-step workflow model is generated. The described algorithm is implemented by the workflow mining tool InWoLVe [12]. InWoLVe also contains two additional induction algorithms that are restricted to sequential workflow models. In addition InWoLVe has an ability of interchanging process models and workflow logs with the business management system ADONIS. InWoLVe is tested with real-life and artificial logs. Fig 3 is one of the real-life workflow models that is generated by InWoLVe.

### **2.3.3 Mining Block-structured Workflows**

The last approach is used for mining block-structured workflow models. It is different from previous two approaches in two points. First it uses rewriting technique instead of graph-based techniques. The other important difference in this approach is that only the block structured patterns are taken into consideration. Goal of this algorithm is to mine minimal and complete workflow models. In the complete workflow model all cases are covered by the mined model, in the minimal workflow model only the recorded cases are covered. To achieve this goal it uses a stronger completeness notion than the notion in the rediscovery problem that considers only the direct succession.

In order to mine a workflow model from event-based data it is required to have an idea on the type of mined workflow model. Types of workflow models are different from each other in terms of the workflow language being used and the class of the workflow models considered. Workflow meta-models are divided into two groups: graph-oriented meta-model and block-oriented meta-model, in the most general case. This approach is built on block-oriented meta-model. Models of block-oriented meta-model is always well formed.

Block structured workflow models are composition of nested blocks. There



are two types of building blocks that are constants and operands. Operands model the process flows, constants model the tasks or nested sub-workflows. To design a block structured workflow model one operand is set as start point then other operands and constants are embedded to it until the nested structure satisfy the desired structure. Operands are embedded to enable nesting and constants are embedded to stop nesting.

Block structured workflow model can be assumed as a tree whose leaves are the operands and constants. Besides their tree representation, they can be expressed as a set of terms. For example, Assume S is a sequential working operand, P is a parallel execution operand and T1, T2, T3 are different tasks then  $P(S(T1,T2),T3)$  means that T1 and T2 are performed sequentially and T3 is performed in parallel both T1 and T2. By definition, all of the terms are well formed. Furthermore, with the help of axioms on commutativity, associativity, identity and inversion, algebra can be specified. These axioms form a basis for term rewriting systems that are used in workflow mining. The process that constructed block-structured workflow model from event-based workflow log consists of the following five steps that are performed sequentially:

In the first step of the procedure, workflow log is read to build a trace for each process instance in the log. Trace is a data structure that keeps all start and complete events of the task instance ordered by the time of events. After building the traces, they are combined according to their start and complete time. Each trace group constitutes a path in the process model.

In the second step, initial process model is constructed from all trace groups. This model is in Disjunctive Normal Form (DNF). A process model in DNF starts with an alternative operator and enumerates inside this block all

possible process flows as a block is constructed by the algorithm and added to alternative operator that builds the root of two models.

The third step deals with the relation between tasks that result from the random order of performing task without a real precedence between them. This made-up precedence relation has to be identified and removed from the generated model. To be able to identify made-up precedence relations, generated model needs to be transformed. Transformation is done by a term rewriting system into form that enumerates all sequences of tasks inside parallel operators embedded into the overall alternative, then made-up precedence relation is detected by the search algorithm. In the search algorithm, the smallest subset of sequences that completely explains the corresponding blocks in the initial model is found to determine the made-up precedence relation. All sequences out of subset are made-up precedence relations and therefore removed. Initial transformation is reversed by a term rewriting system at the end of step.

The fourth step deals with the DNF of the process model, which is built in the second step. Overall alternatives of the model need to be split before the point where the decisions have to be released. So, partial alternatives matching to that condition have to be moved. This is done by another term rewriting system. It is built on top of distributivity algorithm. It also merges block while shifting these alternative operator later position in their model.

The last step is the decision making step which is done by inductive decision tree. Induction is applied to all decision points in the workflow. This step requires the trace data of the data of workflow to build the decision tree. By the help of context data kept in the traces, decision tree induction builds the decision trees. These trees are transformed into rules and then associated to partial

alternative operators. In fact this step is optional. Without performing the last step, the condensed form of model could be obtained. However after performing the last step the desired minimal and complete block structured workflow model comes out. All other details of this approach are in [17].

Process miner [17], is a workflow mining tool that supports mining block-structured model. The tool reads data from the event-based data i.e. workflow logs or XML formatted files. After reading data, it performs all the steps up to decision-making step. Decision-making step is not performed unless context data are provided. It has a graphical interface, so it displays the output workflow in a graphical editor in the form of the diagram and a tree. In addition, it enables users to edit the output workflow model and to export it for later use. Full description and abilities of process miner is explained in [17].

## CHAPTER 3

### MINING TIMED WORKFLOW LOGS

In this chapter we present the workflow mining algorithm that we have developed. The algorithm takes a log of timed events and extracts the workflow graph using the timing information of the events. Before we present the algorithm, we give some formal definitions that will be used in the explanation of the algorithm. First, we formally define what the event logs are. Then, a new notion of ordering relations on tasks based on the two event types START and COMPLETE and their timestamps is defined. Next, the measurements that are used in the determination of orderings are given. Last, identifying the ordering relations is defined.

#### 3.1 Event Logs with Two Types of Events

Existing approaches do not consider event timestamp value [1-7,17,18]. They do not consider the types of events [1-7,17,18] either. Tasks are either considered to be atomic or only the completion of a task is considered (i.e., just event type COMPLETE). One way to deal with this is to consider the start and completion of a task as two atomic tasks. EMiT [3] uses some pre- and post-processing to incorporate multiple event types, but it does not incorporate this in the mining algorithm and ordering relations. In this thesis, we propose a

fundamentally different approach where parallelism and or-joins are detected explicitly by registering overlapping activities. There are two event types: START and COMPLETE. Therefore, a task name, an event type and the timestamp of the event characterize each event.

**Definition 1 (Event).** Let  $T$  be a set of tasks and  $ts$  is a timestamp.

$E = T \times \{\text{START}, \text{COMPLETE}\} \times ts$  is a set of events over  $T$ .  $(t, \text{START}, ts) \in E$  denotes the start of some task  $t$  with timestamp  $ts$  and  $(t, \text{COMPLETE}, ts) \in E$  denotes the completion of  $t$  with timestamp  $ts$ . For convenience, we also introduce the following notation for  $e \in E$ :  $e.task$  refers to the task,  $e.type$  refers to the event type and  $e.ts$  refers to the event timestamp. If  $e = (t, \text{START}, '12-10-2004 10:30:50')$ , then  $e.task = t$ ,  $e.type = \text{START}$  and event occurs on 12.10.2004 at 10:30:50. If  $e = (t, \text{COMPLETE}, '12-10-2004 10:35:22')$ , then  $e.task = t$ ,  $e.type = \text{COMPLETE}$  and event occurs on 12.10.2004 at 10:35:22.

Note that Definition 1 abstracts from other information that may be present in the log, e.g., the performer executing the task, and data linked to the event. An event always occurs in the context of a single case. The ordering of events corresponding to different cases is not important. Therefore, we consider a log to be a set of traces where each trace corresponds to a case.

**Definition 2 (Event trace, Event log).** Let  $E = T \times \{\text{START}, \text{COMPLETE}\} \times ts$  be a set of events over  $T$ .  $\sigma \in E^*$  is an event trace and  $W \subseteq E^*$  is an event log.

Note that the log shown in Table 1 is consistent with this notation. For example, the event trace for the first case is  $\sigma = (\text{TASK A}, \text{START}, 12.09.2004$

21:39:10) (TASK A, COMPLETE, 12.09.2004 21:39:17) (TASK J, START, 12.09.2004 21:39:25)(TASK B, START, 12.09.2004 21:39:25)(TASK C, START, 12.09.2004 21:39:26) (TASK C, COMPLETE, 12.09.2004 21:39:30) (TASK J, COMPLETE, 12.09.2004 21:39:34) (TASK B, COMPLETE, 12.09.2004 21:39:34)  
 ...

Event traces are sequences. We use the following standard notation for sequences.

**Definition 3.** Let  $E = T \times \{\text{START}, \text{COMPLETE}\} \times ts$ ,  $\sigma \in E^*$  a sequence containing  $n$  elements, and  $t \in T$  some task.

1.  $dom(\sigma) = \{1, 2, \dots, n\}$  is the domain of  $\sigma$ ,
2.  $\sigma_i$  is the  $i$ -th element,  $i \in dom(\sigma)$ ,
3.  $t \in \sigma$  iff there exists an  $i \in dom(\sigma)$  such that  $\sigma_i.task = t$ ,
4.  $first(\sigma) = \sigma_1.task$  is the first task to start, and
5.  $last(\sigma) = \sigma_n.task$  is the last task to complete.

### 3.2 Ordering Relations

An essential prerequisite for process mining is the ordering of tasks. To define suitable ordering relations on tasks, we need to consider pairs of events with their occurrence time, i.e., a START event with timestamp and a corresponding COMPLETE event with timestamp. Therefore, we define the notion of task occurrence.

**Definition 4 (Task occurrence).** Let  $\sigma \in E^*$  and  $\sigma = e_1 e_2 \dots e_n$ .  $t(e_i, e_j)$  is a task occurrence of  $t$  in  $\sigma$  iff

1.  $1 \leq i < j \leq n$ ,
2.  $e_i.task = e_j.task = t$ ,

3.  $e_i.type = \text{START}$ ,
4.  $e_j.type = \text{COMPLETE}$ , and
5.  $\forall_{i < k < j} (\sigma_k.task \neq t)$ .

Note that every event in the event trace corresponds to precisely one task occurrence. However, for one task there may be multiple task occurrences in the same event trace. Intuitively, a task occurrence can be represented as a line segment. The left end is the START event and the right end is the COMPLETE event. These line segments represent the period of time the task is being executed and can be used to define succession (i.e., “directly” follows), following (i.e., “indirectly” follows) and intersection (i.e., overlapping task occurrences).

**Definition 5 (Succession).** Let  $W \subseteq E^*$  an event log such that  $E = T \times \{\text{START}, \text{COMPLETE}\} \times ts$ . Let  $a, b \in T$  be two tasks.  $a$  is directly followed by  $b$  in  $W$ , notation  $a >_w b$ , iff there exists a  $\sigma \in E^*$  such that  $\sigma = e_1 e_2 \cdots e_n$  and two task occurrences  $a(e_i, e_j)$  and  $b(e_k, e_l)$  in  $\sigma$  such that  $j < k$  and there is no task occurrence  $c(e_p, e_q)$  in  $\sigma$  satisfying  $j < p < q < k$ .

$a$  is succeeded by  $b$  if and only if in at least one event trace  $a$  is “directly followed” by  $b$ , i.e., there is not any another complete task occurrence in-between the two tasks  $a(e_i, e_j)$  and  $b(e_k, e_l)$ . Figure 5 illustrates the possible succession situations between two tasks T1 and T2.

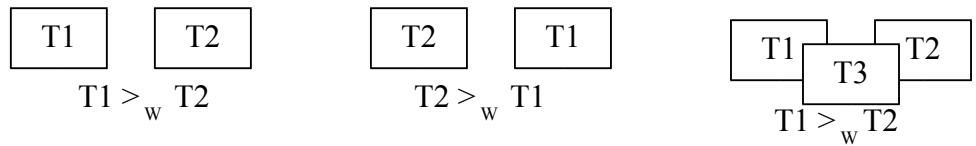


Figure 5 Succession relations

**Definition 6 (Following).** Let  $W \subseteq E^*$  an event log such that  $E = T \times \{\text{START}, \text{COMPLETE}\} \times ts$ . Let  $a, b \in T$  be two tasks.  $a$  is followed by  $b$  in  $W$ , notation  $a \gg_w b$ , iff there exists a  $\sigma \in E^*$  such that  $\sigma = e_1 e_2 \cdots e_n$  and two task occurrences  $a(e_i, e_j)$  and  $b(e_k, e_l)$  in  $\sigma$  such that  $j < k$ .

$a$  is followed by  $b$  if and only if in at least one event trace  $a$  is “indirectly followed” by  $b$ , i.e., there may be another complete task occurrence in-between the two task  $a(e_i, e_j)$  and  $b(e_k, e_l)$ . Figure 6 illustrates the possible following situations between two tasks T1 and T2.

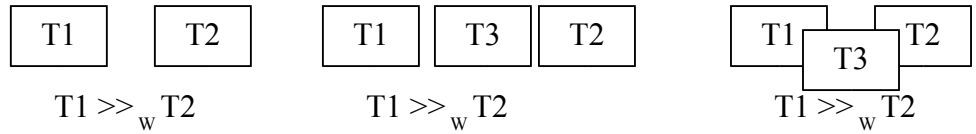


Figure 6 Following relations

**Definition 7 (Intersection).** Let  $W \subseteq E^*$  an event log such that  $E = T \times \{\text{START}, \text{COMPLETE}\} \times ts$ . Let  $a, b \in T$  be two tasks.  $a$  intersects with  $b$  in  $W$ , notation  $a \times_w b$ , iff there exists a  $\sigma \in E^*$  such that  $\sigma = e_1 e_2 \cdots e_n$  and two task occurrences  $a(e_i, e_j)$  and  $b(e_k, e_l)$  in  $\sigma$  such that  $i < k < j$  or  $k < i < l$ .

$a$  intersects with  $b$  if and only if in at least one event trace where an occurrence of  $a$  overlaps with an occurrence of  $b$ . Note that the intersection relation is symmetric, i.e.,  $a \times_w b$  if and only if  $b \times_w a$ . Figure 7 illustrates the possible intersection situations between two tasks T1 and T2.

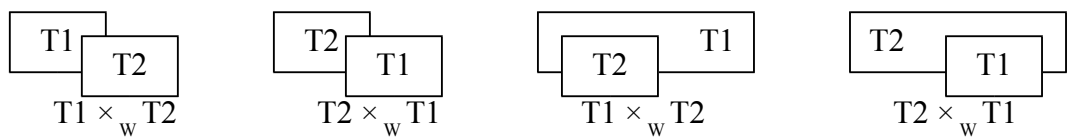


Figure 7 Intersection relations



Using the notation introduced in this section we can represent the finite set of tasks  $T_W = \{t \in T \mid \exists_{\sigma \in W} t \in \sigma\}$ , the finite set of initial task  $T_I = \{t \in T \mid \exists_{\sigma \in W} t = \text{first}(\sigma)\}$  (the first task to start), and the finite set of final task  $T_O = \{t \in T \mid \exists_{\sigma \in W} t = \text{last}(\sigma)\}$  (the last task to complete). It is also fairly straightforward to calculate the relations  $>_W$  and  $\times_W$ . The complexity of an efficient algorithm to calculate these relations and sets is  $O(n)$ , where  $n$  is the number of total events in the corresponding traces.

The notions of  $T_W$ ,  $T_I$ ,  $T_O$ ,  $>_W$ , and  $\times_W$  are the basic ingredients for the mining algorithm presented in this work. In order to prove the correctness of the mining algorithm we need to assume some notion of completeness, i.e., for a complex process with many possible event traces we need a log that somehow reflects the possible behavior.

### 3.3 Measurement Used in Identifying the Relations

Determining the ordering in workflow log requires some measurement. We define the measurements based on occurrence time of the events.

**Definition 8 (Average Succession Waiting Time).** Let  $W \subseteq E^*$  be an event log such that  $E = T \times \{\text{START}, \text{COMPLETE}\} \times ts$ . Let  $a(e_i, e_j), b(e_k, e_l) \in T$  be two tasks where  $a >_W b$ . Average succession waiting time, notation  $\Delta t_{a>wb}$ , is the average length of the time interval between the events  $e_j$  and  $e_k$ .

$$\Delta t_{a>wb} = \frac{\sum |e_k.ts - e_j.ts|}{\sum a >_W b}$$

The average succession waiting time is calculated as the division of the total time difference between the start event of task  $b$  and the complete event of task  $a$  to the total number of successions of  $a$  by  $b$ .

**Definition 9 (Average Following Waiting Time).** Let  $W \subseteq E^*$  be an event log such that  $E = T \times \{\text{START}, \text{COMPLETE}\} \times ts$ . Let  $a(e_i, e_j), b(e_k, e_l) \in T$  be two tasks where  $a \gg_w b$ . The average following waiting time, notation  $\Delta t_{a \gg_w b}$ , is the average length of the time interval between the events  $e_j$  and  $e_k$ .

$$\Delta t_{a \gg_w b} = \frac{\sum |e_k.ts - e_j.ts|}{\sum a \gg_w b}$$

The average following waiting time is calculated as the division of the total time difference between the start event of task  $b$  and the complete event of the task  $a$  to the total number of followings of  $a$  by  $b$ .

**Definition 10 (Edge Validity Ratio)** Let  $\Delta t_{a >_w b}$  be the average succession waiting time and  $\Delta t_{a \gg_w b}$  be average following waiting time. The edge validity ratio, notation  $a \sim b$ , between the activities  $a$  and  $b$  is the ratio of the average succession waiting time to the average following waiting time:

$$a \sim b = \frac{\Delta t_{a >_w b}}{\Delta t_{a \gg_w b}}$$

**Definition 11 (Average Execution Time).** Let  $W \subseteq E^*$  be an event log such that  $E = T \times \{\text{START}, \text{COMPLETE}\} \times ts$ . Let  $a(e_i, e_j) \in T$ . The average execution time, notation  $\Delta a_{\text{Exec}}$ , is the average length of the time interval between the events  $e_j$  and  $e_i$ .

$$\Delta a_{\text{Exec}} = \frac{\sum |e_j.ts - e_i.ts|}{\sum a}$$

The average execution waiting time is calculated as the division of the total time difference between the complete events and the start events of task  $a$  to the total number of occurrences of task  $a$ .

**Definition 12 (Average Intersection Time).** Let  $W \subseteq E^*$  be an event log such that  $E = T \times \{\text{START, COMPLETE}\} \times ts$ . Let  $a(e_i, e_j), b(e_k, e_l) \in T$  be two tasks where  $a \times_w b$ . The average intersection time, notation  $\Delta t_{a \times_w b}$ , is the average length of the time interval between events  $e_j$  and  $e_k$ .  $f_i(a,b)$  is the function that returns 1 or 0 according to its intersection type (see Figure 7 for the four possible intersection types). For instance,  $f_1(a,b)$  return 1 if  $a$  and  $b$  intersects as in the first case in Figure 7 else 0.  $f_2(a,b), f_3(a,b), f_4(a,b)$  behaves similar to  $f_1(a,b)$  as in second, third or fourth case respectively.

$$\Delta t_{a \times_w b} = \frac{\sum |e_j.ts - e_k.ts| * f_1(a,b) + |e_i.ts - e_l.ts| * f_2(a,b) + |e_j.ts - e_i.ts| * f_3(a,b) + |e_l.ts - e_k.ts| * f_4(a,b)}{\sum a \times_w b}$$

The average intersection time is calculated as the division of the total time of intersection time of tasks  $a$  and  $b$  to the total number of intersections of  $a$  and  $b$ .

**Definition 13 (Task Overlapping Ratio).** Let  $\Delta a_{\text{Exec}}$  be the average execution time and  $\Delta t_{a \times_w b}$  be the average intersection time of tasks  $a$  and  $b$ . The task overlapping ratio, notation  $a \wedge b$ , between the activities  $a$  and  $b$  is the ratio of the average intersection time to the minimum of average execution times of  $a$  and  $b$

$$a \wedge b = \frac{\Delta t_{a \times_w b}}{\min(\Delta a_{\text{Exec}}, \Delta b_{\text{Exec}})}$$

### 3.4 Identifying the Routings

After establishing the basic relations  $>_w$  and  $\times_w$  with the time constraints, we identify three derived relations. These derived ordering relations will be used to detect typical routings in the process model, such as sequential( $\rightarrow$ ), parallel ( $//$ ), alternative, iterative (i.e., loops) routing and their combinations.

**Definition 14 (Log-based ordering relations).** Let  $W$  be an event log over  $E$  where  $E = T \times \{\text{START, COMPLETE}\} \times ts$ . For any  $a, b \in T$ :

1. (Sequential)  $a \rightarrow_w b$                       iff  $a \sim b > \epsilon_-$  and  $a \wedge b < \epsilon_x$
2. (Parallel)      $a //_w b$                       iff  $a \wedge b > \epsilon_x$
3. (Disjoint)      $a \#_w b$                       iff  $a \wedge b < \epsilon_x$  and  $a \sim b < \epsilon_-$

To determine whether  $a$  and  $b$  are sequential, parallel or disjoint (neither sequential nor parallel), we use threshold values  $\epsilon_-$  and  $\epsilon_x$ . These values are defined heuristically.  $\epsilon_-$  is considered as a threshold for succession and  $\epsilon_x$  is considered as a threshold for intersection.

From Definition 14, the following property can be inferred directly.

**Property 1.** Let  $W$  be an event log over  $E$  where  $E = T \times \{\text{START, COMPLETE}\} \times ts$ . For any  $a, b \in T$ :  $a \rightarrow_w b$ ,  $a \#_w b$ , or  $a //_w b$ . Moreover, the relations  $\rightarrow_w$ ,  $\#_w$ , and  $//_w$  are mutually exclusive and partition  $T \times T$ .

**Definition 15 (Extra Log-based ordering relations).** Let  $W$  be an event log over  $E$  where  $E = T \times \{0, 1\} \times ts$ . For any  $a_1, \dots, a_n, c \in T$ : where  $n \geq 2$

1.  $\forall a_i \sim c > \oplus_- \text{ and } a_i \wedge c < \oplus_x, 1 \leq i \leq n \Rightarrow \forall a_i \rightarrow c$
2.  $\forall (a_i \rightarrow b \wedge c \rightarrow b) \Rightarrow a_i \# b$

To determine this kind of ordering, we use threshold values  $\oplus_-$  and  $\oplus_x$ . These values are defined heuristically.  $\oplus_-$  is considered as a threshold for succession for this extra ordering and  $\oplus_x$  is considered as a threshold for intersection for this extra ordering.

It is clear from the definition of join that at least two activities are needed. In the first inequality OR-Joins are detected in the following way: Let us examine the task  $c \in T$ , whether there is an OR-Join on  $c$ . To determine the OR-Join we need a set of tasks from  $T$  whose element count is greater than one since for a join at least two tasks required. This set is constructed by the tasks from  $T$  that satisfy the first condition of Definition 15. All of tasks in  $T$  are tested, tasks which satisfy the condition are added to the set. If the result set of testing after all tasks of  $T$ , has an element count more than one, we can conclude that there is an OR-Join on task  $c$  from tasks in result set. In the second inequality, incorrect edges deleted. Edges satisfying the second condition of Definition 15 are considered to be invalid edges. They are mined as if they are edges between tasks because of the nature of the OR-Join. It will be clarified in section 3.5.

### 3.5 The Algorithm

In this section, a new mining algorithm is presented. The algorithm consists of two major steps:

1. The construction of the average values tables;
2. Determination of the edges between tasks.

An average values table (AVT) is constructed for each task in the workflow. An AVT of a given task keeps the average succession waiting time, the average following waiting time, the edge validity ratio, the average intersection time and the task overlapping ratio of that task to every other task in workflow and The average execution time of the task is also kept in its AVT. In the construction of the average values table process, equations that are defined from *Definition 8* to *Definition 13* are used. This process is performed for all tasks that occur in the workflow log. While determining the edges between tasks we need two passes on the AVTs. The first pass is used to determine the edges that are sequential, parallel, alternative or iterative (i.e., loops) routing. The second pass is used for determining OR-Join.

In the algorithm noise and incompleteness of the workflow are considered too. In order to deal with these two facts, there are two constants,  $\epsilon_-$  and  $\epsilon_x$ , in the determination of the inequalities that are used against the noise and incompleteness of the workflow (see Def.14). During this study the  $\epsilon$  constants have been considered as follows:  $\epsilon_- = 0.45$  and  $\epsilon_x = 0.03$ . These values have been concluded heuristically. It is based on the observation done on the average values tables of known workflow diagrams.

In addition to these basic workflow constructs, multi-join that is one of the advanced constructs can be mined by using extra inequalities and constants, defined in *definition 15*, in the second pass. Similarly, the  $\oplus$  constants are considered as followings  $\oplus_- = 0.80$  and  $\oplus_x = 0.45$ . These values have been concluded heuristically too.

### 3.5.1 Constructing Average Values Tables

AVT keeps the average succession waiting time, the average following waiting time, the edge validity ratio, the average intersection time and the task overlapping ratio and the average execution time of the task. Hence for every task in the workflow log an AVT is constructed. We assume all workflow logs considered are sorted by workflow instance case.

While constructing the average values table, first a trace line is read from the workflow log. Then the case, activity, event type, and event time values are extracted from the trace line. If the extracted case value is different from current, we reset all the temporary variables. If the extracted event type is a start event, we keep this time in the variable of the corresponding activity event to use later. If there were any preceding complete events, total succession waiting time and total succession count could be calculated by using this start event time and the saved complete time of preceding activities. Similarly the average following waiting time could be calculated. If it is a complete event, we keep this time in the variable of the corresponding activity event to use later. In addition, the average execution time of the extracted activity can be calculated by using its start time; and also the average intersection time can be calculated by using the start time of the activities that have started and not completed yet. If the start time of the extracted activity is later than the start time of that activity, then start time of the extracted activity is used for start time of the interval. Otherwise, the start time of the earlier activity is used, and the complete time of interval is used as the extracted time. After the average succession waiting time, average following waiting time, average execution time and average intersection time are calculated, edge validity ratio and task overlapping ratio can be calculated.

According to the above explanation, constructing average values table requires full reading of workflow log just once, therefore it can be done in  $O(n)$  where  $n$  is the count of trace lines in the workflow.

Table 2 Average values table for some Task C

Task	Count	$\Delta a_{Exec}$	$\Delta t_{a>wb}$	$\Delta t_{a>>wb}$	$a \sim b$	$a \wedge b$
<b>A</b>	300	3.70				
<b>B</b>	199	4.61		0.50		0.57
<b>C</b>	122	5.01				
<b>D</b>	119	5.88				
<b>E</b>	181	3.76	4.58	4.63	0.9892	
<b>F</b>	119	3.45				
<b>G</b>	119	3.60				
<b>H</b>	300	5.32		14.50		
<b>I</b>	300	6.02	5.77	11.75	0.4908	0.12
<b>J</b>	300	8.82		0.33		0.98
<b>K</b>	300	3.58		27.02		

For example, the AVT in Table 2 is constructed for Task C. First column lists all tasks in the workflow. The total occurrences of the each task in the workflow are in the second column of the table. The average execution times of each task in the workflow are in the third column of the table. In the forth column the average succession time of Task C is kept. For instance, nothing is written in the forth column of first row nothing is written because Task A did not follow C in the workflow log. In other words no complete event of C occurred just before the start event of A occurred in log. However in fifth row of the forth column there is a value of 4.58. This means that the start event of E occurred, on the average, 4.58 seconds immediately after the occurrence of complete events of C in the log. The fifth column is similar to forth column except that instead of immediate



successions other followings are considered. In the sixth column the division of the fourth column to the fifth column is calculated. If there is no average succession time from Task C to the task in workflow log, no edge validity ratio exists from Task C to that task. In the last column the task overlapping ratio of C and the other tasks is calculated.

### 3.5.2 Determining the edges between tasks

In the determination of the edges between tasks we need two passes over the constructed AVTs. In the first pass sequential, parallel, alternative or iterative routing edges are determined. An initial workflow graph is generated at the end of first pass. In the second pass the graph generated in the first pass is refined for OR-Join. The beginning time of the first activity of the workflow is considered to be the beginning of the workflow. Therefore every other time value is calculated relative to it

#### **First Pass:**

In the first pass of the process of determining the edges between tasks, the algorithm works in the following way: Assume the task pair (TASK A, TASK B) is taken to hand. First, the edge validity ratio,  $a \sim b$ , of TASK A to TASK B and the task overlapping ratio,  $a \wedge b$ , of TASK A to TASK B are fetched from the stored 2-Dimensional arrays (i.e. AVTs). If the first condition in Definition 14 is satisfied, the edge value from TASK A to TASK B is set to 1 else it remains 0. This work is done for all task pairs. Complexity of the first pass is  $O((t_n)^2)$ , where  $t_n$  is the total number of tasks occurring in the workflow. Since, fetching from the 2-Dimensional array is done in  $O(1)$  and total number of all task pair is  $(t_n)^2$ , the overall complexity is  $(t_n)^2 * O(1) = O((t_n)^2)$ .

Formal description of the algorithm used in the first pass is shown in Figure 8. Here  $n$  is the number of the tasks that are occurred in the workflow log. In the algorithm tasks are enumerated hence  $t_i$  is the  $i^{th}$  task of the workflow.  $\epsilon_-$  and  $\epsilon_x$  are heuristic values. The *edge* array keeps the determined edges. For example, *edge*[ $i$ ][ $k$ ]=1 means there is flow from  $t_i$  to  $t_k$  in the workflow.

```

for i:0 to n
  for k:0 to n
    if  $t_i \sim t_k > \epsilon_-$  and  $t_i \wedge t_k < \epsilon_x$ 
      then edge[ $i$ ][ $k$ ]=1
    end
  end
end
end

```

Figure 8 Algorithm to generate the initial workflow graph

The output of the first pass is a 2-dimensional array representation of the workflow graph. Table 3 illustrates an example edge table. Row-column headings show the task names. An edge between two tasks is denoted by the value 1.

Table 3 Edge table after first pass

A→wb	A	B	C	D	E	F	G	H	I	J	K
A	0	1	1	1	0	0	0	0	0	1	0
B	0	0	0	0	1	0	0	0	0	0	0
C	0	0	0	0	1	0	0	0	0	0	0
D	0	0	0	0	0	1	1	0	0	0	0
E	0	0	0	0	0	0	0	1	0	0	0
F	0	0	0	0	0	0	0	0	0	0	1
G	0	0	0	0	0	0	0	0	0	0	1
H	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	1
J	0	0	0	0	0	0	0	0	1	0	0
K	0	0	0	0	0	0	0	0	0	0	0

The graph in Figure 9 is generated according to edge information taken from Table 3. Nodes in graph represent the tasks that occurred in the workflow logs. Edges in graph corresponds the control flow from one task to other tasks. The length of the nodes in the graph is drawn proportional to the average execution time of the tasks that they represent. The distances between nodes are proportional to the average waiting times between tasks. Thus the positions of the nodes on the horizontal axis are determined by using the average waiting times and average execution times of the tasks.

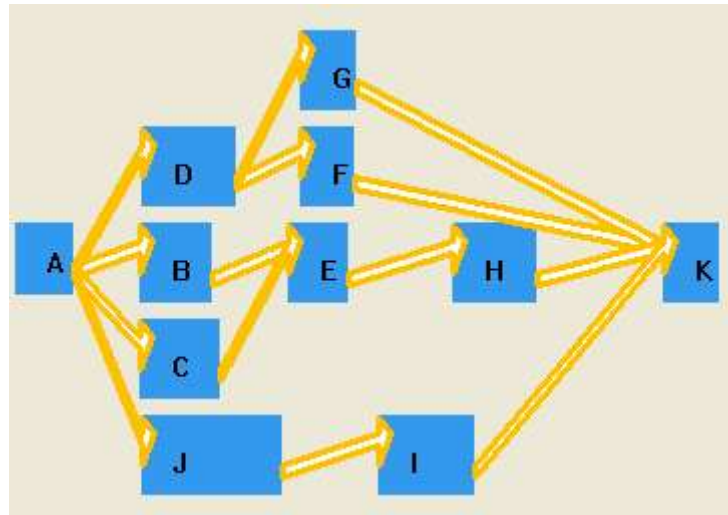


Figure 9 Induced workflow diagram after first pass

The graph in Figure 9 is the first attempt to find the workflow model. However it must be refined since there might be OR-Joins in the workflow model. In the second pass the refinement process is handled.

### **Second Pass:**

The second pass over the AVTs is performed for mining the OR-Joins. In this pass, the following steps are done: Assume the tasks are enumerated by alphabetic order and the iteration is done for TASK H. First, the edge validity ratio,

$a \sim b$ , of TASK A to TASK H and the task overlapping ratio,  $a \wedge b$ , of TASK A to TASK H are fetched from the stored AVTs. Then the first condition of the Definition 15 is tested with the substituted values fetched above. If the condition is satisfied, then TASK A is kept as a candidate task involved in the OR-Join into TASK H. The same is done for other tasks. If there are at least two tasks satisfying the condition, then the edge value between the task satisfying the condition and task H is set to 2 to denote that it is an OR-Join. Assume TASK F and TASK G are the tasks satisfying the condition. Then the edge values from TASK F to TASK H and from TASK G to TASK H are set to 2. After that, the edges that must be removed are searched. We consider only those edges that come out of the joining nodes and terminate at some node which can be reached directly from the joined node. In our example F and G are joining tasks and H is the joined task. Edges, which are from F to some node that can be reached from H directly by an edge are searched. If there are any such edges from F, they are deleted by setting their edge value to 0. This needs to be done for TASK G and TASK H too. By Comparing Fig. 9 to Fig. 11 we can see that edges from TASK F to TASK H and edges from TASK G to TASK H are added and edges from TASK F to TASK K and edges from TASK G to TASK K are removed.

Complexity of the second pass is  $O((t_n)^2)$ , where  $t_n$  is the total number of tasks occurring in the workflow. Since, fetching from the 2-Dimensional array is done  $O(1)$  and the total number of all task pairs is  $(t_n)^2$ , the overall complexity for finding the added edges is  $(t_n)^2 * O(1) = O((t_n)^2)$ . Since fetching the edge values from the 2-Dimensional array is  $O(1)$  and the total number of all tasks is  $(t_n)$ , the overall complexity for finding the removed edges is  $(t_n) * O(1) = O(t_n)$ . Thus the overall complexity is  $O(t_n) + O((t_n)^2) = O((t_n)^2)$ .

Formal description of the algorithm used in the second pass is shown in Figure 10. Here  $n$  is the number of tasks that are occurred in the workflow log. In the algorithm tasks are enumerated hence  $t_i$  is the  $i^{th}$  task of the workflow.  $\oplus_{\sim}$  and  $\oplus_x$  are the heuristic values. *candidate* is a list that contains the tasks satisfying the first if condition in the algorithm. *candidate* list is initialized to empty for each task by *clear* method and tasks inserted to *candidate* list by *add* method. The *edge* array keeps the determined edges For example,  $edge[i][k]=2$  means there is a flow from  $t_i$  to  $t_k$  in the workflow. In Figure 10, *removeWrongEdges* is used for deleting the edges from the task joining as OR-Join.

```

for i:0 to n
    candidate.clear()
    for k:0 to n
        if  $t_k \sim t_i > \oplus_{\sim}$  and  $t_k \wedge t_i < \oplus_x$ 
            then candidate.add(k)
        end
    end
    if candidate.size() > 1 then
        for j:0 to candidate.size()
            edge [candidate.get(j)] [i]=2
            removeWrongEdges(candidate.get(j),i)
        end
    end
end
end

```

Figure 10 Formal description of the second pass

Table 4 is refined form of the Table 3. The changed values are written in bold. As it can be seen from Table 4 edges from F to K and G to K are deleted and new edges from F to H and from G to H are added. The edge from F to H has a value of 2. This is to identify that this edge is an advanced control structure. It is not because two tasks are joining. For instance, if three tasks join as OR-Join, their edge values to the joined task will be still 2.

Table 4 Edge table after second pass

$a \rightarrow_w b$	A	B	C	D	E	F	G	H	I	J	K
A	0	1	1	1	0	0	0	0	0	1	0
B	0	0	0	0	1	0	0	0	0	0	0
C	0	0	0	0	1	0	0	0	0	0	0
D	0	0	0	0	0	1	1	0	0	0	0
E	0	0	0	0	0	0	0	1	0	0	0
F	0	0	0	0	0	0	0	2	0	0	0
G	0	0	0	0	0	0	0	2	0	0	0
H	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	1
J	0	0	0	0	0	0	0	0	1	0	0
K	0	0	0	0	0	0	0	0	0	0	0

In Figure 11 the final workflow graph is shown. This graph is generated from Table 4. If we compare the graphs in Figure 9 and Figure 11, we can see that there is a small difference between them. The difference comes from the refinement in the second pass. In the refinement process, edges from G to K and from F to K are deleted and new edges from F to H and from G to H are added. Furthermore, In Figure 11 the split from A is AND-Split and from D is AND-Split; the join on E is AND-Join, on H is OR-Join from F and G, on K is AND-Split. Or-Joins are represented in different color in the graph.

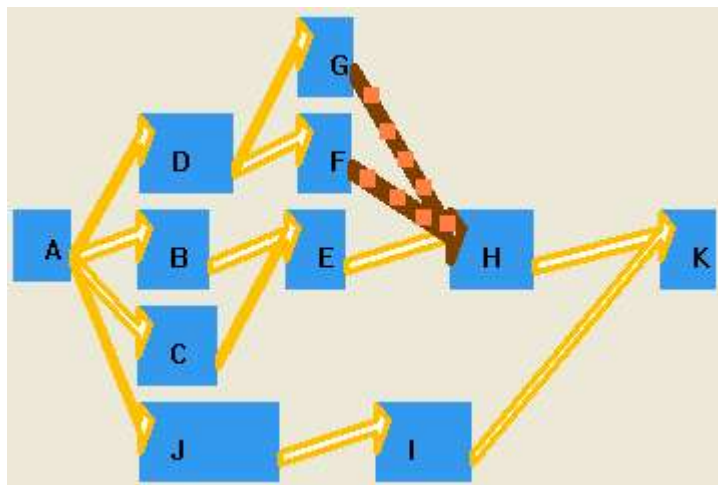


Figure 11 Induced workflow diagram of second pass

In both the first and second passes  $\oplus$  and  $\epsilon$  values are used to deal with noise and incompleteness of the workflow log. In the complete logs if a task follows the other task in the workflow then in the log following task must be seen just after the followed task at least once. Since the proposed algorithm uses interval, it does not require complete log in order to detect following relation. In addition, in some cases the disjoint events, for instance TASK D and TASK I in Figure 11, has a succession waiting time and it is possible that TASK D is succeeded by TASK I in some workflow traces. Such workflow traces are possibly exceptional cases of the workflow log and they can be eliminated by using the  $\oplus$  and  $\epsilon$  thresholds values. In this example edge validity ratio of D to I is 0.23 and the value of  $\epsilon$  for succession is 0.45. Since 0.23 is smaller than 0.45 the edge value from D to I is concluded as 0. This workflow does not contain loop if exists nothing is done to detect it. Loop is a cyclic routing. Routings are discovered one by one. Hence routing in the loops discovered regardless of its pattern

### **3.5.3 Timed Workflow Miner (TWM)**

Timed Workflow Miner (TWM) is the name of the tool that we have implemented. TWM is based on the proposed algorithm. TWM has three main functions:

- i. creating the workflow log, for rediscovery purposes
- ii. constructing average values tables for each task in the workflow log,
- iii. generating the workflow graph.

In TWM, in order to create a workflow log, the workflow descriptor file “WorkflowDescr.txt” is required. The workflow descriptor file contains the

information about each task such as its name, executing time, waiting time, incoming edges, outgoing edges information. A sample workflow descriptor file is placed in APPENDIX-A. A random workflow log is generated from this description. After reading the description of workflow is constructed as in the description. The control flows and executions are handled by random values. In order to construct the average values tables this created workflow log is used. After constructing the average values tables, workflow model is generated using the algorithm in section 3.5.2. Average values tables for Task-A is shown in Figure 12. The other constructed average values are tables placed in APPENDIX-B. After constructing the average values tables, workflow model is generated.

In Figure 12 a snapshot of TWM is shown. The table in the Figure is the AVT of A. In 4<sup>th</sup> column average succession time of other tasks to A, in 7<sup>th</sup> column average succession time of A to other tasks, is shown. 6.19 seconds is the average succession time of B to A and 4.76 seconds is the average succession time of A to B. In 5<sup>th</sup> column average following time of other tasks to A, in 8<sup>th</sup> column average following time of A to other tasks, is shown. 17.53 seconds is the average following time of C to A and 4.76 seconds is the average following time of A to B. In the 6<sup>th</sup> and 9<sup>th</sup> columns the edge validity ratios of the A to B and B to A are shown. In the 10<sup>th</sup> column the task overlapping ratio is shown. Messages in the list of Figure 12 give information about the distances between the tasks and tasks that are executed in parallel.

To identify the routings clearly TWM draws the edges with different colors. It uses dark color edges for loops, dotted color edges for OR-Joins and light color edges for the rest. According to these colorings the resulting workflow model generated by TWM is shown in Figure 13.



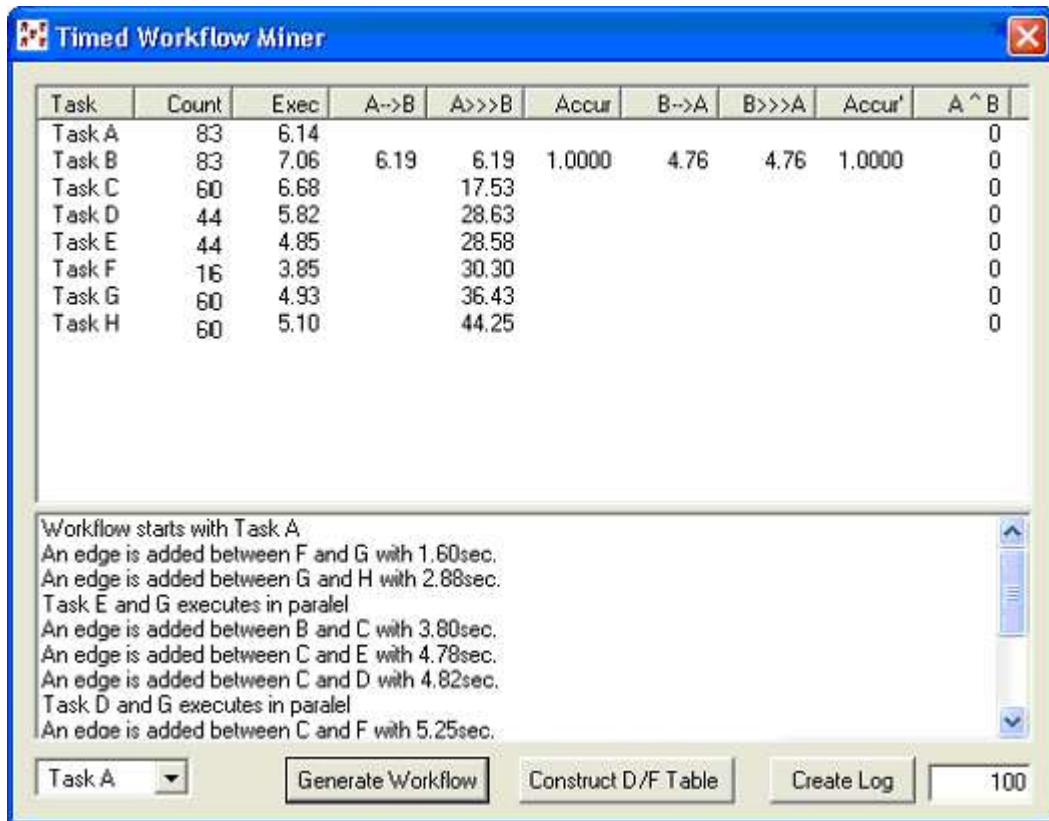


Figure 12 TWM showing Average Values Table of Task A

In Figure 13, a dark colored edge exists from B to A which implies there is a loop from B to A. Light colored edges from C to D, E and F imply an AND-Split from C. The dotted colored edges from D to G and E to G are imply on OR-Join into the task G. The numbers on the task nodes are the average execution times of the tasks. And the numbers on the edges are the average waiting times between tasks. All time values are referenced from the start task; therefore a reference time scale is drawn at the bottom of the generated graph. By using this time scale relative average starting time of the tasks in workflow can be seen.

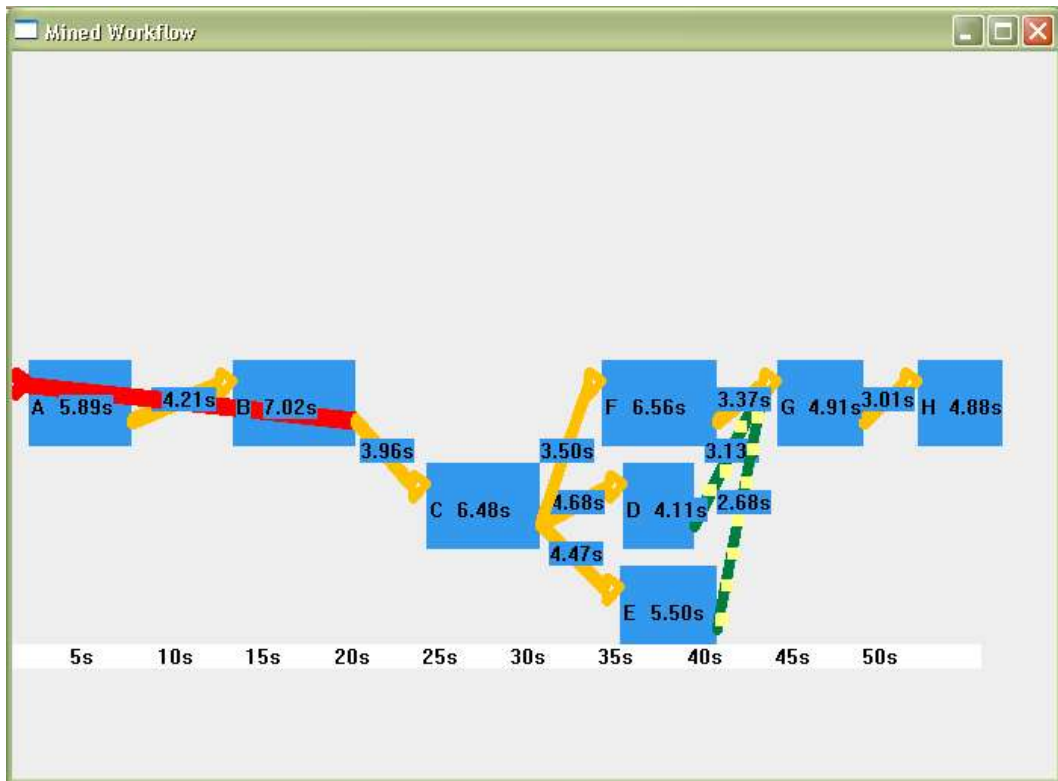


Figure 13 Mined workflow model

## CHAPTER 4

### COMPARISON

In this chapter a comparison of the proposed algorithm and  $\beta$ -algorithm is done. We chose the  $\beta$ -algorithm since it is the latest approach introduced in the literature. On the other hand the approach of mining block-structured workflow is not compatible with the proposed algorithm since the proposed algorithm has graph-oriented meta model and the other has a block-oriented meta model.

Proposed algorithm implemented in TWM and  $\beta$ -algorithm is in ProM framework. While comparing the algorithms, three logs are taken into consideration. Two of these logs obtained from process mining web page ([www.processmining.de](http://www.processmining.de)) and one of them is generated by TWM. Logs are sorted according to the case numbers. The logs obtained from process mining web page are in XML format, log generated by TWM is the in the format shown in Table 1.

First log that is used in comparison contains just three activities but it has a loop of order 1. It is known that mining loop of order 1 and of order 2, i.e. short loops, is a hard problem. Therefore this log is chosen to test whether the tools TWM and ProM can successfully mine this compact workflow model. The second log contains loops both of order 1 and of order 2. These two logs obtained from process mining web page and  $\beta$ -algorithm is introduced as an improvement of  $\alpha$ -

algorithm that is capable of mining short loops. Therefore ProM is expected to mine the desired workflow models. The workflow model generated by TWM needs to be equivalent to the workflow that is generated by ProM since it is expected to mine desired workflow model.

The third log used in comparison contains a parallel routing of order 3. It is generated by our system TWM. This is chosen to test whether  $\beta$ -algorithm can successfully mine the workflow model containing a parallel routing of order 3. Since TWM uses the intervals to detect successive tasks, it can successfully mine this kind of workflow models. The workflow model generated by ProM needs to be equivalent to workflow that is generated by TWM because TWM is expected to mine desired workflow.

TWM and ProM are compared for hidden task identification, complexity, and noise in the section 4.4. Performance analysis for TWM and ProM could not be done since no benchmark data exists for workflow mining.

#### 4.1 Case Study 1 : Log $W_A$

In this case Workflow Log that is obtained from process mining web page is considered. This log consists of 400 instances and 3043 trace lines. There are three activities in the log, namely A, B and C. The property of the workflow being mined is that it contains an activity that has a self-loop (loop of order 1). This log is chosen since  $\beta$ -algorithm is proposed to mine short loops of order 1 or 2. Table 5 summarizes the properties of log  $W_A$ .

Table 5 Log  $W_A$  summary

<b>Name</b>	Log A
<b>Number of cases</b>	400
<b>Number of entries</b>	3043
<b>Source</b>	<a href="http://www.processmining.de">www.processmining.de</a> file:pn_ex_05.xml
<b>Property</b>	Contains loop of order 1
<b>Activities</b>	A , B , C

The workflow models generated by TWM and ProM are in Figure 14 and Figure 15 respectively. In Figure 14, either B or C succeeds A and B has a loop into itself and it is succeeded by C. The same structure is found by ProM as shown in Figure 15. Table 6 and Table 7 presents the Vertex-Edge matrices of the Figure 14 and Figure 15 respectively. It can be seen that Table 6 and Table 7 are equivalent in cell-by-cell comparison. Therefore workflow models mined from  $\log W_A$  by TWM and ProM are equivalent.

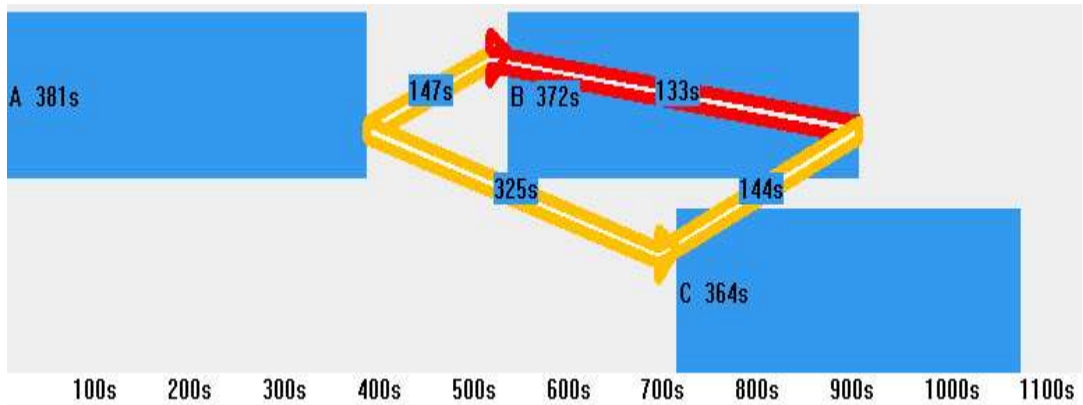


Figure 14 Workflow diagram of  $\log W_A$  generated by TWM

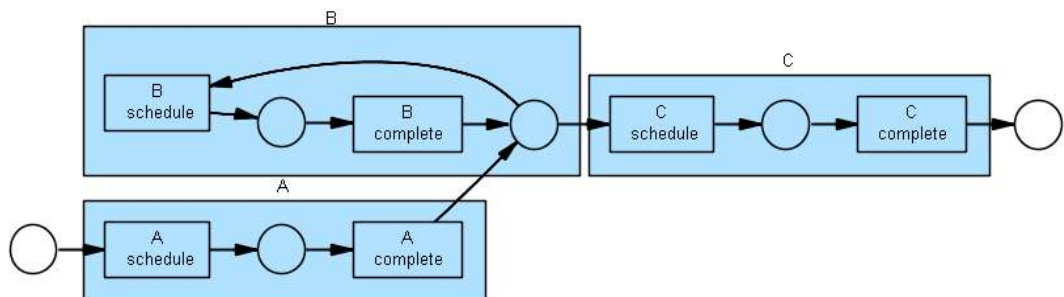


Figure 15 Workflow diagram of  $\log W_A$  generated by ProM

Table 6 Vertex-Edge matrix for Figure 14

	<b>A</b>	<b>B</b>	<b>C</b>
<b>A</b>	0	1	1
<b>B</b>	0	1	1
<b>C</b>	0	0	0

Table 7 Vertex-Edge matrix for Figure 15

	<b>A</b>	<b>B</b>	<b>C</b>
<b>A</b>	0	1	1
<b>B</b>	0	1	1
<b>C</b>	0	0	0

The representation of the workflow graphs in the compared system is different. ProM uses Petri-Nets to represent the control flow of the mined workflow. TWM uses workflow graph similar to [5] to represent the mined workflow model. As shown in Figure 14, TWM generates workflow graphs which include timing information as well. In Figure 14, the example workflow totally takes nearly 1100 seconds on average. The white strip at the bottom of the Figure is the time axis referenced from the start event. For instance A takes 381 seconds, B takes 372 seconds, C takes 364 seconds, B waits 147 seconds to start after A finishes, to start, C waits 325 seconds after A to start, B waits 133 seconds to loop and C waits 144 seconds after B to start. B and C look like they are concurrent activities. This is originated from the calculation of time values by using the averages of all occurrences of the activities, in whole log. C succeeds both A and B so the beginning time of C is sometimes closer to A, sometimes closer to B then on the average C begins in mid-time of the B's execution. None of this timing information about the workflow model can be mined in ProM.

## 4.2 Case Study 2: Log $W_B$

In this case we consider a workflow log that is obtained from process mining web page again. This log consists of 200 instances and 5529 trace lines. In the log there are eleven activities. The workflow contains a loop with two activities (loop of order 2) and a loop with a single activity (loop of order 1). This log is chosen since  $\beta$ -algorithm is proposed for mining short loops of order 1 or 2.

Table 8 Log  $W_B$  summary

<b>Name</b>	Log $W_B$
<b>Number of cases</b>	200
<b>Number of entries</b>	5529
<b>Source</b>	<a href="http://www.processmining.de">www.processmining.de</a> file:pn_ex_13.xml
<b>Property</b>	Contains loop of order 1 and 2
<b>Activities</b>	A, B, C, D, E, F, G, H, I, K, J

TWM generates the workflow models shown in Figure 16 and ProM generates the workflow model shown in Figure 17. The key points of the workflow of Log  $W_B$  are the following:

- (i) Log  $W_B$  contains Log  $W_A$  as a sub graph that contains a loop of order 1.
- (ii) Log  $W_B$  contains a loop of order 2.

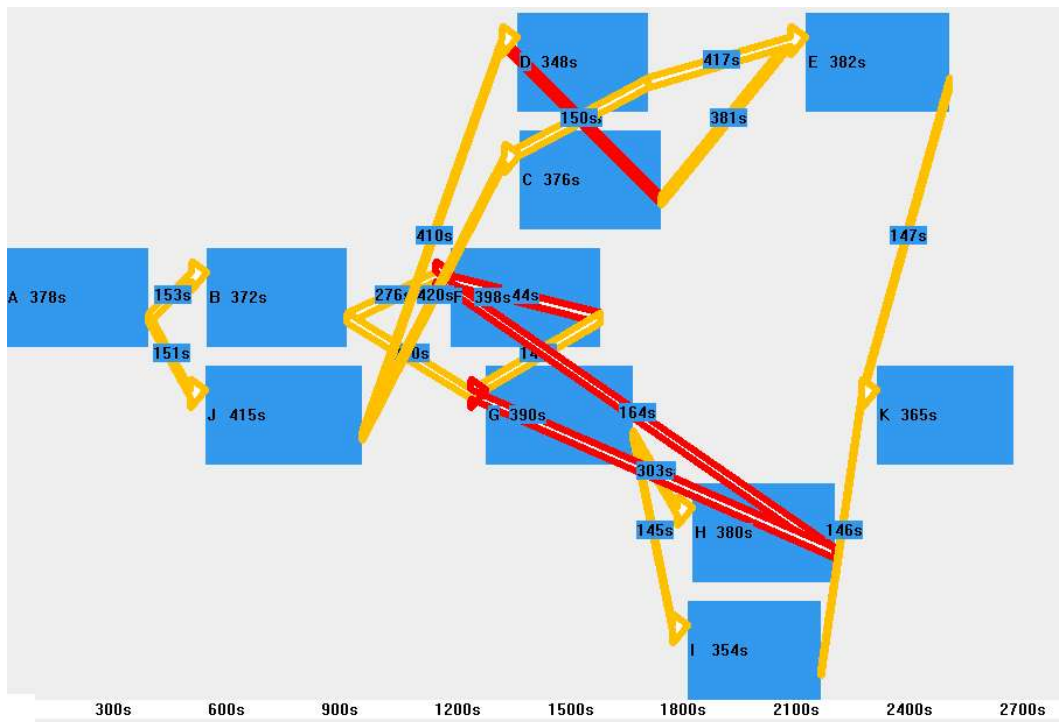


Figure 16 Workflow diagram of Log  $W_B$  generated by TWM

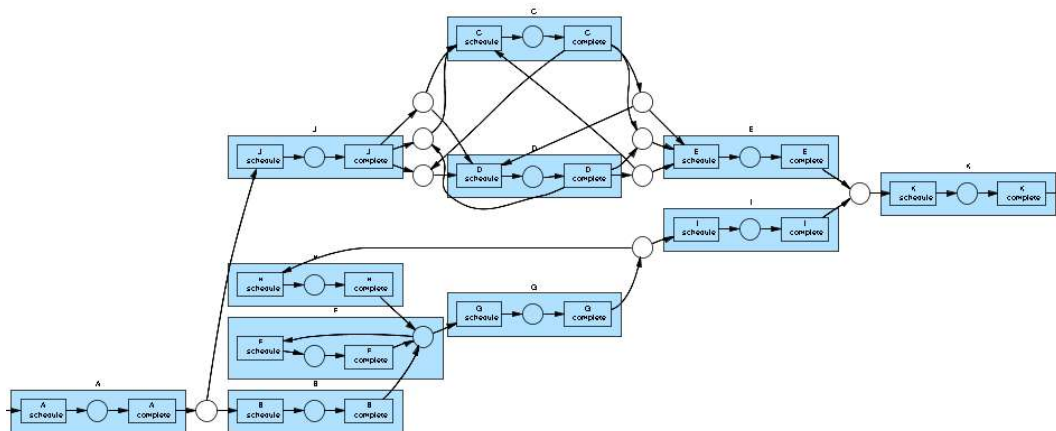


Figure 17 Workflow diagram of Log  $W_B$  generated by ProM

It can be clearly seen from Figure 16 and Figure 17 that F, G and H constitute the subgraph same as the workflow of Log  $W_A$  and the loop that includes C and D is of order 2. Table 9 and Table 10, are the Vertex-Edge matrices of Figure 16 and Figure 17 respectively. It can be seen that Table 9 and Table 10 are equivalent in cell-by-cell comparison. Therefore workflow models mined from Log B by TWM and ProM are equivalent.

Table 9 Vertex-Edge matrix for Figure 16

	A	B	C	D	E	F	G	H	I	J	K
A	0	1	0	0	0	0	0	0	0	1	0
B	0	0	0	0	0	1	1	0	0	0	0
C	0	0	0	1	1	0	0	0	0	0	0
D	0	0	1	0	1	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0	0	1
F	0	0	0	0	0	1	1	0	0	0	0
G	0	0	0	0	0	0	0	1	1	0	0
H	0	0	0	0	0	1	1	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	1
J	0	0	1	1	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0

Table 10 Vertex-Edge matrix for Fig 17

	A	B	C	D	E	F	G	H	I	J	K
A	0	1	0	0	0	0	0	0	0	1	0
B	0	0	0	0	0	1	1	0	0	0	0
C	0	0	0	1	1	0	0	0	0	0	0
D	0	0	1	0	1	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0	0	1
F	0	0	0	0	0	1	1	0	0	0	0
G	0	0	0	0	0	0	0	1	1	0	0
H	0	0	0	0	0	1	1	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	1
J	0	0	1	1	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0

If we examine the timing information of workflow of Log  $W_B$ , C and D seem to begin nearly, at the same time but they are not concurrent. In fact J precedes either C or D but not both. Then there is loop from D to C or from C to D otherwise control moves to E from C or D. Therefore, the average starting time points of C and D relative to initial activity are nearly the same time. This structure, shown in Figure 18, is really hard to mine. In order to be able to detect this structure, parallelism has to be detected correctly. Both TWM and ProM detect parallelism accurately by using the intervals between start and complete time of the activities.



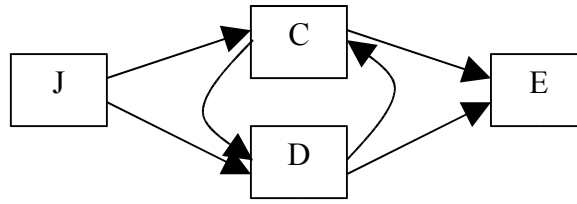


Figure 18 Loop of order 2

The workflow model shown in Figure 16 looks a little bit complex. It is because of the time-based graph of the workflow model. While drawing the graph, activities are placed according to their average beginning time and also as close as possible to the time axis in the middle. Therefore edges between activities intersect with each other.

### 4.3 Case Study 3: Log $W_C$

In this case a Workflow Log that is generated by TWM is considered. This log consists of 100 instances and 2824 trace lines. There are 12 activities in the log. The property of the workflow is that it contains parallel routings of order 3. This log is chosen since  $\beta$ -algorithm is insufficient to mine workflows containing parallel routings of order 3.

Table 11 Log  $W_C$  summary

<b>Name</b>	Log $W_C$
<b>Number of cases</b>	100
<b>Number of entries</b>	2824
<b>Source</b>	Generated by TWM
<b>Property</b>	Contains parallel routing of degree 3
<b>Activities</b>	TASK-A, TASK-B, TASK-C, TASK-D, TASK-E, TASK-F, TASK-G, TASK-H, TASK-I, TASK-K, TASK-J, TASK-L

The workflow to be discovered has two parallel branches, where the first branch includes activities A, D, F and G, H, J and the second branch includes activities A, B and C, E, I, J. (B and C) and D are in the first order, E and (G and F)

are second order, H and I are third order parallel tasks. This situation illustrated in Figure 19.

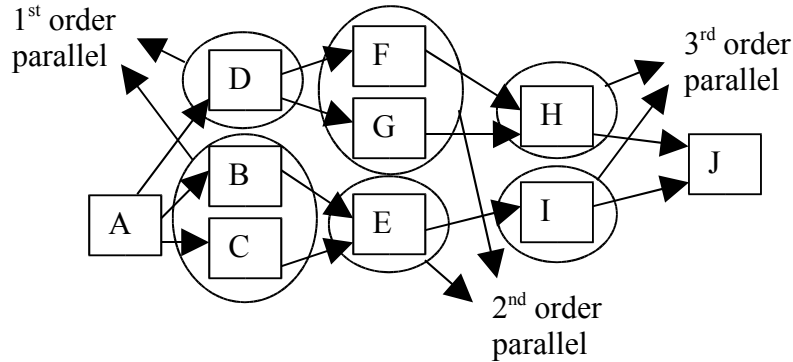


Figure 19 Workflow containing third order parallel tasks

Workflow models generated by TWM and ProM are presented in Figure 20 and Figure 21 respectively (see APPENDIX C for a larger version of Figure 21). With TWM, two branches are discovered in exactly the same as the workflow used in log generation. However, ProM fails to generate the actual workflow graph. In Figure 21 there are edges between (B and H) and between (C and H). This is because of the approach in  $\beta$ -algorithm. Table 12 and Table 13 are the Vertex-Edge matrices of the Figure 20 and Figure 21 respectively. We can see that Table 12 and Table 13 are equivalent in cell-by-cell comparison except (A and F), (B and H) and (C and H). We explain the reason for the incorrect edge between (B and H) and (C and H) in the following, but the reason behind the incorrect edge between (A and F) cannot be clearly identified during this thesis. We conclude that, workflow model mined from Log  $W_c$  by TWM is equivalent to the workflow used in log generation. However workflow mined by ProM contains extra incorrect routings.

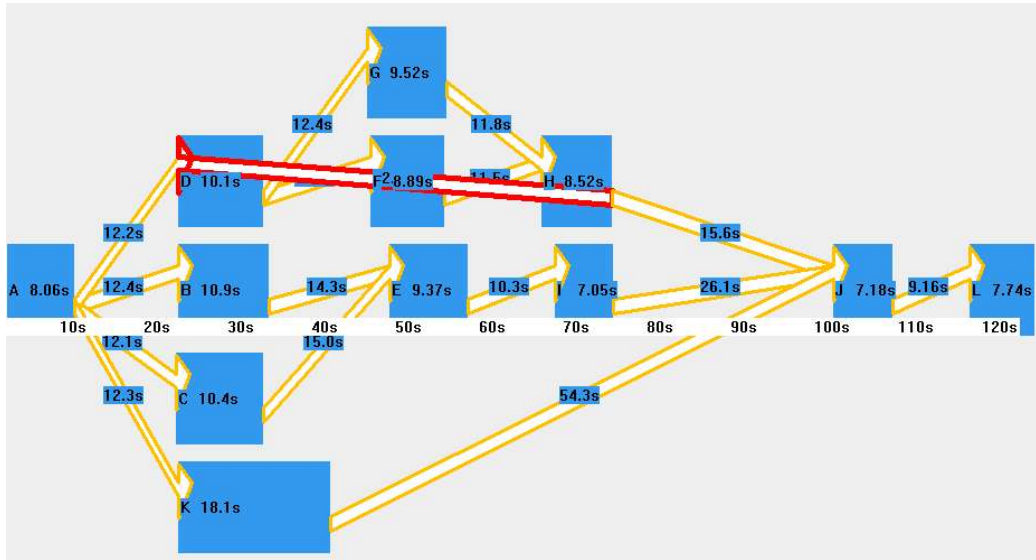


Figure 20 Workflow diagram of Log  $W_c$  generated by TWM

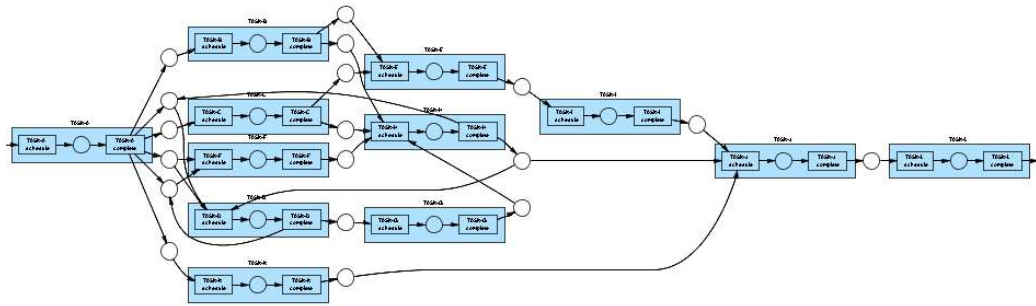


Figure 21 Workflow diagram of Log B obtained from ProM

Table 12 Vertex–Edge matrix for Figure 20

	A	B	C	D	E	F	G	H	I	J	K	L
A	0	1	1	1	0	0	0	0	0	0	1	0
B	0	0	0	0	1	0	0	0	0	0	0	0
C	0	0	0	0	1	0	0	0	0	0	0	0
D	0	0	0	0	0	1	1	0	0	0	0	0
E	0	0	0	0	0	0	0	0	1	0	0	0
F	0	0	0	0	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	0	1	0	0	0	0
H	0	0	0	1	0	0	0	0	0	1	0	0
I	0	0	0	0	0	0	0	0	0	1	0	0
J	0	0	0	0	0	0	0	0	0	0	0	1
K	0	0	0	0	0	0	0	0	0	1	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0

Table 13 Vertex–Edge matrix for Figure 21

	A	B	C	D	E	F	G	H	I	J	K	L
A	0	1	1	1	0	1	0	0	0	0	1	0
B	0	0	0	0	1	0	0	1	0	0	0	0
C	0	0	0	0	1	0	0	1	0	0	0	0
D	0	0	0	0	0	1	1	0	0	0	0	0
E	0	0	0	0	0	0	0	0	1	0	0	0
F	0	0	0	0	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	0	1	0	0	0	0
H	0	0	0	1	0	0	0	0	0	1	0	0
I	0	0	0	0	0	0	0	0	0	1	0	0
J	0	0	0	0	0	0	0	0	0	0	0	1
K	0	0	0	0	0	0	0	0	0	1	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0

The incorrect implication in ProM is done in the following way: Consider the simple workflow model Figure 22.a. Some possible event trace combinations are shown in Figure 22.b and Figure 22.c. In Figure 22.b and Figure 22.c the length of boxes represents the execution time of the task and tasks are placed according to their START event time relatively. According to Figure 22.b F indirectly follows B and Figure 22.c F succeeds B. Since B and F do not have an intersection and indirect following is ignored, workflow model in Figure 22.d is mined by  $\beta$ -algorithm. The combinations illustrated in Figure 22.b and Figure 22.c may not constitute a single case. They can be formed many times in the execution of the workflow depending on the execution times of tasks in each instance. The problem is serious because many workflows contain the workflow in Figure 22.a as a sub-workflow.

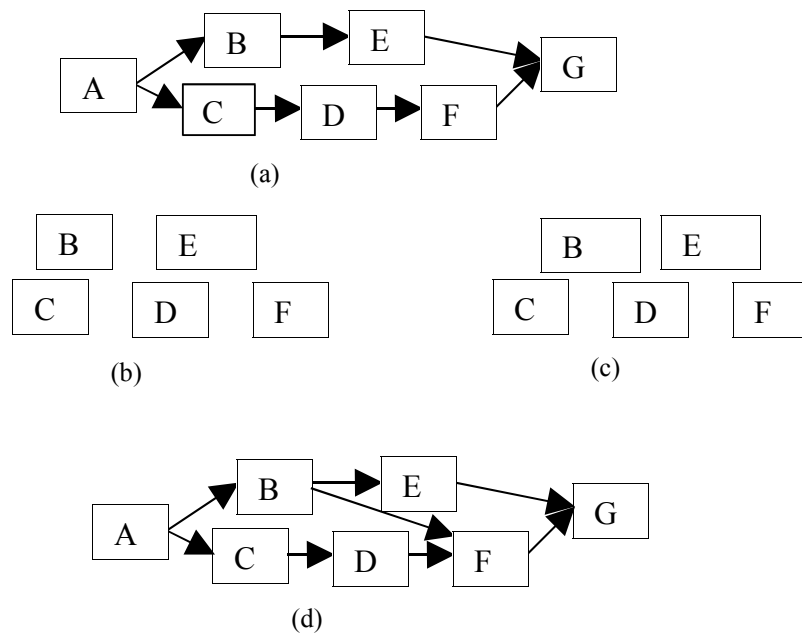


Figure 22 States for incorrect implication

In TWM, this incorrect implication is prevented by the edge validity ratio. In order to add an edge between activities, edge validity ratio must be greater than 0.45. The edge between (B and H) and (C and H) has not been added since the edge validity ratio of B and H is 0.3193 and C and H is 0.3324. In the Figure 23 these values are shown in circle.

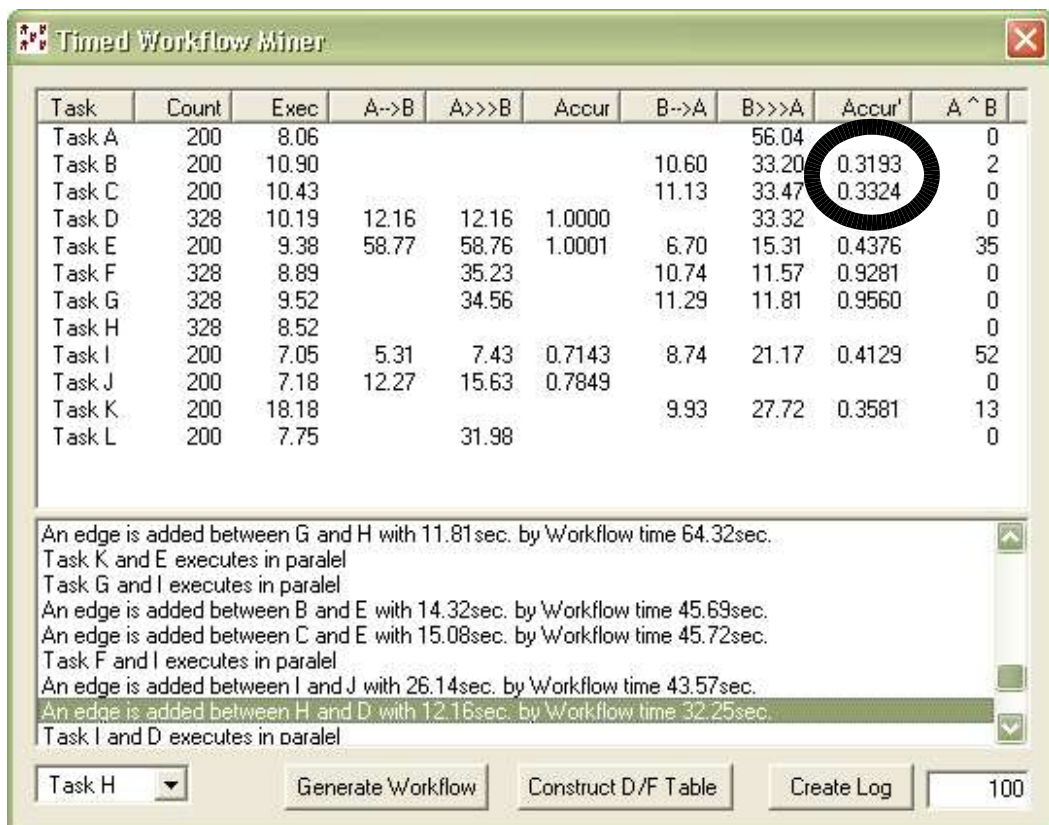


Figure 23 Average values table for activity H

#### 4.4 Other Issues

We have shown with examples that TWM is able to discover the same workflow structure as ProM can discover. We have also shown that ProM cannot discover all Workflow structures that TWM can (see Case Study 3). In this section we consider other issues in comparing the two systems.

#### 4.4.1 Noise

Workflow instances that are used in  $\beta$ -algorithm need to be consistent. That is every activity in a workflow instance with a START event is required to have a COMPLETE event in the same instance or vice versa. Otherwise that instance is discarded. In TWM all workflow instances are taken into consideration. All instances are included in the average value calculations, therefore exceptional cases cannot change the result of the edge detection. Thus TWM can mine workflow logs that contain incomplete data.

#### 4.4.2 Complexity

In [19] it is stated that the number of trace lines is the dominant factor on the execution time of the  $\beta$ -algorithm. In TWM the log is read once. Then two passes are performed over average values tables. The workflow log can be read in  $O(L)$  where  $L$  is the number of trace lines. The complexity of the first pass is of the edge detection algorithm is  $O((t_n)^2)$  where  $t_n$  is the total number of tasks occurring in the workflow log and the complexity of the second pass is  $O((t_n)^2)$  (see Sec 3.5). Hence the total complexity of TWM is  $O(L + (t_n)^2)$ . For large  $L$  values  $O(L + (t_n)^2)$  converges to  $O(L)$ . Consequently, complexity of the  $\beta$ -algorithm and TWM are asymptotically equivalent.

#### 4.4.3 Hidden Tasks

Another issue in mining workflows is the detection of hidden tasks. Hidden task is a task in workflow that exists in workflow and performed by an agent that cannot log the executed task. Therefore, these tasks are not included in the workflow logs. This is why this task is called *hidden task*. By the nature of the problem, it is difficult to be certain about the existence of hidden tasks in the mining process.

Our algorithm TWM might help in discovering hidden tasks in workflow mining. The timing information on the mined workflow graphs can be used to comment on the possibility of having hidden tasks. For instance consider the workflow model mined by TWM in Figure 21. In this workflow graph the average waiting time between the activity K and the activity J is two times longer than the longest waiting time in the whole graph. This might be a very useful information for the purpose of hidden task identification. Since, if a hidden task exists between two activities the waiting time between these two activities will include the waiting time from the first activity to the hidden task, hidden task duration and the waiting time from the hidden task to the second activity. Thus compared to the other waiting times, a relatively longer waiting time would be discovered for hidden tasks. Such information could not be retrieved using  $\beta$ -algorithm.

## CHAPTER 5

### CONCLUSION

In this thesis, a new approach for mining workflow graphs is presented. The mined workflow graph can contain all of the basic workflow patterns and also non-free choice structures. The proposed algorithm uses timed workflow logs and mine workflows with timing information such as the average waiting times and average execution times of the activities. While mining the workflow, first, average values tables for each task is constructed by scanning the log once. Then two passes are performed on AVTs: the first pass is for mining the basic constructs such as sequential, sequential, parallel, alternative or iterative routing. The second pass is for detecting the OR-Join. The noise problem is addressed by means of using threshold values that are decided heuristically. The proposed workflow mining algorithm is compared to other existing tools in the literature. We have concluded that it can show the same results, if not better, in the context of case studies. In the case studies TWM can mine the workflows that  $\beta$ -algorithm can mine, however  $\beta$ -algorithm fails to mine the workflow that TWM can mine (see case study 3). TWM and  $\beta$ -algorithm has an asymptotically equivalent complexity. Performance evaluation cannot be done because there is no benchmark data exists for workflow mining. Furthermore, proposed algorithm mines useful information for



commenting on possible hidden task identification problem. This mining approach can be used in different application such as mining web logs to detect navigation graphs since navigational routing expected to include parallel routing of higher order.

## REFERENCES

- [1] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros, Workflow Patterns.
- [2] W.M.P. van der Aalst, A.J.M.M. Weijters, L. Maruster, Workflow Mining: Which Processes can be Rediscovered? BETA Working Paper Series, WP 74, Eindhoven University of Technology, Eindhoven, 2002.
- [3] W.M.P. van der Aalst, B.F. van Dongen, Discovering Workflow Performance Models from Timed Logs, in: Y.Han, S. Tai, D. Wikarski (Eds.), International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002), Lecture Notes in Computer Science, vol. 2480, Springer-Verlag, Berlin, 2002, pp. 45–63.
- [4] W.M.P. van der Aalst, The application of petri nets to workflow management, Journal of Circuits, Systems and Computers 8 (1) (1998) 21–66.
- [5] R. Agrawal, D. Gunopulos, F. Leymann, Mining Process Models from Workflow Logs, in: Sixth International Conference on Extending Database Technology, 1998, pp. 469–483.
- [6] J.E. Cook, A.L. Wolf, Discovering models of software processes from event-based data, ACM Transactions on Software Engineering and Methodology 7 (3) (1998) 215–249.
- [7] J.E. Cook, A.L. Wolf, Event-based detection of concurrency, in: Proceedings of the Sixth International Symposium on the Foundations of Software Engineering (FSE-6), 1998, pp. 35–45.

- [8] J.E. Cook, A.L. Wolf, Software process validation: Quantitatively measuring the correspondence of a process to a model, *ACM Transactions on Software Engineering and Methodology* 8 (2) (1999) 147–176.
- [9] J. Desel, J. Esparza, Free choice petri nets, *Cambridge Tracts in Theoretical Computer Science*, vol. 40, Cambridge University Press, Cambridge, UK, 1995.
- [10] D. Georgakopoulos, M. Hornick, A. Sheth An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure
- [11] J. Herbst, D. Karagiannis, An inductive approach to the acquisition and adaptation of workflow models, in: M.Ibrahim, B. Drabble (Eds.), *Proceedings of the IJCAI 99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business*, Stockholm, Sweden, August 1999, pp. 52–57.
- [12] J. Herbst, Dealing with concurrency in workflow induction, in: U. Baake, R. Zobel, M. Al-Akaidi (Eds.), *European Concurrent Engineering Conference, SCS Europe*, 2000.
- [13] S. Junginger, H. Kühn, R. Strobl, D. Karagiannis, Ein Geschäftsprozessmanagement-Werkzeug der nächsten Generation—ADONIS: Konzeption und Anwendungen, *Wirtschaftsinformatik* 42 (3) (2000) 392–401.
- [14] B. Kiepuszewski, Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows, Ph.D. thesis, Queensland University of Technology, Brisbane, Australia, 2002. Available via <http://www.tm.tue.nl/it/research/patterns>.

- [15] M. Rusinkiewicz, A. Sheth. Specification and Execution of Transactional Workflows. In: W. Kim (ed.), *Modern Database Systems, The Object Model, Interoperability, and Beyond*, Addison-Wesley 1995, pp 592-620.
- [16] M. Sayal, F. Casati, M.C. Shan, U. Dayal, Business process cockpit, in: *Proceedings of 28th International Conference on Very Large Data Bases (VLDB\_02)*, Morgan Kaufmann, 2002, pp. 80–883.
- [17] G. Schimm, Process miner—a tool for mining process schemes from event-based data, in: S. Flesca, G. Ianni (Eds.), *Proceedings of the 8th European Conference on Artificial Intelligence (JELIA)*, Lecture Notes in Computer Science, vol. 2424, Springer-Verlag, Berlin, 2002, pp. 525–528.
- [18] A.J.M.M. Weijters, W.M.P. van der Aalst, Process mining: discovering workflow models from event-based data, in: B. Kroose, M. de Rijke, G. Schreiber, M. van Someren (Eds.), *Proceedings of the 13th Belgium–Netherlands Conference on Artificial Intelligence (BNAIC 2001)*, 2001, pp. 283–290.
- [19] L. Wen, J. Wang, W.M.P. van der Aalst, Z. Wang, and J. Sun. A Novel Approach for Process Mining Based on Event Types. BETA Working Paper Series, WP 118, Eindhoven University of Technology, Eindhoven, 2004.
- [20] Workflow patterns web page (<http://www.tm.tue.nl/it/research/patterns>)

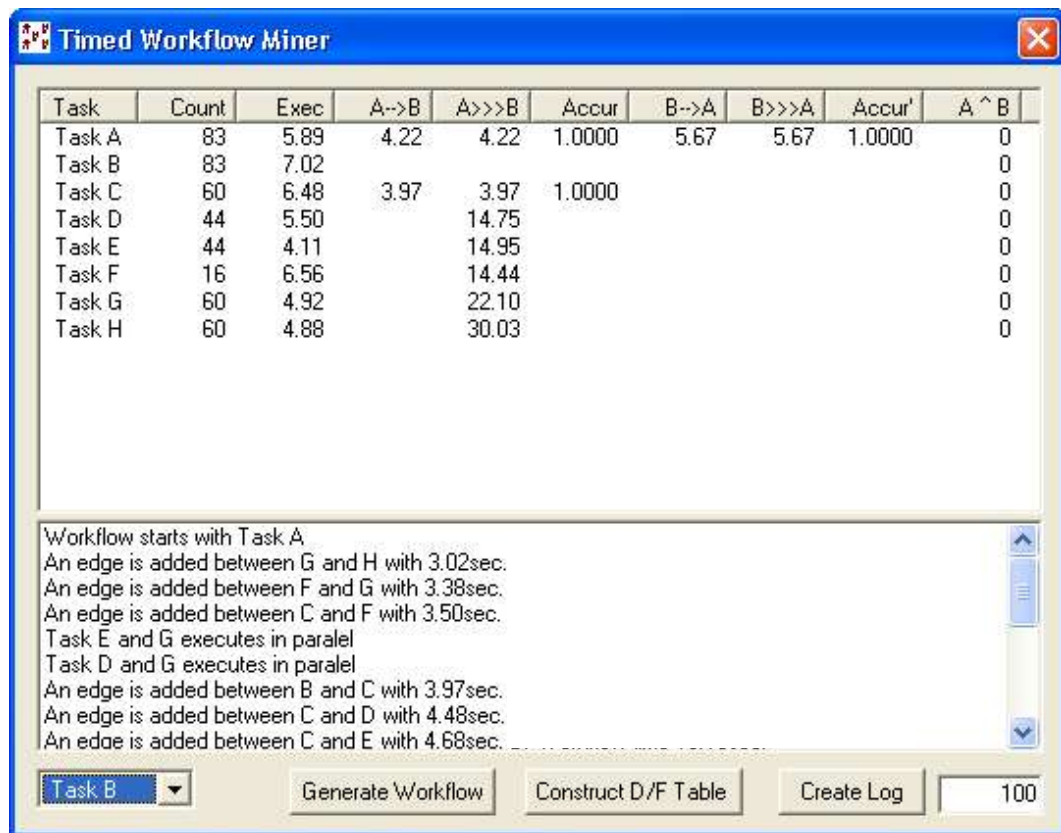
## APPENDIX-A

### A sample workflow descriptor file

TASK-A <task name>	12000
12000 <execution time>	1
12000 <waiting time>	1000
1 <preceding task count>	1
1000 <split waiting time>	TASK-G
1 <succeeding task count>	NOT
TASK-B <succeeding task name>	0
NOT <is succeeding task form loop>	99
0 <random(100) lower to run>	TASK-E
99 <random(100) upper to run>	8000
TASK-B	8000
14000	1
8000	1000
1	1
1000	TASK-G
2	NOT
TASK-A	0
LOOP	99
0	TASK-F
29	18000
TASK-C	8000
NOT	1
30	1000
99	1
TASK-C	TASK-G
12000	NOT
10000	0
1	99
1000	TASK-G
3	10000
TASK-E	6000
NOT	1
0	1000
59	1
TASK-F	TASK-H
NOT	10000
0	10000
99	1
TASK-D	1000
10000	0

## APPENDIX-B

### Used Average Values Tables



Task	Count	Exec	A->B	A>>>B	Accur	B->A	B>>>A	Accur'	A ^ B
Task A	83	5.89	4.22	4.22	1.0000	5.67	5.67	1.0000	0
Task B	83	7.02							0
Task C	60	6.48	3.97	3.97	1.0000				0
Task D	44	5.50		14.75					0
Task E	44	4.11		14.95					0
Task F	16	6.56		14.44					0
Task G	60	4.92		22.10					0
Task H	60	4.88		30.03					0

Workflow starts with Task A  
An edge is added between G and H with 3.02sec.  
An edge is added between F and G with 3.38sec.  
An edge is added between C and F with 3.50sec.  
Task E and G executes in paralel  
Task D and G executes in paralel  
An edge is added between B and C with 3.97sec.  
An edge is added between C and D with 4.48sec.  
An eddae is added between C and E with 4.68sec.

Task B

Figure 24 AVT for Task B

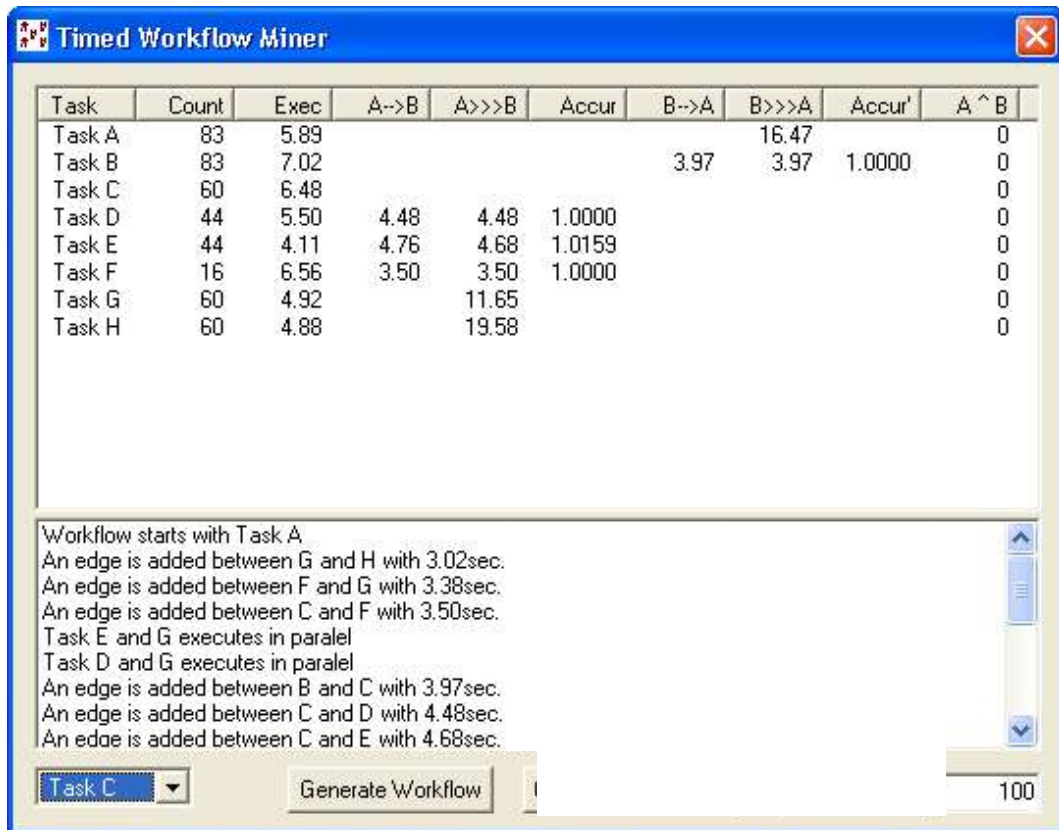


Figure 25 AVT for Task C

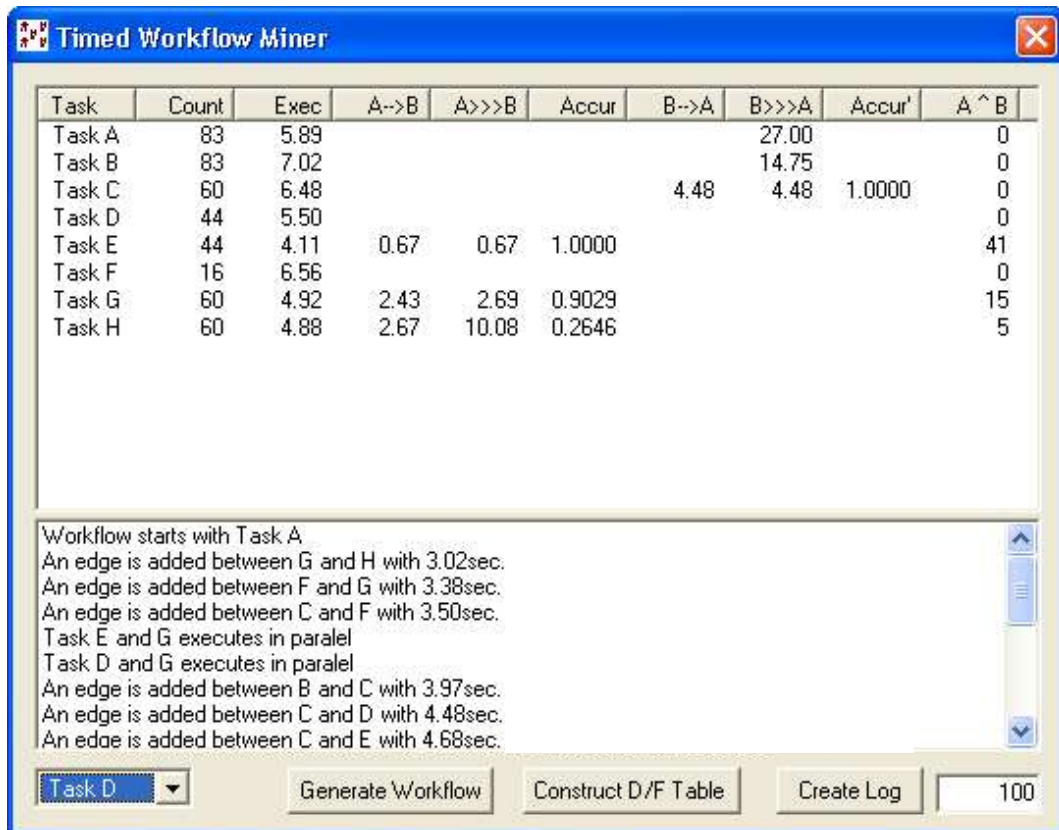


Figure 26 AVT for Task D

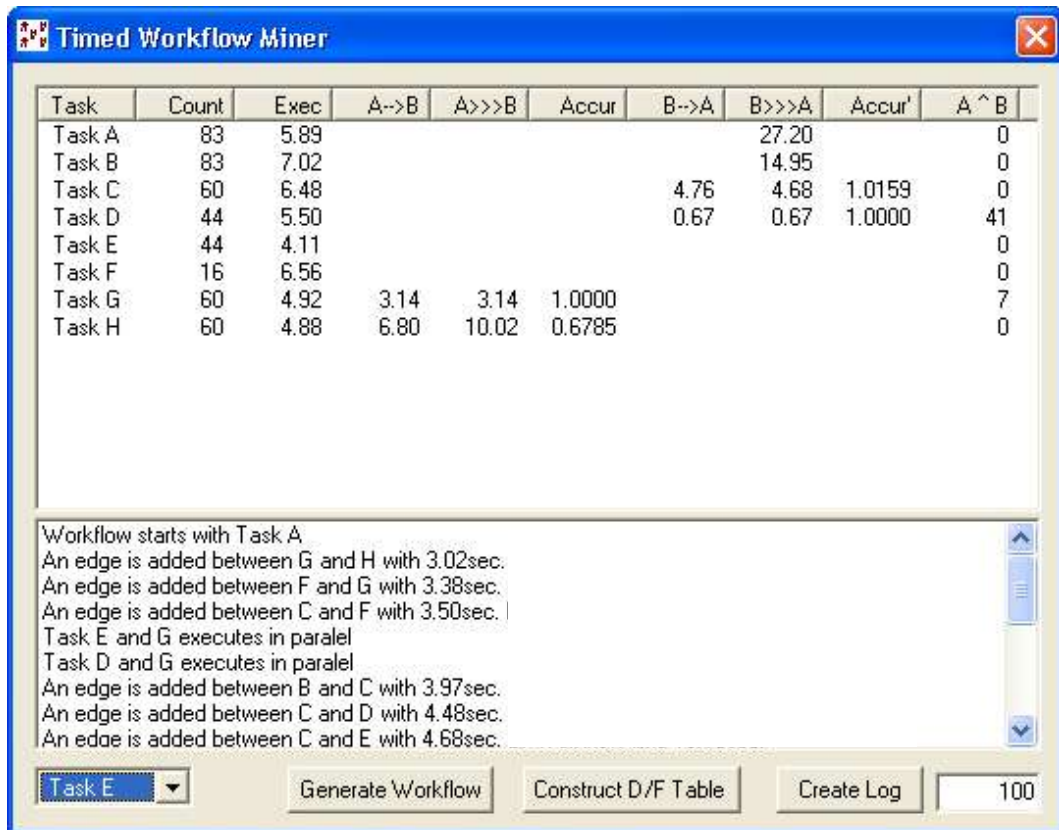


Figure 27 AVT for Task E

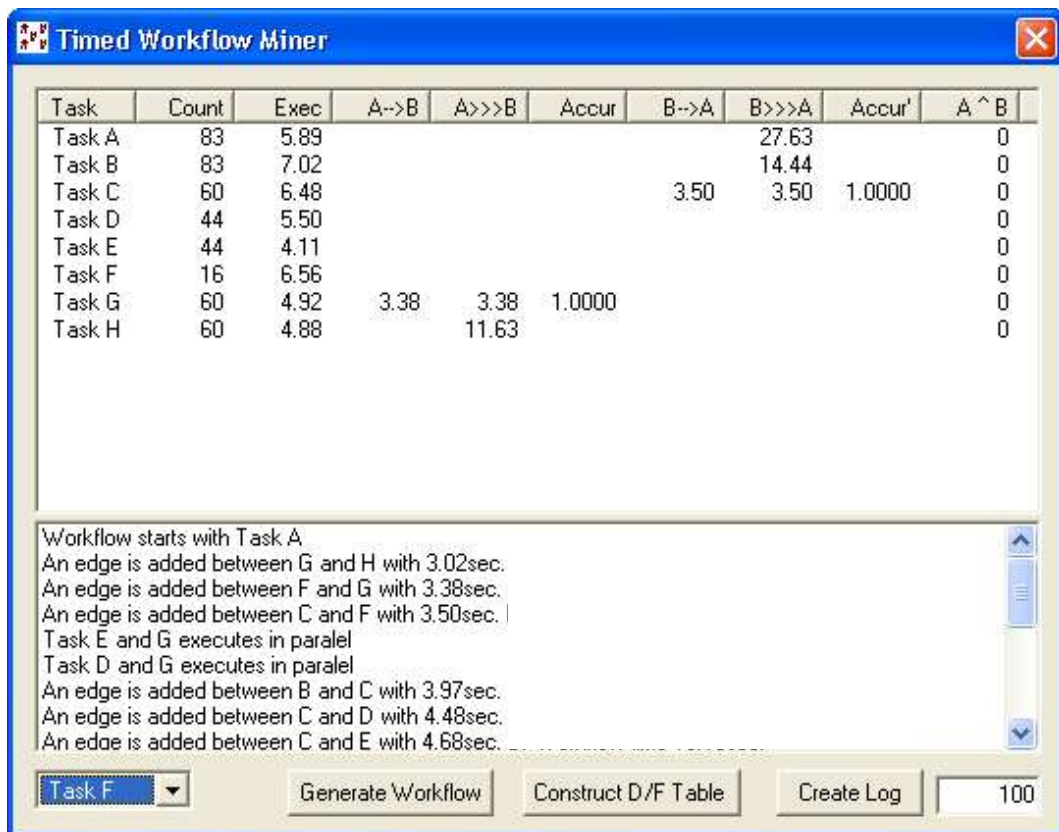


Figure 28 AVT for Task F



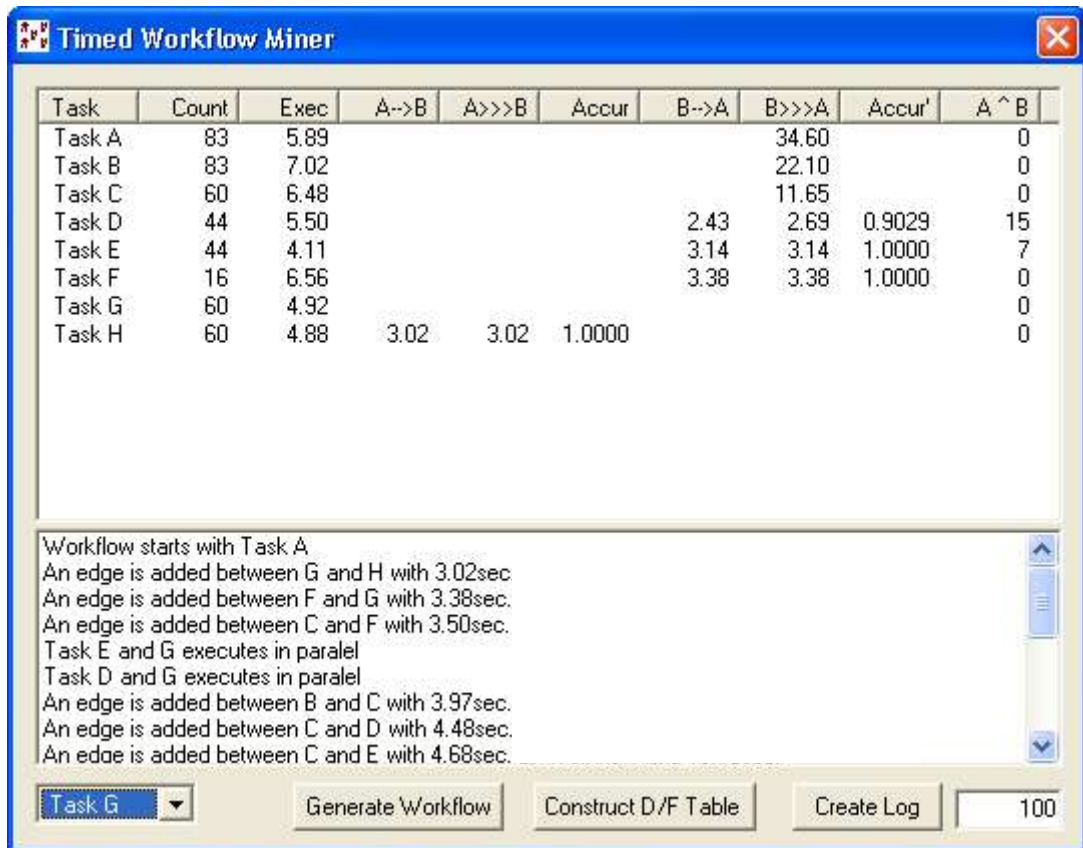


Figure 29 AVT for Task G

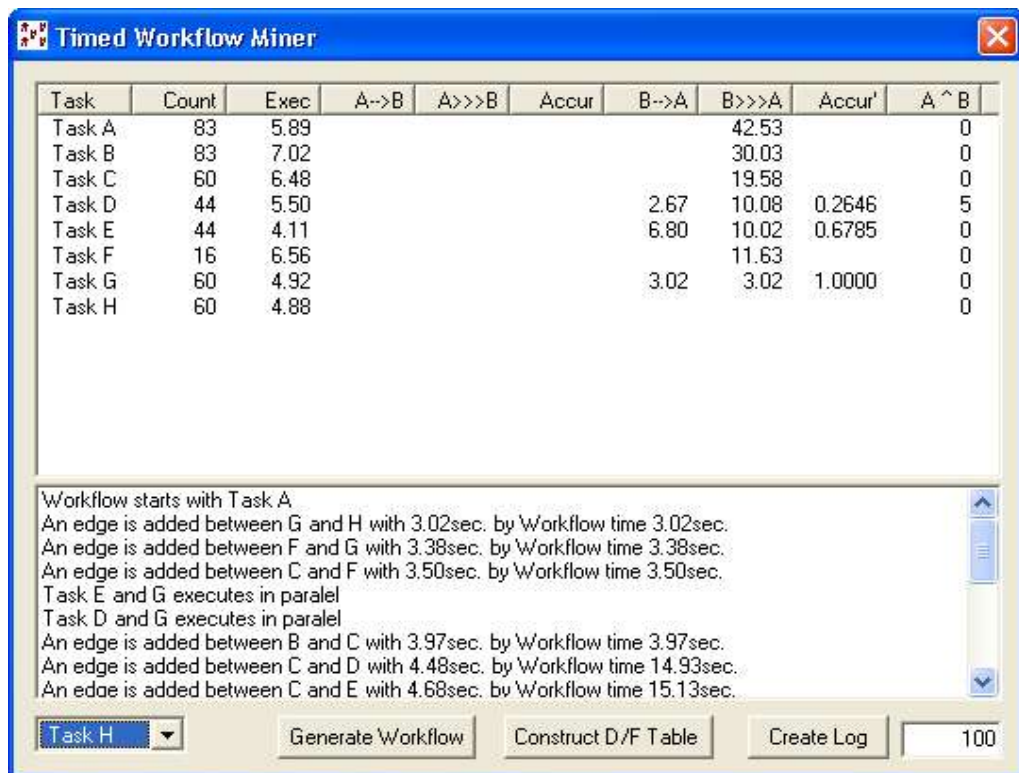


Figure 30 AVT for Task H

# APPENDIX-C

## Larger version of Figure 21

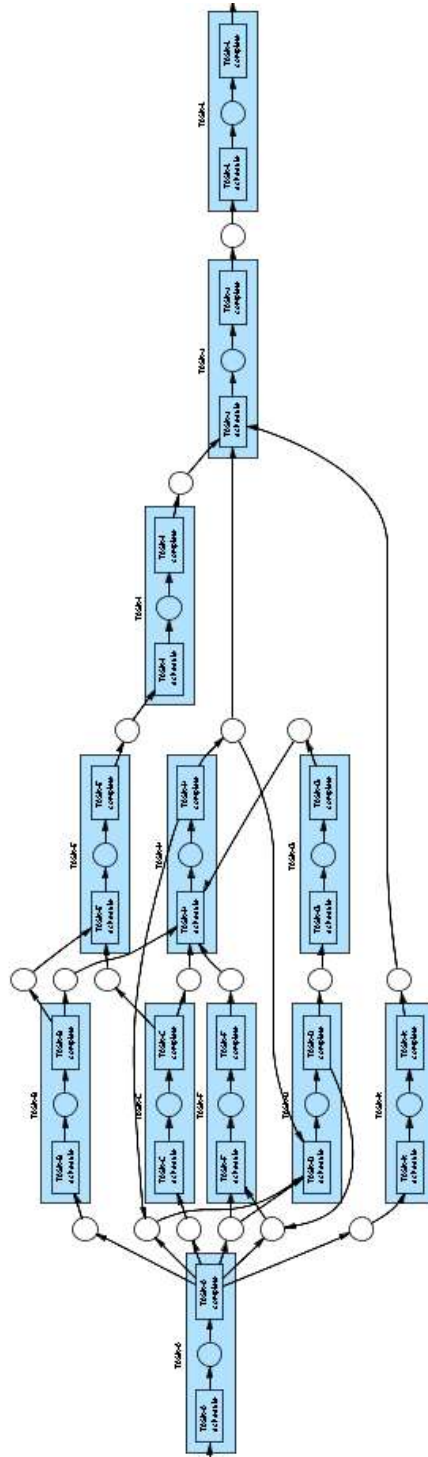


Figure 31 Workflow Model