Research Collection

Doctoral Thesis

Deadline analysis of workflow graphs and workflow performance optimization

Author(s):
Botezatu, Mirela

Publication Date:
2019

Permanent Link:
https://doi.org/10.3929/ethz-b-000356588 →

Rights / License:
In Copyright - Non-Commercial Use Permitted →

This page was generated automatically upon download from the ETH Zurich Research Collection. For more information please consult the Terms of use.

ETH Library

DISS. ETH NO. 26035

Deadline Analysis of Workflow Graphs and Workflow Performance Optimization

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

Mirela Madalina Botezatu

Bsc. Computer Science, Polytechnic University of Bucharest – Bucharest, Romania
Msc. Data Mining - Universite Pierre et Marie Curie - Paris, France

born on 4 October 1987
citizen of Romania

accepted on the recommendation of
Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Wil M.P. van der Aalst, co-examiner
Dr. Hagen Völzer, co-examiner

2019

To my parents.

Acknowledgments

I am extremely thankful to my supervisor Dr. Hagen Völzer at IBM Research - Zurich. His high standards, patience and deep expertise have guided me to the scientific explorations and results I am the most proud of. It has been a privilege and an honor to have Prof. Lothar Thiele as supervisor at ETH Zurich. Prof. Thiele, with his broad and deep scientific insights, has turned any meeting into an ocean of pointers, ideas and questions.

I am very grateful to Prof. Aalst who has patiently read my whole thesis and provided me with lots of insightful comments.

The group at IBM Research has been a source of fruitful collaborations and friendships. I would like to thank my managers, Dorothea and Paolo who have supported this endeavour through many inspiring conversations and funding. I would like to thank my co-authors Jasmina, Ioana who kindly extended their research expertise through my work. I am very grateful to my friends from the lab: Maria, Andreea, David with whom I share many joyful memories.

I would like to thank my parents for offering an empowering blend of candid and firm support in all my professional pursuits.

Last but surely not least, I would like to thank Michal, for always being there.

Abstract

A workflow process represents a collection of tasks, logically connected towards accomplishing a goal. Having a representation of the workflow, e.g., a workflow graph, this becomes the central element in identifying inefficiencies, bottlenecks towards accomplishing the goal.

There are multiple aspects of interest in the analysis of a workflow graph such as the structural correctness of the workflow graph, the data flow, the feasibility of the temporal constraints. These elements can be grouped into qualitative analysis of the workflow graphs and the quantitative analysis. While qualitative analysis provides valuable insights for the execution of a process, its output lacks information that is relevant in decision making. Namely, a process owner may be interested in the time or cost expected for running the process or parts of it to substantiate a decision in quantitative terms.

In this thesis we perform deadline analysis of workflow graphs and performance optimization of workflows. Workflow graphs can be translated into workflow nets which in turn are a class of Petri nets adapted for workflow analysis. The class of workflow graphs we analyze are those that can be modeled by sound, free-choice workflow nets. The main results in the deadline analysis of the workflow graphs presented in the thesis are:

- Polynomial time algorithm for computing the minimum duration of an execution of a sound workflow graph executed by a single resource.
- Hardness result for computing the probability of a deadline transgression in a sound workflow graph executed by a single resource.
- Polynomial time algorithm for computing the expected duration of a sound workflow graph executed by a single resource.
- Polynomial time algorithm for computing the minimum duration of an execution of a sound workflow graph executed by unbounded number of resources.
- Hardness result for computing the expected duration of an execution of a sound workflow graph executed by an unbounded number of resources.

Workflow optimization can bring direct benefits for achieving the objectives by improving predefined performance indicators, such as the cost or the execution time. We analyze two real world scenarios and we propose data-centric algorithms to streamline the execution

of their corresponding processes. The main results in the performance optimization of workflows presented in the thesis are:

- A novel technique to streamline the dispatching process in the IT Service Delivery industry. At the core of this technique is a novel clustering algorithm to group incident tickets into categories that both reflect the problem they document and are homogeneous in the duration it takes an agent to resolve them.
- An algorithm to automate a part of the life cycle process of a computer's hard disk. Concretely, we devise a novel algorithm able to automatically predict the disk replacement with high accuracy.

Zusammenfassung

Ein Workflow-Prozess stellt eine Sammlung von Arbeitsschritten dar, die logisch verbunden sind, um ein Ziel zu erreichen. Durch Darstellung des Workflows, zum Beispiel durch einen Workflow-Graphen, wird dies das zentrale Element zum Identifizieren von Ineffizienzen und Bottlenecks, um das Ziel zu erreichen.

Es gibt mehrere interessante Aspekte bei der Analyse eines Workflow-Graphen, wie die strukturelle Korrektheit eines Workflow-Graphen, der Datenfluss, die Durchführbarkeit der zeitlichen Einschränkungen. Diese Elemente können in qualitative Analysen des Workflow-Graphen und quantitative Analysen eingeteilt werden. Während qualitative Analysen wertvolle Einsichten in die Ausführung eines Prozesses geben, fehlt es ihrem Output an Informationen, die relevant sind zur Schlussfolgerung. Ein Prozess-Besitzer kann an Folgendem interessiert sein: Der erwarteten Zeit, den Kosten zur Durchführung eines Prozesses oder Teilen davon, um eine qualitative Entscheidung zu begründen.

In dieser Doktorarbeit führen wir eine Fristanalyse von Workflow-Graphen und Leistungsoptimierungen von Workflows durch. Workflow-Graphen können in Workflow-Netze konvertiert werden, die eine Klasse von Petri-Netzen sind, die für Workflow-Analysen adaptiert sind. Wir analysieren jene Klasse von Workflow-Graphen, die durch korrekte (im Englischen ”sound”) free-choice Workflow-Netze modelliert werden können.

Die Hauptresultate der Fristanalysen der Workflow-Graphen, die in dieser Doktorarbeit vorgestellt werden, sind:

- Polynomialzeit-Algorithmus zum Berechnen der Minimalzeit einer Ausführung eines korrekten Workflow-Graphen, ausgeführt von einer einzelnen Ressource.
- Schwere-Resultat zur Berechnung der Wahrscheinlichkeit einer Transgressionsfrist in einem korrekten Workflow-Graphen, ausgeführt von einer einzelnen Ressource
- Polynomialzeit-Algorithmus zur Berechnung der erwarteten Dauer eines korrekten Workflow-Graphen, ausgeführt von einer einzelnen Ressource.
- Polynomialzeit-Algorithmus zur Berechnung der minimalen Dauer der Ausführung eines korrekten Workflow-Graphen, ausgeführt von einer unbeschränkten Anzahl von Ressourcen.
- Schwere-Resultat zur Berechnung der erwarteten Dauern einer Ausführung eines korrekten Workflow-Graphen, ausgeführt von einer unbeschränkten Anzahl von Ressourcen.

Workflow-Optimierungen können direkte Vorteile zum Erreichen der Ziele durch Verbessern der vorbestimmten Leistungsindikatoren bringen, wie die Kosten oder Ausführungszeit. Wir analysieren zwei Echtzeitszenarios und schlagen datenzentrierte Algorithmen vor, um die Ausführung deren jeweiligen Prozesse zu streamlinen. Die Hauptresultate der Workflow-Optimierungen, die in dieser Doktorarbeit vorgestellt werden, sind:

- Eine neue Technik zum Streamlinen der Verteilungsprozesse in der IT-Servicebereitstellungsindustrie. Im Kern dieser Technik ist ein neuer Clustering-Algorithmus zum Gruppieren von Tickets von Ausfällen in Kategorien, die sowohl das Problem, das sie dokumentieren, reflektieren, wie auch homogen sind in der Dauer, die es für einen Agenten braucht, um sie zu lösen.
- Ein Algorithmus zum Automatisieren eines Teils des Lebenszyklus einer Festplatte eines Computers. Konkret, entwickeln wir einen neuen Algorithmus, der in der Lage ist, automatisch das Ersetzen von Festplatten mit hoher Genauigkeit vorherzusagen.

Contents

1 Introduction **1**
- 1.1 Formal representation of a process . . . 2
- 1.2 Deadline analysis . . . 5
  - 1.2.1 Incorporating time in workflows and analysis techniques . . . 7
- 1.3 Performance optimization . . . 9
  - 1.3.1 Positioning with respect to the Process Mining Framework . . . 9
  - 1.3.2 Optimizing the dispatching task in IT Service Delivery . . . 11
  - 1.3.3 Automating disk replacement decisions . . . 12
- 1.4 Dissertation Structure . . . 14

I Deadline Analysis of Workflow Graphs **15**

2 Foundations **17**
- 2.1 Petri nets . . . 17
- 2.2 Workflow Graphs . . . 20
  - 2.2.1 Workflow Graphs Semantics . . . 22
  - 2.2.2 An Algebraic Characterization of Workflow Graphs . . . 25

3 Workflow Graphs Executed by a Single Resource **29**
- 3.1 Workflow graphs with nondeterministic choice . . . 30
  - 3.1.1 The minimum duration of a workflow graph . . . 31
    - 3.1.1.1 Minimum cost downstream from an edge . . . 31
    - 3.1.1.2 Algorithm for computing the minimum duration of an execution . . . 34
    - 3.1.1.3 Correctness proof of Algorithm 3.1 . . . 36
  - 3.1.2 The maximum duration of a workflow graph . . . 40
  - 3.1.3 Regular and acyclic Workflow Graphs . . . 43
- 3.2 Workflow graphs with probabilistic choice . . . 44
  - 3.2.1 Probability of deadline transgression . . . 44
  - 3.2.2 Expected duration . . . 45
- 3.3 Conclusion . . . 47

4 **Workflow Graphs executed by Unbounded Resources** **49**
4.1 Introduction 49
4.2 Workflow graphs with nondeterministic choice 51
4.2.1 The minimum duration of a workflow graph 51
4.2.1.1 Correctness proof of Algorithm 4.1 55
4.2.2 Regular and acyclic workflow graphs 59
4.3 Workflow graphs with probabilistic choice 60
4.3.1 Expected duration 60
4.4 Minimum number of resources 62
4.4.1 An algorithm to compute the maximum degree of concurrency of a workflow graph 63
4.5 Workflow graphs executed by a fixed number of resources 66
4.6 Case Study 67
4.6.1 Mining the loan application process 67
4.6.2 Data and data processing 67
4.7 Conclusion 76

II Workflow Performance Optimization **77**

5 **Data-informed Work Assignment in Incident Ticket Resolution** **79**
5.1 Problem context 79
5.2 Incident ticket clustering 82
5.2.1 Preliminaries 82
5.2.2 Multi-view similarity matrix with induced sparsity 83
5.2.3 Semantic similarity metric between words 85
5.2.4 Clustering 86
5.3 Dispatching 90
5.4 Empirical evaluation 91
5.4.1 Data preparation 91
5.4.2 Experiments 92
5.5 Related work 98
5.6 Conclusion 99

6 **Predicting Disk Replacement from Sensor Data** **101**
6.1 Context 101
6.2 Predicting disk replacement 103
6.2.1 Selection of relevant SMART attributes 104
6.2.2 Compact time series representation 105
6.2.3 Class balancing via informative downsampling 106
6.2.4 Classification for disk replacements 106
6.2.5 Transfer learning 108
6.3 Evaluation 108
6.3.1 Data description and experimental setup 108
6.3.2 Feature selection – computation of relevant SMART attributes . . . 109

6.3.3 Compact time series representations 112
6.3.4 Classification for disk replacements 113
6.3.5 Transfer learning 114
6.3.6 Comparison with human designed replacement policies 115
6.3.7 Early vs. late replacement detection 116
6.3.8 SMART indicator rules 117
6.4 Deployment 118
6.5 Related work 119
6.6 Conclusions 120
6.6.1 Instantiation of the predictive pipeline to workflows 121

7 Concluding Remarks **123**

Curriculum Vitae **139**

List of Figures

1.1 The relationship between process, event logs and the different analyses they enable. 1
1.2 An example of a workflow graph and one of its executions (red) 3
1.3 Transition firing in a Petri net 4
1.4 A Petri net equivalent to the workflow graph in Fig. 1.2 4
1.5 A Petri net which is not free-choice 5
1.6 Classes of wokflow graphs (left) and the research questions we tackle (right). 6
1.7 Overview of the refined process mining framework. 10
1.8 A simple incident resolution process 11
1.9 Our data processing pipeline for predicting disk replacement. 12

2.1 An example of a state machine - (a), and a marked graph - (b) 19
2.2 Workflow graph where a node has multiple incoming edges and multiple otugoing edges (left) and an equivalent workflow graph (right) where we convert the node into a merge followed by a split. 21
2.3 An example of a workflow graph modeling a booking request 21
2.4 Regular patterns 21
2.5 Regular graph 21
2.6 Translation rules for the edges of the workflow graph. 22
2.7 A workflow graph, its edges and transitions 26
2.8 A live and bounded Petri net 27
2.9 An unbounded Petri net 27

3.1 Workflow graph with edge weights 32
3.2 Minimum duration execution and the accumulated costs 32
3.3 A cyclic workflow graph with edge weights 35
3.4 A workflow graph with unstructured loops 40
3.5 Example of a workflow graph with loop constraints. 41
3.6 Constructed workflow graph for proof of Thm. 3.2 42
3.7 A chain of XOR-blocks 45
3.8 Expected duration in a cyclic graph 46

4.1 A simple workflow graph with edge weights and a marking change. 51
4.2 Workflow graph with edge weights 53
4.3 Minimum duration execution and the accumulated costs 53
4.4 A cyclic workflow graph with edge weights 54
4.5 A probabilistic workflow graph 61
4.6 Tighter bound example 63
4.7 An example of a workflow graph for which we want to compute the maximum degree of concurrency 65
4.8 The decomposition for the workflow graph in Figure 4.7 into its fragments and its corresponding RPST (a), the workflow graph and RPST after computing the concurrency degree for the *Marked Graph Fragment* (b) the e workflow graph and RPST after computing the concurrency degree for *Sequence Fragment 1* (c) the workflow graph and RPST before the algorithm ends (d) 65
4.9 Regular workflow graph with *n* parallel threads 66
4.10 Histogram of durations of process runs. 69
4.11 Workflow graph for the loan application process (part 1). 71
4.12 Workflow graph for the loan application process (part 2). 72
4.13 Workflow graph for the loan application process (part 3). 73
4.14 Workflow graph for the loan application process (part 4). 74
4.15 Workflow graph for the loan application process (part 5). 75

5.1 Integration of the clustering and dispatching component. 80
5.2 Illustration of a possible ticket partitioning (left) where tickets are semantically related but not homogeneous in agent s performance and a different partitioning (right) where tickets are again semantically related possibly denoting different categories and where tickets are homogeneous in an agent's performance. 81
5.3 Example of similarity demotion between tickets T_1 and T_3 which have been executed by the same agent in 6 and 80 minutes respectively and between tickets T_2 and T_4 which have also been executed by the same agent in 5 and 70 minutes respectively. 83
5.4 Sparsity: similarity matrix (left), divergence matrix (right). 87
5.5 Agent's performance for a given cluster and complexity level. 90
5.6 Percentage of edges removed when increasing λ (for DU1). 92
5.7 Cluster abundances density plot. 93
5.8 Number of tickets to move, tickets successfully placed in other clusters and tickets dropped for each delivery unit (DU1, DU2, DU3). 94
5.9 Speed-up in service time for each delivery unit using the data-informed dispatching policy on the clustering obtained with fuzzy k-means – (DI) Fuzzy, with spectral clustering – (DI) spectral and using the non data informed policy – (non DI). 95
5.10 Differences in agent's performance in resolving tickets of the same complexity from different clusters. 96

5.11 Resolution time for networking tickets in minutes to exemplify an agent with skill level “A” being slower than an agent with skill level “B”. 96
5.12 Homogeneity of different clustering methods. 97

6.1 Availability – number of read or write operations: without proactive/preventive replacement (left) vs. with proactive replacement(right) 102
6.2 Differences between the forecasted and the observed values for SMART_187_raw. 110
6.3 Distribution of the temperature and of the power on/off cycles across the replaced disks for Hitachi and Seagate. 110
6.4 Distribution of the number of days before replacement when the change-point was observed. 112
6.5 Covariate shift for the two Seagate models 115
6.6 Percentage of disks correctly predicted as replaced on snapshots taken 1,3,10 and 30 days before the actual replacement event. 116
6.7 Integration of the predictive replacement component with storage arrays . . 119

List of Tables

3.1 Overview of results; new contributions in bold. 29
3.2 Initial values for $\delta[e]$. 36
3.3 Values for $\delta[e]$ after one iteration of the algorithm. 36
3.4 Values for $\delta[e]$ after two iterations of the algorithm. 36
3.5 Values for $\delta[e]$ after three iterations of the algorithm. 36

4.1 Overview of results; new contributions in bold, * we give a heuristic for this in Sect. 5 50
4.2 Initial values for $\delta[e]$. 54
4.3 Values for $\delta[e]$ after one iteration of the algorithm. 54
4.4 Values for $\delta[e]$ after two iterations of the algorithm. 55
4.5 Values for $\delta[e]$ after three iterations of the algorithm. 55
4.6 Examples of durations for the resume state of the tasks. 68
4.7 Examples of computed probabilities for the source event to be followed by the target event. 69
4.8 Minimum duration, expected duration and the running times. 69
4.9 Minimum duration, expected duration and the running times on the process with removed edges 70

5.1 Example semantic similarity between the words corresponding to two tickets $T_1 = w_1, w_2$ and $T_2 = w_3, w_4$. 84
5.2 Percentage of synonyms identified. 86
5.3 Examples of ticket descriptions. 91
5.4 Frequent words in the clusters obtained by spectral clustering using $\mathbf{S}'$ for DU1. 93
5.5 Values computed for τ_{min} for each delivery unit 93
5.6 Frequent words in the clusters obtained homogeneity optimized fuzzy k-means for DU1. 94

6.1 Healthy (H) vs. replaced (R) disks in the raw dataset and after data cleaning and aggregation for Hitachi and Seagate. 109
6.2 SMART correlation frequencies for SgtA and HitA. A ✓indicates the predictor is included in the classification task. 111

6.3 Precision, Recall, F-score, Deviation of different classifiers - median on 100 runs , each of which using randomly-drawn training and test data points 113
6.4 Precision, recall and F-score to illustrate the importance of transfer learning 116
6.5 Simple decision tree with (insufficient but commonly used) subset of SMART indicators 116
6.6 Examples of rules extracted from a decision tree model trained on the Seagate and Hitachi. 117
6.7 SMART attributes and their definitions. 122

Chapter 1

Introduction

Process modeling is the activity of representing business operations within an enterprise such that they can be better monitored or improved. Process modeling is wide spread across multiple industries such as banking, insurance and logistics [77, 147, 160]. In process modeling, the notion of a *process* is fundamental and serves as a starting point for visualizing and understanding business operations. A *workflow*, or an executable *process model* [25], captures all the activities and their execution order, the data that needs to be passed on and the way the resources are involved.

An organization can benefit in multiple dimensions from having its processes modeled such as: strategic – by facilitating long-range planning, mergers and acquisitions; organizational – stemming from better visibility in process execution; or operational by revealing opportunities for productivity improvement [106].

In the area of workflow management, detailed process descriptions are used to guide the execution of business activities [25]. Work is assigned to specific *resources* – humans or machines. Execution progress can be monitored and can enable escalations when completion takes longer than expected or deadlines are exceeded. Events such as the start and completion of work items together with data regarding the executing agents can be persisted in log files, which can be further examined for various types of analyses. Fig. 1.1 is inspired from [155] and it illustrates the relationship between processes, logs and the different types of run-time and design-time analyses they enable.

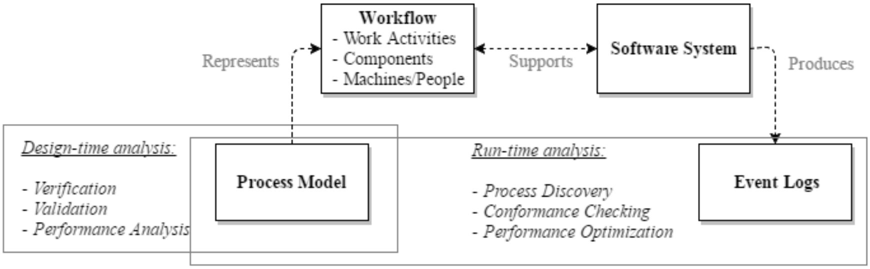


Figure 1.1: The relationship between process, event logs and the different analyses they enable.

From the design to the production phase, a process can be analyzed qualitatively or quantitatively. Qualitative analysis consists in the verification and validation of certain semantic properties determined by the expected functionality. It can answer questions such as: will a process element - given the structure of the process - ever be executed (*reachability*) or does the repeated execution of a process element prevent the execution of another process element (*fairness*). On the other hand, quantitative analysis evaluates the workflow against specific performance indicators, such as cycle time, waiting time or cost.

This thesis contributes to the area of quantitative analysis of workflows, and to the area of *process mining*. Process mining is a discipline which connects model-based process analysis and data-centric analysis techniques. It aims at improving process understanding and process efficiency. The thesis consists of two parts. In the first part, we study the workflow at design-time and we perform deadline analysis of the process models (e.g., we study whether all executions can finish within a given deadline). It is important to analyze a process before it is instantiated because an incorrect process definition may may lead to delays or low service levels. In this chapter, we present novel algorithms for deadline analysis of processes.

After presenting algorithms to be used at design-time, we move to the second part of the thesis where we perform process mining. We combine historic event data of processes with data collected at *run-time* (cf. Fig. 1.1) with the goal to achieve performance improvements.

Processes are very dynamic and complete information on process execution – such as actual task durations – is rarely known before the actual instantiation. However, today's information systems generate detailed information on the completed activities. Such rich event logs can provide a better understanding of the run-time behavior of the processes [155]. Their analysis enables us to pinpoint runtime inefficiencies and places which could benefit from improvements in process execution towards cost or performance gain. Therefore, in this part, we analyze two real world scenarios and we propose algorithms to streamline the execution of their corresponding processes.

In the following, we will introduce the formal model we use to represent and analyze processes, in Sect. 1.1. Subsequently, in Sect. 1.2 and Sect. 1.3, we will present the main problems we tackle and the solutions we have devised in the two parts of the thesis.

1.1 Formal representation of a process

The core subject of our analysis is the notion of a process. A process can be modeled in industrial process modeling languages such as BPMN [5], UML-Activity Diagrams [5], or Event Process Chains [5].

While these languages are useful for devising comprehensible descriptions of an organization's activities, most analysis techniques are based on a different formal model - namely, on Petri nets. A Petri net is a mathematical modeling language for the description and analysis of concurrent processes. We will discuss about Petri nets in more detail shortly. Results obtained through Petri net-based algorithms can not be applied directly for the processes designed in a certain industrial language. The reason is that applying a Petri net-based algorithm on such a process, requires the existence of a well-understood translation procedure from Petri nets to the language the process was designed in [32], which may not always be available.

To mediate these aspects, we use *workflow graphs* as a formal model to *represent* and *analyze* processes. Workflow graphs are control flow graphs extended by parallel fork and join. They have the advantage that they are both expressive and easy to analyze with Petri net based algorithms as they are equivalent to a certain class of Petri nets, as we will discuss next.

In terms of expressiveness, workflow graphs can capture the main control flow of processes modeled in languages such as BPMN, UML-Activity Diagrams, and Event Process Chains, cf. [70]. That is, the core routing constructs of these languages can be mapped to the routing constructs of workflow graphs, which are alternative choice and merge (XOR gateways), and concurrent fork and join (AND gateways). However, not all constructs of industrial languages can be translated to workflow graphs. Favre et al. proved in [33] that there exist simple BPMN process models with an IOR-join whose synchronization behavior cannot be modeled through any combination of AND and XOR gateways. Also, some EPC constructs contain non-local semantics which cannot be translated to Petri net patterns [32, 53].

In terms of analyzability, as mentioned above, workflow graphs are equivalent to a class of Petri nets for which several analysis techniques are known, namely free choice workflow nets [32].

Fig. 1.2 shows an example of a workflow graph modeling a ticket resolution workflow. After a task to categorize the ticket (“Label ticket”), there is a choice $s1$ whether the ticket documents a database issue (DB) or a disk issue (HDD). Following the case of HDD, there is a preliminary step to fetch the disk logs followed by a fork $f2$ that spawns two concurrent threads. One thread follows “Consistency check”, the other thread follows “Analyze HDD logs”. Then each thread is merged with the corresponding thread of the case DB through the merge gateways $m1$ and $m2$. After merging, there are some additional tasks “Identify error” and “Report usage pattern”, before the threads are synchronized at the join $j1$. Finally there are some wrap-up tasks, common to both cases.

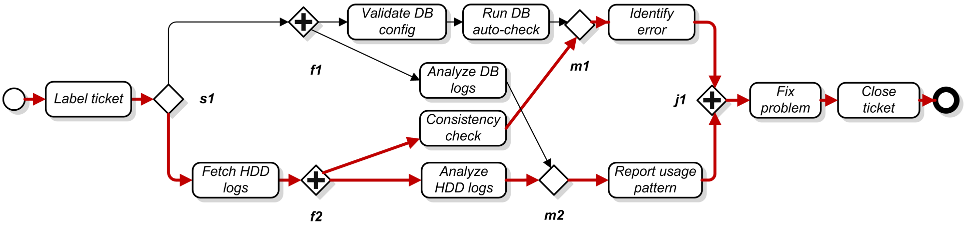


Figure 1.2: An example of a workflow graph and one of its executions (red)

For a better understanding of Petri nets, we provide a simple example in Fig. 1.3. As shown in Fig. 1.3 Petri nets are composed of *places* – for representing the state and *transitions* – corresponding to the actions. Places are drawn as circles, transitions as rectangles, and the flow relation is expressed using arcs connecting places and transitions. The state of the Petri net is represented by tokens which are represented as black dots. The activities of transitions can be divided into the removal of tokens from input places and the placement of tokens in the output places. A formal description of the semantics of Petri nets is given in Chapter 2. Due to the local state-based process description and the numerous analysis techniques [153, 78], the use of Petri nets as the formal model for workflow analysis is an

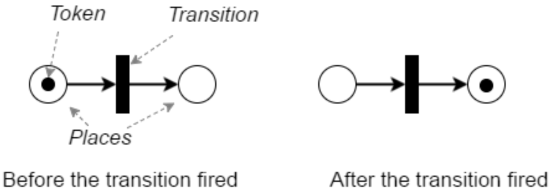


Figure 1.3: Transition firing in a Petri net

attractive option.

As mentioned above, the class of Petri nets workflow graphs are equivalent to is free-choice Petri nets. A workflow graph can be represented as a two terminal free-choice Petri net i.e., a connected net with a unique starting and ending place, satisfying the so called free-choiceness property, and which is also called a *free-choice workflow net* [32]. An example of a free-choice Petri net equivalent to the workflow graph in Fig. 1.2 is given in Fig. 1.4, obtained in conformity with the translation rules presented in Chapter 2. Due to this equivalence, the theory of free-choice Petri nets directly applies to workflow graphs. A workflow graph can be seen as a compact representation of the corresponding free-choice net.

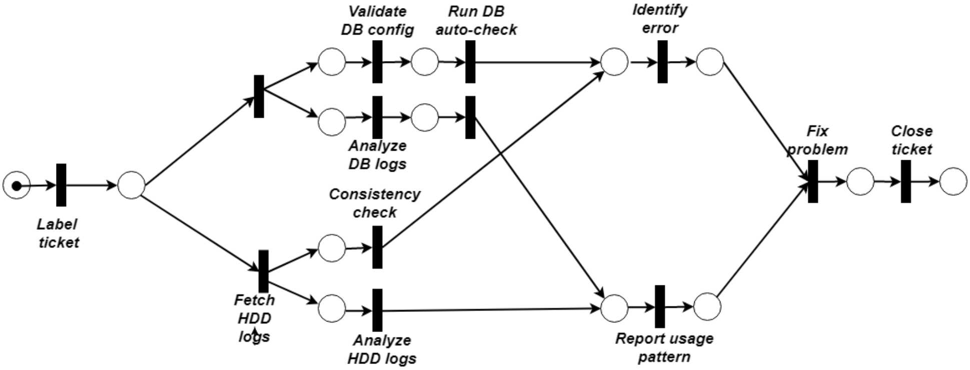


Figure 1.4: A Petri net equivalent to the workflow graph in Fig. 1.2

Next we will explain what free-choice denotes. In order to do this, we need to introduce some important structures that exist in Petri nets such as *conflict* and *synchronization*.

- Conflict arises when two transitions can fire, but the execution of one of them hinders the execution of the other.
- Synchronization exists when a transition can only be executed if two or more independent transitions have been executed before.

A Petri net is free-choice if the decision of which transition will be executed from a certain *state* can never be influenced by the *rest* of the system - and therefore, choices are free [78]. Free-choice rules out unexpected behavior due to dependence on the timing of events. We illustrate this through an example net – Fig. 1.5 which we borrow from [78].

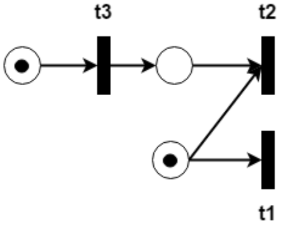


Figure 1.5: A Petri net which is not free-choice

In Fig. 1.5 we have a pattern which exists in general Petri nets but not in free-choice nets. Transitions t_1 and t_3 are concurrent and causally independent, but the order in which they fire determines whether a conflict between transitions t_1 and t_2 exists. Transitions t_1 and t_2 are not in conflict as t_2 can not fire because it is missing inputs. However t_2 will be in conflict with t_1 if t_3 fires before t_1. Therefore, due to the synchronization at t_2, transition t_3 influences which one of the transitions t_1 and t_2 can fire.

In free-choice Petri nets, conflicts and synchronization can not interfere. More precisely, if there is an arc from a place ps to a transition t, then either t is the only output transition of p (which implies that t cannot be in conflict with any other transition) or p is the only input place of t (which implies that there is no synchronization at t).

The expressiveness of free-choice workflow nets is strictly between the control-flow of sequential programs (state machines, flow charts) and general concurrent processes (Petri nets). While free-choice workflow nets permit concurrency as well as choice, the overlap of concurrency and choice is restricted such that no race conditions can arise. While these restrictions limit their usage, many processes in practice can be modeled in this formalism. For example, the set of 735 of industrial process models used in [85] could be mapped completely to workflow graphs. Some of these cases are complex and often represent time critical processes.

An important property of the workflow graphs we analyze is *soundness*. Soundness is a notion of correctness for workflow nets. Soundness requires that a workflow can always terminate, that upon termination no other tokens remain elsewhere in the net, and that it does not contain any dead tasks [161]. This notion is explained in more detail in Chapter 2. Sect. 2.2.1.

1.2 Deadline analysis

In Fig. 1.6, we provide a brief overview of the classes of workflow graphs that we analyze and the research questions we are trying to answer in the context of deadline analysis.

In the first part of the thesis, we study whether the executions of a sound workflow graph meet a given deadline, where tasks are annotated with fixed durations. While this assumption – tasks are assumed to have fixed durations – limits the applicability of our model, it represents a stepping stone towards more realistic settings. We distinguish the cases where the workflow graph is executed by a single resource (i.e., agent or processor, and when the time that is needed to execute two concurrent tasks is the sum of the times

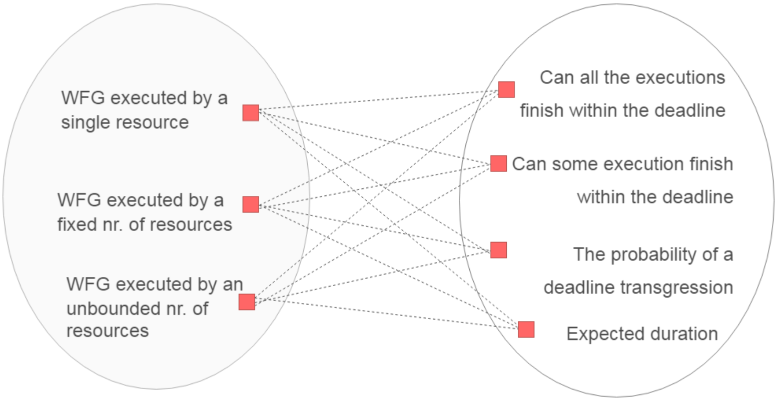

WFG executed by a single resource
WFG executed by a fixed nr. of resources
WFG executed by an unbounded nr. of resources
Can all the executions finish within the deadline
Can some execution finish within the deadline
The probability of a deadline transgression
Expected duration

Figure 1.6: Classes of wokflow graphs (left) and the research questions we tackle (right).

needed for each task) or an unbounded number of resources. We also briefly discuss the case where a fixed number of resources is given. Resources are assumed to be independent, identical and non-preemptive. Whenever a resource completes a task it immediately starts executing the next pending task – there is no idle-time between the execution of tasks.

Some of the simplifying assumptions were made so that we can solve the problems we tackle. For instance, in the case where a fixed number of resources is given, answering whether all the executions meet a given deadline is an NP-hard problem. However, we are able to provide polynomial algorithms for the single resource and the unbounded number of resources cases. Also, the algorithms we provide under some simplifying assumptions can be used as a basis for more complex models. For instance, when durations are specified as intervals - the process owner may replace the given interval with a rough approximation of the duration - e.g., with the average of the two extremes, and one can still have an idea of the total duration of an execution of the process.

As shown in Fig. 1.6, some of the questions we consider are: can we say whether all the executions of a workflow graph finish within a given amount of time? Or conversely, can we guarantee that there exists an execution that meets a given deadline? Assessing whether there exists an execution that meets a given deadline can be established by computing the *minimum duration of an execution* of a workflow graph. Similarly, assessing whether all the executions of the workflow graph meet the deadline corresponds to computing the *maximum duration of an execution* of a workflow graph. The minimum duration and the maximum duration of an execution are important attributes of processes which need to meet rigid timing constraints, e.g., traffic control, flight mission control [88], etc.

In practice, it is likely that not all executions of a workflow graph are equally probable, hence we also study the class of stochastic workflow graphs, where each choice of a path in a workflow graph is annotated with the probability of being taken. In this setting, we address a different set of questions such as: what is the probability that all executions of a workflow graph meet a given deadline and what is the expected duration of an execution of a workflow graph.

1.2.1 Incorporating time in workflows and analysis techniques

The importance of incorporating time in workflow analysis has been frequently addressed in the literature, see [39, 79, 80]. There are multiple ways to introduce time in a Petri net and a survey on the different approaches is given in [59]. The various ways to introduce time in a Petri net differ from each other in several essential aspects such as: i) the type of timing constraint that is assumed, ii) the Petri net element for which the timing constraint is specified, iii) and the influence of time on the control of the net.

The timing constraint can be given deterministically, non-deterministically or stochastically. A deterministic timing constraint is given by a fixed value – which may denote a duration or a delay. A non-deterministic timing constraint is given by a time interval constraining the duration of execution of an activity. Lastly, a stochastic timing constraint is given by a probability distribution of the duration or the delay. The temporal specifications can be associated with any Petri net element: places, transitions, arcs or with the tokens. In the majority of the Petri net models augmented with time, the delays are associated with transitions, and only in a few, the delays are specified for places or arcs. The rationale behind this is the fact that transitions represent activities and activities take time. In the literature, the term *Timed Petri nets* is used to refer Petri nets for which a finite firing duration is associated to each transition in the net.

As mentioned, we consider deterministic timing constraints – tasks are annotated with fixed durations. Our model of time annotated workflow graphs, where tasks have fixed durations, can be modeled by Timed Petri nets. This can be achieved simply by adding in the equivalent Petri net model, for each task of a workflow graph a transition with a delay equal to the execution time of the corresponding task while all the other transitions have a zero delay.

There are various methods for analyzing (Timed) Petri nets and these can be grouped into three main categories: i) methods based on the reachability graph, ii) methods based on the state equation and iii) methods based on reduction or decomposition techniques [140]. The reachability graph entails the enumeration of all the reachable markings. On the other hand, the latter two methods are more efficient but they are applicable only to certain subclasses of Petri nets or to certain problems. In the following, we will discuss in more detail the advantages and the shortcomings of these different approaches for timing analysis.

General Petri nets can be analyzed for timing behavior in terms of their reachability graph, and there are various techniques and tools that support this [151, 68, 108, 110, 76]. This holds also for non-Petri-net like models, e.g., timed automata where the minimum cost reachability problem is addressed through exponential branch-and-bound based algorithms [102]. Since the construction of the reachability graph incurs an exponential blowup, these techniques do not run in polynomial time in the size of the original model.

A decomposition approach for the deadline analysis of free-choice workflow nets is given in [94]. To perform temporal validation of the net, the authors present a procedure to decompose a free-choice timed Petri net into a set of free-choice subnets each of which describes a routing path. The method has exponential running time as the complexity grows exponentially with the number of choice places. A different type of decomposition of the net, proposed for computing the minimum execution time of a process, is given in [97].

The authors map all the routing constructs of a workflow into a stochastic Petri net and they give formulas for computing the execution time of elementary design blocks: sequence, iteration, choice and parallelism. However this method can not be applied to a Petri net with complex control structures, not just *regular* structures which we will explain in Chapter 2 (i.e., nesting of concurrent and alternative control structures as shown in Fig. 1.4).

As shown, many of the approaches face the combinatorial blow-up of the state-space they analyze, a problem also known as the *state space explosion*. There are different approaches to deal with the state space explosion in the analysis of Petri nets. The most prominent approach is the reduction of the Petri net specification while preserving important properties with respect to a certain analysis problem (e.g., in the area of workflow verification, the aim is to reduce the specification of the net while preserving soundness, or for the timing analysis we study, the reduction should preserve the minimum or maximum delays between events). There are several works that address the idea of reduction in General Petri nets, such as [61, 113], subclasses of Petri nets [78] or timed extensions of Petri nets [128, 57].

The application of net reduction to timed versions of Petri nets is very limited due to the challenges posed by the preservation of timing constraints [57]. That's why the results that are particularly relevant for us are the ones in [128, 57], as the authors apply reduction techniques to study the timing behavior of Petri nets. However, they have the following shortcomings. In [57] the authors show that the reduction rules in [128] do not guarantee that the timing constraints are preserved from the original time Petri net to the reduced one. On the other hand the reduction rules in [57] alleviate but do not fully rule out the state space explosion rendering nets that are only more amenable to the reachability analysis.

Apart from the analytical techniques, simulation can also be a useful method for estimating durations of workflows [159] or to provide confidence intervals for performance indicators in general and there are various tools that support it such as [11, 6]. However, simulation can be expensive in terms of the compute time, and it can not be used to prove a certain property (e.g., all executions can be executed within a given deadline).

Despite significant advances in the analysis of Petri nets, the sub-class of sound (no local deadlock or lack of synchronization) free-choice Petri nets still lacked algorithms for their temporal analysis (e.g., the computation of the minimum duration of an execution) that exploit the specific properties of this model.

For certain subclasses of Timed Petri nets, i.e. the Petri nets where each place has exactly one input and one output transition, also known as timed event-graphs or marked graphs, the exact execution time can be computed efficiently, see e.g. [38, 116]. The same holds for Petri nets where each transition has exactly one input and one output place.

In this work, we show that some deadline analysis problems for workflow graphs can be solved in polynomial time. Our main contributions are polynomial-time algorithms for computing the minimum duration of an execution of a sound workflow graph executed by a single resource and by an unbounded number of resources respectively. We show that for acyclic workflow graphs, the maximum duration can be computed in linear time both for single and unbounded resources.

Note that when we fix the number of resources – to some value greater than one, then all the questions that we consider cannot be solved in polynomial time. Even computing the minimum duration of an execution of a sound acyclic workflow graph becomes NP-hard.

As mentioned, we also study the class of stochastic workflow graphs, where each choice of a path in a workflow graph is annotated with the probability of being taken. We prove that computing the probability of a deadline transgression is an NP-hard problem. This is proven for the single resource case and it also holds for the unbounded number of resources case. We also show that for the single resource case, the expected duration of an execution of a workflow graph can be computed in polynomial time. However, in contrast to this result, we show that for workflow graphs executed by an unbounded number of resources, computing the expected duration is NP-hard.

As shown, this part consists of algorithms to be used (largely) at design-time. We also have some results on how we can optimize processes after they are established, based on insights collected from past executions. These are presented in the second part of the thesis where we perform process mining for performance optimization.

1.3 Performance optimization

In Part II of this thesis we study performance optimization of workflows. Performance optimization of workflows consists in the development of tools or techniques to achieve operational gains in process execution on predefined quantitative measures or performance objectives.

A systematic optimization technology for workflows is challenging to devise. Workflow optimization can adopt techniques from different disciplines such as operations research to data mining or machine learning. However primary data mining approaches are process agnostic so it is not immediately obvious how to incorporate such results. On the other side the process centric approaches are rather focused on modeling [157]. When workflow execution data is available, it opens the opportunity for process analysis and improvement. The discipline which leverages all these three – data, processes and data science is process mining.

Process mining can tell us *what* to improve by unveiling process issues that were not visible at the design-time but that are apparent in practice, for example bottlenecks. It can also tell *how* to improve the process by providing knowledge which can be directly used in making better decisions at process run-time or in the automation of certain tasks as we will illustrate in this thesis.

1.3.1 Positioning with respect to the Process Mining Framework

Before proceeding to presenting our work, we would like to provide an overview of the Process Mining Framework, cf. [157], see Fig.1.7 and position our work with respect to it.

As described in [157], there are three main types of process mining: *discovery*, *conformance* and *enhancement*. The first type – discovery – corresponds to a set of techniques which process event logs to produce process models that explain the behavior observed in the logs. Conformance analysis can tell whether the event logs conform to a given process and the reciprocal. Enhancement aims at enriching process models with novel performance information e.g., cycle times, processing times, identifying bottlenecks, etc. Orthogonal to the types enumerated above are several distinct perspectives: the *control-flow* perspective

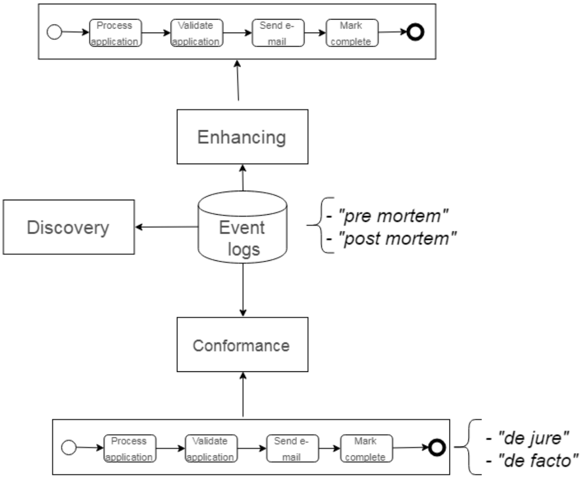


Figure 1.7: Overview of the refined process mining framework.

– concerned with the ordering of activities, the *organizational* perspective – it focuses on resources and their roles, and the *case/data* perspective – aims at understanding different properties of cases, such as durations, resource utilization, etc.

Process models can be split into “de jure models” and “de facto” models. The former represent normative process models which specify how work should be done and the latter ones are descriptive, simply aiming at capturing reality.

Data in event logs is split into “pre mortem” and “post mortem” event data. “Pre mortem” data corresponds to cases which are still alive and it can be used for better handling the corresponding cases. In contrast, “post mortem” data corresponds to executed cases and can be used for offline process analysis and improvement.

In part II of the thesis, we are mainly studying the data and organizational perspectives but nonetheless we incorporate the control flow perspective as well. We use the “de facto” process models obtained from process owners, we leverage the “pre mortem” and “post mortem” data to enhance our understanding on the process, i.e., to identify bottlenecks or wasted cycles and to provide recommendations on how the process execution can be streamlined.

Research in process mining has brought new algorithms for processing event logs and turning them into valuable process insights [105, 17, 103, 146]. More specifically, event logs were used to check the recorded behavior against already defined process models to identify potential deviations, see [23, 133]. Since processes are often not available, process discovery has also been a prolific area of research [150, 156]. In [22], the authors aim at the discovery of dependencies between the data and the routing decisions within a process. This is largely what we do in Chapter 5 and Chapter 6 of the thesis. In Chapter 5 we use the process execution logs and devise several text mining techniques to streamline the incident ticket dispatching process. In Chapter 6 we use monitoring data (time series) from hard disks to understand when their lifecycle is about to reach the end. The value of jointly analyzing time series data with process information is stressed also in this paper [122] where the authors provide a method for incorporating time series data which is external to the process into the log files.

We focus on two real world scenarios, and present how their afferent business process can be improved by leveraging process execution data. In the first scenario, we show a method to optimize the dispatching task in IT Service Delivery, and in the second scenario, we show how log data can help in fully automating a business task, namely the task of establishing when a hard disk needs to be replaced.

1.3.2 Optimizing the dispatching task in IT Service Delivery

For the first scenario, we analyze an incident resolution process in the IT Service Delivery industry, as shown in Fig. 1.8. In such a process (see e.g. [3]), an incoming incident ticket is dispatched to a *solving agent* by a human who reads the ticket to select a solving agent that is trained on resolving the class of problems documented in the ticket and also matches the required skill level. We noticed that in the past work [1, 16, 15], the historic execution data which captures who solved which ticket and how long it took is disregarded . However, we demonstrate in our study, that this data can serve to optimize and potentially automate the dispatching task.

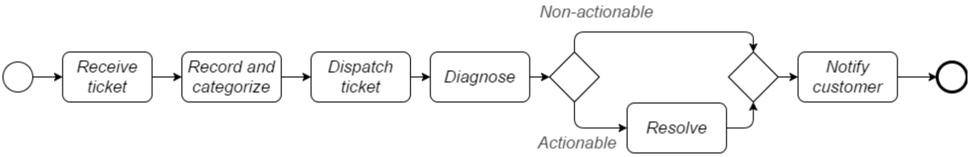


Figure 1.8: A simple incident resolution process

Optimizing the dispatching has multiple benefits: execution costs are decreased as the human dispatcher is displaced and the dispatching decisions are more informed as historic execution data is accounted for. This, in turn, as shown by our study, has the potential to decrease the total resolution time for incident tickets, as historic execution data reveals the agent which is provably fastest in resolving tickets documenting similar issues.

However, automating the dispatching task is a challenging problem. First, one needs to understand the type of issue documented in the ticket from the short, unstructured text in the ticket. Second, we need to discover classes of tickets that are homogeneous in the time it takes an agent to solve it such that we can reliably dispatch the ticket without major variations in the execution time. To this end, we devise a novel technique able to cluster incident tickets into categories that both reflect the problem they document and are homogeneous in the duration it takes for an agent to solve them.

We are not aware of any previous work on ticket clustering that aims at discovering topics which are semantically similar and are also homogeneous in the agent's performance. Clustering alerts and incident tickets has been attempted in the past [134, 46] for both structured and unstructured text using either graph theoretic approaches [46] or a combination of a latent semantic indexing based technique with a hierarchical technique [134]. We have observed that standard text similarity measures used by the authors in [46] perform poorly when used in clustering tasks due to data sparseness and the lack of context. We differentiate from these approaches by proposing a similarity metric between tickets that overcomes the vocabulary mismatch problem, by using semantic similarity between words inferred from a large domain specific corpus. The main contribution of this first study is a novel clustering algorithm which discovers incident categories that are both topically related but also homogeneous in terms of agent's performance which is of high importance for the subsequent dispatching task.

1.3.3 Automating disk replacement decisions

The second scenario also illustrates an opportunity to automate a part of a business process by considering log data. More specifically, we consider the life cycle of a computer's hard disk: a disk is installed, then monitored and ultimately when the administrator decides, the disk is replaced. We devise a novel data mining technique able to automatically predict disk replacements with high detection rate, i.e. 80-98%, whereas previous works only attained 50%-60% accuracy [62, 60]. Our prediction pipeline is based on time series data collected from the disk self-monitoring subsystem, also known as SMART data. Our model can automate the administrator's decision for when to replace a disk. Our pipeline correctly predicts the necessity of a disk replacement even 10-15 days in advance. This in turn enables timely replacement decisions and better protection against data loss.

There are several challenges in building such a prediction pipeline (see e.g., Fig. 1.9). As mentioned, the prediction pipeline receives as input the SMART data. The SMART data consists of values for different disk health indicators such as internal temperature or the number of uncorrected errors. Each such daily observation on its own is not *stable* due to the recovery mechanisms embedded in the disk. That is, at a certain point, one may see large number of errors but the next observation shows zero errors as all errors have been corrected by the error correcting unit embedded in the hard disk. Second, due to the lack of standards when implementing SMART attributes, e.g., manufacturer specific normalizations in the SMART reporting, one needs to discover the SMART attributes that are indicative of emerging failures for each disk model. Third, the disk data that we used are highly imbalanced, viz. only about 2% of disks are replaced, which makes the task of building high quality predictive models very challenging [137].

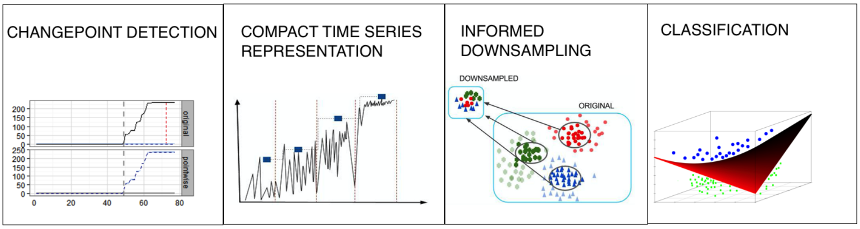


Figure 1.9: Our data processing pipeline for predicting disk replacement.

We devise a novel prediction pipeline that addresses all the challenges above and that discriminates with high accuracy between healthy and failing disks. Our pipeline consists of four steps as shown in Fig. 1.9. First, the important SMART attributes are selected through *changepoint detection*. Next, each of the selected SMART attributes is brought to a more compact representation via *smoothing*. The data is balanced through an informed downsampling procedure and lastly the data is fed to a highly accurate classification algorithm. All these steps are presented in detail in Chapter 6.

Predicting disk replacement is an instantiation of a more general topic – *proactive maintenance*. Proactive maintenance refers to a degradation-based anticipation on when a system will fail or stop working as expected. Industries have shown a growing interest in this field also known as condition based maintenance (CBM), see [19, 58]. As noted in [24], a generic and scalable prognostic methodology does not exist, as most of the developed prediction approaches are application or equipment specific.

Approaches to predict component failure vary wildly from simple failure rate models to complex physics-based models [31]. They may rely on various inputs such as current and past operating conditions, known fault patterns, engineering model, maintenance history, etc. The approaches can be grouped into two categories: *model-based* approaches and *data-driven* approaches [115].

Model-based approaches are used when a mathematical model can be built from first principles as in [36, 63, 21]. In such a method, the metric used for predicting component failure is the difference between the actual values of the real systems and the outcomes of the mathematical model. A large difference in this value is indicative of anomalous behavior. Threshold values for this difference such that one can distinguish between the presence or absence of faults can be inferred statistically. One concrete example would be the micro architectural thermal modeling tool for processors described in [95] that could be used in assessing processor health based on monitored heat.

Data-driven approaches, on the other hand, use real data (such as sensor data) to identify features that are indicative of component degradation and to predict the component's behavior. There are multiple techniques that can be employed here such as multivariate statistical methods, discriminant analysis, signal analysis, neural networks, Bayesian networks, etc., and some examples of such data-driven approaches are [117, 138, 121].

Our approach for predicting disk replacement is a data-driven approach. The authors in [145, 62, 60, 163] have also worked on building a predictive model for the timely discovery of impending disk failures. There are several key differences between the aforementioned studies and ours. First, our approach focuses on selecting the SMART indicators that correlate with disk replacements unlike previous works where the authors guess the subset of relevant attributes [62] or just use all predictors in the model [60]. In addition, we propose stable representations of the time series data for each disk as input to the predictive model unlike previous studies which operate directly on the raw data. Also, some studies are based on monitoring data from drives used in accelerated life tests, whereas we rely solely on field data collected when the disks were in actual use. The problem with data collected during testing in uniform controlled environments is that although it can be insightful in understanding the role of certain environmental factors, it has been shown to be not informative enough with respect to actual failure rates observed in the field [92]. Also, we note that some manufacturers deploy the disks with embedded failure predictive models. However, these models are based on simple methods, such as threshold-based normalizations. According to field observations, these models are built such that they avoid false alarms at the expense of a weak predictive power.

The algorithms for deadline analysis of workflow graphs and the process mining based solutions for performance optimization of workflows are presented in more detail in the rest of the thesis. In the following, we present the dissertation structure.

1.4 Dissertation Structure

The contents of the thesis are structured in two main parts: Part I – Deadline analysis of workflow graphs, consisting of Chapter 2, Chapter 3, and Chapter 4 and Part II – Workflow performance optimization which in turn consists of Chapter 5 and Chapter 6

In Chapter 2, we present formal definitions for the fundamental concepts of the thesis such as workflow graphs and their semantics. In Chapters 3 and 4, we present the results in the deadline analysis of workflow graphs executed by a single resource and by unbounded resources respectively. We showcase on real data extracted from a Dutch financial institution the usefulness and the performance of these algorithms in practice, see 4.6. Next, we move to the workflow optimization part and in Chapters 5 and 6, we present the two use cases on which we show how log data can serve in optimizing processes. Finally, we present our concluding remarks in Chapter 7.

Part I

Deadline Analysis of Workflow Graphs

Chapter 2

Foundations

In this section, we define the necessary fundamental notions for understanding Chapter 3 and Chapter 4. We start by giving the formal definition of Petri nets and we present some of the important subclasses of Petri nets such as: state machines, marked graphs and workflow nets in Sect.2.1. Next, we introduce workflow graphs, their semantics and an algebraic characterization of Petri nets in Sect.2.2.

2.1 Petri nets

A Petri net is a special type of a directed graph, see e.g., Fig. 1.4, which has an initial state, also known as initial marking. The graph corresponding to a Petri net N is a directed, bipartite graph which has two types of nodes: *places* and *transitions* and where arcs are either from a place to a transition or from a transition to a place. Arcs between two nodes of the same type are not allowed. More formally:

Definition 1 (Petri net)**.** *A* Petri net *is a triple* $N = (P, T, F)$ *where:*

- $P = \{p_1, p_2, \cdots, p_m\}$ *is a finite set of* places*,*
- $T = \{t_1, t_2, \cdots, t_n\}$ *is a finite set of* transitions*,*
- $P \cap T = \emptyset$ *and,*
- $F \subseteq (P \times T) \cup (T \times P)$ *is the set of* arcs *corresponding to the* flow *relation* F*.*

We write ${}^\bullet p = \{t \mid (t, p) \in F\}$ and $p^\bullet = \{t \mid (p, t) \in F\}$ to denote the input and output transitions of a place p, respectively. Similarly ${}^\bullet t = \{p \mid (p, t) \in F\}$ and $t^\bullet = \{p \mid (t, p) \in F\}$ for the input and output places of a transition t.

In the following, we present the semantics of Petri nets. First, we introduce some necessary preliminary notions.

Definition 2 (Multi-set)**.** *Let* A *be a set. A multi-set over* A *is a mapping* $m : A \to \mathbb{N}$*. For two multi-sets* m_1*,* m_2*, and each* $x \in A$*, we have:* $(m_1 + m_2)(x) = m_1(x) + m_2(x)$ *and* $(m_1 - m_2)(x) = m_1(x) - m_2(x)$*.*

A *marking* of a Petri net $m : P \rightarrow \mathbb{N}$ is a multi-set over P.

We say a place p is *marked* by a marking m if $m(p) > 1$. A transition $t \in T$ is *enabled* in a marking m if m marks every place of ${}^{\bullet}t$.

A transition t, enabled in the marking m can fire, leading to a new marking m', we write this as $m \xrightarrow{t} m'$ and the following equation holds:

$$m'(p) = \begin{cases} m(p) - 1, & \text{if } p \in {}^{\bullet}t \\ m(p) + 1, & \text{if } p \in t^{\bullet} \\ m(p), & \text{otherwise} \end{cases}$$

Some additional important concepts regarding Petri nets are the notions of liveness and boundness.

Definition 3 (Live, bounded)**.** *Let $N = (P, T, F)$ be a strongly connected Petri net, and let m_0 be the initial marking.*

- *A transition $t \in T$ is* live *if for each marking m reachable from m_0 there exists a marking m' such that m' is reachable from m and t is enabled in m'.*
- *The Petri net N is* live *if each transition in T is live.*
- *The Petri net N is* k-bounded *if for each reachable marking m and each place $p \in P, m(p) \leq k$.*
- *N is* bounded *if there exists k such that N is* k-bounded.

In this dissertation we use a restricted class of Petri nets for modeling and analyzing workflows. One of the important properties of the nets we analyze is named *free choice*.

Definition 4 (Free choice)**.** *A Petri net $N = (P, T, F)$ is a free choice net if for every two places $p_1, p_2 \in P$ either $p_1^{\bullet} \cap p_2^{\bullet} = \emptyset$ or $p_1^{\bullet} = p_2^{\bullet}$.*

Petri nets which model workflow procedures [152] have some particular properties. Most importantly, they have two special places, i and o, also known as the *source* and the *sink* of the net respectively. These places denote the beginning and the termination of a procedure. These Petri nets are called *Workflow nets*. Formally, a Workflow net is defined as follows:

Definition 5 (Workflow net)**.** *A Workflow net is a quintuple (P,T,F,i,o) where:*

- *(P, T, F) is a Petri net,*
- *$i, o \in P$ are places such that i has no incoming arcs and o has no outgoing arcs,*
- *The graph $(P \cup T, F \cup (o, i))$ is strongly connected.*

A Petri net which is also a Workflow net is given in Fig. 1.4.

There are other interesting subclasses of general Petri nets also. These classes are of interest because they model some specific applications of Petri nets and also some important analysis problems in these classes can be solved, in contrast to the class of general Petri nets, in polynomial time. The two classes we will present in the following are *state machines* and *marked graphs*.

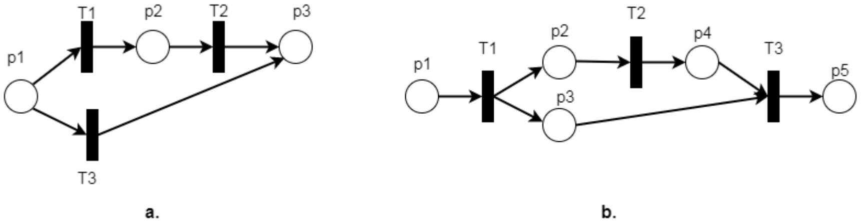


Figure 2.1: An example of a state machine - (a), and a marked graph - (b)

Definition 6 (State machine)**.** *A Petri net N is a* state machine *if and only if* $\forall t \in T, |^\bullet t| = |t^\bullet| = 1$.

An example of a state machine is given in Fig. 2.1 (a).

Definition 7 (Marked graph)**.** *A Petri net N is a* marked graph *if and only if* $\forall p \in P, |^\bullet p| = |p^\bullet| = 1$.

An example of a marked graph is given in Fig. 2.1 (b).

The nondeterministic nature of Petri net executions can lead to *conflict*. In conflict, one place enables two or more transitions. One of the enabled transitions may fire and therefore remove the enabling token for the other transitions. For example, when we have a token in place p_1 in Fig. 2.1 (a), transitions T_1 and T_3 are in conflict.

Concurrency in Petri nets is given by the existence of a forking transition that places tokens simultaneously in two or more output places, as is the case for transition T_1 in Fig. 2.1 (b). As noted by Petri [40], concurrency can be seen as a binary relation which is reflexive (transition T is concurrent with itself), symmetric (transitions T_i and T_j are concurrent implies transitions T_j and T_i are concurrent) but not transitive (transitions T_i and T_j are concurrent, transitions T_j and T_k are concurrent does not imply that transitions T_i and T_k are concurrent). Petri nets can model also the mechanism of *synchronization*, when an event occurs only after all the required inputs are available. For example in Fig. 2.1 (b), places p_3 and p_4 are synchronized for the firing of transition T_3.

Marked graphs and state machines differ fundamentally in their modeling capability. Marked graphs model concurrency and synchronization. A state machine admits no synchronization and a marked graph allows no conflict.

An important partitioning of the set of places and of the set of transitions of the net is the partitioning into certain subnets named *clusters*. Free choice nets can be decomposed into a disjoint set of clusters [78].

Definition 8 (Cluster)**.** *The* cluster *of a node $x \in P \cup T$, denoted as $[x]$ is the unique smallest set $[x] \subseteq P \cup T$ satisfying the following conditions:*

- $x \in [x]$
- $\forall p \in P \cap [x]$, *then* $p^\bullet \subseteq [x]$, *and*
- $\forall t \in T \cap [x]$, *then* $^\bullet t \subseteq [x]$.

A set $X \subseteq P \cup T$ is a cluster if $X = [x]$ for some x.

For example, the cluster of p_1 in Fig. 2.1 (a) is the set $\{p_1, T_1, T_3\}$. Similarly, the cluster of transition T_3 in Fig. 2.1 (b), is the set $\{T_3, p_3, p_4\}$.

2.2 Workflow Graphs

Workflow Graphs are a special type of multi-graphs where each node has a routing logic associated with it. In the following, we will provide the definition of a workflow graph, their semantics, and we introduce the notion of an execution of a workflow graph.

Definition 9 (Weighted, directed multi-graph). *A* weighted, directed multi-graph $G = (V, E, c, w)$ *consists of a set of* nodes V*, a set of* edges E*, a mapping* $c : E \to V \times V$ *that maps each edge to an ordered pair of nodes and a mapping* $w : E \to \mathbb{N}$ *that maps each edge to a nonnegative integer, called its* weight *or* duration*. For each edge* e *with* $c(e) = (v, z)$*, we assume* $v \neq z$ *for simplicity throughout the paper.*

Definition 10 (Workflow graph). *A* workflow graph $\Gamma = (V, E, c, l)$*, is a multi-graph* $G = (V, E, c)$ *with distinct and unique source and sink nodes, denoted* v_{source} *and* v_{sink}*, respectively, equipped with an additional mapping* $l : V \backslash \{v_{source}, v_{sink}\} \to \{\text{XOR}, \text{AND}, \text{TASK}\}$ *that associates a* branching logic *with* XOR *and* AND*. Furthermore, we assume that every node is on a path from the source to the sink, that the source has a unique outgoing edge, called the* source edge *(*e_{source}*), and that the sink has a unique incoming edge, called the* sink edge *(*e_{sink}*).*

Tasks don't affect the routing logic, from a control flow stand point being equivalent with an AND or XOR -node with a single incoming edge and a single outgoing edge. It is natural to assign durations to tasks.

Note: we will henceforth omit tasks for simplicity, and keep all the other types of nodes. We annotate each edge $e = c(u, v)$ with a duration $w(e)$ such that $w(e)$ represents the total duration of the tasks that existed on the path between nodes u and v.

This leads us to the following definition:

Definition 11 (Workflow graph (omitting tasks)). *A* workflow graph $\Gamma = (V, E, c, l, w)$*, is a weighted multi-graph* $G = (V, E, c, w)$ *with distinct and unique source and sink nodes, denoted* v_{source} *and* v_{sink}*, respectively, equipped with an additional mapping* $l : V \setminus \{v_{source}, v_{sink}\} \to \{\text{XOR}, \text{AND}\}$ *that associates a* branching logic *with every node, except for the source and the sink. Furthermore, we assume that every node is on a path from the source to the sink, that the source has a unique outgoing edge, called the* source edge *(*e_{source}*), and that the sink has a unique incoming edge, called the* sink edge *(*e_{sink}*).*

Similarly as for Petri nets, for each node v, we define the *pre-set* of v, ${}^\bullet v = \{e \in E \mid \exists\, z \in V : c(e) = (z, v)\}$ and the post-set of v, $v^\bullet = \{e \in E \mid \exists\, z \in V : c(e) = (v, z)\}$. A node with a single incoming edge and multiple outgoing edges is called a $split$. A node with multiple incoming edges and a single outgoing edge is called a $join$.

We don't allow any node that has multiple incoming edges as well as multiple outgoing edges. Note that this does not restrict expressiveness as such a node can be converted into a join followed by a split without changing the semantics, see an example in Fig.2.2.

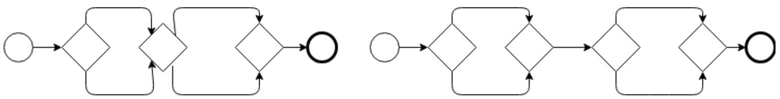


Figure 2.2: Workflow graph where a node has multiple incoming edges and multiple otugoing edges (left) and an equivalent workflow graph (right) where we convert the node into a merge followed by a split.

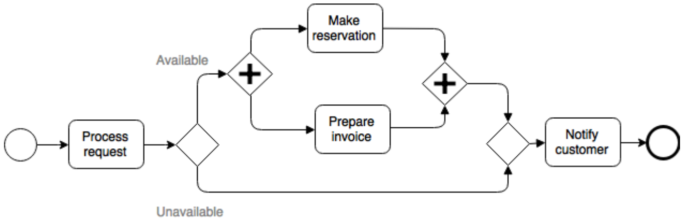


Figure 2.3: An example of a workflow graph modeling a booking request

Fig. 2.3 shows a workflow graph in BPMN notation, [5]: An XOR gateway is depicted as a diamond, an AND gateway as a diamond decorated with a plus sign. Source and sink are depicted as circles.

Let Γ be a workflow graph; Γ is *sequential* if it contains no AND-split and no AND-join. It is *acyclic* if the underlying graph has no cycles.

A *regular* workflow graph is a workflow graph that can be generated from a regular expression. To formalize, let ϵ be a workflow graph consisting of a single edge. Let us also introduce the following operations on a pair of workflow graphs X and Y: $(X\ ;\ Y)$ – denoting workflow graphs X and Y are in a sequence, $(X\ \mathrm{AND}\ Y)$ – X and Y are in a parallel block, $(X\ \mathrm{XOR}\ Y)$ – X and Y are in a choice block, and $(X\ \mathrm{LOOP}\ Y)$ – X and Y are in a loop. They can be visualized as shown in Fig. 2.4.

Let RWG represent the family of regular workflow graphs. We have $\Gamma \in RWG$ if and only if $\Gamma = \epsilon$ or there exist $X, Y \in RWG$ such that Γ equals one of $(X\ ;\ Y)$, $(X\ \mathrm{AND}\ Y)$, $(X\ \mathrm{XOR}\ Y)$, and $(X\ \mathrm{LOOP}\ Y)$. Such recursive definition is valid, as both X, Y have smaller number of edges than Γ. Defining RWG in this way enables us to establish a one-to-one correspondence between RWG and a family of regular expressions (i.e. formal strings defined recursively in exactly the same way as RWG). For instance formal expression $((\epsilon;\epsilon)\ AND\ (\epsilon\ LOOP\ \epsilon))$ has its corresponding regular graph as shown in Fig. 2.5.

It can be decided in linear time whether a workflow graph is a regular workflow graph using graph parsing techniques [86].

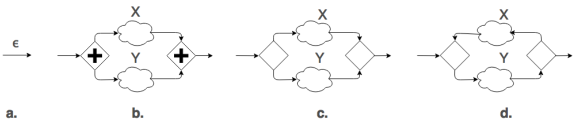


Figure 2.4: Regular patterns

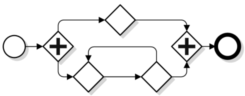


Figure 2.5: Regular graph

2.2.1 Workflow Graphs Semantics

As mentioned in Chapter 1, workflow graphs are equivalent to free-choice petri nets. In [158], the authors provide a structural translation from workflow graphs to workflow nets, based on a set of rules for translating the nodes of the workflow graph and a set of rules for translating the edges. Each task and each AND-gateway X of the worklfow graph is translated to a transition t_x and each XOR-gateway Y is translated to a place P_y. Next, each edge between two nodes x and y is translated to a pattern for the corresponding Petri net nodes, as shown in Fig.2.6. These rules can serve for translating any given workflow graph into a free choice workflow net, however, they can not be used for translating in the reverse direction. A different set of rules is given in [32]. These new rules permit also the translation of every free-choice workflow net into an equivalent workflow graph. The authors devise a normal form for workflow graphs and a normal form for free-choice workflow nets and they show that these two normal forms are isomorphic.

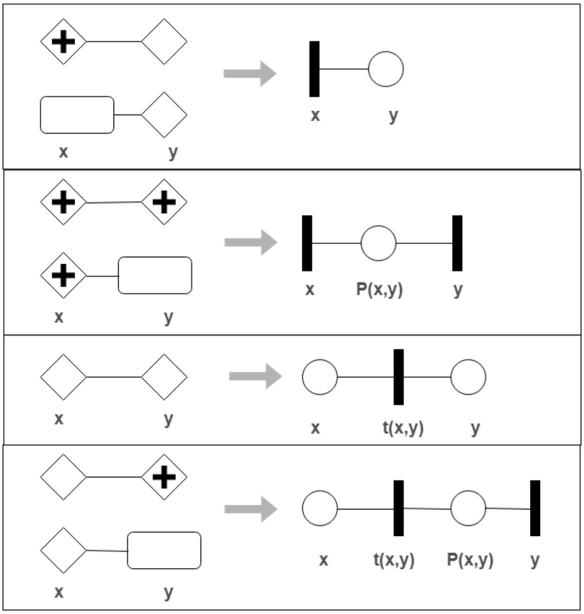


Figure 2.6: Translation rules for the edges of the workflow graph.

The *semantics* of workflow graphs is defined as a token game as it is in Petri nets. The execution of a node with AND-logic removes one token from each of its incoming edges and adds one token to each of the outgoing edges. The execution of a node with a XOR-logic removes non-deterministically a token from one of its incoming edges that has a token, then non-deterministically adds one token to one of the outgoing edges. Although we omit tasks, we allow nodes with just one incoming and one outgoing edge for technical reasons. For such nodes, XOR- and AND-logic behave the same.

The state or *marking* of a workflow graph is a distribution of tokens over the edges.

Definition 12 (Marking). *A* marking $m : E \to \mathbb{N}$ *of a workflow graph* $\Gamma = (V, E, c, l, w)$ *is* *a multi-set over* E*. If* $m(e) = i$*, we say that there are* i tokens *on edge* e*. The marking with exactly one token on the source edge and no token elsewhere is called the* initial marking *of* Γ*, denoted by* m_s*. The marking with exactly one token on the sink edge and no token elsewhere is called the* final marking *of* Γ*, denoted by* m_f*.*

Definition 13 (Transition)**.** *A triple* $t = (E_1, v, E_2)$ *is called a transition of* Γ *if* $v \in V$*,* $E_1 \subseteq {}^\bullet v$*, and* $E_2 \subseteq v^\bullet$ *such that:*

- *if* $l(v)$ *= AND, then* $E_1 = {}^\bullet v$*, and* $E_2 = v^\bullet$*,*
- *if* $l(v)$ *= XOR, and it is a split node, then* $E_1 = {}^\bullet v$ *and there exists an edge* $e \in v^\bullet$ *such that* $E_2 = \{e\}$*.*
- *if* $l(v)$ *= XOR, and it is a merge node, then* $E_2 = v^\bullet$ *and there exists an edge* $e \in {}^\bullet v$ *such that* $E_1 = \{e\}$*.*

A transition (E_1, v, E_2) is *enabled* in a marking m if for each edge $e \in E_1$ we have $m(e) > 0$.

We will use ${}^\bullet t$ to denote E_1 and $t^\bullet$ to denote E_2.

A transition t can be *executed* in a marking m if t is enabled in m. When t is executed in m, a marking m' results such that $m' = m - E_1 + E_2$ (recall that we defined m as a multi-set over E). We write $m \rightarrow m'$ if there exists a transition t, enabled in a marking m and its execution results in a marking m'. We write $m \xrightarrow{t} m'$ when the transition t is enabled in a marking m and its execution results in the marking m'. We use $\xrightarrow{*}$ to denote the transitive and reflexive closure of $\rightarrow$. We say m' is *reachable from a marking* m if $m \xrightarrow{*} m'$. We say m' is a *reachable marking* of Γ if $m_s \xrightarrow{*} m'$.

Definition 14 (Execution of a workflow graph)**.** *An* execution *of a workflow graph* Γ *is an alternating sequence* $\sigma = \langle m_s, t_0, m_1, t_1, \cdots \rangle$ *of markings* m_i *of* Γ *and transitions* t_i *such that* $m_i \xrightarrow{t_i} m_{i+1}$*, for each* $i \geq 0$*, and* $m_0 = m_s$*. We will be using also the shorter notation* $\sigma = \langle m_s, m_1, \cdots \rangle$ *to denote an execution.*

Definition 15 (Transition sequence)**.** *If* $\sigma = \langle m_0, t_0, m_1, t_1, \cdots, t_n, m_{n+1} \rangle$ *is an execution, then* $\tau_\sigma = \langle t_0, t_1, \cdots, t_n \rangle$ *is a* transition sequence *leading from* m_0 *to* m_{n+1} *and we write* $m_0 \xrightarrow{\tau_\sigma} m_{n+1}$*.*

Definition 16 (Maximal execution)**.** *An execution* σ *of a workflow graph* Γ *is maximal if either* σ *is of infinite length or* σ *ends in a marking from which no other marking is reachable.*

For example the execution $\langle m_s, t_1, m_1, t_3, m_2, t_4, m_3, t_5, m_f \rangle$ of the workflow graph in Fig. 2.7 is a maximal execution , where m_1, m_2, m_3 are respectively equal to $\{e_2\}$, $\{e_4, e_5\}$ and $\{e_6\}$.

As defined, an execution of a Petri net is finite or infinite sequence of transition occurrences. We additionally impose a fairness assumption on infinite executions. An execution in which some transition is enabled over and over again but does not occur from a point on in the corresponding transition sequence is *unfair*.

We say an edge e is *taken* at i in an execution $\sigma = \langle m_s, t_0, m_1, t_1, \cdots \rangle$, if $\exists\, t_i$ such that $e \in t_i^\bullet$. This is particularly relevant for XOR-split nodes where the taken edge corresponds to the outcome of the choice.

Definition 17 (Fairness)**.** *A maximal execution σ, of a workflow graph Γ, is* fair *if for each XOR-split v, that is executed infinitely often in σ, each edge $e \in v^\bullet$ is taken infinitely often in σ.*

An important property of Petri nets which, informally speaking, reflects whether the modeled system is correct or not is the *soundness* property. The notion of soundness was originally given for workflow nets in [154]:

Definition 18 (Sound Workflow net)**.** *A workflow net $N = (P, T, F, i, o)$ is sound if:*

1. *For every marking m, reachable from the initial marking of N, there exists an execution sequence σ such that $m \xrightarrow{\sigma} m_f$ and m_f is the final marking of N.*
2. *The final marking of N is the only reachable marking of N which marks the final place.*
3. *For every transition $t \in T$ there is a reachable marking which enables t.*

There exist multiple notions of soundness [161]. In our work, we consider soundness in the classical sense, which means that every execution of a Petri net is maximal, and any transition has a chance to fire.

In [158], van der Aalst et al. have proven that for acyclic workflow graphs, soundness is equivalent to the condition that neither a *local deadlock* nor a an *unsafe* marking are reachable from the initial state. To understand this statement, we need to define the notions of local deadlock and unsafeness.

We use the definition of local deadlock of Favre et al. [34].

Definition 19 (Local deadlock)**.** *A marking m of a Petri net N is a* local deadlock *if there exist a place $p \in P$ which is not the final place of N, such that m marks p and all the markings reachable from m mark p.*

The name *local deadlock* is used to distinguish from the notion of *global deadlock* which is a marking in which no transition is enabled. For example, for the workflow graph in Fig. 2.7, the marking $m = \{e_4, e_3\}$ is a local deadlock whereas the marking $m = \{e_4\}$ is a global deadlock.

Definition 20 (Unsafeness)**.** *A reachable marking m is* unsafe *or exhibits* lack of synchronization *if one edge has more than one token in m.*

We say a workflow graph is *safe* if it has no unsafe reachable marking.

We use a generalization of the result of van der Aalst [158], to define soundness for arbitrary workflow graphs as given in [85]. This definition gives a local view of correctness for arbitrary workflow graphs:

Definition 21 (Equivalent soundness characterization)**.** *A workflow graph is said to be sound if it has no local deadlock and no unsafe reachable marking.*

Definition 22 (Workflow graphs with probabilistic choice)**.** *A workflow graph has probabilistic choice if each XOR-node* v *is assigned a distribution* $\mu : v^\bullet \to [0,1]$ *such that* $\mu(e) > 0$ *for each* $e \in v^\bullet$ *and* $\sum_{e \in v^\bullet} \mu(e) = 1$.

We will henceforth also use the name probabilistic workflow graphs to denote workflow graphs with probabilistic choice.

Soundness guarantees that every fair execution terminates in the final marking of Γ. Soundness has various equivalent characterizations and can be decided in cubic time by help of the rank theorem for free-choice Petri nets [78], also cf. [158].

2.2.2 An Algebraic Characterization of Workflow Graphs

Workflow graphs, similarly to Petri nets, can be represented as an incidence matrix. In the following we give the algebraic characterization of workflow graphs by following the algebraic characterization of Petri nets given by Desel and Esparza in [78]. Such a characterization is useful because it allows us to use notions and results of linear algebra in the domain of workflow graphs.

Several of the definitions and lemmas we will present in the following, are originally phrased in the context of free choice Petri Nets. Due to the equivalence between workflow graphs and free choice Petri Nets stated in [32], the results hold for workflow graphs as well.

Definition 23 (Incidence matrix of a workflow graph)**.** *The incidence matrix* $\mathbf{N}$ *of a workflow graph is a matrix whose rows represent the edges of the workflow graph, and the columns represent the transitions of the workflow graph. The entry* $\mathbf{N}[i,j]$ *corresponds to the change of the marking of the edge* i *caused by the occurrence of the transition* $j = (E_1, v, E_2)$.

$$\mathbf{N}[i,j] = \begin{cases} -1 & \text{if } i \in E_1 \setminus E_2 \\ 1 & \text{if } i \in E_2 \setminus E_1 \\ 0 & \text{otherwise} \end{cases}$$

In Fig. 2.7 we show a workflow graph where we label its *AND* and *XOR* -nodes, edges and its transitions.

The transitions of this workflow graph are $t_1 = (\{e_1\}, v_1, \{e_2\})$, $t_2 = (\{e_1\}, v_1, \{e_3\})$, $t_3 = (\{e_2\}, v_2, \{e_4, e_5\})$, $t_4 = (\{e_4, e_5\}, v_3, \{e_6\})$, $t_5 = (\{e_6\}, v_4, \{e_7\})$, and $t_6 = (\{e_3\}, v_4, \{e_7\})$.

The incidence matrix of the workflow graph in Fig. 2.7 is:

$$\begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \end{array} \begin{array}{c} \begin{array}{cccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t6 \end{array} \\ \begin{pmatrix} -1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

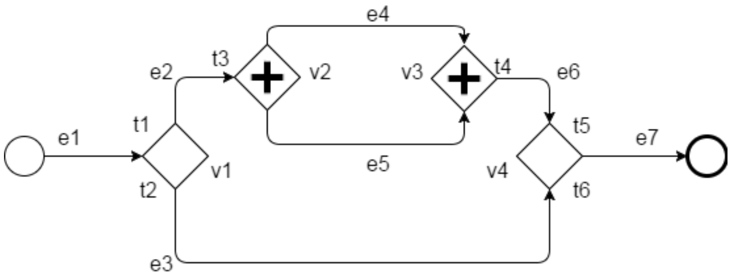


Figure 2.7: A workflow graph, its edges and transitions

The *Parikh vector* of a transition sequence τ, written $\vec{\tau}$, maps every transition t to the number of occurrences of t in τ. More formally,

Definition 24 (Parikh vector)**.** *Let (P, T, F) be a net and let σ be a finite transition sequence. The* Parikh vector $\vec{\tau}\colon T \to \mathbb{N}$ *of σ maps every transition $t \in T$ to the number of occurrences of t in σ.* [78]

For example, let $\tau_1 = \langle t_1, t_3, t_4, t_5 \rangle$, and $\tau_2 = \langle t_2, t_6 \rangle$ be two transition sequences of the workflow graph in Fig. 2.7. Note that for this workflow graph $|T| = 6$. The Parikh vectors for the two transition sequences are $\vec{\tau_1} = (1, 0, 1, 1, 1, 0)$ and $\vec{\tau_2} = (0, 1, 0, 0, 0, 1)$ respectively.

An important lemma that gives an algebraic description of the marking change in a workflow graph is the *marking equation lemma* [78]:

Lemma 2.1 (Marking equation lemma)**.** *For every finite transition sequence τ of a workflow graph with the incidence matrix* $\mathbf{N}$, *with $m \xrightarrow{\tau} m'$, the following equation holds:*

$$m' = m + \mathbf{N} \cdot \vec{\tau}$$

The expression in Lemma 2.1 is also called the *state equation* of a workflow graph and it compactly captures the relation between markings and the number of transition occurrences within a transition sequence.

Lemma 2.1 gives a necessary condition for a marking m' to be reachable. That is, any reachable marking must fulfill the state equation but the converse is not true [149]. Conversely, if the marking equation does not have vector solution for $\vec{\tau}$ with its components in $\mathbb{N}$, then the marking m' is not reachable from m.

Two transition sequences τ_1 and τ_2 of Γ are *permutations* of each other if $\vec{\tau_1} = \vec{\tau_2}$.

Let τ_2 be a permutation of the transition sequence τ_1. If $m_0 \xrightarrow{\tau_1} m$ and $m_0 \xrightarrow{\tau_2} m'$, then it follows from the marking equation lemma that $m = m'$.

We say a transition t is included in an execution σ, denoted $t \in \sigma$, if $\vec{\tau_\sigma}\ [t] > 0$ (recall that the Parikh vector $\vec{\tau}$, cf. Def. 24, maps every transition $t \in T$ to the number of occurences of t in σ). We write $\vec{\sigma}$ instead of $\vec{\tau_\sigma}$, henceforth, for convenience.

For two vectors $\vec{\tau_1}$ and $\vec{\tau_2}$ we set $\vec{\tau_1} \leq \vec{\tau_2}$ if $\forall\ t$, $\vec{\tau_1}\ [t] \leq \vec{\tau_2}\ [t]$.

For two vectors $\vec{\tau_1}$ and $\vec{\tau_2}$ we will denote their coordinate wise difference by $\vec{\tau_1}$ - $\vec{\tau_2}$.

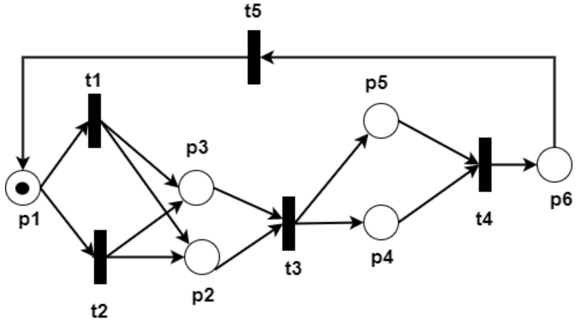


Figure 2.8: A live and bounded Petri net

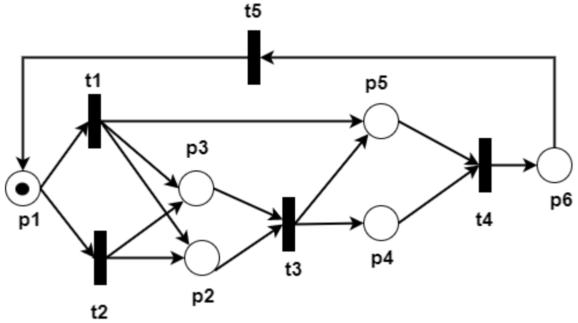


Figure 2.9: An unbounded Petri net

Let $\mathit{Trans}(v)$ denote the set of transitions of a node v. $\mathit{Trans}(v) = \{t \mid t = (E_1, v, E_2)\}$ where $E_1 = {}^\bullet t$ and $E_2 = t^\bullet$.

In the following we will present an important lemma proven by Gaujal et al. [27] for live and bounded free choice Petri nets. We will use this lemma to prove the correctness of the algorithms for computing the minimum duration of an execution of a workflow graph executed by a single resource or by an unbounded number of resources. The lemma is as follows (Corllary 3.1 in [27]):

Lemma 2.2. *Let N be a live and bounded Free Choice net. Let b be any transition of N and let $[b]$ be the cluster of b. There exists a unique reachable marking $m[b]$ in which the set of enabled transitions is exactly the set of transitions in $[b]$. Furthermore, the marking $m[b]$ can be reached from any reachable marking and without firing any transitions in $[b]$.*

We will illustrate this lemma both through an example and a counterexample.

First let's consider the (bounded) net in Fig. 2.8 with initial marking $m_0 = \{p_1\}$. We can see that for the cluster $[t_3] = \{p_2, p_3, t_3\}$, where t_3 is the only transition, there exists a single marking reachable from m_0 and in which t_3 is the only enabled transition, namely $m' = \{p_2, p_3\}$.

Now, consider the net N in Fig. 2.9 with initial marking $m_0 = \{p_1\}$, obtained from the previous net but by adding also arc $(t1, p_5)$. Note that this net is unbounded. We can see that for the cluster $[t_3] = \{p_2, p_3, t_3\}$, there exist two markings reachable from m_0 and in which t_3 is the only enabled transition, namely $m' = \{p_2, p_3\}$ and $m'' = \{p_2, p_3, p_5\}$. If we removed the arc (t_1, p_5).

Due to the equivalence between workflow graphs and free choice workflow nets [32], Lemma 2.2 also holds for workflow graphs.

Chapter 3

Workflow Graphs Executed by a Single Resource

In this chapter, we study whether the executions of a sound workflow graph (WFG) meet a given deadline, where tasks, or, equivalently, edges are annotated with execution times.

Studying this problem is relevant for workflow graphs modeling processes which need to respect precise timing and a high level of reliability. For such processes it is necessary to guarantee their completion within certain timing constraints [30].

We restrict in this chapter to the case where all tasks are executed by a single resource. The case where a workflow graph is executed by multiple resources is presented in the next chapter.

	1. All executions	2. Some execution	3. Probability of transgression	4. Expected duration
A. Acyclic Sequential WFG	$O(\|V\|+\|E\|)$	$O(\|V\|+\|E\|)$	Weakly **NP-hard**	$O(\|V\|+\|E\|)$
B. Sequential WFG	NP-hard	$O(\|E\|+\|V\|\cdot log\|V\|)$	Weakly **NP-hard**	$O(\|E\|^3)$
C. Regular WFG	$O(\|V\|+\|E\|)$	$O(\|V\|+\|E\|)$	Weakly **NP-hard**	$O(\|V\|+\|E\|)$
D. Acyclic Sound WFG	$\boldsymbol{O(\|V\|+\|E\|)}$	$\boldsymbol{O(\|V\|+\|E\|)}$	**NP-hard**	$\boldsymbol{O(\|V\|+\|E\|)}$
E. Sound WFG	NP-hard	$\boldsymbol{O(\|V\|\|E\|)}$	**NP-hard**	$\boldsymbol{O(\|E\|^3)}$

Table 3.1: Overview of results; new contributions in bold.

Table 3.1 shows the results for deadline analysis for different classes of workflow graphs, and our contributions are written in bold. The first two columns refer to the question whether all executions or some execution of the workflow graph finish before the deadline, respectively. These correspond to the problem of computing the maximum and the minimum duration of an execution respectively. In the former case, cycles in the graph are constrained by a termination order. For the third and fourth columns of Table 3.1, alternative branches have probabilities associated to them and we ask whether the probability to terminate within the given deadline is above a given threshold (third column) or what the expected duration is (fourth column).

Sequential graphs refers to the subclass of classical control-flow graphs without concurrency. We can use Dijkstra's algorithm [52] for computing the shortest path of a sequential graph and therefore determine the minimum duration (Cell B.2). If the sequential graph is acyclic, then the shortest and the longest path can be computed in linear time through a combination of topological sort and dynamic programming [144] (Cells A.1 and A.2). To define the maximum duration of a workflow graph, we need to constrain the number of times a loop can be traversed. We propose a general model of loop constraints for cyclic workflow graphs in this chapter. For this general model, we adapt the known result that computing the longest simple path in a sequential graph is NP-hard to show that it is NP-hard to compute whether all *admissible* executions (we explain what an admissible execution is in Section 3.1.2) of a sequential workflow graph meet a given deadline (Cells B.1 and E.1).

The expected duration of a probabilistic sequential graph, i.e., Markov chain, can be computed in polynomial time [35] (Cell B.4), and again there is a linear time solution for the acyclic case [35] (Cell A.4). *Regular graphs* refers to the subclass where the graph can be generated by a regular expression, i.e., every split corresponds to a join of the same logic (alternative or concurrent), see Fig. 2.5 for an example. For regular workflow graphs, solutions are simple recursive algorithms which we briefly mention in the paper and which run in linear time (Cells C.1, C.2, and C.4).

The main contribution we present in this chapter is an algorithm that computes the minimum duration of an execution of a sound workflow graph executed by a single resource in polynomial time (Cell E.2). The algorithm together with the correctness proof are given in Sect. 3.1.

Furthermore, we show that computing the probability of transgressing a deadline is NP-hard even for sequential regular graphs (Cell D.3), while it is known that there is a pseudo-polynomial solution for sequential graphs. This is shown in Sect. 3.2.

The chapter is structured as follows. In Section 3.1, we present a new algorithm for computing the minimum duration for a worfklow graph in polynomial time, and we present the NP-hardness proof for computing the maximum duration of an execution in our general model. In Section 3.2, we present the hardness result for assessing the probability of deadline transgression and, we present a polynomial time algorithm for computing the expected duration for probabilistic cyclic workflow graph.

3.1 Workflow graphs with nondeterministic choice

In this section, we study deadline analysis where choices in the workflow graph are assumed to be nondeterministic. We distinguish two cases. First, we assume that choices are made by the process internally, e.g., based on data-based decisions that we abstract from. In this case, we are interested in whether the process always terminates within the deadline, i.e., whether all its fair executions meet the deadline. Secondly, we consider that the choices are made by a superimposed scheduler that has the goal to make choices in order to meet the deadline, i.e., we ask whether there exists an execution that meets the deadline. It is clear that for the first case, it is sufficient to compute the maximum duration of an execution and for the second case, the minimum duration of an execution.

In Sect. 3.1.1 , we present the polynomial time algorithm to determine the minimum duration of an execution of a sound workflow graph. In Sect. 3.1.2, we define a system model to rule out infinite executions of a workflow graph and adapt the known NP-hardness result of the longest path problem to that system model. Finally, we discuss in Sect 3.1.3, the special cases of regular and acyclic workflow graphs.

3.1.1 The minimum duration of a workflow graph

In the following, we present, as a main contribution, an algorithm to compute the minimum duration of an execution of a sound workflow graph. We start by presenting several preliminaries that are needed for the algorithm such as the definition for the duration of an execution. In the following, let Γ be a sound workflow graph.

Since we assume a single resource, i.e., processor or agent executing the workflow graph, the *duration (or cost) of a fair execution* σ, of Γ, which we denote as $c(\sigma)$, is defined as follows:

Definition 25 (Duration of an execution - single resource). *For a workflow graph executed by a single resource, we define the duration of an execution* σ *as*

$$c(\sigma) = \sum_{e \in E} w(e) \cdot \sigma(e) \tag{3.1}$$

where $\sigma(e)$ *is the number of times edge* e *is marked in* σ*, which is either* 0 *or* 1 *if* Γ *is acyclic, but can be any non-negative integer or* ∞*, otherwise.*

$$\sigma(e) = \begin{cases} |\{i \mid m_i, m_{i+1} \text{ are consecutive markings of } \sigma \text{ such that } m_i(e) < m_{i+1}(e)\}|, \\ \hfill \text{for } e \in E \setminus \{e_{source}\} \\ 1, \ \text{for } e = e_{source}. \end{cases} \tag{3.2}$$

As mentioned, in this chapter we provide an algorithm for computing the minimum duration of an execution and we will prove its correctness. The algorithm for computing the minimum duration of an execution of a workflow graph, works on a weighted workflow graph, and for each node v, and each edge $e \in {}^\bullet v$, it updates a variable $\delta[e]$ that represents the currently known *minimum cost to reach the sink from* e.

In the following we will formally define the *minimum cost to reach the sink from* e which we will call the *minimum cost downstream from* e and next we will give the algorithm that computes the minimum duration of an execution. Lastly, we will prove that upon termination of the algorithm, the value associated to e_{source}, $\delta[e_{source}]$, represents the minimum cost downstream from e_{source} and hence, it represents the duration of the minimum duration execution (see Algorithm 3.1 and Algorithm 3.2).

3.1.1.1 Minimum cost downstream from an edge

To understand what we denote by *minimum cost downstream from* e, we will re-write the cost of a fair execution (3.1) in terms of recursive equations, that represent the *accumulated cost* for reaching the sink from a given edge, for a chosen execution. This will serve us later

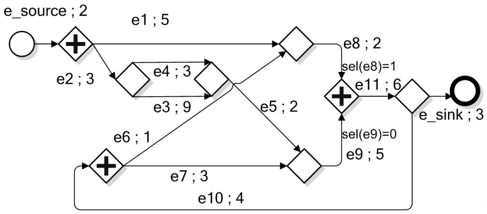


Figure 3.1: Workflow graph with edge weights

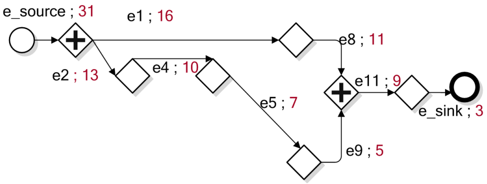


Figure 3.2: Minimum duration execution and the accumulated costs

when we will prove that the algorithm we propose for computing the minimum duration of an execution is correct.

We will define the (minimum) cost for reaching the sink in an execution for each edge. The minimum cost for reaching the sink from e_{source} represents the cost of the minimum cost execution.

As an example, consider the workflow graph in Fig. 3.1. In Fig. 3.1, edges are labeled (e.g. $e8$; 2) with an edge name ($e8$) and a duration (2). Fig. 3.2 represents the workflow graph restricted to the elements that are contained in the execution with minimum duration, i.e., it is a representation of the minimum duration execution. Each edge in Fig. 3.2 is labeled with the cost for reaching the sink (the cost incurred traversing the workflow graph backwards from the sink to the edge).

For e_{11}, the cost to reach the sink is: $w(e_{11})$ to which we add the cost of e_{sink} therefore, $6 + 3 = 9$. Since in any fair execution in which edge e_8 is marked, edge e_9 is also marked we carry the cost to reach the sink on only one of the edges (otherwise the cost is overcounted). The cost associated to e_9 stays $w(e_9)$ and the cost associated to e_8 becomes $w(e_8)$ plus the cost of e_{11}, and we obtain 2+9=11, etc. The duration of the entire execution (Fig. 3.2) equals the accumulated cost associated with e_{source} which is 31.

As presented in the example, for each node v such that $l(v)$ =AND, and $|^{\bullet}v| > 1$, the cost associated to the outgoing edge is accumulated on only one incoming edge. For this, we define a mapping $sel : E \rightarrow \{0, 1\}$ that specifies which of the incoming edges of an AND-join, accumulates the cost of the outgoing edge. Note that for all the v of this type, $\sum_{e \in {}^{\bullet}v} sel(e) = 1$.

In order to define the duration of an execution as a set of recursive equations that represent the *accumulated cost* for reaching the sink from a given edge, we need first to represent a fair execution σ as the sequence of edges that get marked in σ. Let $\tau_\sigma = \langle T_0, T_1, \cdots, T_n \rangle$ be the transition sequence that corresponds to σ, and let S_i denote the sequence composed of the elements in $T_i^{\bullet}$ ordered by a predefined edge labeling. The sequence of edges that get marked in σ is then given by $\langle e_{source}, S_0, S_1, \cdots, S_n \rangle$.

We use the notation $\sigma = \langle e_{source}, \cdots, e, \cdots, e_{sink} \rangle$ to express an execution σ as the sequence of edges that get marked in σ. Recall that we refer to fair, maximal executions of a sound workflow graph, and therefore the sequence of edges is finite and ends with e_{sink}.

Expressing a fair execution as the sequence of the edges that get marked in the execution serves to compute the *accumulated cost of an edge*in a particular fair execution. We compute this cost by accumulating backwards the cost on the edges in the execution. By this, we denote traversing the sequence of edges from the last to the first edge in the sequence and updating the cost of an edge $e \in {}^\bullet v$ at position i in the sequence based on the cost already computed for the edges in the sequence that belong to $v^\bullet$ at smallest positions greater than i.

Let e^i be the edge at position i in the sequence of edges that get marked in the execution.

Since we update based on the edges in $v^\bullet$, for the XOR nodes, we define a function $next_\sigma(e^i)$ such that for the edge at position i, $e^i \in {}^\bullet v$, it returns the edge in $v^\bullet$ that get marked after e^i gets marked. Formally, for an edge $e \in {}^\bullet v$, at position i in an execution σ , $next_\sigma(e^i) = e^{\{\min j > i | e^j \in v^\bullet\}}$.

For each position i in the sequence of edges that get marked in the execution, starting from the last index we update the cost of the edge e^i, which we denote by $d_\sigma(e^i)$.

Note that in this procedure, we may update the cost of an edge e multiple times. As the final accumulated cost associated with the edge e in σ, we take the value of $d_\sigma(e)$ after the last update.

$$d_\sigma(e^i) = \begin{cases} w(e^i) & \text{if } e^i = e_{sink} \\ w(e^i) + d_\sigma(next_\sigma(e^i)) & \text{if } l(v) = \text{XOR} \\ w(e^i) + \sum\limits_{e' \in v^\bullet} d_\sigma(e') & \text{if } l(v) = \text{AND and } |v^\bullet| > 1 \\ w(e^i) + d_\sigma(e') & \text{if } l(v) = \text{AND and } \{e'\} = v^\bullet \\ & \text{and } sel(e^i) = 1 \\ w(e^i) & \text{if } l(v) = \text{AND and } |v^\bullet| = 1 \\ & \text{and } sel(e^i) = 0 \end{cases} \tag{3.3}$$

Since e_{source} is always the first edge in the sequence of edges that get marked in a fair execution that starts in the initial marking m_s, it follows that $e^0 = e_{source}$ and $d_\sigma(e^0) = d_\sigma(e_{source})$. Since the computation of $d_\sigma(e)$ follows the semantics of workflow graphs, it can be proven by induction on i that $d_\sigma(e_{source}) = c(\sigma)$, the duration of the execution σ.

Let e be an edge of Γ and v the node of Γ such that $e \in {}^\bullet v$. We define an *edge enabling marking* m_e, as a reachable marking for which $m_e(e) = 1$, v is enabled in m_e and no other node is enabled in m_e.

Due to the Lemma 2.2 presented in Section 2.2.1, and due to the equivalence between workflow graphs and free choice Petri nets proven in [32], it holds that for a sound workflow graph, the edge enabling marking is unique.

We define $d^*(e)$, *the minimum cost downstream from* e, by considering executions that start in m_e. Note that m_s is the edge enabling marking for e_{source}.

$$d^*(e) = \min\{d_\sigma(e) \mid \sigma \text{ is a fair execution that starts in } m_e\} \tag{3.4}$$

Note that since m_e is a reachable marking, it holds that $m_e \xrightarrow{*} m_f$. Recall that by m_f we denote the final marking.

Since $d_\sigma(e_{source})$ represents the cost of a fair execution σ, $d^*(e_{source})$ represents the duration of the minimum duration execution.

3.1.1.2 Algorithm for computing the minimum duration of an execution

The algorithm that computes the minimum duration of an execution of Γ is Algorithm 3.1, which contains a call to the subroutine represented by Algorithm 3.2. The outer loop of the algorithm is similar to the Bellman-Ford algorithm [120] for sequential graphs, but in order to deal with concurrency we need a different relaxation procedure. In addition, the correctness proofs are more complex due to the characteristics of sound workflow graphs.

Algorithm 3.1 Minimum duration

```
function WFGMIN($\Gamma = \{V, E, c, w\}$)
    for $e \in E \setminus \{e_{sink}\}$ do
        $\delta[e] \leftarrow \infty$
    end for
    $\delta[e_{sink}] \leftarrow w(e_{sink})$
    for $i = 1 \cdots |V|$ do
        for all $e \in E$ do
            Let $u, v$ be nodes of $\Gamma$ s.t. $e = c(u, v)$
            RELAX(e,v)
        end for
    end for
end function
```

Algorithm 3.2 Relaxation of an edge $e \in {}^{\bullet}v$

```
function RELAX(e,v)
    if $l(v)$ = XOR, $|v^{\bullet}| = 1$, and $\{e'\} = v^{\bullet}$ then
        if $\delta[e] > w(e) + \delta[e']$ then
            $\delta[e] \leftarrow w(e) + \delta[e']$
        end if
    end if
    if $l(v)$ = XOR, $|v^{\bullet}| > 1$, and $e' \in v^{\bullet}$ then
        if $\delta[e] > w(e) + min_{e'}(\delta[e'])$ then
            $\delta[e] \leftarrow w(e) + min_{e'}(\delta[e']))$
        end if
    end if
    if $l(v)$ = AND, $|v^{\bullet}| = 1$, and $\{e'\} = v^{\bullet}$ then
        if $\delta[e] > w(e) + \delta[e']$ then
            if $sel(e) = 1$ then
                $\delta[e] \leftarrow w(e) + \delta[e']$
            else
                $\delta[e] \leftarrow w(e)$
            end if
        end if
    end if
    if $l(v)$= AND and $|v^{\bullet}| > 1$ then
        if $\delta[e] > w(e) + \sum_{e' \in v^{\bullet}} \delta[e']$ then
            $\delta[e] \leftarrow w(e) + \sum_{e' \in v^{\bullet}} \delta[e']$
        end if
    end if
end function
```

For an example execution of the algorithm, consider the workflow graph in Fig. 3.3, and let $sel(e_5) = 1$ and $sel(e_6) = 0$.

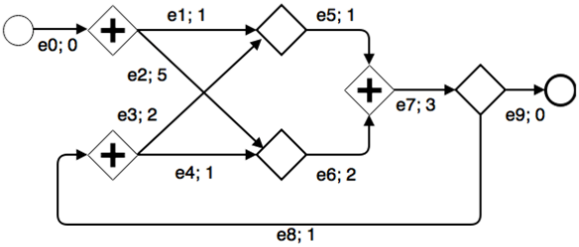


Figure 3.3: A cyclic workflow graph with edge weights

In Table 3.2, we have the initial values for $\delta[e]$ for each edge e of the workflow graph.

	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9
Initial	∞	∞	∞	∞	∞	∞	∞	∞	∞	0

Table 3.2: Initial values for $\delta[e]$.

After one iteration of the algorithm, considering the order of edge relaxations matches the order of the edges in Table 3.3, we have the values for $\delta[e]$ shown in Table 3.3.

	e_9	e_8	e_7	e_3	e_4	e_2	e_1	e_0	e_5	e_6
After the first iteration	0	∞	3	∞	∞	∞	∞	∞	4	2

Table 3.3: Values for $\delta[e]$ after one iteration of the algorithm.

After the second iteration, the values of $\delta[e]$ are given in Table 3.4.

	e_9	e_8	e_7	e_3	e_4	e_2	e_1	e_0	e_5	e_6
After the second iteration	0	∞	3	6	3	7	5	12	4	2

Table 3.4: Values for $\delta[e]$ after two iterations of the algorithm.

After the third iteration, the values of $\delta[e]$ are given in Table 3.5.

	e_9	e_8	e_7	e_3	e_4	e_2	e_1	e_0	e_5	e_6
After the third iteration	0	9	3	6	3	7	5	12	4	2

Table 3.5: Values for $\delta[e]$ after three iterations of the algorithm.

Notice that after this iteration all $\delta[e]$ have their final values (they won't change in potential subsequent iterations of the algorithm). Therefore, we have the minimum duration of an execution – the value associated to e_0, 12.

3.1.1.3 Correctness proof of Algorithm 3.1

In the following, we will show the correctness of the algorithm.

Lemma 3.1. *Let e be an edge and v a node such that $e \in {}^{\bullet}v$. We always have $\delta[e] \geq d^*(e)$.*

Proof. We prove the lemma by induction on k, the number of calls of $Relax(e, v)$.

Base case: $k = 0 : \delta[e_{sink}] = w(e_{sink})$, therefore clearly, $\delta[e_{sink}] = d^*(e_{sink})$, and for all $e \in E \setminus \{e_{sink}\}$ $\delta[e] = \infty$, and therefore $\delta[e] > d^*(e)$.

Induction step: Suppose that after the k-th call of $Relax(e, v)$ we have $\delta[e] \geq d^*(e)$ for all e. At the $(k + 1)$-th call of $Relax(e, v)$, only $\delta[e]$ may get updated, while all the other $\delta[e'], e' \in E \setminus \{e\}$ remain unchanged.

We will show that from the definition of $d^*(e)$ and from the induction hypothesis, it follows that $\delta[e] \geq d^*(e)$, for each of the relaxation cases. We will present the reasoning for one of the relaxation cases, as the justification for the remaining ones is similar.

Let v be a node, with $l(v) =$ XOR and $|v^{\bullet}| > 1$. Let ${}^{\bullet}v = \{e\}$ and $v^{\bullet} = \{e_1, e_2, \cdots, e_m\}$. Before the $(k + 1)$-th relaxation step, it holds that $\delta[e_i] \geq d^*(e_i)$ $\forall i, 1 \leq i \leq m$. After the

$(k+1)$-th relaxation step, $\delta[e]$ gets updated, such that $\delta[e]$ becomes $w(e) + min\{\delta[e_i] \mid e_i \in v^\bullet\}$, as presented in Algorithm 3.2. Therefore, due to the induction hypothesis, $\delta[e] \geq w(e) + min\{d^*(e_i) \mid e_i \in v^\bullet\}$ (i).

Using the definition of $d^*(e)$ and of $d_\sigma(e)$, we can prove that $d^*(e) = w(e) + min\{d^*(e_i) \mid e_i \in v^\bullet\}$ (ii). In the following, we provide the proof for (ii):

By definition, for $e \in \Gamma$, it holds that $d^*(e) = min\{d_\sigma(e) \mid \sigma$ is a fair execution that starts in $m_e\}$.

Also, by definition, recall that for a fair execution σ and for an edge $e \in \Gamma$, $d_\sigma(e) = w(e) + d_\sigma(next_\sigma(e))$ such that this is the last update.

$d^*(e) = \min\{w(e) + d_\sigma(next_\sigma(e)) \mid \sigma$ is a fair execution that starts in $m_e\}$. Therefore, since $w(e)$ is constant:

$d^*(e) = w(e) + \min\{d_\sigma(next_\sigma(e)) \mid \sigma$ is a fair execution that starts in $m_e\}$

Let m_{e_i} be the marking obtained when executing v and choosing $e_i \in v^\bullet$ from the marking m_e. Note that since m_e is an edge enabling marking, m_{e_i} is also an edge enabling marking. This can be easily verified by analyzing the different possibilities for $l(v')$ where $e_i \in {}^\bullet v'$.

Since $next_\sigma(e)$ is any e_i, $e_i \in v^\bullet$, we can rewrite $d^*(e)$:

$d^*(e) = w(e) + \min\{d_\sigma(e_i) \mid \sigma$ is a fair execution that starts in $m_{e_i}\}$.

Let σ_i be the execution that starts in m_{e_i}. For any choice of $e_i \in v^\bullet$, $\sigma(e) = \langle e_i, \sigma_i(e_i)\rangle$. We obtain:

$d^*(e) = w(e) + \min\{d_{\sigma_i}(e_i) \mid \sigma_i$ is a fair execution that starts in $m_{e_i}\}$. Therefore, $d^*(e) = w(e) + min\{d^*(e_i) \mid e_i \in v^\bullet\}$.

From (i) and (ii) it follows that $\delta[e] \geq d^*(e)$. □

Note that at each relaxation step, we can only decrease the value of $\delta[e]$. Once $\delta[e] = d^*(e)$, it doesn't change (it can not decrease further) as otherwise it would contradict the claim that $\delta[e] \geq d^*(e)$.

Lemma 3.2. *Let e be an edge. Let σ be a fair execution such that $d_\sigma(e) = d^*(e)$. Let $S = \langle e_{i-1}, \cdots, e_{sink}\rangle$ be the sequence of edges that get marked after e gets marked for the last time in σ. Each sequence of calls of $Relax(e, v)$ that has the property that edges $e_{sink}, \cdots, e_{i-1}, e$ have been relaxed in this order, after the sequence of calls to $Relax(e, v)$ we have $\delta[e] = d^*(e)$.*

Proof. We prove the lemma by induction on $k = |S|$.

Base case: k=0: $\delta[e_{sink}] = w(e_{sink})$, therefore $\delta[e_{sink}] = d^*(e_{sink})$.

Induction step: Suppose that after having relaxed $e_{sink}, \cdots, e_{k-2}, e_{k-1}$ in order, we have that $\delta[e_{k-1}] = d^*(e_{k-1})$. We will show that when we relax the edge e_k, given the definition of $d^*(e)$ and the induction hypothesis, it follows that $\delta[e_k] = d^*(e_k)$. We present the reasoning for one of the cases, as the justification for the remaining ones is similar.

Let v be a node, with $l(v) =$ XOR and $|v^\bullet| > 1$. Let ${}^\bullet v = \{e_k\}$ and $\langle e_{k-1}, \cdots, e_{sink}\rangle$ are the edges that get marked after e gets marked, in σ. Assume without loss of generality that $e_{k-1} \in v^\bullet$.

From the induction hypothesis, we have $\delta[e_{k-1}] = d^*(e_{k-1})$. Since e_{k-1} is the edge that gets marked after e_k gets marked in σ for which $d_\sigma(e_k) = d^*(e_k)$, we have that $d^*(e_k) = w(e_k) + d^*(e_{k-1})$ (recall claim (ii) from the proof of Lemma 3.1).

We have demonstrated in Lemma 3.1 that after each call of $Relax(e, v)$ it holds that $\delta[e] \geq d^*(e)$, therefore for our case $\delta[e_k] \geq w(e_k) + d^*(e_{k-1})$ (i).

Also, after the relaxation of e_k we have that $\delta[e_k] = w(e_k) + min\{\delta[e'] \mid e' \in v^\bullet\}$. Since $e_{k-1} \in v^\bullet$ it follows that $\delta[e_k] \leq w(e_k) + \delta[e_{k-1}]$ and since $\delta[e_{k-1}] = d^*(e_{k-1})$, it follows $\delta[e_k] \leq w(e_k) + d^*(e_{k-1})$. (ii)

From (i) and (ii) it follows that after relaxing e_k, we have $\delta(e_k) = w(e_k) + d^*(e_{k-1})$, and therefore $\delta[e_k] = d^*(e_k)$. ☐

Definition 26 (Loop-free execution)**.** *A fair execution σ of Γ, is a* loop-free execution *if no node is executed more than once in σ, and therefore no edge is marked more than once in σ.*

Lemma 3.3. *Some fair execution of Γ with minimum duration is loop-free.*

Proof. Let σ^* be a fair execution of Γ of minimum cost. Suppose σ^* is not loop-free. Then, we have $\sigma^* = \langle m_0, T_0, \cdots, m_i, T_i, \cdots, m_j, T_j, \cdots, m_f\rangle$ such that $T_i = (E_1, v, E_2)$ and $T_j = (E'_1, v, E'_2)$ (there exists a node v such that a transition in $Trans(v)$ is executed more than once). Let τ_{σ^*} be the transition sequence corresponding to σ^*.

We construct σ' such that $\tau_{\sigma'}$ is a permutation of τ_{σ^*}. We will show that in σ', a marking is repeated which contradicts the claim that σ' (implicitly σ^*) is an execution of minimum cost of Γ.

We divide σ^* into three parts:

$$\sigma^* = \underbrace{\langle m_0, \cdots, m_i\rangle}_{\alpha}\langle T_i\rangle\underbrace{\langle m_{i+1}, \cdots, m_f\rangle}_{\beta}$$

The constructed sequence σ' starts with α, followed by a permutation of the transitions contained in $\tau_{T_i\beta}$.

Let $\beta' = \langle m'_0, \cdots, m_k\rangle$ be a maximal sequence such that:

1. $m'_0 = m_i$
2. $\vec{\beta'} \leq \vec{\beta}$

Claim: T is enabled in m_k implies $T \in Trans(v)$.

We prove indirectly that such a β' exists. Let $A = \vec{\beta} - \vec{\beta'} - [T_i]$. Suppose $\exists\, T', T' \in Trans(v')$ enabled in m_k. We distinguish two cases:

Case 1: $\forall\, T'' \in A : T'' \notin Trans(v')$. In this case if we fire all the transitions in A we obtain m_f (due to the marking equation lemma). But m_f enables T', because of the assumption of case 1, therefore we reach a contradiction, namely that m_f is not the final marking.

Case 2: $\exists\, T'' \in A$ such that $T'' \in Trans(v')$. Then, T'' is enabled in m_k and therefore β' is not maximal and we reach a contradiction.

At this point, we have an intermediary prefix for σ', let's call it σ_{temp}, $\sigma_{temp} = \alpha\, \beta'\, T_i$. σ_{temp} does not include all the transitions in σ^*, more precisely we still have to add the transitions in $\vec{\beta} - \vec{\beta'}$.

Note that $T_j \in \vec{\beta} - \vec{\beta'} - [T_i]$. Let π denote the transition sequence obtained from τ_β after removing from it the transitions in $\tau_{\beta'}$ and T_i.

Since $m_0 \xrightarrow{\tau_\alpha} m_i \xrightarrow{\tau_{\beta'}} m_k \xrightarrow{T_i} m_k' \xrightarrow{\pi} m_f$, it means $\exists\, m_l \in \pi$ such that T_j is enabled in m_l (because T_j, $T_i \in$ $\mathit{Trans}(v)$ and they were both blocked in β' and $T_j \in \overrightarrow{\pi}$. We repeat the same procedure for the sequence π as we did for the original sequence and we reach a marking m_p in which the only enabled transitions belong to $\mathit{Trans}(v)$ (recall that $T_j \in \mathit{Trans}(v)$).

It holds that, for $l(v) \in \{$XOR-split, AND-split, AND-join $\}$, $m_k = m_p$ due to Lemma 2.2, stated by the authors in [27]. Since $\sigma' = \langle m_0, \cdots, m_k, \cdots, m_p, T_p, m_{p+1} \cdots, m_f \rangle$ and $m_k = m_p$, one can construct a lower cost execution $\sigma'' = \langle m_0, \cdots, m_k, T_p, m_{p+1} \cdots, m_f \rangle$ by removing the execution in σ' that led to the repetition of the marking. This implies that σ^* can not be the execution of minimum cost and tlghus we reach a contradiction.

If $l(v) =$XOR-join the result follows easily by noting that the edge e where $\{e\} = v^\bullet$ is marked twice and therefore the transition $T = (E_1, v, E_2)$ such that $e \in E_1$ is repeated, and thus the unique edge enabling marking of e is repeated.

□

Lemma 3.4. *For a sound workflow graph, after running Algorithm 3.1, it holds that* $\delta[e_{source}] = d^*(e_{source})$.

Before presenting the proof we would like to make some short remarks and establish a notation that we will use in the proof.

An equivalent way of expressing Lemma 2.2 is by referring to edge enabling markings. For any edge e of Γ, there exists a unique edge enabling marking m_e.

Gor a workflow graph Γ and a loop-free execution σ of Γ, we define Γ_σ as the workflow graph Γ restricted to σ such that it contains only the nodes of Γ that are executed in σ and the edges of Γ such that $\sigma(e) = 1$. For a loop-free execution σ of Γ, it follows that Γ_σ is an acyclic workflow graph.

The elements of an acyclic workflow graph are in a partial order defined by the flow of the graph: Let $G = (V, E, c)$ be an acyclic multi-graph. If x_1, x_2 are two elements in $V \cup E$ such that there is a path from x_1 to x_2, then we say that x_1 *precedes* x_2, denoted $x_1 \preceq x_2$, and x_2 *follows* x_1.

Now we have all the ingredients necessary for proving Lemma 3.4.

Proof. Lemma 3.3 states that some fair execution of Γ, with minimum duration, is loop-free (i). Recall that for a given fair execution σ, $d_\sigma(e_{source})$ represents the duration of execution of σ (ii). From (i) and (ii) it follows that some execution that minimizes $d_\sigma(e_{source})$ is loop-free (iii).

Note that m_s is the edge enabling marking for e_{source}.

Using (iii) and the definition for $d^*(e)$ instantiated to e_{source}, we obtain:

$d^*(e_{source}) = \min\{d_\sigma(e_{source}) \mid \sigma$ is a fair execution that starts in m_s $\}$. It follows that some σ^* for which $d_{\sigma^*}(e_{source}) = d^*(e_{source})$, is a loop-free execution.

Since σ^* is loop-free, it means that at most $|V|$ nodes are executed in σ^*. In each complete relaxation step (one iteration of the loop in line 6 in Algorithm 3.1), we relax all the edges. Therefore, at the $|V|$-th iteration we have relaxed all the edges, in decreasing order with respect to the partial order on the edges of Γ_{σ^*}. It means that at the $|V|$-th iteration, we will have relaxed all the edges that get marked after e gets marked in σ^*. Therefore, from Lemma 3.2, $\delta[e] = d^*(e)$. □

Therefore, we computed the duration of the minimum duration execution of the workflow graph, which is $d^*(e_{source})$.

For Algorithm 3.1, the initialization of the edge costs takes $O(|V|)$ time and each of the $|V|$ iterations over the edges of the workflow graph is performed in $O(|E|)$ time. The cost update is performed in constant time. Hence, we have proven the following:

Theorem 3.1. *The minimum duration of a fair execution of a sound workflow graph can be computed in time* $O(|V||E|)$.

3.1.2 The maximum duration of a workflow graph

To define the maximum duration of a workflow graph, we need to constrain the number of times a loop can be traversed. We propose a general model of loop constraints for cyclic workflow graph in this section. For this general model, we show that it is NP hard to compute whether all *admissible* executions of a sequential workflow graph meet a given deadline by using the known result that computing the longest *simple path* in a sequential graph is NP-hard. Note that a simple path is a path with no repeating vertices.

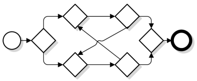


Figure 3.4: A workflow graph with unstructured loops

For graphs with *structured loops* as produced by the structured constructs (while, repeat loops), loop bounds represent the maximum iteration count of the loop body relative to the header. In a workflow graph however, there may be *unstructured loops* - loops with multiple entry points.

For unstructured loops, the specification of loop bounds is more complex [91] due to the multiple loop entries. An example of a graph containing unstructured loops is given in Fig. 3.4. As a generalization of loop bounds in regular graphs, we now define a system model that rules out infinite executions and we then show that computing the maximum duration execution for workflow graphs with cycles is NP-hard:

Assume a sound workflow graph Γ is given. We furthermore assume that the following two specifications are given with Γ:

1. A partition $\mathscr{V} = V_0, V_1, \cdots, V_k$ of the nodes V of Γ, where each V_i is called a *termination layer* such that V_0 contains only the sink, and each node in layer V_i is at distance i from the sink.

2. For each XOR-split v a mapping $\phi : v^\bullet \to \mathbb{N} \cup \{\infty\}$, called a *decision outcome restriction*, which denotes the maximal number of times an *outcome edge* (i.e., outgoing edge of an XOR-split, i.e., partial loop condition) may be traversed in an *admissible* execution such that

(i) each XOR-split v has an *unrestricted* outcome edge, such that $\phi(e) = \infty$, and (ii) each unrestricted edge leads into a *lower* termination layer, i.e., $\phi(e) = \infty$ where $c(e) = (v, w)$ implies $v \in V_i$, $w \in V_j$ where $i > j$.

A workflow graph equipped with $\mathscr{V}$ and ϕ is then called a *workflow graph with loop constraints*.

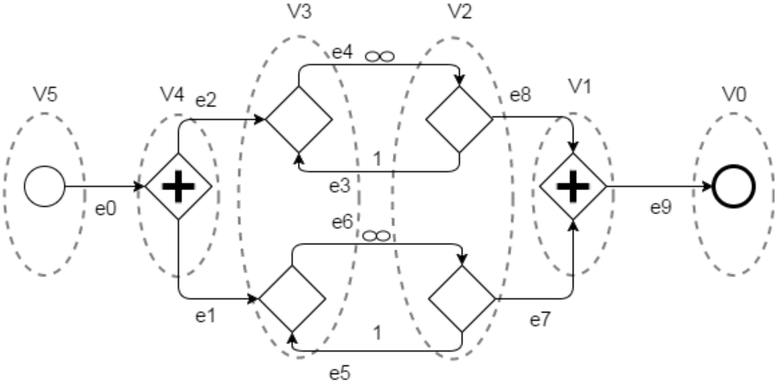


Figure 3.5: Example of a workflow graph with loop constraints.

We then call an execution σ of Γ *admissible* if for each outcome edge o of Γ, $\sigma(o) \leq \phi(o)$, i.e., if the specified outcome restrictions are obeyed by σ. An example of such an workflow graph is given in Fig. 3.5 with termination layers $V_0, V_1, \cdots, V_5$, and where $\phi(e_3) = 1$, $\phi(e_4) = \infty$, $\phi(e_5) = 1$, $\phi(e_6) = \infty$.

An example of an admissible execution σ of the workflow graph in Fig. 3.5, which we will give as the sequence of edges that get marked, is $\sigma = \langle e_0, e_1, e_2, e_4, e_3, e_4, e_6, e_8, e_7, e_9 \rangle$. In contrast, $\sigma = \langle e_0, e_1, e_2, e_4, e_3, e_4, e_3, e_4, e_6, e_8, e_7, e_9 \rangle$ is not an admissible execution as $\sigma(e_3) > \phi(e_3)$.

Proposition 3.1. *Each admissible execution terminates.*

Proof. First note that in any infinite exectuion there exists an XOR-split that is executed infinitely often. We will provide a proof of this claim. Assume that we have an infinite execution σ and from some point there exists no more XOR-split being executed. Let $\sigma = \sigma_1, \sigma_2$ and let σ_2 be the part of σ in which no XOR-split is executed. Since σ is infinite then σ_2 is infinite. There exists a node that is executed infinitely often in σ_2 which is not an XOR-split. We take such a node from the lowest termination layer, and let it be $v \in V_i$. Note that from the construction of the layers, from any node in a layer there exists an edge from this node to a node in the next lower termination layer. Therefore there exists $e = (v, w)$ where $w \in V_{i-1}$. Note that due to soundness and from the fact that v is not an XOR-split, it implies that w also is executed infinitely often. But we had chosen v to be the node that is executed infinitely and which is in the lowest termination layer, therefore we have a contradiction.

Tot each admissible execution terminates we make use of the claim we have just proved. We take the XOR-split that is executed infinitely often and which is in the lowest termination layer. Let it be $v \in V_x$. If this node is executed infinitely often, it means there exists an outcome edge $o \in v^\bullet$ which is taken infinitely often. This edge o, must be unrerstricted, i.e. $\phi(o) = \infty$ and from the definition of ϕ we have that o must be an edge who connects v to a node $w \in V_{i-1}$. Then this node w also is executed infinitely often. This node w must also be an XOR-split and we reach a contradiction that v is not in the lowest termination layer. Note that if w is not an XOR-split, then this node which also is executed infinitely often and which has an edge from w to $z \in V_{i-2}$ will also cause z to be executed infinitely often, and so on, until we get a token in the lowest termination layer (which contains the sink) – which means the exectuion is not infinite. ☐

Proposition 3.2. *The restrictions given by ϕ do not create an artificial deadlock, i.e., a partial execution that cannot be extended into an admissible execution because of an exhausted outcome restriction. Hence, an admissible execution always exists.*

Proof. Any execution in Γ which has the restrictions given by ϕ is also an execution in *unrestricted* Γ. Assume there exists a partial execution σ which can not be extended into an admissible execution. It means that there exists a node v which can not be executed. But since σ is also an execution in unrestricted Γ, and in Γ any partial execution can be extended, it means that the node that can not be executed in σ is an XOR-split which can not be executed due to the outcome restriction. But any XOR-split can be executed because it has an unrestricted outcome edge. Therefore we have reached a contradiction, and hence any partial execution can be extended into an admissible execution. ☐

Any termination order that satisfies these two natural requirements may be specified using the node partition and ϕ. We now present a hardness result for sequential workflow graphs with cycles:

Theorem 3.2. *The problem to determine whether all admissible executions of a sequential workflow graph with loop constraints meet a given deadline is NP-hard.*

Proof. We reduce from the problem of computing the longest simple path between any nodes in a directed graph. This, in turn, is a reduction from the Hamiltonian path problem. Given a weighted directed graph G, we construct a sequential workflow graph Γ with loop constraints as follows, cf. Fig. 3.6.

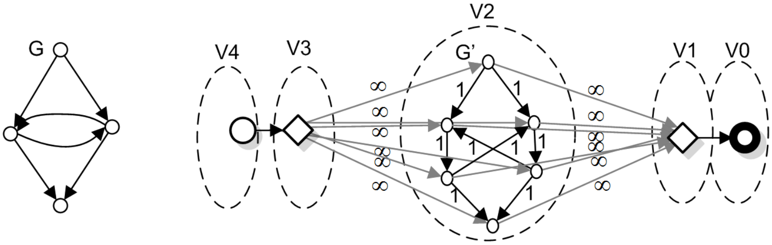


Figure 3.6: Constructed workflow graph for proof of Thm. 3.2

First we expand each node of G that is a split as well as a join (e.g., the two interior nodes of G in Fig. 3.6) into a separate join and a separate split with a single edge from the join to the split. These added edges are weighted with duration 0. Note that we do not represent these edge weights in Fig. 3.6 for simplicity. The obtained graph is called G', cf. subgraph of the right hand side graph in Fig. 3.6 encircled with label $V2$. Note that each path in G' corresponds to a path in G of the same duration and vice versa.

We add a fresh source and a fresh sink and we add an edge from the source to each node in G' and an edge from each node in G' to the sink, which all have duration 0 and are unrestricted ($\phi(e) = \infty$). The termination layers are specified as in Fig. 3.6, all edges in G'

are restricted with 1, i.e., must not be traversed more than once. It is easy to check that all conditions of the system model above are met.

Suppose the maximum duration admissible execution in Γ can be computed in polynomial time. That execution is a path from the source to the sink in Γ and due to the construction of Γ, it contains the longest duration path π' of G' between any nodes that contains each edge of G' at most once. This path π' of G' corresponds to a path π in G of the same duration that contains each node at most once and π must be the longest path of G with that property. Any longer path of G that visits each node at most once would correspond to path in G' of the same duration that visits every edge at most once. Hence we would be able to solve the longest simple path problem in polynomial time which is known to be NP-complete. ☐

Thm. 3.2 settles Cells B.1 and E.1 of Table 3.1.

3.1.3 Regular and acyclic Workflow Graphs

For a regular workflow graph, with a structured cycle, i.e., a while or repeat loop, or more general, of the form X LOOP Y, cf. Fig. 2.4, in order to compute the maximum duration, one needs the number of iterations for each loop. If we assume that the backedge of each loop (i.e., edge “x” in Fig. 2.4) of the regular graph is annotated with a positive integer k that represents the maximum number of times the backedge can be traversed, then the maximum duration of X LOOP Y is $(k+1) \cdot d_X + k \cdot d_Y$ where d_X denotes the maximum duration of the loop body X, and d_Y represents the duration associated to re-entering the loop. For computing the minimum duration we take $k = 0$. We still obtain the minimum/maximum duration of such an annotated regular workflow graph in linear time (Cell C.1, C.2 of Table 3.1).

If the graph is not regular but sequential, the minimum duration execution can be computed using Dijkstra’s algorithm (Cell B.2 of Table 3.1).

If the acyclic graph is not regular, but sequential, there is still a well-known dynamic programming solution to finding the longest path between two nodes, which runs in time $O(|V| + |E|)$ (Cell A.1 of Table 3.1). Analogously, the same algorithm can be applied to compute the shortest simple path by taking the minimum (Cell A.2 of Table 3.1).

Algorithm 3.3 Minimum duration, acyclic

```
function AcyclicWFGMin($\Gamma = \{V, E, c, w\}$)
    for $e \in E \setminus \{e_{sink}\}$ do
        $\delta[e] \leftarrow \infty$
    end for
    $\delta[e_{sink}] \leftarrow w(e_{sink})$
    TopologicalSort($\Gamma$)
    while $V \neq \emptyset$ do
        Select $v \in V$ s.t. $v$ is maximal w.r.t the topological sort
        $V \leftarrow V \setminus \{v\}$
        for all $e \in {}^\bullet v$ do
            Relax(e,v)
        end for
    end while
end function
```

For acyclic workflow graphs, we can use the algorithm for the cyclic case but without the need to perform $|V|$ iterations. Instead we exploit the fact that the elements of an acyclic workflow graph are in a partial order defined by the flow of the graph. Therefore, in order to make sure that the edges are relaxed respecting the partial order, first, the graph is sorted topologically in time $O(|V| + |E|)$. Secondly, the edges are relaxed in descending order with respect to the topological sorting in time $O(|E|)$. The algorithm that formalizes this idea is Algorithm 3.3.

Theorem 3.3. *The minimum duration of a fair execution of a sound acyclic workflow graph can be computed in linear time $O(|V| + |E|)$.*

Note that, in the acyclic case, for computing the maximum duration execution, one only needs to select the maximum instead of the minimum in the $Relax(e, v)$ procedure when $l(v) =$ XOR and $|v^\bullet| > 1$.

3.2 Workflow graphs with probabilistic choice

If not all executions of a workflow graph meet the deadline, we could ask whether at least a large portion of the executions does. We approach this question by assuming that decisions are resolved through a coin flip, i.e., the choices are probabilistic, see the definition in Chapter 2. Although some executions may not terminate, their probability is zero. We can then take the duration of an execution as a random variable and ask whether the probability of an execution terminating before the deadline exceeds a given threshold. We address this question in Sect. 3.2.1 and contrast the obtained results with results on computing the expected duration in Sect. 3.2.2

3.2.1 Probability of deadline transgression

We will show that computing whether the probability of an execution terminating before the deadline exceeds a given threshold is NP-hard. The hardness result can be obtained even for the simplest of graphs:

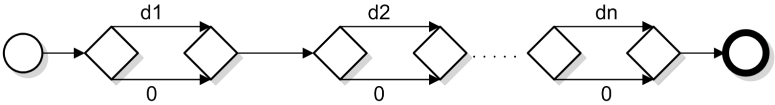


Figure 3.7: A chain of XOR-blocks

Theorem 3.4. *Given a regular, sequential, acyclic probabilistic workflow graph, a deadline* $\alpha \in \mathbb{N}$ *and a threshold* $p \in [0, 1]$*, computing whether* $\mathbb{P}(c(\sigma) \leq \alpha) \geq p$ *is NP-hard.*

Proof. The proof consists of a reduction from the *subset sum problem*, which is, given a set $D = \{d_1, \cdots, d_n\}$ of integers and an integer α, to determine whether any non-empty subset sums up to exactly α. This problem is known to be NP-hard [139]. Given these parameters, we consider the (regular, acyclic, sequential) probabilistic workflow graph in Fig. 3.7, where each decision outcome has probability 0.5. Suppose we can answer in polynomial time whether $\mathbb{P}(c(\sigma) \leq \alpha) \leq p$. We can then also compute $\mathbb{P}(c(\sigma) \leq \alpha)$ in polynomial time. Note that for the class of graphs of Fig. 3.7, $\mathbb{P}(c(\sigma) \leq \alpha) = \frac{k}{2^n}$, for some k where $0 < k < 2^n$ and n is the number of XOR gateways. One way to compute $\mathbb{P}(c(\sigma) \leq \alpha)$ is to run binary search with queries for $\mathbb{P}(c(\sigma) \leq \alpha) \leq p$ with varying p. Binary search takes $log(2^n) = n$ operations.

Because we now know $\mathbb{P}(c(\sigma) \leq \alpha)$, we can also answer in polynomial time if there exists an execution σ such that $c(\sigma) = \alpha$: We have $\mathbb{P}(c(\sigma) \leq \alpha - 1) \neq \mathbb{P}(c(\sigma) \leq \alpha)$, if and only if there exists σ such that $c(\sigma) = \alpha$. This in turn is the case exactly when there is a subset of $\{d_1, d_2, \ldots d_n\}$ which sums to α. ☐

The subset sum problem can be solved in pseudo-polynomial time, i.e., in polynomial time if numbers are represented in unary form. One way to represent the durations in unary form in a workflow graph is to assume that each edge needs one time unit and represent a duration of k time units by a sequence of k edges. For such a model, it is known for the case of sequential graphs, i.e., for Markov chains, that the probability of deadline transgression can be computed in polynomial time, e.g., by using the model checking algorithm of pCTL [107]. Therefore, the problem is said to be *weakly* NP-hard for sequential graphs. This can be extended to regular graphs, because each regular AND-block with subblocks X and Y can be treated as a sequence of X and Y under the assumption of a single resource. Therefore, in this case, regular graphs can be reduced to sequential graphs.

3.2.2 Expected duration

In some use cases, it may be sufficient to compute the expected duration, which turns out to be easier than the probability of transgression. The main contribution of this section is a polynomial-time algorithm for computing the expected duration for general sound workflow graphs. Subsequently we discuss some subclasses which have a linear-time solution.

General sound workflow graphs. For probabilistic sequential graphs, i.e., Markov chains, it is known that computing the expected duration can be done in polynomial time. In this context, it is often phrased as computing the mean hitting h_{xy} time in a Markov chain, which is the expected time of a random walk starting at node x to reach node y. The mean

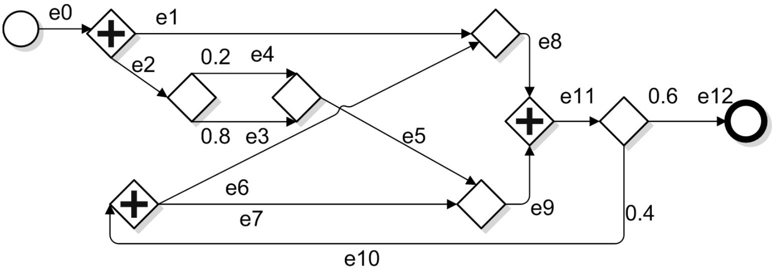


Figure 3.8: Expected duration in a cyclic graph

hitting times are the minimal non-negative solution to a set of n linear equations, as in [45], of which the computational cost is $O(n^3)$, cf. Cell B.4 of Table 3.1.

We can use a similar approach by identifying a suitable set of equations. Due to the linearity of the expectation, we can compute the expected duration of an execution as follows:

$$\mathbb{E}(c) = \mathbb{E}(\sum_{e \in E} w(e) \cdot X_e) = \sum_{e \in E} w(e) \cdot \mathbb{E}(X_e) \tag{3.5}$$

where the random variable $X_e(\sigma)$ is defined as the number of times an execution σ produces a token on e, defined in equation 2, which can be any non-negative integer in a cyclic workflow graph. To compute $\mathbb{E}(X_e)$, we can define a set of equations. For each AND-gateway v, we have $\mathbb{E}(X_e) = \mathbb{E}(X_{e'})$ for each $e, e' \in {}^\bullet v \cup v^\bullet$. For each XOR-gateway v and each $o \in v^\bullet$, we have

$$\mathbb{E}(X_o) = \mu(o) \cdot \sum_{e \in {}^\bullet v} \mathbb{E}(X_e) \tag{3.6}$$

In addition, we know $\mathbb{E}(X_{e_0}) = 1$ for the source edge e_0. We can now solve this system of linear equations in time $O(|E|^3)$ and we use Eq. (3.5) to compute the final result.

To sum up, we obtained the following result for Cell E.4 of Table 1:

Theorem 3.5. *The expected duration of a sound, probabilistic workflow graph can be computed in time $O(|E|^3)$.*

As an example, we consider the cyclic workflow graph in Fig. 3.8.

For the example from Fig. 3.8, we obtain the following set of linear equations, where a variable e stands for $\mathbb{E}(X_e)$:

$$\begin{array}{llll} e_0 = e_1 = e_2 = 1 & e_5 = e_3 + e_4 & e_9 = e_8 & e_{10} = 0.4 \cdot e_{11} \\ e_4 = 0.2 \cdot e_2 & e_6 = e_7 = e_{10} & e_{11} = e_9 = e_8 & \\ e_3 = 0.8 \cdot e_2 & e_9 = e_5 + e_7 & e_{12} = 0.6 \cdot e_{11} & \end{array}$$

For this example, assuming for simplicity, that all edges have duration 1, we obtain an expected duration of 12.95.

Regular graphs with cycles. For regular workflow graphs, the expected duration can be computed in linear time (Cell C.4 of Table 3.1) recursively, by exploiting the linearity of the expectation, as follows:

- Sequential and concurrent composition: The expected duration of $(X\ ;\ Y)$ and X AND Y is the sum of the expected durations of X and Y.
- Alternative composition: The expected duration of X XOR Y is $p_X \cdot d_X + p_Y \cdot d_Y$, where p_X ($p_Y = 1 - p_X$) is the probability of branching into subgraph X (Y resp.) and d_X is the expected duration of X.
- Loops: The expected duration of X LOOP Y, where $1 - p$ is the probability of re-entering the loop and p is the probability of exiting, can be computed by solving the system of two linear equations, which yields the following closed formula:

$$\sum_{k=0}^{\infty} p(1-p)^k((k+1)d_X + kd_Y)$$

Sequential workflow graphs. It is known for acyclic sequential graphs that the expected duration can be computed in linear time [50]. We approach the problem for acyclic sound workflow graphs in a similar way. We compute the expected number of times each edge is taken iteratively which is possible by processing the edges in the partial order defined by the flow of the graph. Having computed the expected frequencies for each edge of the graph, the expected duration is just the inner product of the expected frequencies and the durations of the edges. This settles Cell D.4 of Table 3.1.

3.3 Conclusion

Accurate timing information of a processes model at design time can be useful in assessing the exisiting processes and in generating new alternative ones that fulfil the timing constraints [30].

We presented new results on the deadline analysis of workflow graphs. In particular, as a main contribution, we presented an algorithm that computes the minimum duration execution of a sound workflow graph in polynomial time.

We have shown that where efficient algorithms for deadline analysis of sequential programs exist, we were able to define efficient algorithms for the corresponding workflow graph classes executed by a single resource exploiting the linear-algebraic properties of workflow graphs. In the next chapter we will address the case of workflow graphs executed by more than one resource.

Chapter 4

Workflow Graphs executed by Unbounded Resources

4.1 Introduction

In this chapter, we study whether the executions of a time-annotated sound workflow graph meet a given deadline when an unbounded number of resources (i.e., executing agents) is available. We assume the resources are non-constraining (e.g., we abstract away from resources' schedules, working hours, and such). We present polynomial-time algorithms and NP-hardness results for different cases. In particular, we show that it can be decided in polynomial time whether some executions of a sound workflow graph meet the deadline. For acyclic sound workflow graphs, it can be decided in linear time whether some or all executions meet the deadline. Furthermore, we show that it is NP-hard to compute the expected duration of a sound workflow graph for unbounded resources, which is contrasting the earlier result that the expected duration of a workflow graph executed by a single resource can be computed in cubic time. We also propose an algorithm for computing the maximum concurrency of a workflow graph. Knowing the maximum degree of concurrency can serve as an estimation of the number of resources needed for executing a workflow graph such that at any time, any enabled task is executed by a different resource.

As in Chapter 3, we analyze whether the executions of a sound workflow graph meet a given deadline, where tasks, or, equivalently, edges are annotated with execution times. We are not aware of any similar work for the model class we investigate. In the previous chapter we considered the case where the workflow graph is executed by a single resource (i.e., executing agent). In this chapter, we provide results for the case where the workflow graph is executed by an unbounded number of resources. We also discuss the case of a fixed number $n > 1$ of resources in Section 4.5.

Table 4.1 shows the results for deadline analysis of sound workflow graphs with unbounded resources, where our new contributions in this chapter are written in bold.

First, we ask whether all executions of a workflow graph finish before a given deadline. This is a question that arises when the choices made in the process at runtime are not under our control. This corresponds to Column 1 in Table 4.1. For the general case (Cell A.1), loops in the graph are constrained by a termination order. The complexity result for this

	1. All executions	2. Some execution	3. Probability of transgression	4. Expected duration	5. Min. nr. resources
A. Sound WFG	NP-hard	$\boldsymbol{O(\lvert V\rvert\lvert E\rvert)}$	NP-hard	**NP-hard**	**open***
B. Acyclic Sound WFG	$\boldsymbol{O(\lvert V\rvert + \lvert E\rvert)}$	$\boldsymbol{O(\lvert V\rvert + \lvert E\rvert)}$	NP-hard	**NP-hard**	**open***
C. Regular WFG	$O(\lvert V\rvert + \lvert E\rvert)$	$O(\lvert V\rvert + \lvert E\rvert)$	NP-hard	**NP-hard**	$O(\lvert V\rvert + \lvert E\rvert)$

Table 4.1: Overview of results; new contributions in bold, * we give a heuristic for this in Sect. 5

case follows directly from Theorem 2 in the previous chapter, Sect. 3.1.2. For acyclic workflow graphs, this question can be answered in linear time (Cell B.1) and we provide an algorithm for this in Subsection 4.2.1. For *regular graphs*, (as defined in Sect.2.2, see Fig. 2.5 for an example), the solutions again consist of simple recursive algorithms that run in linear time (Cell C.1).

Next, we assume we have control over the choices made in the process at runtime. Therefore, we ask the question whether there exists an instantiation of the process – an execution – that meets a given deadline. This corresponds to Column 2 in Table 4.1. In particular, as one of our main contributions, we show that for general sound workflow graphs, finding the minimum duration over all executions can be solved in polynomial time (Cell A.2). When restricting to acyclic workflow graphs (Cell B.2, similarly as for Cell B.1), the problem can be solved in linear time. As above, for regular graphs, the minimum duration of an execution can be computed recursively in linear time (Cell C.2).

Suppose not all executions meet a given deadline but only some. We can then ask whether the probability of a deadline transgression exceeds a given threshold - Column 3 in Table 4.1. Results carry over from our previous results in Chapter 3 where we have proven that computing whether the probability of an execution with a single resource terminating before the deadline exceeds a given threshold is NP-hard (Cells A.3, B.3 and C.3).

Also in the probabilistic framework, another valuable information is the expected duration of an execution of a given workflow graph. The results related to this question map to Column 4 in Table 4.1. We show that computing the expected duration is NP-hard even for regular graphs. This is in contrast to the execution with a single resource where, the expected duration can be computed in cubic time for general sound workflow graphs, as shown in Chapter 3.

Finally, we ask what is the optimal number of resources for the workflow graph where optimal means the minimum number k of resources such that each execution achieves its minimal execution time under k resources (Column 5 in Table 4.1). The maximum number of tokens that can exist in the graph, M_t, is an upperbound for k. Note that if we have at most M_t tokens in the graph, it means that at most M_t tasks can be enabled simultaneously, and since we don't consider task level parallelism, having more than M_t resources would not help in reducing the execution time. We are not aware of any prior work for computing M_t for workflow graphs. A simple algorithm to compute M_t would be to enumerate all the reachable markings of the workflow graph and return the maximum number of tokens that can exist in a marking. However, such algorithm runs in EXPTIME and in practice, this can be prohibitively expensive. We provide an algorithm to compute M_t, which despite not being a polynomial time algorithm, in practice it can be fast as it runs in EXPTIME only for some classes workflow graphs. We motivate the problem in more detail and we present our algorithm for computing the maximum concurrency of a workflow graph in Sect. 4.4.

4.2 Workflow graphs with nondeterministic choice

In this section, we present a polynomial time algorithm that computes the minimum execution time of a workflow graph executed by an unbounded number of resources. This can be used to determine whether some execution of a time annotated workflow graph with an unbounded number of resources meets a given deadline.

4.2.1 The minimum duration of a workflow graph

We start by presenting several preliminary notions such as the definition of the duration of an execution of a workflow graph executed by an unbounded number of resources.

To give the definition of the duration of an execution of a workflow graph, we make use of the token game. We first equip each token with an integer-valued clock initialized to zero. Then the *state* of the workflow graph is given by the tuple (m, c) where m is the marking and $c : E \to \mathbb{N} \cup \emptyset$, where $c(e) = \emptyset$ if $m(e) = 0$, i.e., when there is no token on edge e. We extend the token-game semantics to clocks and we set $(m, c) \xrightarrow{T} (m', c')$ when $m \xrightarrow{T} m'$ and $c'(e) = c(e)$ for $e \in m' \setminus T^\bullet$ and $c'(e) = w(e) + \max\{\, c(e') \mid e' \in {}^\bullet T\}$ for $e \in T^\bullet$.

In the initial marking, the state of the workflow graph is given by (m_s, c_s), where $c_s(e_{source}) = w(e_{source})$. Similarly, in the final marking, the state of the workflow graph is given by (m_f, c_f), where c_f is determined through the sequence of transitions from the initial state. In particular, the final state is not unique in general, as different during the execution may lead to a different value of c_f.

Definition 27 (Duration of an execution - unbounded number of resources)**.** *For a workflow graph executed by an unbounded number of resources, we define the duration of an execution σ as $c_f(e_{sink})$, where σ ends in the final marking m_f.*

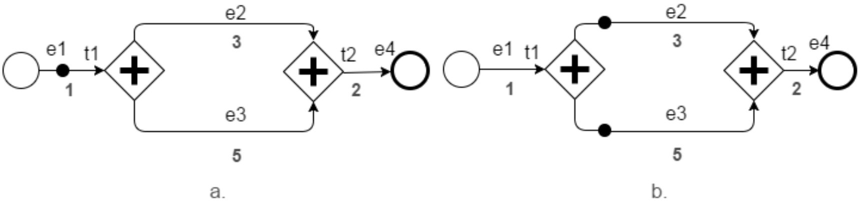


Figure 4.1: A simple workflow graph with edge weights and a marking change.

For an example of the clock update with the marking change, consider Fig. 4.1. In Fig. 4.1 a.), the state of the workflow graph is given by the marking $m_1 = \{e_1\}$, and clock value 1 for the token on the edge in the marking. Upon the marking change, after t_1 is executed, we have a new state, as shown in Fig. 4.1 b.), given by the marking $m_2 = \{e_2, e_3\}$ and clock values 4 and 6 respectively. We follow the same resoning for when t_2 fires, and thus the token reaching the sink has clock value 8 ($max\{4, 6\} + 2$).

In this chapter, we will again use a notion of *accumulated cost associated with an edge* in a fair execution similar to the approach in Chapter 3. We will adapt this definition for the new setting - an unbounded number of resources are available to execute the workflow graph and we will use it in proving the correctness of the algorithm that computes the minimum duration of an execution of a workflow graph. As before, the cost accumulated on the *source* edge represents the cost of a fair execution.

In the following, let Γ be a sound workflow graph.

To facilitate the computation of the cost accumulated on an edge in a fair execution σ, we proceed similarly as in the previous chapter and we express the execution as the sequence of edges that get marked in σ and we write $\sigma = \langle e_{source}, \cdots, e, \cdots, e_{sink}\rangle$. Recall that since we are interested in fair executions (and we assume soundness), the sequence of edges is finite and ends with e_{sink}.

We similarly traverse the sequence backwards, from the last to the first edge in the sequence and *update* the cost of an edge $e \in {}^\bullet v$ at position i in the sequence – denoted by e^i– based on the cost already computed for the edges in the sequence that belong to $v^\bullet$.

However a difference appears in the definition of $d_\sigma(e^i)$ - the function which we apply at each position i in the sequence of edges that get marked in the execution, starting from the last index, to compute the accumulated cost of an edge. Contrary to the definition of $d_\sigma(e^i)$in Chapter 3, for updating the cost of an incoming edge of an AND-split we don't take the sum of the accumulated cost on the outgoing edges anymore but we take the maximum of the accumulated costs on the outgoing edges. We have:

$$d_\sigma(e^i) = \begin{cases} w(e^i) & \text{if } e^i = e_{sink} \\ w(e^i) + d_\sigma(next_\sigma(e^i)) & \text{if } l(v) = \text{XOR} \\ w(e^i) + \max\{d_\sigma(e') \mid e' \in v^\bullet\} & \text{if } l(v) = \text{AND and } |v^\bullet| > 1 \\ w(e^i) + d_\sigma(e') & \text{if } l(v) = \text{AND and } \{e'\} = v^\bullet \end{cases}$$

As the final accumulated cost associated with the edge e in σ, we take the value of $d_\sigma(e)$ after the last update and using the same arguments as in Chapter 3 we have that $d_\sigma(e_{source}) = c(\sigma)$, the duration of the execution σ.

As an example, consider the workflow graph in Fig. 4.2. In Fig. 4.2, edges are labeled (e.g. $e8$; 2) with an edge name ($e8$) and a duration (2). Fig. 4.3 represents the workflow graph restricted to the elements that are contained in the fair execution with minimum duration, i.e., it is a representation of the minimum duration execution. Each edge in Fig. 4.3 is labeled with the accumulated cost for reaching the sink in that execution.

For e_{11}, the accumulated cost to reach the sink is: $w(e_{11})$ to which we add the cost of e_{sink} therefore, $6{+}3 = 9$. Based on our update rule for AND-join nodes, the cost associated to e_9 becomes $w(e_9)$ plus the cost of e_{11} and we obtain 14 and the cost associated to e_8 becomes $w(e_8)$ plus the cost of e_{11}, and we get 2+9=11. For edges e_5 and e_1, we update the cost by adding the edge weight to the accumulated cost on the outgoing edge of the

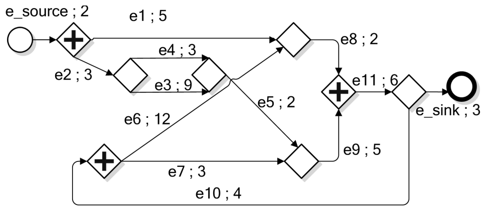


Figure 4.2: Workflow graph with edge weights

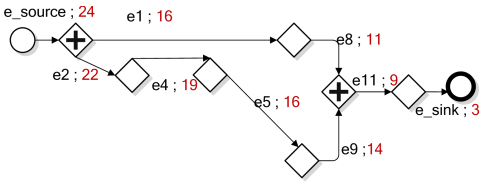


Figure 4.3: Minimum duration execution and the accumulated costs

XOR-split, and we obtain costs 16 (11+5) for $e1$ and 16 (14+2) for e_5. We apply the same rule for e_4 and we obtain an accumulated cost of 19 (16+3) and subsequently also for e_2 and we obtain 22 (19+3). Now we can compute the cost of the execution. Note that the AND-split we are about to process spawns two threads. The cost of the execution is decided by the longest thread (in terms of duration). Therefore, we update the cost accumulated on e_{source} to be equal to $w(e_{source}) + max(16, 22)$ which equals 24 and this equals the cost of the execution.

The algorithm for computing the minimum duration of a fair execution of a workflow graph with an unbounded number of resources, Algorithm 4.1, is given below and it resembles Algorithm 3.1. Note however that the relaxation is not the same. It works on a weighted workflow graph, and for each node v, and each edge $e \in {}^{\bullet}v$, it updates a value $\delta[e]$ that represents the currently known *minimum cost* to reach the sink from e based on relaxation rules specific to each node type (see Algorithm 4.2). All edge costs are updated at most $|V|$ times for a cyclic workflow graph (see Algorithm 4.1) and only once for an acyclic workflow graph (see Algorithm 4.3). Upon termination of our algorithm, the value associated to e_{source}, $\delta[e_{source}]$, represents the duration of the minimum duration execution.

Algorithm 4.1 Minimum duration

```
function WFGMin( $\Gamma = \{V, E, c, w\}$)
    for $e \in E \setminus \{e_{sink}\}$ do
        $\delta[e] \leftarrow \infty$
    end for
    $\delta[e_{sink}] \leftarrow w(e_{sink})$
    for $i = 1 \cdots |V|$ do
        for all $e \in E$ do
            Let $u, v$ be nodes of $\Gamma$ s.t. $e = c(u, v)$
            Relax(e,v)
        end for
    end for
end function
```

Algorithm 4.2 Relaxation of an edge $e \in {}^\bullet v$

```
function RELAX(e,v)
    if ($l(v)$ = XOR or $l(v)$ = AND) and $\{e'\} = v^\bullet$ then
        if $\delta[e] > w(e) + \delta[e']$ then
            $\delta[e] \leftarrow w(e) + \delta[e']$
        end if
    end if
    if $l(v)$ = XOR and $|v^\bullet| > 1$ then
        if $\delta[e] > w(e) + \min\{\delta[e'] \mid e' \in v^\bullet\}$ then
            $\delta[e] \leftarrow w(e) + \min\{\delta[e'] \mid e' \in v^\bullet\}$
        end if
    end if
    if $l(v)$= AND and $|v^\bullet| > 1$ then
        if $\delta[e] > w(e) + \max\{\delta[e'] \mid e' \in v^\bullet\}$ then
            $\delta[e] \leftarrow w(e) + \max\{\delta[e'] \mid e' \in v^\bullet\}$
        end if
    end if
end function
```

For an example execution of the algorithm, consider the workflow graph in Fig. 4.4.

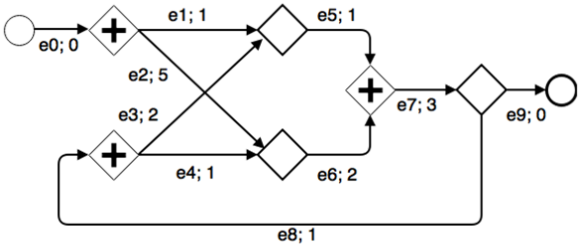


Figure 4.4: A cyclic workflow graph with edge weights

In Table 4.2, we have the initial values for $\delta[e]$ for each edge e of the workflow graph.

	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9
Initial	∞	∞	∞	∞	∞	∞	∞	∞	∞	0

Table 4.2: Initial values for $\delta[e]$.

After one iteration of the algorithm, considering the order of edge relaxations matches the order of the edges in Table 4.3, we have the values for $\delta[e]$ shown in Table 4.3.

	e_9	e_8	e_7	e_3	e_4	e_2	e_1	e_0	e_5	e_6
After the first iteration	0	∞	3	∞	∞	∞	∞	∞	4	5

Table 4.3: Values for $\delta[e]$ after one iteration of the algorithm.

After the second iteration, the values of $\delta[e]$ are given in Table 4.4.

	e_9	e_8	e_7	e_3	e_4	e_2	e_1	e_0	e_5	e_6
After the second iteration	0	∞	3	6	6	10	5	10	4	5

Table 4.4: Values for $\delta[e]$ after two iterations of the algorithm.

After the third iteration, the values of $\delta[e]$ are given in Table 4.5.

	e_9	e_8	e_7	e_3	e_4	e_2	e_1	e_0	e_5	e_6
After the third iteration	0	7	3	6	6	10	5	10	4	5

Table 4.5: Values for $\delta[e]$ after three iterations of the algorithm.

Notice that after this iteration all $\delta[e]$ have their final values (they won't change in potential subsequent iterations of the algorithm). Therefore, we have the minimum duration of an execution – the value associated to e_0, 10.

4.2.1.1 Correctness proof of Algorithm 4.1

In the following, we will show the correctness of the algorithm. For the correctness proof, we will again make use of the notion of the minimum cost that can be accumulated on an edge e: $d^*(e)$. Recall the definition of $d^*(e)$ which we gave in Eq.(3.1.1.1):

$$d^*(e) = \min\{d_\sigma(e) \mid \sigma \text{ is a fair execution that starts in } m_e\} \tag{4.1}$$

The notion of minimum cost accumulated on an edge e is necessary for the proofs, as we will demonstrate that the algorithm computes the minimum cost accumulated on the *source edge*.

Some key ideas behind the proof, that we will present in detail next are:

1. While relaxations always decrease the accumulated cost of an edge e, at any point of executing Algorithm 4.1, the accumulated cost on an edge e is always an upper bound for $d^*(e)$ – we will refer to this as the *upper bound property*, and,
2. There exists a property of a sequence of calls of $Relax(e, v)$ which guarantees that after performing it, $\delta[e]$ attains $d^*(e)$, and we will show that our algorithm satisfies this property.

Recall that since $d_\sigma(e_{source})$ represents the cost of a fair execution σ, $d^*(e_{source})$ represents the duration of the minimum duration execution.

Next we will provide a set of lemmas and proofs, similarly to how we proceeded in Chapter 3, that will contribute to the proof of the correctness of Algorithm 4.1. The lemmas that we will present next are similar to the lemmas we stated for proving Algorithm 3.1 but note that their interpretation and proofs are sometimes different as now we have a underlying different model – a workflow graph executed by an unbounded number of resources.

We will start by stating an important property of executions with minimum duration which holds for both workflow graphs executed by a single resource and workflow gaphs executed by an unbounded number of resources.

Lemma 4.1. *Some fair execution of* Γ *with minimum duration is loop-free.*

We had this lemma also in Chapter 3 when we considered workflow graphs executed by a single resource and accordingly, we had a different definition of the duration of an execution. However, technically, the proof of Lemma 4.1 is the same as the proof of Lemma 3.3.

This lemma is important because we will later use Γ_σ (Γ restriced to the nodes that are executed in σ and the edges of Γ such that $\sigma(e) = 1$) and the fact that for a loop-free execution σ, Γ_σ is acyclic – as stated in Chapter 3.

Next, we will state and prove the lemma that gives the upperbound property.

Lemma 4.2. *Let* e *be an edge and* v *a node such that* $e \in {}^\bullet v$*. We have at any point of executing Algorithm 4.1 that* $\delta[e] \geq d^*(e)$*.*

Proof. We prove the lemma by induction on k, the number of calls of $Relax(e, v)$.

Base case: $k = 0 : \delta[e_{sink}] = w(e_{sink})$, therefore clearly, $\delta[e_{sink}] = d^*(e_{sink})$, and for all $e \in E \setminus \{e_{sink}\}$, $\delta[e] = \infty$, and therefore $\delta[e] > d^*(e)$.

Induction step: Suppose that after the k-th call of $Relax(e, v)$ we have $\delta[e] \geq d^*(e)$ for all e. At the $(k + 1)$-th call of $Relax(e, v)$, only $\delta[e]$ may get updated.

We will show that from the definition of $d^*(e)$ and from the induction hypothesis, it follows that $\delta[e] \geq d^*(e)$, for each of the relaxation cases.

For the case $l(v) =$ XOR and $|v^\bullet| > 1$, the proof relies on Lemma 4.2.1, and for the case $l(v) =$ AND and $|v^\bullet| > 1$, the proof relies on Lemma 4.2.2. In the following, we state these necessary sub-lemmas (Lemma 4.2.1, Lemma 4.2.2) and their proofs and after that we will finalize the proof of Lemma 4.2.

Lemma 4.2.1. *Let* v *be a node such that* $l(v) =$ *XOR and* $v^\bullet > 1$*. Let* $\{e\} = {}^\bullet v$*. We have* $d^*(e) = w(e) + min\{d^*(f) \mid f \in v^\bullet\}$*.*

Proof. From the definition of $d_\sigma(e)$, we have: $d_\sigma(e) = w(e) + d_\sigma(next_\sigma(e))$ (such that this is the last update of $d_\sigma(e)$). Also, by definition, we have: $d^*(e) = min\{d_\sigma(e) \mid \sigma$ is a fair execution that starts in $m_e\}$. Therefore, $d^*(e) = min\{w(e) + d_\sigma(next_\sigma(e)) \mid \sigma$ is a fair execution that starts in $m_e\}$.

As $w(e)$ does not depend on σ, we have: $d^*(e) = w(e) + min\{d_\sigma(next_\sigma(e)) \mid \sigma$ is a fair execution that starts in $m_e\}$.

Note that σ is an execution like: $\langle e\ f \cdots \rangle$ where $f \in v^\bullet$. Let σ_f be the execution when $f = next_\sigma(e)$.

We can re-write the definition of $d^*(e)$:
$d^*(e) = w(e) + min_{f \in v^\bullet} min\{d_\sigma(f) \mid \sigma$ is a fair execution that starts in m_e $\}$.

Recall that σ started in m_e. Upon executing v, we obtain the marking m_f. Note that since m_e is an edge enabling marking, m_f is also an edge enabling marking. This can be easily verified by analyzing the different possibilities for $l(v')$ where $f \in {}^\bullet v'$. We obtain: $d^*(e) = w(e) + min_{f \in v^\bullet} min\{d_\sigma(f) \mid \sigma$ is a fair execution that starts in m_f $\}$. From this it follows that $d^*(e) = w(e) + min_{f \in v^\bullet} d^*(f)$. □

Lemma 4.2.2. *Let* v *be a node such that* $l(v) =$ *AND and* $v^\bullet > 1$*. Let* $\{e\} = {}^\bullet v$*. We have* $d^*(e) = w(e) + max\{d^*(f) \mid f \in v^\bullet\}$*.*

Proof. Our proof relies on some notions we would like to fix here:

- From Lemma 4.1 it follows we can choose $\sigma^* = argmin_\sigma\{d_\sigma(e) \mid \sigma$ is a fair execution that starts in $m_e\}$, such that σ^* is a loop-free execution.
- For a loop-free execution σ, and an unbounded number of resources, the duration of the execution is equal to the longest path in Γ_σ - the *critical path* [37].

I. We prove that $d^*(e) \geq w(e) + max_{f\in v^\bullet} d^*(f)$.

From the definition of $d_\sigma(e)$ we have: $d_\sigma(e) = w(e) + max\{d_\sigma(f) \mid f \in v^\bullet\}$. Also, by definition, we have: $d^*(e) = min\{d_\sigma(e) \mid \sigma$ is a fair execution that starts in $m_e\}$.

Therefore, $d^*(e) = min\{w(e) + max\{d_\sigma(f) \mid f \in v^\bullet\} \mid \sigma$ is a fair execution that starts in m_e $\}$.

$d^*(e) = w(e) + min\{max_{f\in v^\bullet} d_\sigma(f) \mid \sigma$ is a fair execution that starts in $m_e\}$.

We use the max-min inequality [131] which says that, for any function $f : Z \times W \to \mathbb{R}$ it holds that:

$$min_{w\in W} max_{z\in Z} f(z, w) \geq max_{z\in Z} min_{w\in W} f(z, w) \tag{4.2}$$

We obtain that:

$min\{max_{f\in v^\bullet} d_\sigma(f) \mid \sigma$ is a fair execution that starts in $m_e\} \geq max_{f\in v^\bullet} min\{d_\sigma(f) \mid$ σ is a fair execution that starts in $m_e\}$. Therefore,

$$d^*(e) \geq w(e) + max_{f\in v^\bullet} min\{d_\sigma(f) \mid \sigma \text{ is a fair execution that starts in } m_e\} \tag{4.3}$$

Note that since we first take the minimum in the right hand side of the inequality, in $min\{d_\sigma(f) \mid \sigma$ is a fair execution that starts in $m_e\}$, due to Lemma 4.1, we can restrict to loop-free executions (some execution that minimizes $d_\sigma(f)$ is loop-free).

Note that for any σ starting in m_e, there exists σ' starting in m_e such that $\sigma' = \langle m_e, \cdots, m_f, \cdots\rangle$, such that m_f is the edge enabling marking for $f \in v^\bullet$ and $\vec{\sigma} = \vec{\sigma'}$.

Let $f = (u_f, v_f)$. From Lemma 2.2, the edge enabling marking m_f can be reached without firing any transition in $Trans(v_f)$.

$\sigma' = \underbrace{\langle m_e, \cdots\rangle}_{\sigma^{Pref}} \underbrace{\langle m_f, \cdots\rangle}_{\sigma^{Suf}}$.

We have that $d_{\sigma'}(f) = d_{\sigma^{Suf}}(f)$ (i).

We also have that $d_{\sigma'}(f) = d_\sigma(f)$ (ii). The reason for this is the fact that from $\vec{\sigma} = \vec{\sigma'}$ and from the acyclicity of σ and σ' it follows that $\Gamma_\sigma = \Gamma_{\sigma'}$. It is clear that $d_\sigma(f) = d_{\sigma'}(f)$ both represent the longest path starting from f in the same acyclic workflow graph.

From (i) and (ii) it follows that $d_{\sigma^{Suf}}(f) = d_\sigma(f)$. Therefore, $min\{d_\sigma(f) \mid \sigma$ is a fair execution that starts in $m_e\}$ = $min\{d_\sigma(f) \mid \sigma$ is a fair execution that starts in $m_f\}$. Plugging this into Equation 4.3, we have that: $d^*(e) \geq w(e) + max_{f\in v^\bullet} min\{d_\sigma(f) \mid \sigma$ is a fair execution that starts in $m_f\}$, and therefore $d^*(e) \geq w(e) + max_{f\in v^\bullet} d^*(f)$.

II. We prove that $d^*(e) \leq w(e) + max_{f\in v^\bullet} d^*(f)$.

We construct an execution σ'' such that $max_{f\in v^\bullet}(d_{\sigma''}(f)) \leq max_{f\in v^\bullet} d^*(f)$.

Let $\Psi = argmin\{d_\sigma(f) \mid \sigma$ is a fair execution that starts in $m_f\}$.

Case 1: All $\sigma^*_f \in \Psi$ agree on all choices $\forall f \in v^\bullet$, hence σ^*_f is unique. Then σ'' is σ^*_f.

Case 2: In the opposite case, we need to construct σ''. Let $\sigma^*_f \in \Psi$. We will iteratively modify σ^*_f, $\forall f \in v^\bullet$ without increasing $d_{\sigma^*_f}(f)$ until all of them will become identical to common σ''.

Let v' be a choice for which σ^*_f don't agree. Let $\Gamma_{\sigma^*_f}$ be a workflow graph restricted to elements contained in σ^*_f. Let also $\pi(\Gamma_{\sigma^*_f}, v')$ be the longest path in $\Gamma_{\sigma^*_f}$, starting from vertex v'. By taking $f_{min} = argmin_f\{\pi(\Gamma_{\sigma^*_f}, v')\}$, to modify each σ^*_f it is enough for each of them to take $choice(\sigma^*_f, v') = choice(\sigma^*_{f_{min}}, v')$. Due to the choice of f_{min}, such modification will not increase $d_{\sigma^*_f}(f)$ and will eventually converge to common σ'' for all f.

From the way we constructed σ'', we have:

$$w(e) + max_{f\in v^\bullet} d^*(f) \geq w(e) + max_{f\in v^\bullet} d_{\sigma''}(f) \tag{4.4}$$

From the definition, we have:

$$w(e) + max_{f\in v^\bullet} d_{\sigma''}(f) = d_{\sigma''}(e) \tag{4.5}$$

We have:

$$d_{\sigma''}(e) \geq d^*(e) \tag{4.6}$$

From Eq. 4.4, Eq. 4.5 and Eq. 4.6 we have $d^*(e) \leq w(e) + max_{f\in v^\bullet} d^*(f)$. ☐

Now we can resume the proof of Lemma 4.2. We will present the reasoning for one of the relaxation cases, as the justification for the remaining ones is similar.

Let $l(v) =$ XOR and $|v^\bullet| > 1$. Let ${}^\bullet v = \{e\}$ Before the $(k+1)$-th relaxation step, it holds that $\delta[f] \geq d^*(f)\ \forall f \in v^\bullet$ (due to the induction hypothesis). After the $(k+1)$-th relaxation step, $\delta[e]$ gets updated, such that $\delta[e]$ becomes $w(e) + min\{\delta[f] \mid f \in v^\bullet\}$, as presented in Algorithm 4.1. Therefore, due to the induction hypothesis, $\delta[e] \geq w(e) + min\{d^*(f) \mid f \in v^\bullet\}$ (i).

From Lemma 4.2.1, we have that $d^*(e) = w(e) + min\{d^*(f) \mid f \in v^\bullet\}$ (ii).

From (i) and (ii), it follows that $\delta[e] \geq d^*(e)$. ☐

Note that at each relaxation step we can only decrease the value of $\delta[e]$. Once $\delta[e] = d^*(e)$, it doesn't change (it can not decrease further) as otherwise it would contradict the claim that $\delta[e] \geq d^*(e)$.

Now that we have proven the upper bound property we will state formally and prove the second key insight for demonstrating the correctness of Algorithm 4.1 – the sufficient condition for attaining $d^*(e)$.

Lemma 4.3. *Let e be an edge. Let σ be a fair execution such that $d_\sigma(e) = d^*(e)$. Let $S = \langle e_{i-1}, \cdots, e_{sink}\rangle$ be the sequence edges that get marked after e gets marked for the last time in σ. Each sequence of calls of $Relax(e, v)$ that has the property that edges $e_{sink}, \cdots, e_{i-1}, e$ have been relaxed in this order, after the sequence of calls to $Relax(e, v)$ we have $\delta[e] = d^*(e)$.*

The proof of Lemma 4.3 is identical to the proof of Lemma 3.2.

Next, we will instantiate the lemmas given above, namely Lemma 4.2 and Lemma 4.3 to conclude the proof of the correctness of Algorithm 4.1.

Lemma 4.4. *For a sound workflow graph, after running the Algorithm 4.1, it holds that* $\delta[e_{source}] = d^*(e_{source})$.

Proof. Lemma 4.1 states that some fair execution of Γ, with minimum duration, is loop-free (i). Recall that for a given fair execution σ, $d_\sigma(e_{source})$ represents the duration of execution of σ (ii). From (i) and (ii) it follows that some execution that minimizes $d_\sigma(e_{source})$ is loop-free (iii).

Note that m_s is the edge enabling marking for e_{source}.

Using (iii) and the definition for $d^*(e)$ instantiated to e_{source}, we obtain:

$d^*(e_{source}) = \min\{d_\sigma(e_{source}) \mid \sigma$ is a fair execution that starts in m_s $\}$. It follows that some σ^* for which $d_{\sigma^*}(e_{source}) = d^*(e_{source})$, is a fair, loop-free execution.

Since σ^* is loop-free, it means that at most $|V|$ nodes are executed in σ^*. In each complete relaxation step (one iteration of the loop in line 6 in Algorithm 4.1), we relax all the edges. Therefore, at the $|V|$-th iteration we have relaxed all the edges, in decreasing order with respect to the partial order on the edges of Γ_{σ^*} (recall that Γ_{σ^*} was defined in Chapter 3) . It means that at the $|V|$-th iteration, we will have relaxed all the edges that get marked after e gets marked in σ^*. Therefore, from Lemma 4.3, $\delta[e] = d^*(e)$. ☐

Therefore, we computed the duration of the minimum duration execution of the workflow graph, which is $d^*(e_{source})$.

Since Algorithm 4.1 and Algorithm 3.1 have the same complexity, we have proven the following:

Theorem 4.1. *The minimum duration execution time of a sound workflow graph with unbounded number of resources can be computed in time* $O(|V||E|)$.

4.2.2 Regular and acyclic workflow graphs

In the following, we briefly present the ideas for computing the maximum duration of execution for regular and acyclic workflow graphs.

As for the single resource case, for a regular workflow graph executed by an unbounded number of resources and with a structured cycle the computation of the maximum duration requires the specification of the maximal number of iterations for each loop. As discussed in Sect.3.1.3, if we assume that the backedge of each loop of the regular graph is annotated with a positive integer k that represents the maximum number of times the backedge can be traversed, then the maximum duration of a structured cycle is $(k+1)\cdot d_X + k\cdot d_Y$ where d_X denotes the maximum duration of the loop body X, and d_Y represents the maximal duration associated to reentering the loop. For computing the minimum duration we take $k = 0$ and the minimum duration of the loop body. We still obtain the minimum/maximum duration of such an annotated regular workflow graph in linear time (Cell C.1, C.2 of Table 4.1).

For acyclic workflow graphs, we can exploit the fact that its elements are in a partial order defined by the flow of the graph as we did in the previous chapter. To compute the minimum duration of the workflow graph we only need to sort topologically its elements and subsequently apply the relaxation procedure on edges in descending order with respect to this sorting.

Algorithm 4.3 Min duration, acyclic

```
function AcyclicWFGMin($\Gamma = \{V, E, c, w\}$)
    for $e \in E \setminus \{e_{sink}\}$ do
        $\delta[e] \leftarrow \infty$
    end for
    $\delta[e_{sink}] \leftarrow w(e_{sink})$
    TopologicalSort($\Gamma$)
    while $V \neq \emptyset$ do
        Select $v \in V$ s.t. $v$ is maximal with respect to the topological sort
        $V \leftarrow \{V \setminus v\}$
        for all $e \in {}^{\bullet}v$ do
            Relax(e,v)
        end for
    end while
end function
```

The algorithm that formalizes this idea is Algorithm 4.3, which similarly to Algorithm 3.3 has complexity $O(|V| + |E|)$. Therefore, it holds that:

Theorem 4.2. *The minimum duration execution of a sound acyclic workflow graph with unbounded number of resources can be computed in time* $O(|V| + |E|)$.

Note that, in the acyclic case, for computing the maximum duration execution, one only needs to select the maximum instead of the minimum in the $Relax(e, v)$ procedure when $l(v) =$ XOR and $|v^{\bullet}| > 1$.

4.3 Workflow graphs with probabilistic choice

For workflow graphs with probabilistc choice, computing the probability of a deadline transgression is NP-hard and this carries over directly from our previous result presented in Chapter 3, Sect.3.2. The reason is that for proving Theorem 3.4 we are using a class of workflow graphs which are sequential and therefore the time needed for executing the workflow graph when a single resource is available is the same as when an unbounded number of resources is available.

4.3.1 Expected duration

In the following we will prove that computing the expected duration of a workflow graph executed by an unbounded number of resources is NP-hard in contrast to the single resource case where the expected duration can be computed in polynomial time (see Chapter 3, Sect.3.2.2).

Theorem 4.3. *Given a regular, acyclic probabilistic workflow graph* Γ*, computing the expected duration of* Γ *executed by an unbounded set of resources is NP-hard.*

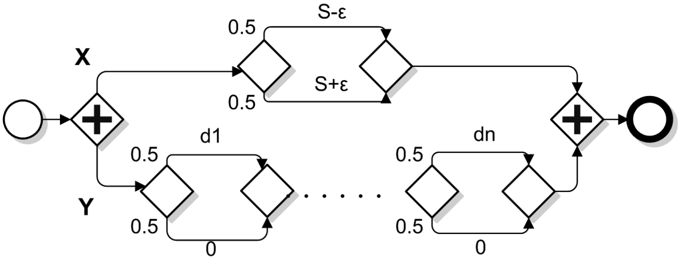


Figure 4.5: A probabilistic workflow graph

The proof consists of a reduction from the *subset sum problem*. Recall that the statement of the subset sum problem is: given a set $D = \{d_1, \cdots, d_n\}$ of integers and an integer S, to determine whether any non-empty subset $D' \subseteq D$ sums up to exactly S. Note that this is equivalent to solving a problem where all the values $d_1, \cdots, d_n, S$ are multiples of 4 (this statement will be used in the proof of Theorem 4.3 below). For the proof, we use the class of (regular, acyclic) probabilistic workflow graphs $\Gamma_{\epsilon,n}$ in Fig. 4.5, where each decision outcome has probability 0.5.

Proof. Let X, Y be random variables that denote the duration of each of the two parallel flows of $\Gamma_{\epsilon,n}$. The expected duration of the workflow graph $\Gamma_{\epsilon,n}$ is:

$$\mathbb{E}(\Gamma_{\epsilon,n}) = \mathbb{E}(max(X,Y))$$

$$\mathbb{E}(\Gamma_{\epsilon,n}) = \tfrac{1}{2}\mathbb{E}(max(S-\epsilon,Y)) + \tfrac{1}{2}\mathbb{E}(max(S+\epsilon,Y))$$

Let f be the probability distribution of Y. We rewrite the terms of $\mathbb{E}(\Gamma_{\epsilon,n})$ as follows:

$$\mathbb{E}(max(Y,S-\epsilon)) = (S-\epsilon)\Pr(Y \leq S-\epsilon) + \sum_{y>S-\epsilon} yf(y) \tag{4.7}$$

$$\mathbb{E}(max(Y,S+\epsilon)) = (S+\epsilon)\Pr(Y \leq S+\epsilon) + \sum_{y>S+\epsilon} yf(y) \tag{4.8}$$

By using equations (4.7), (4.8) we obtain the following expression for $\mathbb{E}(\Gamma_{\epsilon,n})$:

$$\mathbb{E}(\Gamma_{\epsilon,n}) = \tfrac{1}{2}\Big[(S+\epsilon)\Pr(Y \leq S+\epsilon) + \sum_{y>S+\epsilon} yf(y) + (S-\epsilon)\Pr(Y \leq S-\epsilon) + \sum_{y>S-\epsilon} yf(y)\Big].$$

Let us choose $\epsilon > 0$ such that no subset of $\{d_1, \cdots, d_n\}$ has sum in $[S-\epsilon, S)$ nor in $(S, S+\epsilon]$. Note that the sum, can still potentially equal exactly S. Such ϵ is easy to find. It is enough to choose $\epsilon = 2$ as all the numbers $d1, \cdots, d_n, S$ are multiples of 4.

We will show that $\mathbb{E}(\Gamma_{\epsilon,n}) = \mathbb{E}(\Gamma_{\frac{\epsilon}{2}})$ if there is no non-empty subset of $\{d_1, \cdots, d_n\}$ that sums up to exactly S (i), and $\mathbb{E}(\Gamma_{\epsilon,n}) \neq \mathbb{E}(\Gamma_{\frac{\epsilon}{2}})$ otherwise (ii). If we can compute the expected duration of a workflow graph with unbounded resources in polynomial time, we can solve the subset sum problem in polynomial time. Note that both ϵ and $\epsilon/2$ are integers, so we are always considering workflow graphs with integer weights.

(i) There is no non-empty subset of $\{d_1, \cdots, d_n\}$ that sums up to exactly S

.

In this case, it holds that $\Pr(Y \le S - \epsilon) = \Pr(Y \le S + \epsilon)$. Therefore we update the equation for $\mathbb{E}(\Gamma_{\epsilon,n})$):

$$\mathbb{E}(\Gamma_{\epsilon,n}) = \frac{1}{2}\Big[\Pr(Y \le S + \epsilon)(S + \epsilon + S - \epsilon) + \sum_{y>S-\epsilon} yf(y) + \sum_{y>S+\epsilon} yf(y)\Big].$$

$\mathbb{E}(\Gamma_{\epsilon,n}) = \frac{1}{2}\Big[2S\Pr(Y \le S + \epsilon) + \sum_{y>S-\epsilon} yf(y) + \sum_{y>S+\epsilon} yf(y)\Big]$. One can easily observe that $\mathbb{E}(\Gamma_{\epsilon,n}) = \mathbb{E}(\Gamma_{\frac{\epsilon}{2}})$.

(ii) There exists a non-empty subset of $\{d_1, \cdots, d_n\}$ that sums up to exactly S

.

In this case, $\Pr(Y \le S - \epsilon) \ne \Pr(Y \le S + \epsilon)$. Therefore,

$$\mathbb{E}(\Gamma_{\epsilon,n}) = \frac{1}{2}\Big[(S + \epsilon)\Pr(Y \le S + \epsilon) + (S - \epsilon)\Pr(Y \le S - \epsilon) + \sum_{y>S-\epsilon} yf(y) + \sum_{y>S+\epsilon} yf(y)\Big].$$

$$\mathbb{E}(\Gamma_{\epsilon,n}) = \frac{1}{2}\Big[(S + \epsilon)(\Pr(Y \le S - \epsilon) + \Pr(Y = S)) + (S - \epsilon)\Pr(Y \le S - \epsilon) + \sum_{y>S-\epsilon} yf(y) + \sum_{y>S+\epsilon} yf(y)\Big].$$

$$\mathbb{E}(\Gamma_{\epsilon,n}) = \frac{1}{2}\Big[\underbrace{2S\Pr(Y \le S - \epsilon)}_{T_1} + \underbrace{(S + \epsilon)\Pr(Y = S)}_{T_2} + \underbrace{\sum_{y>S-\epsilon} yf(y) + \sum_{y>S+\epsilon} yf(y)}_{T_3}\Big].$$

Please note that term T_2 has different value for $\mathbb{E}(\Gamma_{\epsilon,n})$ and $\mathbb{E}(\Gamma_{\frac{\epsilon}{2}})$, while T_1 and T_3 have the same value for $\mathbb{E}(\Gamma_{\epsilon,n})$ and $\mathbb{E}(\Gamma_{\frac{\epsilon}{2}})$. Therefore, $\mathbb{E}(\Gamma_{\epsilon,n}) \ne \mathbb{E}(\Gamma_{\frac{\epsilon}{2}})$. □

4.4 Minimum number of resources

In this section, we compute the maximum *degree of concurrency* of Γ, i.e., the maximum number of tokens that can exist in the graph in a reachable marking. Knowing the maximum number of tokens that can exist in the graph can help in answering a natural question that arises in the quantitative timing analysis of a business process: What is the minimum number k^* of resources one needs, such that each execution achieves its minimal execution time? Note that increasing the number of resources makes each execution faster up to a point. This means that from a certain point there does not exist any execution for which the duration could be decreased by having more than k^* resources. The maximum number

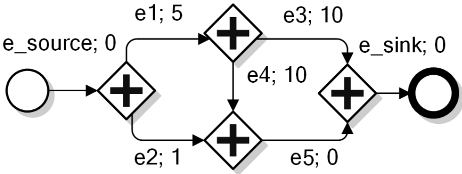


Figure 4.6: Tighter bound example

of tokens that can exist in the graph is an upper bound for k^* as we presented in Sect.4.1. There are cases where a tighter bound exists, as illustrated in Figure 4.6 where the maximum number of tokens is 3, obtained in the marking that marks edges e_2, e_3 and e_4 but 2 resources would suffice for reaching the minimum duration, viz. 15. Note that two resources could execute in parallel the tasks corresponding to edges e_1 and e_2 and after 5 time units, the two resources can execute the tasks corresponding to edges e_3 and e_4 which take other 10 time units. After 15 time units all the tasks of the workflow graph will have beeen completed. Conversely, for each max concurrency k there are weights such that $k = k^*$

4.4.1 An algorithm to compute the maximum degree of concurrency of a workflow graph

There is an EXPTIME algorithm for computing the maximum degree of concurrency for general workflow graphs. It is based on computing the reachability graph of Γ, which is the transition relation $\rightarrow$ restricted to its reachable markings. Note that for sound workflow graphs the reachability graph is finite, but exponential in the size of Γ. Each reachable marking is visited to compute the maximum concurrency degree.

However, efficient algorithms for computing the maximum degree of concurrency are known for subclasses such as marked graphs (see Def.7), regular or sequential workflow graphs (state-machines). Therefore, we propose to leverage this fact and tackle the problem through a *divide and conquer* strategy. In the following we will present our algorithm for computing the maximum degree of concurrency, we will briefly jsutify its correctness and illustrate the steps of the algorithm through an example.

The main idea behind divide and conquer solutions is to use the following three steps:

1. *Divide* - Decompose the problem into subproblems of the same type.
2. *Conquer* - Solve the newly devised subproblems recursively.
3. *Combine* - Combine the answers obtained in step (2).

This approach, which we will describe shortly for our problem, has the potential of speeding up the computation of the maximum degree of concurrency of a workflow graph in practice because it permits us to use the fact that certain subproblems we know how to solve in polynomial time (e.g., computing the maximum degree of concurrency for marked graphs).

In order to divide the problem into smaller parts, we compute the Refined Process Structure Tree (RPST) [86] of the workflow graph. The RPST represents a decomposition of a workflow graph into a hierarchy of sub-workflows that are subgraphs with a single entry and a single exit of control called *fragments* (see e.g., Figure 4.8 (a)). In [86], the authors provide a linear time algorithm for computing the RPST. The decomposition results in a parse tree which reflects the containment relationship of the fragments, see for example the workflow graph in 4.7 and its decomposition in fragments in 4.8.

The algorithm for computing the maximum degree of concurrency works as follows:

1. *Divide* the problem of computing the maximum degree of concurrency to subproblems by decomposing the workflow graph into its fragments. These fragments are labeled with their corresponding subclass (e.g., marked graph, state-machine, etc.).

2. *Conquer* the problem by computing the maximum degree of concurrency of the workflow graph based on the maximum degree of concurrency computed for its fragments. Note that we omit here trivial fragments consisting of a single edge.

3. *Combine* the results obtained in step (2) by replacing fragment for which we computed the maximum degree of concurrency with an edge whose weight equals the maximum degree of concurrency of that fragment.

The complexity of the algorithm depends on the subclass of the fragment f, as follows. For a single edge, the degree of concurrency is given by the weight of the edge. The algorithm runs in linear time for state-machines, where the returned value is the maximum weight of an edge of this fragment. Similarly it runs in linear time for regular fragments. For regular fragments modeling concurrency (cf. Figure 2.c) the maximum degree of concurrency is the *sum* of the weights of the edges. For regular fragments modeling choice (cf. Figures 2.b, d, e), the maximum degree of concurrency is the *maximum* of the weights of the edges. The computation of the concurrency degree runs in polynomial time for marked graphs and in exponential time for the *complex fragments* – the fragments which are not regular nor marked-graphs nor state-machines.

In the following we will present some remarks with respect to the correctness of the algorithm. Note that the algorithm finishes, as the RPST is a tree and we solve the problem starting from the leafs and we finish when we reach the root. Also note that when we iterate, the reduced graph – the one obtained following the *combine* step – has the same degree of concurrency as the original. The reason is that due to the fact that the fragments are with a single entry and a single exit of control, replacing a fragment with an edge whose weight equals the maximum degree of concurreny of the corresponding fragment, does not change the maximum degree of concurrency of the reduced graph.

An example for how our algorithm works is provided in Figure 4.8 where we show the docomposition of the workflow graph in Figure 4.7. In Figure 4.8, edge weights represent the concurrency degree. We decompose the original workflow graph into four fragments: *Sequence Fragment 2*, *Regular Fragment*, *Sequence Fragment 1*, and *Marked Graph*. The root fragment is *Sequence Fragment 2*, each node has exactly one child, and the tree has one leaf – the *Marked Graph Fragment*, Figure 4.8 (a). After computing the concurrency degree of the marked graph (value 3) the workflow graph is updated as shown in Figure 4.8 (b) where the *Marked Graph Fragment* has been replaced with an edge whose weight equals the maximum degree of concurrency of that fragment. In the next iteration, we compute the maximum degree of concurrency of the new leaf - the *Sequence Fragment 1*, and we obtain value 3. Next, the workflow graph is reduced to the graph composed of a regular fragment contained in a sequence fragment, as shown in Figure 4.8 (c). The concurrency degree of the regular fragment is computed (we obtain 3+1=4), we update the workflow graph and we are left with a sequence fragment Figure 4.8 (d). The maximum degree of concurrency of the workflow graph is the concurrency degree of this fragment (4).

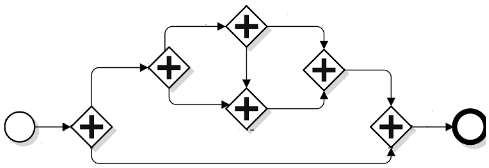
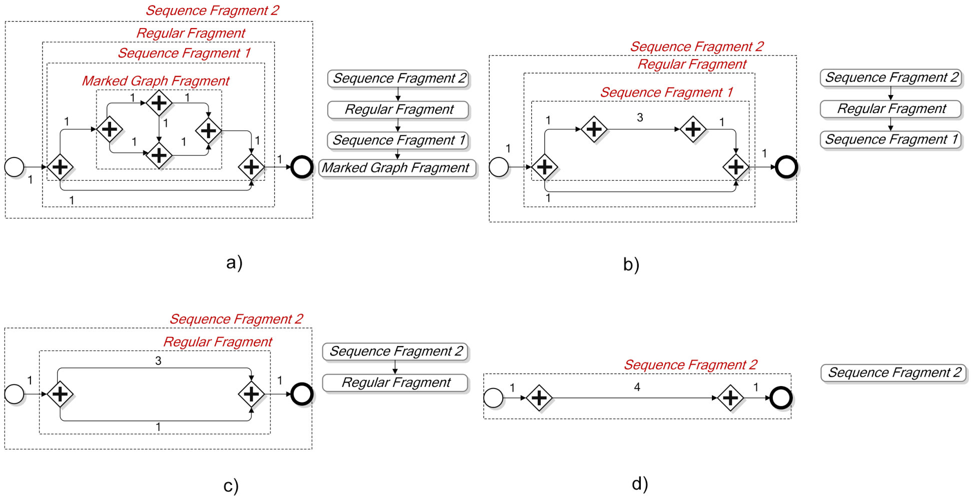


Figure 4.7: An example of a workflow graph for which we want to compute the maximum degree of concurrency



a)

b)

c)

d)

Figure 4.8: The decomposition for the workflow graph in Figure 4.7 into its fragments and its corresponding RPST (a), the workflow graph and RPST after computing the concurrency degree for the *Marked Graph Fragment* (b) the e workflow graph and RPST after computing the concurrency degree for *Sequence Fragment 1* (c) the workflow graph and RPST before the algorithm ends (d)

In the following, we present the approaches for computing the maximum degree of concurrency for marked graphs and for complex fragments.

Let $\vec{w}$ denote a $|E| \times 1$ column vector representing the degree of concurrency associated with each edge of Γ. Finding the maximum degree of concurrency of a marked graph Γ, $deg(\Gamma)$, can be formulated as:

$$deg(\Gamma) = max\{m \cdot \vec{w} \mid m \text{ is a marking reachable from } m_0\} \tag{4.9}$$

The solution we propose for computing $\deg(\Gamma)$ in a marked graph is identical to the computation of the maximum weighted sum of tokens in [140]. In [140] the author formulates this problem as an integer programming (IP) problem with integer data and totally unimodular constraint matrix. Note that any IP problem with with integer data and totally unimodular constraint matrix is solvable in polynomial time.

The worst case complexity of the algorithm is therefore dominated by complex fragments, for which we resort to the EXPTIME algorithm. Only a fragment which is not from a category we can efficiently solve becomes subject to state space exploration. For all the workflow graphs which don't have any complex fragments our algorithm runs in polynomial time. Since complex fragments are rare in practice, this approach has the potential of speeding up the time it takes to compute the maximum degree of concurrency. In a previous study documented in [44] on 735 industrial business process which were translated to workflow graphs, only about 4% of the total of their corresponding fragments were complex with an average number of edges between 29 and 33. Also in [44] the authors noted in their study that it seems difficult to design a sound process with complex fragments as 87% of all complex fragments identified were unsound, 12% of the complex fragments were cyclic, but less than 3% of them were sound.

We have attempted solving this problem in both directions – finding a polynomial time algorithm for computing the maximum degree of concurrency or proving it is NP-hard to compute it but we have not reached a conclusion yet. We have proven through a reduction from MAX-SAT problem [126] that it is NP-hard to compute the maximum degree of concurrency for sound workflow nets that are not necessarily free-choice.

4.5 Workflow graphs executed by a fixed number of resources

The questions we asked in Chapters 3 and 4 can be asked in settings when a fixed number $n > 1$ of resources is available for executing the workflow graph. This constraint makes it hard to find efficient solutions to the questions we were trying to solve. The probability of deadline transgression and the expected duration remain NP-hard and this follows from our justifications in the current work. More specifically to prove that the probability of a deadline transgression to occur in a workflow graph executed by a fixed number of resources is NP-hard, we can use the same reduction we proposed for the single resource case. The same holds for computing the expected duration where we can use the reduction for the same class of graphs as we used in Section 4.3.1.

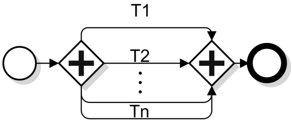


Figure 4.9: Regular workflow graph with *n* parallel threads

For the maximum duration – the worst case execution time is attained when we require all the tasks to be executed by a single resource, which we have studied in Chapter 3. What is different, is the fact that computing the minimum duration of execution becomes NP-hard for a fixed number $n > 1$ of resources. For example, for a simple workflow graph as the one in Figure 4.9, let's assume we need to complete n tasks $T_1, \cdots, T_n$ and we have k identical agents to solve them. Finding an assignment of the tasks to the agents such that the duration of execution (*makespan*) is minimized is NP-hard as one can reduce 2-PARTITION to finding the minimum duration when there are exactly two resources available [111].

4.6 Case Study

In the following, we present a case study where we employ the algorithms presented in this and the previous chapter on real data.The data we use [29] originates from a Dutch financial institution, and it consists of events generated along a loan approval process.

This data is suitable for various types of analyses. We commence by performing a descriptive analysis where we mine the underlying business process. This enables us to visualize the process and understand interdependencies between tasks. The latter is particularly important for us, as capturing the control flow is a mandatory prerequisite for performing the timing analysis.

This process is translated to its equivalent workflow graph representation, which we provide in Fig. 4.11, Fig. 4.12, Fig. 4.13, Fig. 4.14, Fig. 4.15.

The original mined process is a cyclic process consisting of 64 tasks and 115 edges. The translated workflow graph is a cyclic workflow graph with 64 tasks, 94 gatways and 141 edges.

Next, we mine the average duration of execution of different tasks in the process, as well as the probability of transitioning from a task to another. Finally, we run some of our algorithms for deadline analysis, and we report two aspects from the result of running them. The first result we report is the actual output of the algorithms, i.e., the expected time to process an application, or the minimum duration of an execution. The second result we report is the running time of our algorithms on this process.

4.6.1 Mining the loan application process

Process mining allows us to discover the different steps taken in the loan application proces and their ordering. A process model was obtained by importing the logs in ProM and running the fuzzy miner. To unclutter the process and have a clearer picture, the *best edges* filter was applied which keeps the best incoming and outgoing edge for each activity [10]. This process was translated to a workflow graph such that we have a process representation compatible with our algorithm. This transformation was made in accordance with our restrictions presented in Chapter 2 (free choice workflow graph, we don't allow any node that has multiple incoming edges as well as multiple outgoing edges, etc.).

4.6.2 Data and data processing

The data consists of all the loan applications filed trough an online system. The logs consist of 262200 events in 13087 cases collected along 6 months. The events are labeled with the state in their life-cycle, namely: "schedule", "start", "resume", "abort" and "complete".

Event name	Event duration (seconds)
W_Handle leads	219
W_Assess potential fraud	2529
W_Complete application	249
W_Call incomplete files	254
W_Call after offer	94383
W_Personal loan collection	37
W_Shortened completion	1655
W_Validate application	646

Table 4.6: Examples of durations for the resume state of the tasks.

The data contained in each event is the event id, the step in the approval process, whether it is a start/complete or schedule event, the resource id, the time-stamp and the case identifier.

The way we obtained the duration of an event was by subtracting the timestamp of the event from the timestamp of the next event – where the two events occur in the same case. For example, for the "resume" work item events we have the durations shown in Table 4.6:

After obtaining these durations, we notice that the most time consuming events are W_Call after offer and W_Personal loan collection which in the suspend state (when perhaps some user interaction is involved) they take approximately 5 and 7 days respectively.

As one can imagine, to know the appropriate rate/probability to be associated to each event of the stochastic model of a business process may not be a trivial task. In some cases, the rates/probabilities can be defined by a business specialist, based in his/her knowledge of the domain. In other cases, when the real system is already implemented and in use, these values can be inspired by the times/frequencies observed in the real system

Computing the appropriate transition probabilities might be a non-trivial task. In some studies we have seen authors assuming uniform distribution regarding possible process execution flows at conditional routing [132]. In some cases, the transition probabilities are already provided by the business specialist based on his prior knowledge. In other cases when process execution data is available, these values can be inspired by the times/frequencies observed [99].

To compute the transition probabilities we also rely on the observed frequencies with which events occur. Therefore, we devise a squared matrix $M_{n,n}$, where n is the number of events. $M[i,j]$ represents the number of time event i was succeeded by event j divided by total number of occurrences of event i. Some examples of the obtained probabilities are provided in the Table 4.7:

At this point, we have all the prerequisites necessary for running the algorithms for computing the minimum duration and the expected duration of an execution. In the following, we present the results obtained from running them.

After running our algorithms we obtained results for minimum duration of an execution and expected duration and the running times of the respective algorithms. As shown in Table 4.8, the minimum duration of an execution is of 248125 seconds, which means approximatey 3 days. The expected duration of an execution is of 41 days. The runtime of the algorithms on this process are of 40ms for computing the minimum duration of an

Source event name	Target event name	Probability
O_Created:complete	O_Sent (online only)	0.04
O_Created:complete	O_Sent (mail and online)	0.82
A_Concept:complete	A_Accepted:complete	0.29
A_Concept:complete	W_Complete application:start	0.47
W_Call after offers:suspend	A_Cancelled:complete	0.12

Table 4.7: Examples of computed probabilities for the source event to be followed by the target event.

	Value	Runtime
Minimum duration of an execution	270722s	40ms
Exepected duration	3570240s	9ms

Table 4.8: Minimum duration, expected duration and the running times.

execution and 9ms for computing the expected duration of an execution.

We have adapted the algorithm to compute the minimum duration of an execution to also trace back the path that achieves the minimum duration. Therefore, the minimum duration of an execution corresponds to the following sequence of activities: A_Create application:complete, A_Submitted:complete, W_Handle leads:schedule, W_Handle leads:withdraw, W_Complete application:schedule, W_Complete application:start, O_Create offer:complete, O_Created:complete, O_Sent (mail and online) complete, W_Complete application:complete, W_Call after offers:schedule, W_Call after offer:withdraw, W_Validate application:schedule, W_Validate application:start, O_Received:complete, W_Validate application:suspend, O_Accepted complete, A_Pending:complete, W_Validate application:complete.

We also compute the distribution of the durations of process runs (traces).

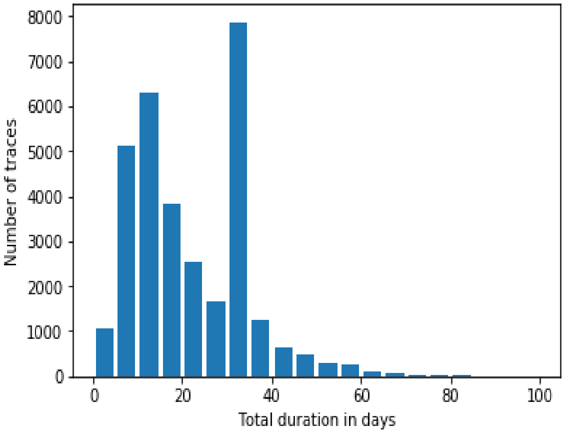


Figure 4.10: Histogram of durations of process runs.

As seen in Fig. 4.13, indeed the minimum duration of an execution is about 5 days and the expected duration is about 35 days.

	Value	Runtime
Minimum duration of an execution	854478s	37ms
Exepected duration	4291460s	9ms

Table 4.9: Minimum duration, expected duration and the running times on the process with removed edges

After looking more closely at the execution path corresponding to the minimum duration – as outputed by our algorithm – we notice that it goes through several withdrawn tasks, e.g., W_Handle leads:withdraw, W_Call after offer:withdraw which correlated to the histogram in Fig. 4.13, makes us suspect that this execution might be rather an exception. To analyze an execution that contains less of these withdrawals, we remove the path between V_0 and V_{77}, V_{43} and V_{51}, V_{44} and V_{51} - which basically exclude the possibility of going through the withdraw state of the aforementioned tasks.

The new path we obtain is: A_Create application:complete, A_Submitted:complete, W_Handle leads:schedule, W_Handle leads:start, W_Handle leads:complete, W_Complete application:schedule, W_Complete application:start, O_Create offer:complete, O_created:complete, O_Sent (mail and online):complete, W_Complete application:complete, W_Call after offers:schedule, W_Call after offers:start, W_Call after offers:suspend, W_Call after offers:resume, W_Validate application:schedule, W_Validate application:start, O_Received:complete, W_Validate application:suspend, O_Accepted complete, A_Pending:complete, W_Validate application:complete.

The minimum duration is of approximately 10 days and the expected duration of an execution is of approximately 49 days, cf. Table 4.9. The runtime of the algorithms has not changed significantly.

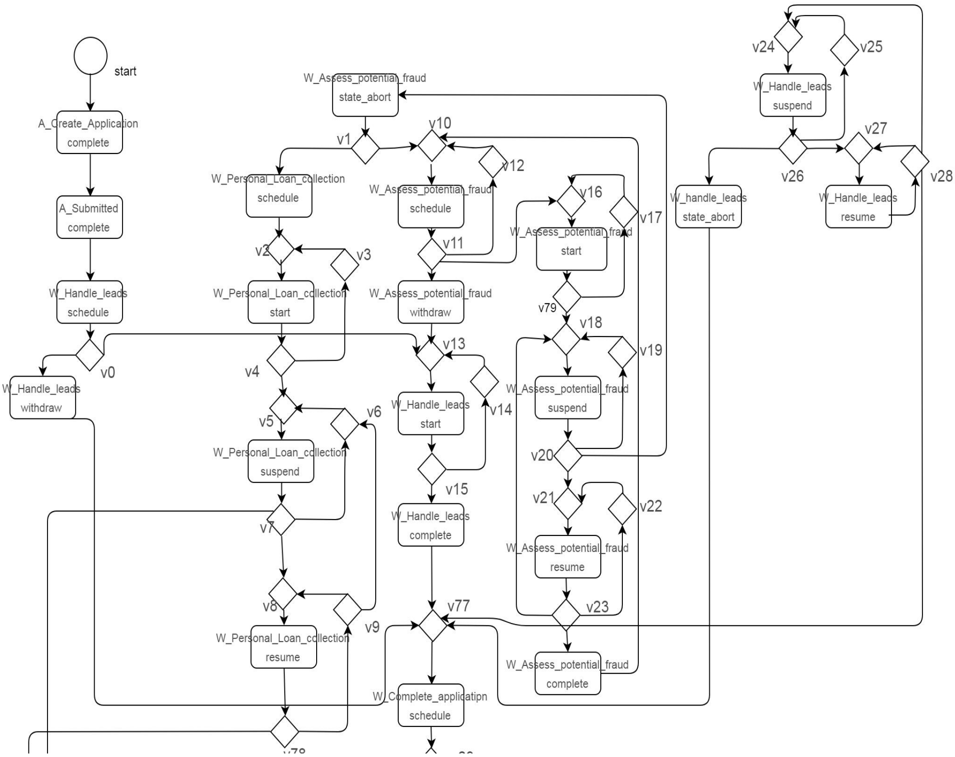


Figure 4.11: Workflow graph for the loan application process (part 1).

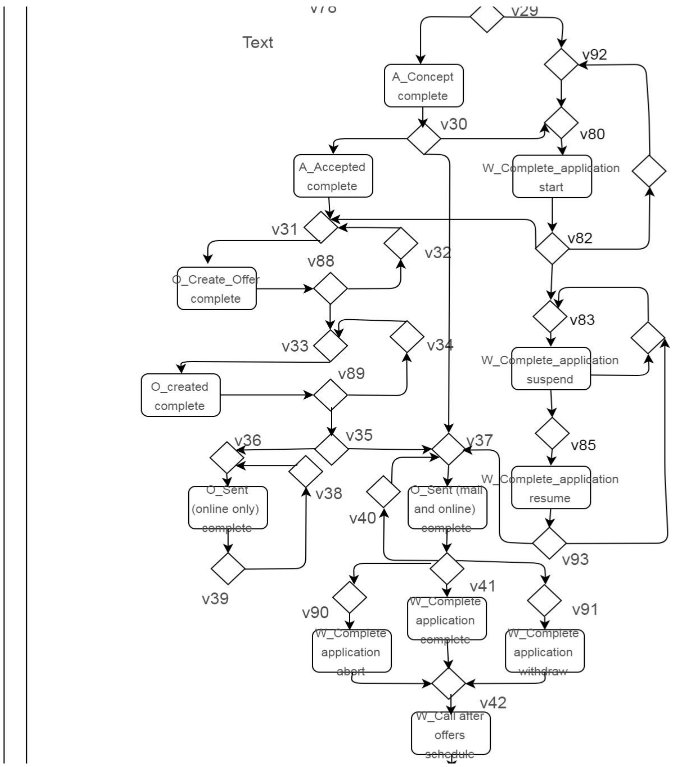

v78
Text
v29
v92
A_Concept complete
v30
v80
W_Complete_application start
v81
A_Accepted complete
v31
v82
v32
v88
O_Create_Offer complete
v83
v84
v33
v34
W_Complete_application suspend
O_created complete
v89
v35
v36
v37
v85
W_Complete_application resume
O_Sent (online only) complete
v38
O_Sent (mail and online) complete
v40
v93
v39
v41
v90
v91
W_Complete application complete
W_Complete application abort
W_Complete application withdraw
v42
W_Call after offers schedule

Figure 4.12: Workflow graph for the loan application process (part 2).

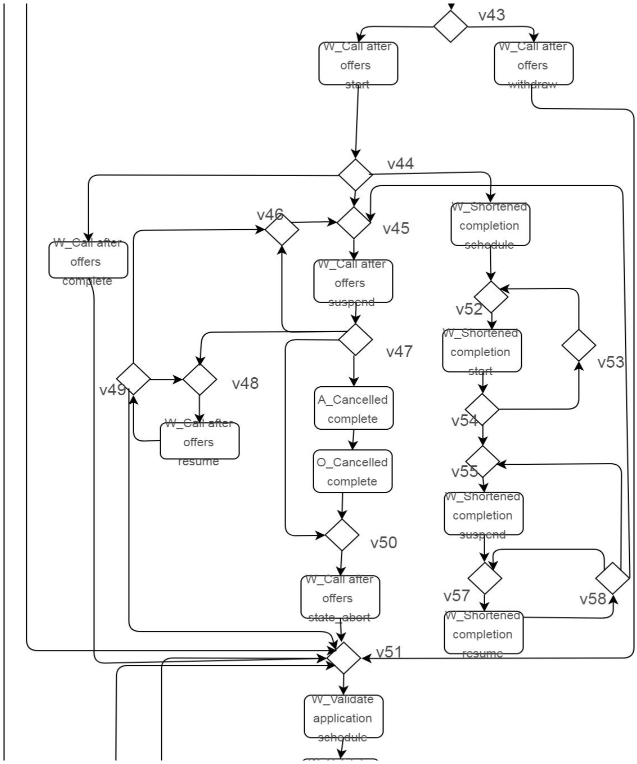

v43
W_Call after offers start
W_Call after offers withdraw
v44
v46
v45
W_Call after offers complete
W_Shortened completion schedule
W_Call after offers suspend
v52
W_Shortened completion start
v53
v47
v49
v48
A_Cancelled complete
W_Call after offers resume
v54
O_Cancelled complete
v55
W_Shortened completion suspend
v50
W_Call after offers state_abort
v57
v58
W_Shortened completion resume
v51
W_Validate application schedule

Figure 4.13: Workflow graph for the loan application process (part 3).

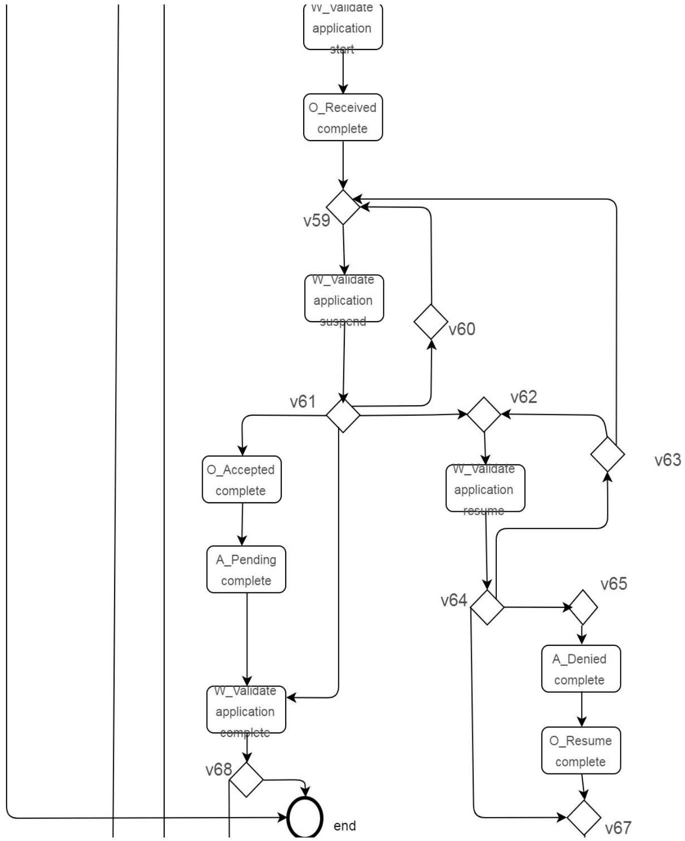


Figure 4.14: Workflow graph for the loan application process (part 4).

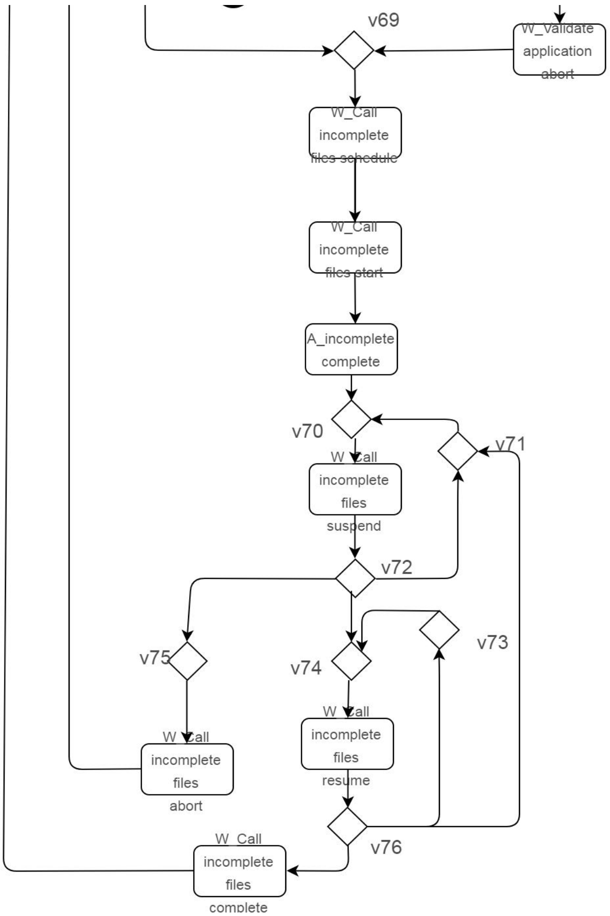

v69
W_Validate application abort
W_Call incomplete files schedule
W_Call incomplete files start
A_incomplete complete
v70
v71
W_Call incomplete files suspend
v72
v73
v75
v74
W_Call incomplete files abort
W_Call incomplete files resume
v76
W_Call incomplete files complete

Figure 4.15: Workflow graph for the loan application process (part 5).

4.7 Conclusion

We presented new results on the deadline analysis of workflow graphs with an unbounded number of resources.

We have shown that computing the minimum duration of an execution of a workflow graph executed by an unbounded number of resources can be obtained in polynomial time. In contrast to the single resource case however, computing the expected duration is NP-hard. We have also discussed the case when a fixed number of resources is given where we have shown that all the problems we studied become NP-hard.

Since the timing perspective and the resource perspective are tightly connected, we were also interested in the number of resources required to attain the minimum duration of an execution. In this regard, we proposed an algorithm for computing the maximum number of tokens that can exist in the graph which has the potential to run fast in practice (as it only resorts to state space exploration for complex fragments).

We have also performed a case study, on data obtained from a Dutch financial institution. In this case study we mined the underlying business process, then translated it to the workflow graph representation, we computed the event durations and transition probabilities and finally employed our algorithms on this data. We concluded that our algorithms are feasible to use in practice and can provide additional insights on the process.

Part II

Workflow Performance Optimization

Chapter 5

Data-informed Work Assignment in Incident Ticket Resolution

In this chapter, we present a novel technique that optimizes the dispatching of incident tickets to the agents in an IT Service Support Environment. Unlike the common skill-based dispatching, our approach also takes empirical evidence on the agent's efficiency from historical data into account.

Our solution consists of two parts. First, a novel technique clusters historic tickets into incident categories that are discriminative in terms of an agent's performance. Second, a dispatching policy selects, for an incoming ticket, the fastest available agent according to the target cluster. We show that, for ticket data collected from several Service Delivery Units, our new dispatching technique can reduce service time between 35% and 44%.

5.1 Problem context

Enterprises and IT service providers are constantly striving to improve the quality of service and in the same time maintain or reduce the cost of service delivery. One of the major challenges in service delivery is resolving the incidents as efficiently as possible. This is an important but labor intensive task, assigned to the service agents that are responsible for solving the customer's incident reports.

In our setting, incidents in the customer environment are submitted to the IT Service Provider in the form of a ticket which is a snippet of text describing a particular IT problem. The reported problems range widely from authentication errors, application crashes, to broken transactions or server unavailabilities.

The dispatching to the service agent is generally done on two levels. First, a human dispatcher reviews the problem description text and decides which delivery unit (a team of service agents) is responsible for addressing that type of ticket. Second, within the delivery unit, a service agent is either chosen by the group leader or on a voluntary basis. The dispatching at this level is based on the qualification of the agents, the availability and the workload of the agents, as well as the *complexity of the ticket* (a label assigned to the ticket reflecting the difficulty level of the problem documented in the ticket). At this level we have identified an opportunity for automation and optimization of the dispatching process.

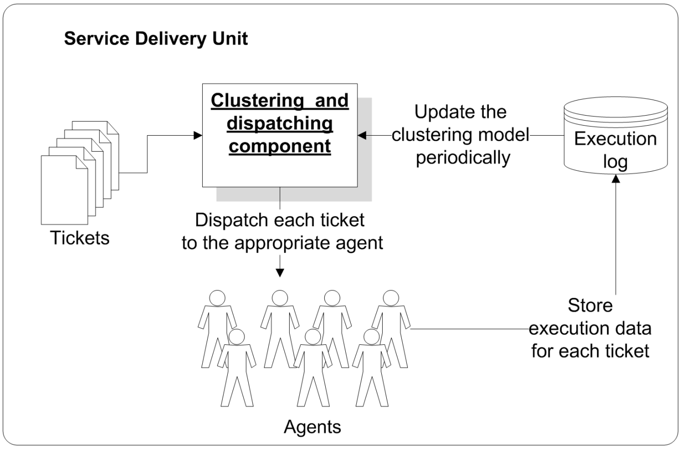

Service Delivery Unit
Clustering and dispatching component
Update the clustering model periodically
Execution log
Tickets
Dispatch each ticket to the appropriate agent
Store execution data for each ticket
Agents

Figure 5.1: Integration of the clustering and dispatching component.

Empirical evidence on which agent is most efficient in solving a certain category of tickets is not explicitly taken into consideration. We consider this as a missed opportunity as we believe and we will show later in this chapter, that considering this information can help reduce the time required for solving a set of tickets. Therefore, we derive a method for ticket clustering, which identifies ticket topics (categories) that are discriminative in terms of an agent's efficiency and dispatches each ticket from every category to the agent that is the most efficient in executing it.

The clustering and dispatching component is designed to be integrated in the IT Service Delivery Environment as shown in Figure 5.1. In this chapter, we show that the existence of such a component that takes informed dispatching decisions, can reduce the resolution time and by extension, the business costs. We demonstrate this by conducting experiments on real data collected from different service delivery units.

More precisely we gathered data from three IBM Service Delivery units which represents logs of past incident resolutions. It corresponds to approximately 8000 tickets. This data comprises:

- Incident information: a) the ticket description, e.g., *Server unresponsive, can not ping*, and b) the ticket complexity, as assessed by the first human dispatcher.
- Agent information: the name and the email address of the person who resolved the ticket.
- Incident resolution information: timestamps corresponding to the moment the ticket was received and when it was closed and some text describing how the incident was resolved.

Our approach partitions the tickets into clusters that are relevant with respect to the problem they describe and that are also *homogeneous* in terms of the agent's performance. We exemplify such a desirable partitioning in Figure 5.2 where ticket data is represented as circles. In this example there are two ticket categories denoted by “Topic 1” and “Topic 2”.

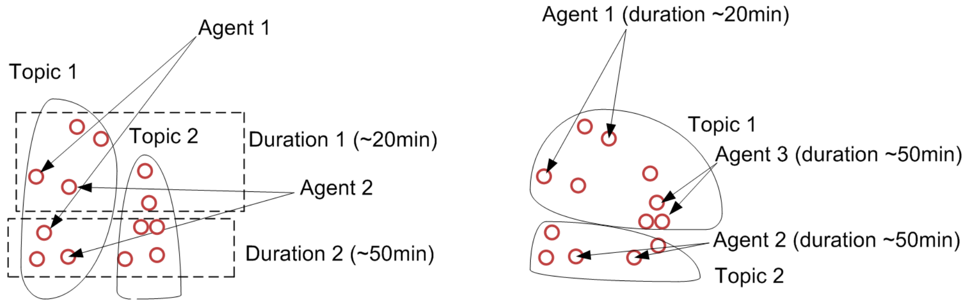


Figure 5.2: Illustration of a possible ticket partitioning (left) where tickets are semantically related but not homogeneous in agent s performance and a different partitioning (right) where tickets are again semantically related possibly denoting different categories and where tickets are homogeneous in an agent's performance.

All the tickets within the cluster of a certain topic should document approximately the same problem. At the same time, if an agent has solved multiple tickets pertaining to a certain cluster, the duration of execution of these tickets (executed by the same agent) should not vary much. If the variation in the time it takes for an agent to solve tickets from a certain cluster is small (the distribution is narrow, as illustrated in Figure 5.2) then we say a cluster is homogeneous in terms of an agent's performance.

We achieve such a partitioning by leveraging the multiple views of the ticket data (e.g., ticket description, ticket complexity, resolving agent, duration for resolving it, etc.). Having such a partitioning, when a new ticket arrives, we can infer its category and the approximate duration needed for an agent to solve the ticket. Further, we use this information in the dispatching process, by always assigning the ticket of a given cluster, to the agent that is the most efficient in solving tickets of that cluster.

The difficulty in finding these clusters is that applying a standard algorithm that groups the tickets based solely on their most prominent topics, leads to clusters that are non-homogenous in terms of the agent's performance. On the other hand, clustering based on the duration of resolution would not allow us to discriminate well between such groups as the duration of resolution is not data intrinsic to the ticket but more to the ticket and resolving agent pair. This poses a novel clustering problem - because we are restricted in the way we combine these main features – textual description and duration it took an agent for solving it. Practically, this problem is challenging also because we don't have a huge amount of data (about 8000 tickets) and the ticket descriptions are small and they have a very specific vocabulary – pertaining to the IT domain. Therefore, to assess the topical similarity we first perform a semi transfer-learning approach where to understand word similarity we gather some statistics on word co-occurrences on a large corpus of text representing the manuals for IT incident resolution.

Therefore, we propose a clustering approach that performs a partitioning that can discover meaningful ticket categories with homogeneous agent performance.

Our contributions are manifold:

1. A clustering method capable to identify ticket categories that are also homogeneous

in terms of the agent's performance;

2. A data-informed dispatching policy that assigns an incoming ticket to the agent that is the most efficient in resolving it;

3. An integrated solution to reduce the service costs.

The outline of this chapter is as follows. Section 5.2 presents the incident ticket clustering method. Section 5.3 provides an overview of the data-informed dispatching policy. The experimental evaluation on real-world datasets is presented in Section 5.4, followed by the related work in Section 5.5 and the conclusions in Section 5.6.

5.2 Incident ticket clustering

To achieve the partitioning described in the previous section and exemplified in Figure 5.2 we devise a method that clusters the set of tickets into groups of similar tickets both in terms of the incident they report and the time needed for an agent to resolve them. The method utilizes the matrix of *semantic similarities* between tickets – a matrix which for any two tickets contains a score of how close the two tickets are from a semantical perspective and the *divergence matrix* - a matrix which captures information on the difference in the duration of resolution of two tickets assigned to the same agent. More specifically, we first modify the *semantic similarity* matrix based on the information represented in the divergence matrix, and then, we run a clustering algorithm that operates on the resulting matrix. For each cluster, we derive an estimate for the time each agent needs to resolve tickets pertaining to that cluster. Ultimately, we devise a dispatching policy that assigns each ticket to the agent who is the fastest at resolving it and compute the achieved reduction in terms of service time due to the new assignment.

In Figure 5.3, we show an example of the desired clustering assuming we had only four tickets. Even though the semantic similarity between tickets T_1 and T_3 and tickets T_2 and T_4 is high, we do not put them in the same cluster due to the large difference in the resolution times. Instead we cut the edge that links them in the similarity graph. Therefore, for this example, instead of reporting the two clusters "servers issues" and "DB issues", we would discover "network issues" (as denoted by bigrams "lost connection" or "ping statistics") and "functional errors" (as denoted by bigrams "server crashed" or "inconsistent state").

5.2.1 Preliminaries

Let T be a the sequence of words representing a ticket consisting of $|T|$ words $w_1, \cdots, w_{|T|}$ and let $K = \{T_1, T_2, \cdots, T_n\}$ be a set of n tickets. Also, let $A = \{a_1, a_2, \cdots, a_{|A|}\}$ denote the set of agents who resolved the tickets. Each ticket T_i, is associated with several fields: $d^{(T_i)}$ the duration for resolving it, $a^{(T_i)}$ the agent who resolved it and $c^{(T_i)} \in W = \{\text{“A”},\text{“B”},\text{“C”}\}$ the complexity of the ticket (where "A" denotes the highest complexity and "C" the lowest).

For each agent a_i there is a mapping to its skill level $s^{(a_i)} \in W$. An agent with skill level "A" is entitled to resolve tickets of any complexity class, an agent with skill level "B"

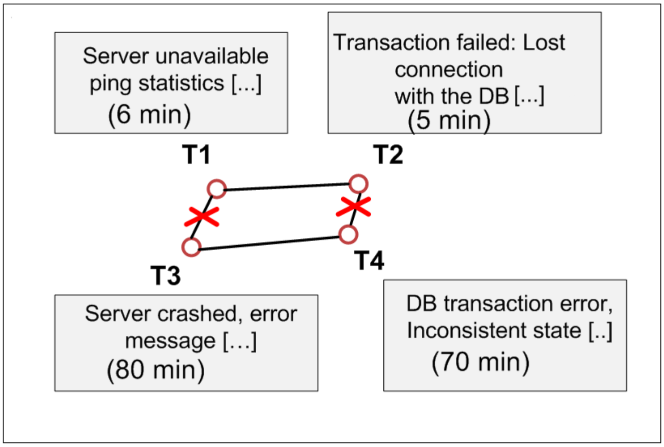

Server unavailable ping statistics [...] (6 min)
Transaction failed: Lost connection with the DB [...] (5 min)
T1
T2
T3
T4
Server crashed, error message [...] (80 min)
DB transaction error, Inconsistent state [..] (70 min)

Figure 5.3: Example of similarity demotion between tickets T_1 and T_3 which have been executed by the same agent in 6 and 80 minutes respectively and between tickets T_2 and T_4 which have also been executed by the same agent in 5 and 70 minutes respectively.

is entitled to resolve tickets of complexity “B” and “C”, and an agent with skill level “C” is entitled to resolve only tickets of complexity “C”.

5.2.2 Multi-view similarity matrix with induced sparsity

The typical approach to measuring the similarity between two blocks of text is to use a lexical matching method, and compute a similarity score based on the number of lexical units (words) that occur in both input texts. Pre-processing the texts via stemming, removal of the stop words, longest subsequence matching and additional normalization and weighting factors have shown to improve the results of such lexical methods [47, 143]. However the lexical similarity measures alone have an important limitation: they fail in identifying the semantic similarity between texts (i.e., the similarity score between “process shutdown” and “application terminated” would be zero).

In this section, we describe the method [125] we used to compute the semantic similarity metric between tickets. It is derived from the semantic similarity metric between words, described in the next section. We define the semantic similarity between two tickets T_i and T_j, given the semantic similarity between words. Each word w in T_i is assigned a semantic similarity score $maxSim(w, T_j)$, which represents the similarity score between w and the most similar word in T_j. The procedure is applied symmetrically, for each word in T_j. The word similarity scores for each ticket are added and each summation is normalized with the length of its corresponding ticket. The final result is the average of the normalized summation of each ticket:

$$\begin{aligned} \mathit{Similarity}(T_i, T_j) = & \frac{1}{2}\Big(\frac{\sum_{w \in T_i} \mathit{maxSim}(w, T_j)}{|T_i|} + \\ & + \frac{\sum_{w \in T_j} \mathit{maxSim}(w, T_i)}{|T_j|}\Big) \end{aligned} \tag{5.1}$$

For example, assuming we have a ticket $T_1 = w_1, w_2$ and a ticket $T_2 = w_3, w_4$ and the semantic similarity scores between words of these two tickets as given in Table 5.1, then the semantic similarity between the two tickets according to equation 5.1 is 0.7.

	w_3	w_4
w_1	0.1	0.6
w_2	0.4	0.9

Table 5.1: Example semantic similarity between the words corresponding to two tickets $T_1 = w_1, w_2$ and $T_2 = w_3, w_4$.

Let $\mathbf{S} \in \mathbb{R}^{n \times n}$ denote the ticket-ticket semantic similarity matrix, where $\mathbf{S}_{ij} = \mathit{Similarity}(T_i, T_j)$.

Another type of proximity relation between tickets is given by the divergence matrix described below. If two tickets T_i and T_j, that are annotated with the same complexity level (i.e., $c^{(i)} = c^{(j)}$) have been resolved by the same agent ($a^{(i)} = a^{(j)}$) and the ratio in their corresponding resolution times is higher than a certain threshold λ then they have different level of difficulty and they should not be clustered in the same group of incidents. We define the divergence matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$:

$$\mathbf{D}_{ij} = \begin{cases} 1 & \text{if } a^{(i)} = a^{(j)} \text{ and } c^{(i)} = c^{(j)} \text{ and} \\ & \dfrac{min(d^{(i)}, d^{(j)})}{max(d^{(i)}, d^{(j)})} > \lambda; \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

Two tickets which are similar in terms of both their topics and the duration it took an agent to resolve them may reveal stronger connection than two other tickets which are similar only in terms of topic.

An example of such a case we have shown in Figure 5.3 where tickets T_1 and T_3 both document a server error. However when these two tickets were executed by the same agent, for T_1 it took 6 minutes to be resolved and for T_3 it took 80 minutes to be resolved. This hints the fact that the category "server error" is not informative enough and that possibly attaching T_1 to the category of networking errors and T_3 to some other outage might be more accurate.

Motivated by these observations we combine the two proximity relations (views) into one matrix $\mathbf{S}' \in \mathbb{R}^{n \times n}$ as follows:

$$\mathbf{S}' = \mathbf{S} - \mathbf{S} \circ \mathbf{D}. \tag{5.3}$$

where $\mathbf{S} \circ \mathbf{D}$ is the element wise product of the two matrices. We induced sparsity on $\mathbf{S}$ by setting the similarity scores in $\mathbf{S}$ to 0 when $\mathbf{D}_{ij} = 1$. Intuitively, when two tickets are executed by the same agent and seemingly have similar topics, but which take very different amounts of time to be resolved by the same agent, it might mean that there is a subtle difference in the problem they descirbe and hence, these two tickets should not be clustered together.

5.2.3 Semantic similarity metric between words

As the same incident may be documented with different words, see e.g., tickets T_1 and T_2 in Figure 5.3, we are interested in assessing the degree of similarity between words in the specific IT Service Delivery domain.

For extracting the semantic similarity scores between pairs of words, we have explored three corpus-based methods which have been reported to give good results [65, 56]. The methods are: PMI-IR [142], the Google similarity distance (GSD) [129] and LSA [118].

PMI-IR only requires simple statistics about two words: their marginal frequencies and their co-occurrence frequency in a corpus.

$$\mathrm{PMI}_{\mathrm{IR}}(w_1, w_2) = \log \frac{p(w_1, w_2)}{p(w_1) * p(w_2)} \tag{5.4}$$

where $p(w_i, w_j)$ denotes the probability that words w_1, w_2 co-occur in the same ticket, $p(w)$ is the probability that the word w occurs in a ticket. Please note that we do not need to specify a window size for estimating the co-occurrence frequencies of w_1 and w_2 as the ticket descriptions are usually short ($<$ 20 words).

GSD is based on information distance and Kolmogorov complexity. In the original paper, the authors rely on Google to retrieve pages for co-occurrence statistics. We adapt the extraction of co-occurrence statistics to our setting, as follows:

$$\begin{aligned}\mathrm{GSD}(w_1, w_2) =& \frac{max\{\log(|f(w_1)|, \log(|f(w_2)|\}}{\log(Q) - min\{\log(|f(w_1)|), \log(|f(w_2)|)\}} - \\ &- \frac{\log(|f(w_1, w_2)|)}{\log(Q) - min\{\log(|f(w_1)|), \log(|f(w_2)|)\}}\end{aligned} \tag{5.5}$$

where $f(w_1, \cdots, w_k)$ represents the set of all tickets in which the words $w_1, \cdots, w_k$ appear together and $Q = \sum_{w_1 \in T_1, w_2 \in T_2} |f(w_1, w_2)|$ is the sum of the numbers of occurrences of search terms in each ticket, summed over all tickets.

In **LSA**, term co-occurrences in a corpus are captured via dimensionality reduction operated by a singular value decomposition (SVD) on the term-by-document matrix representing the corpus.

While, as we will show in Section 5.4, all the methods gave meaningful results regarding the relatedness of words in a broader sense (i.e., flagging associations between pairs of words such as “hung” and “ping” or “disk” and “full” as significant), LSA outperformed the other two methods in identifying synonyms. In order to perform a quantiative evaluation of how well each method performs on our dataset (comprised of a large set of ticket descriptions and ticket resolutions) we did the following:

- We have tagged the words in the corpus into their *parts-of-speech* (e.g., a noun, a verb, an adjective, etc.) and labeled them accordingly (*pos tagging* [96]);
- We selected pairs of words with the same pos tag (lexical category) and with high association score with any of the three methods;
- We asked a domain expert to select out of the pairs given above, pairs of words that are synonyms in the IT Service Delivery domain;
- We evaluated the methods based on their performance in detecting these pairs of synonyms.

The process described above has resulted in a list of 72 pairs of words that are synonyms in the IT Service Delivery domain (e.g., "pingable" and "responsive" or "archive" and "backup"). We say that a method/algorithm has "discovered" a pair of synonyms (w_1, w_2) if the scaled semantic similarity score between (w_1, w_2) is above the mean of the semantic similarity scores between w_1 and any other word in the corpus (or w_2 and any other word in the corpus).

We count the number of synonyms identified by each method and divide the result by the total number of pairs of synonyms. The results are shown in Table 5.2.

LSA	GSD	PMI-IR
0.83	0.42	0.3

Table 5.2: Percentage of synonyms identified.

There are several reasons why LSA outperformed the other two methods. First, the conventional wisdom is that synonym words, with a high degree of relatedness are unlikely to co-occur in a small window size (very close to each other). Second, it is plausible to represent the meaning of a word by a context vector of co-occuring words and the corresponding co-occurrence counts measured in a text window context. LSA does exactly this, as the assumption is that words that are close in meaning will occur in *similar context* and in addition, it transforms the context vectors to a lower dimensional space by applying singular value decomposition (SVD). The similarity is further reduced to the similarity of the context vectors where the cosine of the angle is employed as a similarity metric.

5.2.4 Clustering

The tickets are related via two types of similarity measures that originate from different sources: one that comes from the ticket content - the ticket similarity matrix $\mathbf{S}$, and one that comes from the agents who resolved the tickets - the divergence matrix $\mathbf{D}$.

In Figure 5.4 we show two square matrices ($n \times n$, where n is the number of tickets), the similarity matrix between tickets (left) and the divergence matrix (right). This is just for the reader to visualize the discrepancy in the information content between these two matrices. I.e., due to the extreme sparsity, the divergence matrix alone does not contain complete information of the structure of the clusters. Since both matrices capture important information, we propose an approach that uses the two matrices in the combined matrix $\mathbf{S}'$ as defined in equation (5.3) and has the following objectives:

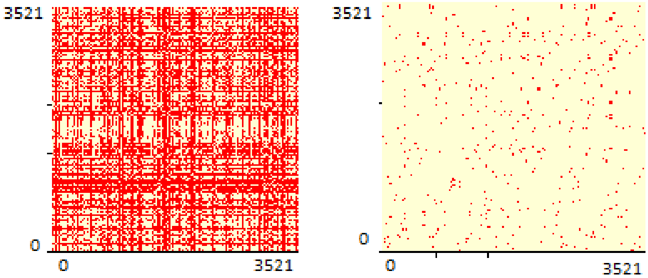


Figure 5.4: Sparsity: similarity matrix (left), divergence matrix (right).

I Cluster similar tickets together (we want to be able to extract topic information from these clusters);

II Minimize the number of pairs of tickets in the same cluster that have similarity equal to zero (we want homogeneous clusters in terms of the agent's performance, which is why we enforced the $\mathbf{S'}_{ij} = 0$ when $\mathbf{D}_{ij} = 1$).

We achieve this using three candidate clustering algorithms namely: hierarchical clustering with complete linkage [43], spectral clustering [164] and our adjusted version of fuzzy k-means [90]: homogeneity optimized fuzzy k-means.

Hierarchical clustering and spectral clustering are *hard clustering* algorithms, i.e., each ticket is a member of exactly one cluster. The input for these algorithms is the matrix $\mathbf{S'}$.

Fuzzy k-means is a *soft clustering* algorithm - and therefore returns a partition of the n tickets $\{T_1, T_2, \cdots, T_n\}$ into k clusters $\{C_1, C_2, \cdots, C_k\}$ specified by a membership matrix $\mathbf{M} \in \mathbb{R}^{n \times k}$, $\mathbf{M}_{ij} \geq 0$ and $\sum_{j=1}^{k} \mathbf{M}_{ij} = 1$, whose components quantify the membership probability of ticket T_i in cluster C_k. The input for fuzzy k-means is the similarity matrix $\mathbf{S}$. We optimize the homogeneity of these clusters by iteratively moving tickets from their most probable clusters to the second, third, up to the least probable clusters, as will be described later on. We selected fuzzy k-means and not LDA – a generative statistical model, which assumes that each document is a mixture of a small number of topics and that each word's creation is attributable to one of the document's topics [49] – because the ticket descriptions are very short, which is too sparse for traditional topic modeling. Therefore we used a soft clustering algorithm that is able to leverage the similarity matrix $\mathbb{S}$.

While hierarchical clustering does not require the specification of the number of clusters, k, spectral clustering and fuzzy k-means take k as input. In order to estimate the optimal number of clusters for the two latter algorithms we use the *silhouette statistic*, a well-balanced coefficient introduced in [101] and which has shown good performance in experiments, as we will show in Section 5.4.

Let $C = \{C_1, C_2, \cdots, C_k\}$ be a clustering of the set of tickets. For each cluster $C_i \in C$, for each complexity class $c \in W$ and for each agent $a_c \in A$ (such that a_c has solved tickets in C_i of complexity c) , one can measure the dispersion in the duration of resolution. The metric we propose for this is the *coefficient of variation* (CV), defined as the ratio of the standard deviation of these durations to their mean:

$$CV = \frac{s_x}{\overline{x}} \tag{5.6}$$

where s_x is the standard deviation of a set of samples x_i and $\overline{x}$ is their mean.

The motivation for using the coefficient of variation is the following: the standard deviations of two sets are not comparable to each other in a meaningful way to determine which set has greater dispersion because the values in the sets may have different magnitudes. The coefficient of variation does however show the extent of variability in relation to the mean.

Hierarchical Clustering with Complete Linkage The agglomerative hierarchical clustering algorithm with complete linkage [64], works as follows:

1. The algorithm starts with n clusters, each containing one object.
2. The most similar pair of clusters C_i, C_j is found using the combined similarity matrix $\mathbf{S}'$ and merged into a single cluster.
3. The similarity matrix is updated (its order is reduced by one by substituting the individual clusters with the newly merged one).

Steps (2) and (3) are repeated until a certain stopping criterion is reached.

This method is relevant for our study because of the distance measure that is used between two clusters in the merging step in the algorithm: C_i, C_j: $d_{complete}(C_i, C_j) = max_{l \in C_i, m \in C_j}(1 - \mathbf{S}'_{lm})$. If s_k denotes the similarity of the two clusters merged in step k, this distance measure ensures that no pair of tickets with similarity 0 (distance 1) will be put in the same cluster. The clusters at step k are maximal sets of points that are completely connected with each other by edges of weights (similarity) $s \geq s_k$.

Spectral Clustering Using the combined similarity matrix $\mathbf{S}'$ in a spectral clustering algorithm [164], there will result a partitioning obtained by minimizing sum of the weights of the edges belonging to the graph cuts in the input graph and thus implicitly maximally satisfying the objective II above.

Spectral clustering works as follows:

1. Construct the Graph Laplacian $\mathbf{L}$ from the combined similarity matrix $\mathbf{S}'$ such that $\mathbf{L} = \mathbf{D}^{-\frac{1}{2}}\mathbf{S}'\mathbf{D}^{-\frac{1}{2}}$ where $\mathbf{D}_{ii} = \sum_{j=1}^{n} \mathbf{S}'_{ij}$;
2. Select the first k eigenvalues $\lambda_1, \cdots, \lambda_k$ of $\mathbf{L}$ and determine their corresponding eigenvectors $v_1, \cdots, v_k$ and let $\mathbf{M} \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $v_1, \cdots, v_k$ as columns, This matrix is furhter normalized;
3. Perform clustering in the new subspace (new matrix $\mathbf{M}$) using K-means.

Homogeneity Optimized Fuzzy K-means We use the similarity matrix $\mathbf{S}$ to perform the soft clustering with fuzzy k-means. The resulting clusters will not be homogeneous in terms of duration because we only pass the semantic similarity matrix to the algorithm. In the following, we propose an approach that utilizes the clusters returned by fuzzy k-means and reduces the variation in duration inside the cluster by iteratively removing tickets.

Let $\mathscr{C} = \{C_1, C_2, \cdots, C_k\}$ be the clusters returned by fuzzy k-means and $\Pr(T \mid C_i)$ denote the probability of ticket T belonging to cluster C_i. At each step, the algorithm selects the ticket that minimizes the variation from a sub-cluster $C_i^{(a)(c)} \subseteq C_i$ (subset of tickets in C_i that have been solved by agent a and have complexity c) and removes it from the cluster. The ticket is then tentatively inserted in the the next most probable clusters in an order given by the ranking from fuzzy k-means. A ticket is inserted in another cluster if the variation in duration of the cluster *with* the ticket does not increase. If the ticket can not be inserted in any of the clusters it is dropped. This procedure continues until the variation in duration inside $C_i^{(a)(c)}$ reaches a target value τ. These steps are formalized in our algorithms: Algorithm 5.1 and Algorithm 5.2:

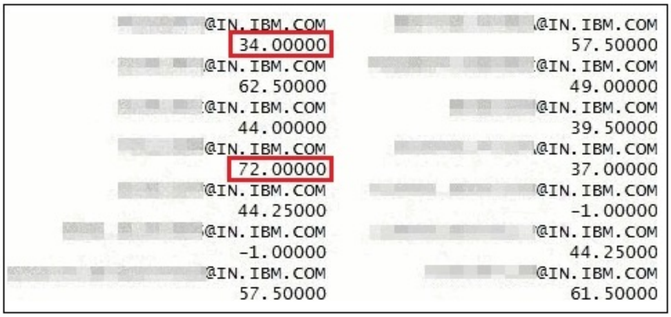
Algorithm 5.1 Optimize homogeneity of the clusters.

```
function OPTIMIZE($\mathscr{C}, \tau$)
    for all $C_i \in \mathscr{C}$ do
        for all $C_i^{(a)(c)} \subseteq C_i$ do
            while $CV_{C_i^{(a)(c)}} \geq \tau$ do
                $T = \text{argmin}_T \, CV_{C_i^{(a)(c)} \setminus \{T\}}$
                $C_i^{(a)(c)} \leftarrow C_i^{(a)(c)} \setminus \{T\}$
                MOVE($T$) (see Algorithm 5.2)
            end while
        end for
    end for
end function
```

Algorithm 5.2 Move a ticket.

```
function MOVE($T$)
    for $C_i \in \mathscr{C} \setminus \{T\}$ sorted in decreasing order of $\Pr(T \mid C_i)$ do
        if $CV_{C_i^{(a)(c)} \cup \{T\}} \leq CV_{C_i^{(a)(c)}}$ then
            $C_i^{(a)(c)} \leftarrow C_i^{(a)(c)} \cup \{T\}$
            return
        else
            drop the ticket $T$
        end if
    end for
end function
```

Homogeneity optimized fuzzy k-means has the following properties: (i) the variation inside the final clusters is not an output of the algorithm but an input parameter of the algorithm and this fact enables finer control. The variation of the resulting clustering will not exceed the one given as an input. (ii) The algorithm can be used for identifying the outliers in the data set. To achieve this we only move the ticket to the second or the third most probable clusters and drop it if it cannot be placed. The advantage of doing this is



```
[redacted]@IN.IBM.COM    [redacted]@IN.IBM.COM
          34.00000                 57.50000
[redacted]@IN.IBM.COM    [redacted]@IN.IBM.COM
          62.50000                 49.00000
[redacted]@IN.IBM.COM    [redacted]@IN.IBM.COM
          44.00000                 39.50000
[redacted]@IN.IBM.COM    [redacted]@IN.IBM.COM
          72.00000                 37.00000
[redacted]@IN.IBM.COM    [redacted]@IN.IBM.COM
          44.25000                 -1.00000
[redacted]@IN.IBM.COM    [redacted]@IN.IBM.COM
          -1.00000                 44.25000
[redacted]@IN.IBM.COM    [redacted]@IN.IBM.COM
          57.50000                 61.50000
```

Figure 5.5: Agent's performance for a given cluster and complexity level.

that the coherence of the clusters with respect to the topics remains unaltered (unlike the base algorithm where we tradeoff topic relevance for high homogeneity). Additionally one can inspect the set of dropped tickets and observe which type of incidents have inherently higher variation in the resolution time. A discussion on the behaviour of these algorithms on our dataset is given in Section 5.4.

5.3 Dispatching

For a given partitioning $\mathscr{C} = \{C_1, C_2, \cdots, C_{|\mathscr{C}|}\}$, we build a matrix $\mathbf{P} \in \mathbb{R}^{|A|\times|\mathscr{C}|\times|W|}$ which stores the median duration for the execution time for each agent $a \in A$ in resolving a ticket pertaining to one of the clusters in $\mathscr{C}$ for a certain complexity class $k \in W$. Some entries in this matrix will be -1. When an entry $\mathbf{P}_{ijk} = -1$ it means that there are no records in the data of agent a_i working on a ticket from cluster C_j of complexity k where k is an id for each complexity class $\{$"A", "B", "C"$\}$.

We implement the data-informed dispatching policy and the non data-informed dispatching policies which we will describe next. Unlike a non data-informed dispatching policy, the data-informed policy, dispatches a ticket T from a cluster C to the agent that is the most efficient in resolving it from all the agents that are available, i.e., it selects a_i. such that $\mathbf{P}_{ijk}$ is minimal. The implementation of the dispatching aligns with the following assumptions:

- The maximum number of working hours of an agent is less than 8. When an agent reaches 8 hours of work in one day, the agent is removed from the pool of available agents to which tickets can be dispatched on that day;

- No tickets are passed from one day to another. This assumption is enforced by the original data, where the ticket resolution is started and completed in the same day in 99.5% of the cases;

- The complexity of the ticket must match the qualification of the resolving agent. Therefore, an agent with skill level "A" is entitled to resolve tickets of any complexity class, an agent with skill level "B" is entitled to resolve tickets of complexity "B" and

serverXYXY error : DISK Utilization :Object = /dev/sda3/var : percent full: 95% : MB free: 219 MB
New User Request Form: XYXY backup missed on this server
We are unable to connect to hosts serverXYXY. Please investigate

Table 5.3: Examples of ticket descriptions.

"C", and an agent with skill level "C" is entitled to resolve only tickets of complexity "C";

- When measuring the service time which represents the total duration of execution of the tickets we did not add any waiting time;
- The arrival order of the tickets based on which we perform the dispatching is the original order, as documented in the dataset.

For the implementation of the dispatching, we split the data in separate chunks, each chunk containing the tickets arrived in a certain working day. For every chunk, we store a list with all the agents available that day by mining the log for agents that have tickets associated to their id in that particular day. We also maintain a queue with the tickets that need to be completed that day by extracting the list of tickets from the logs that were recorded as started and completed that day. Ultimately, we run the data-informed dispatching policy and the policy that works under exactly the same assumptions, but without being data-informed (equivalent to the setting when an agent voluntarily selects the next ticket to resolve, provided that the agent has the necessary qualification - skill based dispatching [127]), and compare the recorded service times for each delivery unit.

5.4 Empirical evaluation

5.4.1 Data preparation

We use three datasets: DU1, DU2, DU3 from three service delivery units comprising 3696, 2373 and 1916 tickets respectively.

The first step in the analysis is the data cleaning procedure where we take the raw unstructured data and remove unnecessary information such as email headers, punctuation, html formatting, server names, stop words and we also perform stemming [109]. This step is important as due to the fact that ticket descriptions are very short, such "noise" in the data severely impacts the performance of the algorithms. The ticket descriptions are a mixture of machine and human-generated text and contain many domain-specific technical words, some of which are not present in a standard dictionary (e.g., "unpingable", "ssd", "hw"), but are still relevant in identifying the incident category. Some examples of ticket descriptions are given in Table 5.3.

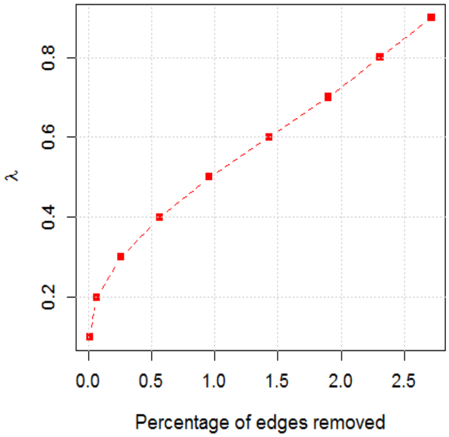


Figure 5.6: Percentage of edges removed when increasing λ (for DU1).

5.4.2 Experiments

In the experiments, we set the threshold to $\lambda = 0.8$, meaning that for two tickets from the same cluster, solved by the same agent and of the same complexity the variation of duration of executing them is within 20%. Using such small variation is reflected in the connectedness of the similarity graph. In Figure 5.6, we show how the percentage of deleted edges in the similarity graph increases by increasing the value of λ.

Clustering In what follows, we show that spectral clustering and homogeneity optimized fuzzy k-means exhibit the best performance for this task in terms of cluster purity and homogeneity in terms of agent perforamnce.

We run hierarchical clustering with complete linkage (stopped at dissimilarity threshold 0.8). For DU1 we obtain a large set of clusters (176 clusters) with a low abundance of tickets (on average 19 tickets per cluster), and also the clusters tend to have similar sizes. This is illustrated in Figure 5.7 where we can observe a skewed distribution when plotting the cluster abundances. The results remain similar when varying both λ and the dissimilarity threshold for stopping (i.e., for $\lambda = 0.8$, dissimilarity threshold=0.9 we obtain 130 clusters, or for $\lambda = 0.5$, dissimilarity threshold=0.8 we obtain 92 clusters). The resulting clusters are small and with low variation in both topic and resolution time per agent, but they are not unique, i.e., they are fragments of larger clusters.

With spectral clustering, we obtain 12 clusters with sizes between 193 and 659 tickets for DU1, 7 clusters with sizes between 232 and 601 for DU2 and 5 clusters with sizes between 134 and 385 for DU3. By analyzing the frequent words in the clusters together with the domain experts, we observed that the algorithm is able to correctly capture relevant incident ticket categories in each cluster. This is shown in the Table 5.4

Homogeneity optimized fuzzy k-means exhibits also good performance in identifying incident categories. We obtain 12 clusters with sizes between 195 and 429 tickets for DU1, 7 clusters with sizes between 206 and 387 for DU2 and 5 clusters with sizes between 127

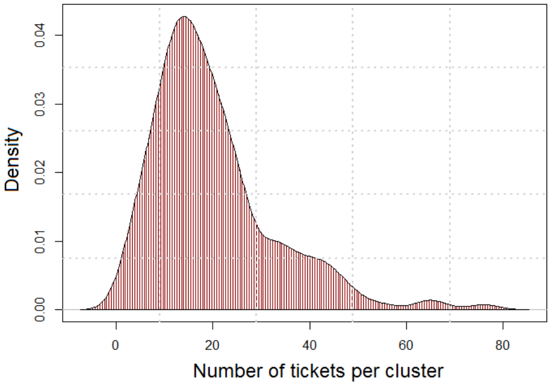


Figure 5.7: Cluster abundances density plot.

Cluster	**Frequent words**
C_1	*lost, contact, agent*
C_2	*ssd, internal, error*
C_3	*file, system, space, full*
C_4	*prod, issue, faulty, disk*
C_5	*cluster, bottleneck*
C_6	*system, restart, request*
C_7	*backup, error*
C_8	*scsi, retry, failed*
C_9	*server, physical, failure*
C_{10}	*application, problem, not, responding*
C_{11}	*node, down, power, supply*
C_{12}	*limit, exceeded, file*

Table 5.4: Frequent words in the clusters obtained by spectral clustering using $\mathbf{S}'$ for DU1.

and 368 for DU3.

We first run the algorithm such that no tickets are dropped. We compute the minimum value for the target coefficient of variation, τ_{min}, for each delivery unit, such that all the tickets are assigned to a cluster. The values for τ_{min} when no tickets are dropped for each delivery unit are given in Table 5.5. Note that the values for the coefficient of variation are small, due to the fact that we allow tickets to be assigned to less probable clusters. The frequent words in the clusters discovered are presented in Table 5.6. We observe that both clustering approaches identify ticket categories documenting “file limit exceeded”, problems related to the “power supply”, or “backup” related issues. While the clusters obtained with spectral clustering place all the disk related tickets in one single cluster,

Delivery Unit	DU1	DU2	DU3
$\tau_{\mathbf{min}}$	0.17	0.23	0.21

Table 5.5: Values computed for τ_{min} for each delivery unit

Cluster	**Frequent words**
C_1	*monitoring, cluster, disk, replacement*
C_2	*fsd, code, error*
C_3	*command, retry, reboot, server*
C_4	*client, backup, file*
C_5	*ssd, database, memory, corruption*
C_6	*disk, error, scsi*
C_7	*node, down, agent, contact*
C_8	*disk, fan, sensor, faulty*
C_9	*docket, frame, replacement*
C_{10}	*power, supply, fault*
C_{11}	*transport, lost, retry*
C_{12}	*limit, exceeded, file*

Table 5.6: Frequent words in the clusters obtained homogeneity optimized fuzzy k-means for DU1.

with the homogeneity optimized fuzzy k-means, we obtain two clusters documenting disk errors: one which was documenting disk replacement, and one reporting disk fan sensor issues. Also for homogeneity optimized fuzzy k-means it is not always straightforward to establish the incident category of a certain cluster. One such example is the cluster with top words “ssd, database, memory, corruption”. Also, it merged tickets documenting “node down” with tickets documenting “lost, contact, agent”. This hints to the fact that for an agent, these two categories of incidents take similar amount of time to be resolved.

When we introduced homogeneity optimized fuzzy k-means, we also mentioned the possibility of identifying outliers, tickets that take either too long or too little time to be resolved by an agent relative to the cluster they pertain to. This can be achieved by modifying the algorithm to move tickets only in the first three most likely clusters. This leads to a significant number of tickets dropped and this grows inversely proportional with the value of τ as illustrated in Figure 5.8. By inspecting the dropped tickets we observe that they

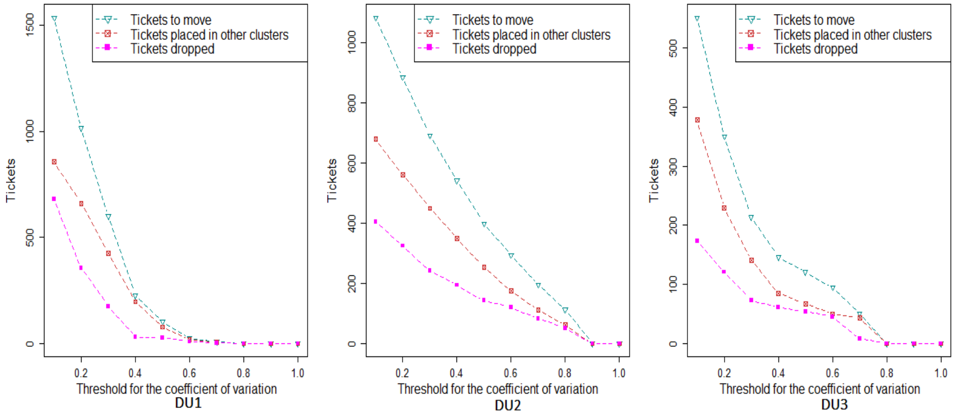


Figure 5.8: Number of tickets to move, tickets successfully placed in other clusters and tickets dropped for each delivery unit (DU1, DU2, DU3).

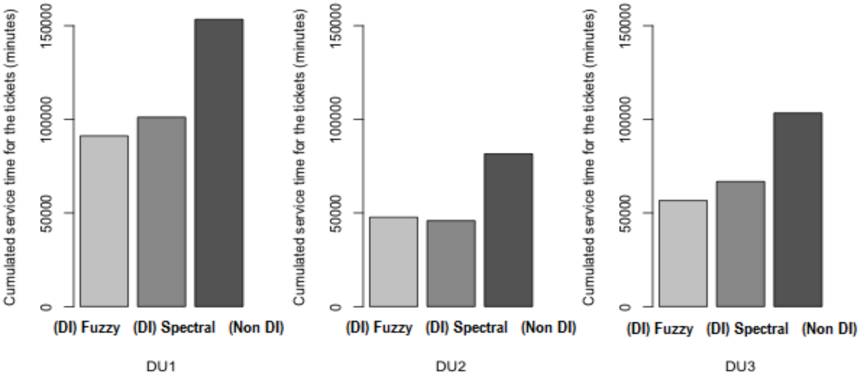


Figure 5.9: Speed-up in service time for each delivery unit using the data-informed dispatching policy on the clustering obtained with fuzzy k-means – (DI) Fuzzy, with spectral clustering – (DI) spectral and using the non data informed policy – (non DI).

mostly document disk errors, and filesystem issues. The recorded duration for the dropped tickets is either very short or very large. This indicates that in the dispatching, for those tickets with larger duration one needs to have some margin with respect to filling the day completely.

In the following, we evaluate the quality of the clustering obtained from running spectral clustering with the combined similarity matrix, and homogeneity optimized fuzzy k-means with the minimum values of τ for which no tickets are dropped, τ_{min}. We inspect both the reduction in service time from implementing the data informed dispatching policy and the homogeneity in agent's performance.

Speed-up in service time Let $C = \{C_1, C_2, \cdots, C_n\}$ be the partitioning of the corpus of tickets C. We measure the service time which is the total number of working hours needed for resolving all the tickets in a given period of time. This is formally defined as:

$$ServiceTime(C) = \sum_{i=1}^{n} \sum_{j=1}^{|C_i|} P_{e,j,k} \tag{5.7}$$

In equation 5.7, the matrix $\mathbf{P}$ was used, which for a resolving agent e, for a ticket from a cluster C_j of complexity k, stores the estimated duration for resolving the ticket.

Next, we run the data-informed dispatching policy, both on the clustering obtained with spectral clustering and with the homogeneity optimized fuzzy k-means. The data informed dispatching policy is evaluated against the dispatching that does not take the insight from the data into account (which agent is fastest in resolving an incoming ticket).

We observe a major reduction in the service time up to 44% (for DU2) when running the data-informed dispatching policy non data-informed policy. The reductions are apparent for dispatching based on both the homogeneity optimized fuzzy k-means clustering and the

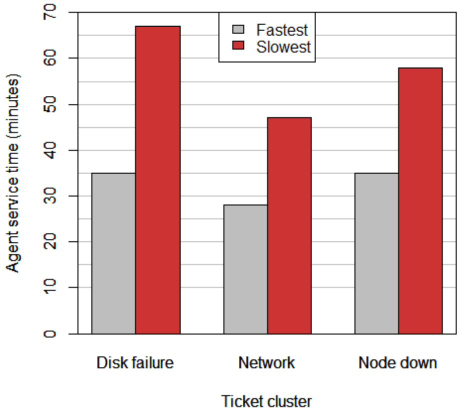


Figure 5.10: Differences in agent's performance in resolving tickets of the same complexity from different clusters.

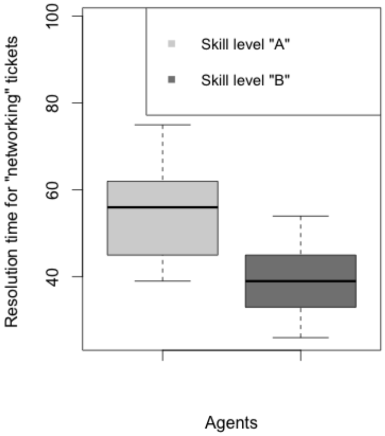


Figure 5.11: Resolution time for networking tickets in minutes to exemplify an agent with skill level "A" being slower than an agent with skill level "B".

spectral clustering, slightly better in the former one (possibly due to the more uniform cluster sizes). The speedup is shown in Figure 5.9 where we denote by (DI) the data-informed dispatching policy and by (non-DI) the non data informed dispatching policy.

Figure 5.11 shows that the discrepancies in agent's performance for a given cluster and complexity are large and exploiting these differences in dispatching leads to a significant reduction in the service time. Also we have identified that agents with skill level "B" are faster than agents with skill level "A" in solving tickets documenting "network errors" of complexity "B", shown in Figure 5.10. This illustrates that a higher skill (which is established based on experience or training level) does not directly translate into higher speed (which can only be established empirically, by analyzing the historical data). Note that we did not include service level agreements in both dispatching policies. A service level agreement (SLA) is a commitment established between the client and the service provider. An example of such an SLA is: "90% of tickets have to be resolved within 24 hours". We kept the order of ticket arrival, but assumed that a ticket can be resolved by the optimal agent even if that meant ticket resolution could only start when this agent becomes free. Naturally this might delay ticket resolution potentially conflicting with SLA requirements on resolution times. In contexts with such SLA constraints, a simulation taking arrival patterns into account can yield more precise assessments. Please note however that both dispatching policies considered in our experiments, work under exactly the same assumptions and are therefore comparable.

Homogeneity in terms of agent's performance As the quality of the clustering is given by the qualities of the individual clusters, we define the homongeneity of a clustering as the

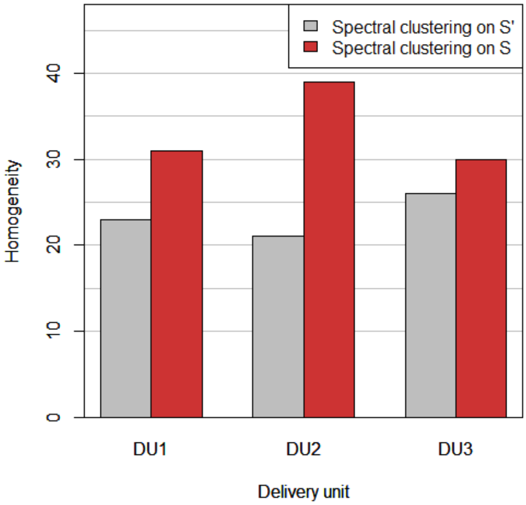


Figure 5.12: Homogeneity of different clustering methods.

average of the homogeneity of each cluster:

$$hom(C) = \frac{1}{|C|} \sum_{C_i \in \mathscr{C}} hom(C_i) \tag{5.8}$$

The homogeneity of the cluster C_i is defined as the average coefficient of variation for the durations recorded for a certain agent and complexity class. Formally, let $C_i^{(a)(c)} = \{d^{(k)} \text{ for } T_k \in C_i | a^{(k)} = a \text{ and } c^{(k)} = c\}$ and $CV_{C_i^{(a)(c)}}$ denote the coefficient of variation of $C_i^{(a)(c)}$. Then, we define the homogeneity of a cluster C_i as:

$$hom(C_i) = \frac{1}{|Z|} \sum_{C_i^{(a)(c)} \in Z} CV_{C_i^{(a)(c)}} \tag{5.9}$$

where $Z = \{C_i^{(a)(c)} \subseteq C_i \mid C_i^{(a)(c)} \neq \emptyset\}$

For the spectral clustering, the results are shown in Figure 5.12 where we compared the homogeneity in terms of agent's performance for the algorithm that uses the combined matrix $\mathbf{S}'$ to the algorithm ran on the semantic similarity matrix $\mathbf{S}$.

We compare these two version to illustrate the homogeneity gain obtained when using the divergence matrix. This also demonstrates that topics alone are not informative enough for estimating the agent's performance.

For homogeneity optimized fuzzy k-means, the coefficient of variation provided as input will reflect the homogeneity of the final clustering. The minimum values for the coefficient of variation for which no tickets are dropped, for each delivery unit, were given previously in Table 5.5.

Deployment As mentioned previously, the clustering and dispatching component is designed to be integrated in the IT Service Delivery Environment and a depiction of the elements the component interfaces with is shown in Figure 1. Our method succeeded in the evaluation phase and demonstrated effectiveness.

One important aspect that must be noted in deploying the dispatcher is the handling of a new agent. A new agent is assigned a default duration for solving a ticket of a particular topic, more precisely the median duration across agents for solving tasks of that topic. This way we ensure that enough tickets are sent to the new agent and therefore, the system can rapidly estimate the agent's actual speed.

Another aspect of the dispatching component is that since it favors the best agent for each ticket category, this may potentially lead to uneven workload distribution. To avoid this, one can adjust the greedy dispatching policy to include some regularization.

The performance gaps between agents need to be continuously reduced. Apart from its utility in the automatic dispatching, our system finds execution differences among agents and can show agent training needs and best practices. It can serve in targeted mentoring for ramp-up of skills for new employees which is very useful in a high turn-over environment.

5.5 Related work

Several works have previously addressed the possibility of improving the efficiency of ticket routing by mining ticket resolution sequence data or ticket descriptions [119, 130]. The authors in [119] capture the ticket transfer decisions embedded in ticket resolution sequences to develop a model to generate ticket routing recommendations. In [130] supervised learning techniques (SVM) and a discriminative term based heuristic are used to analyze ticket descriptions and predict the most appropriate resolution group. While we also investigate optimal ticket routing based on ticket descriptions, our focus is not matching the correct resolution group for a given ticket but rather on matching the most cost effective agent to resolve the ticket within a resolution group.

Researchers have also previously looked into clustering alerts and incident tickets [46, 134] for both structured and unstructured text using either graph theoretic approaches [46] or a combination of a latent semantic indexing based technique with a hierarchical n-gram based technique [134]. We have observed through our experiments that standard text similarity measures as Jaccard used by the authors in [46] perform poorly when used in clustering tasks due to data sparseness and the lack of context. We differentiate from these approaches by proposing a similarity metric between tickets that tries to overcome the vocabulary mismatch problem [67], by using semantic similarity between words inferred from a large corpus.

We are not aware of any previous work on ticket clustering that aims at discovering topics with high homogeneity in the agent's performance. From a theoretical point of view, the problem we are trying to solve is similar to clustering with multiple graphs. Clustering with multiple graphs aims to fully exploit the links between different dimensions of a given network. While there is a rich body of work in the context of single graph clustering [41, 162, 82, 18, 75] the problem of clustering with multiple graphs has gained interest only recently [141, 55, 48]. In [141] the authors propose a factorization method based on linked matrices to solve the multi-graph clustering problem. In this model, each graph is approximated by a graph specific factor with a common factor shared by all the graphs. In [55] the authors propose two multi-graph clustering techniques (one tailored for unweighted graphs and one that performs well also on weighted graphs) with the goal of finding well-defined clusters across all the views of the graph. In [48] the authors propose a generalization of normalized cut for multi-dimensional graphs. Their model leads to a mixture of Markov chains defined on the different graphs.

We want to identify clusters that persist under different measures. Leskovec et al. have shown in [83] that strong communities are still identifiable under various measures. We choose to combine the adjacency matrices in one in a way such that the information in the divergence matrix is manifested in the final matrix as pairwise cannot-link constraints (constraints that express that entity i and entity j should be in different clusters). Having a single matrix, the problem becomes again a single graph clustering problem which we try to solve employing matrix factorization based clustering algorithms or by iteratively adjusting a fuzzy partitioning. The work by Leskovec et al. has recently demonstrated that, although different quality measures produce differences in terms of specific communities, strong communities persist under a variety of measures.

A combination of text mining and process analysis is also present in the work of van der Aa [73, 72]. The author also notes the event logs provide valuable information on data attributes, event durations, and other aspects specifically associated with the enactment of a process. We both try to surface interesting process related information from the logs. Van der Aa focused mostly on the link between logs and process models, e.g., assessing the compliance of process models to textual descriptions [73]. Our contributions correspond to a different path – in the sense that we make some assumptions on what the process looks like – and we try to optimize the process execution. However, the algorithms in the thesis of Van der Aa [7] could be used to confirm such hypotheses as the one we made on the model.

For integration such inference and dispatching pipeline with a workflow management system, we would like to refer to the work of Mans et al., e.g., [124]. They present a proposal on how a workflow management system can be integrated with scheduling facilities rather than simply extending the functionality of a workflow management system or a scheduling system. As a modeling and execution tool they use CPN Tools [104], a modeling and execution tool for Colored Petri Nets. They explain how a workflow language can be augmented with information relevant for scheduling and present the design of a workflow management ststem integrated with scheduling together with a concrete implementation. Their architecture consists of four components: a) a workflow engine which routes cases, b) a workflow client application c) a scheduling service and d) a separate calendar component.

5.6 Conclusion

In this chapter, we presented a novel approach that optimizes the service time in IT Service Delivery industry. We demonstrated on real data that considering empirical evidence on which agent is the most efficient on resolving certain incident tickets in the dispatching process significantly reduces service time.

We first build a model able to cluster the tickets into categories that both reflect the problem they document and are homogeneous in the duration time for an agent to solve them. Then we devise a data-informed dispatching policy that assigns an incoming ticket to the agent that is the fastest in resolving it. Experiments conducted on real data from several IBM Service Delivery Units demonstrate the benefits of our approach. More specifically, we compare the data-informed dispatching with the non-data informed dispatching and observe that the former exhibits 35% to 44% reduction in the service time.

In workflow management systems, various forms of scheduling have been studied from a theoretical perspective, by providing patterns of work item assignment that can be supported by workflow management systems [114]. In current workflow management systems, scheduling is primarily based on the availability of resources that match a role that is authorized to execute a work item. In addition, there exist other algorithms that take qualitative properties into account, such as the suitability of a resource for a particular work item [71], or the appropriateness of a resource, given other resources that worked on the same case [20]. All these approaches could be labeled as *qualitative scheduling*.

Because human resources are one of the most significant factors in the operational cost of workflow, it is also interesting to consider *quantitative* approaches to scheduling. Scheduling can be quantitative in at least two different ways. First, it can be seen not only as a constraint satisfaction problem but as an optimization problem to minimize certain measures of cost – such as service time for our use case – or maximize certain measures of service quality. Second, the suitability of a resource is not a qualitative relation anymore but becomes a measure.

In this chapter, we considered the time a person needs to complete a certain work item – a ticket in this case – as such a measure of resource suitability. It is clear that such a measure can be obtained and adapted over time from information in execution logs. There are very few prior studies for BPM systems that take work item scheduling as an optimization problem and use a measure of resource suitability. We are not aware of any approaches where work item scheduling is based on execution log data.

A scheduling component could be easily embedded into the architecture of a business process execution engine. In order for the scheduler to make use of the new data generated by the engine, we would propose an incremental schema to feed the scheduler by sending the new data to it in batches, one at a time, so that the scheduler can update itself appropriately.

Chapter 6

Predicting Disk Replacement from Sensor Data

Disks are among the most frequently failing components in today's IT environments [4]. Despite a set of defense mechanisms such as RAID, the availability and reliability of the system are still often impacted severely [148].

In this chapter, we present a novel, highly accurate SMART-based analysis pipeline that can correctly predict the necessity of a disk replacement 10-15 days in advance. Our method has been built and evaluated on more than 30000 disks from two major manufacturers, monitored over 17 months. Our approach employs statistical techniques to automatically detect which parameters from the self monitoring facility of drives (SMART) correlate with disk replacement and uses them to predict the replacement of a disk with 98% accuracy.

6.1 Context

Data center downtime costs have increased sharply in the past years from \$5,600/minute in 2010 to \$8,851/minute in 2016 according to a study conducted on 63 data center organizations in the U.S. [8]. IT equipment failure is a significant contributor to such downtimes. Disks are among the most frequently failing components in today's IT environments. It appears that field behavior of disks, in particular failure rates, are fairly different than the one described in the datasheet specifications [28]. Factors such as temperature, operation cycles or workloads may significantly affect both the reliability and the performance of hard drives. Reliability issues are more severe than performance issues as they pose the risk of data loss and manifest themselves as disk failures leading to replacements.

Disk failures can be either predictable or unpredictable. On the one hand, unpredictable failures, ranging from electronic components becoming defective to sudden crashes due to improper handling, cannot be foreseen by monitoring. On the other hand, predictable failures mainly result from slow processes such as wear-and-tear that typically progress over months or years (for example disk head degradation). The latter ones make it possible for predictive failure analysis.

In this chapter, we introduce a novel data mining approach able to automatically predict disk replacements based on historic disk replacement data from an expert-maintained disk environment [9] and hence minimize the effects of component failure as shown in Figure 6.1.

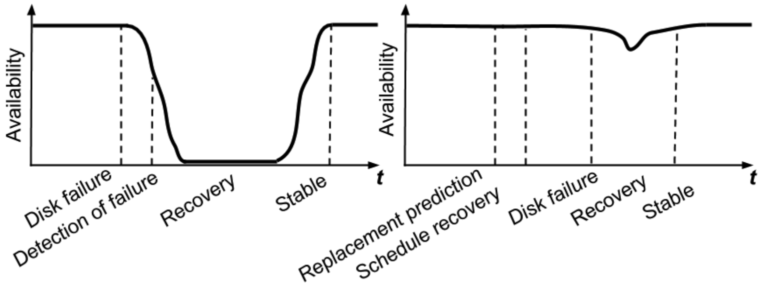


Figure 6.1: Availability – number of read or write operations: without proactive/preventive replacement (left) vs. with proactive replacement(right)

SMART monitoring (disk sensors' data, for which we provide a detailed description in Table 6.7) can be used to determine when disk failures become more likely. Some manufacturers even use them to deploy drives with embedded predictive models. However, these models are proprietary and often times, simple, threshold-based normalizations, that are more conservative – designed to avoid false alarms and therefore have a very weak predictive power [28, 54, 74] .

In this chapter, we focus on the automatic forecasting of disk replacements using SMART attributes. For this purpose, we use data collected from a large population of disks (more than 30000 disks) monitored over 17 months [9]. A drive is labeled as failed when it stopped working, it is non-responsive to commands, the RAID system reports that the drive cannot be written or read, or it shows evidence of failing soon [9]. Therefore, the model goes beyond the expert knowledge used in proactive replacements and is able to detect failures that this knowledge can not capture (see Section 6.3.6).

The goals of our analysis are two-fold: (1) to provide the set of SMART attributes that are indicative for disk replacements; (2) to use these attributes to build a statistical model that automatically predicts impending replacements with high accuracy (81-98%). Such a model not only automates the disk replacement decision, but also allows administrators to proactively replace disks at risk, days in advance.

To achieve these goals, we employ an approach that comprises four steps. First, we use changepoint detection in time series to identify the SMART attributes indicative of impending replacements. Second, we transform the event sequence into a set of examples [66] by encoding multiple events as individual points such that we achieve a compact, yet informative representation of the time series of each disk. Next, we build a predictive classification model that is able to discriminate between healthy and failure-impending drives, by using these data points as inputs. Finally, we propose a *transfer learning* [135] approach to enable replacement decision prediction on data from novel disk models.

There are several challenges one may encounter when performing the aforementioned steps. In the following, we list a few of them. Since SMART indicators are manufacturer-specific, their encoding and normalization varies widely across manufacturers (for instance attribute 9 raw – power on cycles can be reported in differing time units). This hinders the possibility to fit one predictive model for all different disk manufacturers. A separate model needs to be trained for each individual disk manufacturer. Further, due to the lack of standards when implementing SMART attributes, one needs to discover the ones that are indicative of failures. Finally, the disk data are highly unbalanced (only about 2% of disks are replaced), which makes the task of fitting high quality models very challenging.

Therefore, we build and evaluate our approach for disks from two individual manufacturers. Following our efforts to choose the right SMART indicators and tuning the predictive model, results show up to 98% accuracy in identifying both disks that are about to be replaced and those that are healthy, when using only a small set of SMART parameters.

The remainder of this chapter is organized as follows. In Section 6.2, we describe the predictive pipeline, whereas in Section 6.3 we present experimental results. We discuss deployment in Section 6.4 and finally review the state-of-the-art in Section 6.5 and conclude in Section 6.6.

6.2 Predicting disk replacement

Given the longitudinal measurements of the SMART attributes for a large set of disks, from a specific disk model of interest and information on their replacements, we develop a fully automated approach for solving the disk replacement prediction problem. Our method is summarized in Algorithm 6.1 and consists of four consecutive steps: (1) selection of relevant SMART attributes, (2) compact time-series representation, (3) balancing of the healthy and unhealthy disk classes via informed downsampling, and (4) classification model for disk replacements. In the following, we present the details of each step.

Algorithm 6.1 Disk replacement prediction algorithm

Input: A time series collection of SMART attributes along with the disk replacement information for a given target disk type.

1. Find the subset of SMART attributes indicative of disk replacements by identifying significant changepoints in their corresponding time series;
2. Compute a highly-informative compact representation for the time series corresponding to each relevant attribute from Step 1 via exponential smoothing;
3. Perform informative downsampling via K-means clustering to address the high class imbalance in the disk replacement datasets;
4. Use the training dataset from Step 3 to fit a classification model that predicts disk replacements.

Output: Predictive model for disk replacement using a small set of SMART attributes.

6.2.1 Selection of relevant SMART attributes

The main goal of this step is to automatically discover the set of SMART attributes that are indicative of impending disk replacements. This will reveal the most informative predictors with respect to the disks at risk to the domain experts. As SMART attribute data are gathered over time, we address this feature selection problem through changepoint detection in time series. More specifically, when a SMART attribute is informative of disk replacement, we expect a significant shift in its values at some time point before the actual replacement, i.e., at the changepoint. Moreover, this shift should be permanent and unrecoverable to be indicative of a disk replacement (an illustration of such a shift is given in Fig. 6.2. In the following we provide a more formal description of the approach for detecting the permanent changepoints for SMART attributes.

Let $S_i = (s_1, s_2, \cdots, s_p)$ denote the time series (collected at a daily granularity) for a certain SMART attribute comprising p measurements ordered by their timestamps, where s_p is the most recent one when the disk replacement has occurred. If there exists a timestamp $t < p$ when a significant change in the values of the attribute S_i occurs (e.g., the values start increasing), then we consider S_i a potential attribute relevant for the disk replacement. We determine the time point t that indicates a significant change using the approach described in [98]. Briefly, we use a maximum likelihood (ML) based approach as follows: $t = argmax_{\tau} ML(\tau)$ provided that $ML(t)$ is significantly larger than $\log p(\mathbf{s}_{1:p} \mid \hat{\theta})$, where θ is the maximum likelihood estimator of the parameter, and:

$$ML(\tau) = \log(p(\mathbf{s}_{1:\tau} \mid \hat{\theta_1})) + \log(p(\mathbf{s}_{\tau+1:p} \mid \hat{\theta_2})). \tag{6.1}$$

The intuition behind the equation above is that we need to split the data into two parts – before the change occured and after the change and we use maximum likelihood to fit the parameters to each of those parts. The reason for this is that we don't know beforehand where the change occured so we compute the likelihood of each potential time point. The most likely one is selected as the changepoint. Next, we verify whether the change is permanent by checking whether the difference between the time series of the potential SMART attribute and the corresponding time series of the same attribute in the absence of the observed change at time point t is significant. We do this as follows. First, let the time series $\Gamma_t = (s_t, \cdots, s_p)$ denote the subsequent values recorded for the potential SMART indicator S_i starting from the timestamp t to the time of the replacement p.

We then compare this time series with the one that would have been observed for this indicator in the absence of a significant change on day k, noted $\Psi = (\tilde{s}_{k+1}, \cdots, \tilde{s}_p)$. If there is a significant difference between these two, then the indicator values are correlated with the disk failure where Ψ, the synthetic control, is generated via time-series modeling.

More specifically, we compute the posterior distribution of Ψ, $p(\tilde{\mathbf{s}}_{(t+1):p} \mid \mathbf{s}_{1:(t)}, \mathbf{x}_{1:p})$ given the value of the series in the pre-change period $\mathbf{s}_{1:t}$ along with the values of the control time series $\mathbf{x}_{1:p}$ using a Bayesian structural time-series model. The control time series is a sample of the values of the target SMART attribute collected for a healthy disk. Finally, the target SMART attribute is indicative of a disk replacement if the probability distributions of the actual time series measured after the detected change point and the synthetic one generated based on the values of a healthy disk are significantly different. We assess the difference via hypothesis testing [81]. Formally, let Γ_k and Ψ be samples generated from unknown distributions P and Q, respectively. As in any hypothesis testing, we have a null hypothesis H_0 – associated with the theory we want to disprove and an alternative hypothesis H_1 associated with theory we want to prove. The hypothesis to test for our case is the following:

$$\begin{cases} H_0 : P = Q \\ H_1 : P \neq Q \end{cases} \tag{6.2}$$

Then we check whether we can reject the null hypothesis H_0 that the two probability distributions P and Q are equal with high confidence.

6.2.2 Compact time series representation

The previous step results in the set of relevant SMART indicators for the disk replacement problem. Now we move to the next step, whose goal is to provide a compact (reduced to a single data point), but highly informative representation of the time series of each indicator that can readily be employed in the predictive model.

There are several observations that hint the necessity of a compact representation for the time series data: (i) Each daily observation on its own is not enough - we need to consider a time frame longer than a day - this is because the single day record is not stable (it may change from a bad state to a good state) due to the recovery mechanisms embedded in the disk. (ii) Also, if we considered as observations for the failed class only the entries from the last day of the life of the disk then the model will not be able to predict replacement in advance as it can only recognize the instances when drive fails not before.

Therefore, we use a *window* to split the *raw data set* into segments. We aggregate each of the relevant time series to a single value using *exponential smoothing* over a specific time window. This way, we assign the highest weights to the most recent observations and exponentially decreasing weights to the remaining observations as they get older. Intuitively we expect that the observations closer to the time point of the disk replacement are more informative compared to the older ones. The raw data sequence is represented by Y_t, with Y_t starting at time $t = 0$, and the output of the smoothing procedure is denoted by S_t. Formally, for every indicator variable we have:

$$S_t = \alpha \cdot Y_t + (1 - \alpha) \cdot S_{t-1} \tag{6.3}$$

In the equation above, the smoothed value at time t, S_t is computed recursively based on the observation at time t and the smoothed value at time $t - 1$.

$$\begin{aligned} S_t &= \alpha Y_t + (1-\alpha) S_{t-1} \\ &= \alpha Y_t + \alpha(1-\alpha) Y_{t-1} + (1-\alpha)^2 S_{t-2} \\ &= \alpha \left[Y_t + (1-\alpha) Y_{t-1} + (1-\alpha)^2 Y_{t-2} + \cdots + (1-\alpha)^{t-1} Y_1 \right] + (1-\alpha)^t x_0. \end{aligned}$$

When fixing the width of the window to a value k (instead of smoothing over all the observations starting from $t = 0$), S_t becomes the weighted average of a certain number of the past observations up to Y_{t-k}.

A smaller value of k causes a weaker smoothing effect which enables higher sensitivity to new changes in the data. The parameter α controls the speed at which the older observations are dampened. A large α is used for assigning lower weights to observations from the more distant past.

For each relevant SMART attribute, the width of the time window used in the smoothing process is chosen as *the median of the distribution of the time stamps of their corresponding significant change* computed as described in Section 6.2.2.

6.2.3 Class balancing via informative downsampling

The data to be used in the predictive model is highly imbalanced, as only a small percentage of all disks are replaced over time. Since classification algorithms are typically optimized to maximize the overall accuracy, when trained using imbalanced datasets they often exhibit poor predictive performance. To address this issue, we balance the training dataset for our predictive model by using a representative subset of the data for the dense class – in our case the healthy disks. This representative subset is chosen such that it comprises the most informative samples with low or no redundancy. We achieve this by clustering the observations pertaining to the healthy disk set into k clusters using the K-means clustering algorithm [112]. Next, for each cluster, we select the data points closest to the respective cluster centroid as representatives for the healthy disk class. Finally, we generate a balanced training dataset by choosing k close to the number of samples available for the replaced disks. Using this technique is a pretty common way to tradeoff between using the complete dataset which might give us a poor or overfitted model and randomly sampling some data which might lead to underfitting.

6.2.4 Classification for disk replacements

In the following we will describe the classification algorithm we use for deciding whether a disk needs to be replaced or not.

In the final step of our approach we fit a model that utilizes the training dataset generated in the previous step and provides high quality disk replacement predictions for new, unseen data. Formally, let $D = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ denote the training dataset, where $\mathbf{x_i} \in X$ is a multivariate temporal observation. In this data, x_i is not the raw data point anymore but it is the aggregate value between time points t_{i-k} and t_i (using procedures described in Sect. 6.2.2 and Sect. 6.2.3) for the set of relevant SMART attributes, and y is a binary response variable ($y \in \{0, 1\}$) representing the replacement outcome. We want to learn a function $h\colon X \to \{0, 1\}$ that minimizes a *loss function* [87] $\ell(h(\mathbf{x}); y)$ that quantifies the prediction quality. Intuitively the goal is to train a model that correctly predicts whether a disk needs replacement ($y = 1$) or not ($y = 0$).

We tackle this problem using an ensemble model – the regularized greedy forests (RGF) [123] approach that is a powerful, non-linear classification method. Ensemble models consist of multiple base learners (e.g., decision trees) which are trained on random selections of the input variables and on different bootrstrap samples from the training data. For producing a class label, the input feature vector is fed to each base learner in the ensemble, starting from the root, until it gets to the leaves. As each tree provides a class prediction, the class label is computed through a voting mechanism (the class label with most votes is returned).

We show that, for this task RGF delivers better quality predictions compared to other tree ensemble based methods such as gradient boosted decision trees (GBDT) [89] or random forests [100] and also outperforms other classification methods such as SVM [42], or logistic regression [51].

The RGF algorithm is a variation of GBDT in which the structure search and the optimization are decoupled. More specifically, the main differences are the following:

- RGF introduces an explicit regularization term, $R(h)$, that takes advantage of individual tree structures.

$$\hat{h} = argmin_{h \in H}[\ell(h(\mathbf{x}); y) + R(h)] \tag{6.4}$$

- RGF employs a *fully-corrective* greedy algorithm which iteratively modifies the weights of all the leaf nodes (decision rules) currently obtained while new rules are added into the forest by greedy search. Here, an explicit regularization is also included to avoid overfitting and very large models.
- RGF utilizes the concept of *structured sparsity* [69] to perform greedy search directly over the forest nodes based on the forest structure.

The general framework of RGF is given in Algorithm 6.2 which we describe in the following. F represents a forest, and each node v of F is associated with the pair (b_v, a_v), where b_v denotes the nonlinear function – which we also refer as basis function of node v, and a_v the weight assigned to this node. The model of F is given by $h_{F(x)} = \sum_{v \in F} a_v b_v(\mathbf{x})$ with $a_v = 0$ for any internal node v.

In this setting, the regularized loss specified in Eq. 6.4 is a function of F: $Q(F) = \ell(h_F(\mathbf{x}), y) + R(h_F)$. Further, $S(F)$ represents the set of all forest structure-changing operations (i.e. the split of a node or the addition of a new tree) applicable to F.

Algorithm 6.2 Regularized Greedy Forest framework

```
$F \leftarrow \{\}$
while stopping criterion not met do
    Fix weights and adjust forest structure $s$:
    $\hat{s} \leftarrow argmin_{s \in S(F)} Q(s(F))$ (the optimum $s$ that minimizes $Q(F)$ among all the structures that can be obtained by applying one structure-changing operation to $F$).
    if some criterion is met then
        Fix the structure and change the weights in $F$ s.t. the loss is minimized in $Q(F)$ (it can be optimized using a standard procedure (such as coordinate descent) if the regularization penalty is standard e.g., L2-loss
    end if
end while
Optimize leaf weights in $F$ to minimize loss in $Q(F)$
return $h_F(\mathbf{x})$
```

6.2.5 Transfer learning

As illustrated in Figure 6.5, the data collected from different disk models are different. We observe that different models of a single disk manufacturer have similar SMART reporting but different distributions of the values reported for the SMART attributes. Therefore, utilizing an existing predictive model created on the training data of a specific disk model will not deliver the optimal predictive performance when directly applied on the data collected from a different disk model from the same manufacturer. In data mining, this problem is referred to as *sample selection bias, covariate shift or dataset shift*. Therefore, we apply a transfer learning approach, in order to be able to use a prediction model trained on specific disk model for a new disk model of the same manufacturer.

Note that such an approach is valuable as it transfers the expert knowledge gathered over the years through historic data to a new disk model from a given manufacturer of interest. We tackle the described dataset shift issue we have across disk models of a given manufacturer as follows. We leverage the unlabeled data for the target (new) disk model to conduct a sample selection de-biasing, as described in Algorithm 6.3.

The idea behind Algorithm 6.3 is to train a classifier that can rank the observations linked to a specific disk model based on their similarity to observations pertaining to the target disk model – the disk model for which we have labeled data. This corresponds to the function f in our algorithm. Using this function, we can select a sample from the labeled disk data which is more representative for the disk population pertaining to the model for which we don't have labeled data. This data is then used to build a disk replacement prediction model for the disk model which did not have labeled data.

As a paranthesis, this is a central idea to transfer learning – borrowing examples. In [84], the authors exemplify this concept in the context of building a *sofa detector*, where they borrow examples of other related classes, such as annotated images of armchairs. This approach enables us to sample the observations from the original disk model (which are already labeled) that are more representative for learning the class labels for the target disk model, i.e. that matches the distribution of the original disk model to the target disk model. Learning a predictive model using a training sample that reflects the distribution of the new disk model results in higher quality predictions.

6.3 Evaluation

In the following, we present our experimental setup and the results obtained in each step of our approach.

6.3.1 Data description and experimental setup

Our analysis is based on the Backblaze dataset [2]. The set contains public data collected from 50984 hard disks, monitored over 27 months (April 2013 to June 2015), and each entry corresponds to the sensor data for a day. Each entry of the dataset contains the following: (1) timestamp, (2) disk serial number, (3) disk model, (4) disk capacity, (5) failure - '0' if the drive is alive and '1' if the disk has been replaced the following day, and (6) SMART monitoring data. From the disk models, we extract the manufacturers and we restrict our

Algorithm 6.3 Transfer learning for different models

```
Input: $D_{DM_1} = \{x_i, y_i\}_i^n$, the labeled data collected from disk model 1, and $D_{DM_2} = \{x'_i\}_i^m$ the unlabeled data from disk model 2, where $x_i, x'_i$ represent features and $y_i$ are the labels.
1. Let $D_{DM_1} = \{x_i, y_i\}_i^n$ be the labeled data collected from disk model 1, and $D_{DM_2} = \{x'_i\}_i^m$ be the unlabeled data from disk model 2.
2. Let $D_{aug} = \{x_i, “DM_1”\}_i^n \cup \{x'_i, “DM_2”\}_i^m$. Note that we left out the labels $y_i$ from $D_{DM_2}$.
3. Use $D_{aug}$ to learn a function $f : X \rightarrow [0, 1]$, (e.g. logistic regression), such that $f(x)$ represents the probability of a disk being of type $“DM_2”$.
4. Sample a subset $D_{sub}$ from $D_{DM_1}$ such that probability of selecting a point $x = f(x)$.
5. Use $D_{sub}$ to learn a function $g : X \rightarrow [0, 1]$ (call the procedure in Algorithm 6.2) such that $g(x)$ represents the probability of a disk of type $DM_2$ needing replacement.
Output: Predictive model for disk replacement for disk model 2.
```

analysis to Hitachi[2] and Seagate[1] due to the fact that for the other manufacturers, there are only few samples in the dataset, or poor population of the SMART parameters. We also exclude all monitoring data between April 2013 and January 2014, as more than 70% of the SMART parameters are not collected. Thus the dataset we consider is gathered over 17 months.

First, we build and evaluate the predictive model described in this chapter for Seagate ST4000DM000 (SgtA) and Hitachi HDS722020ALA330 (HitA). Then, we evaluate the transfer learning approach on Seagate ST31500541AS (SgtB) and Hitachi HDS5C3030ALA630 (HitB), respectively. Further details on the data are presented in Table 6.1.

	Original		**Post-aggregation**	
	H	R	H	R
SgtA	247524	543	17769	457
SgtB	30859	375	2188	227
HitA	75618	150	4616	115
HitB	74040	80	4662	73

Table 6.1: Healthy (H) vs. replaced (R) disks in the raw dataset and after data cleaning and aggregation for Hitachi and Seagate.

6.3.2 Feature selection – computation of relevant SMART attributes

First, note that for each SMART indicator there are two values recorded – the raw value, and the normalized value. The raw value often represents counts or a physical unit (e.g., degrees Celsius or milliseconds). The normalized values are a vendor specific mapping of the

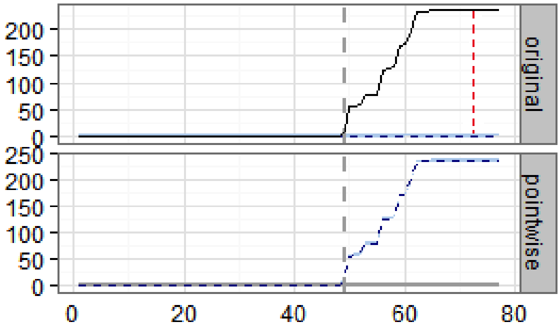


Figure 6.2: Differences between the forecasted and the observed values for SMART_187_raw.

raw values such that, typically, higher values indicate healthy disks with some exceptions (e.g., the *temperature* attribute for Seagate models). A detailed breakdown of the SMART parameters is found in [14].

As presented in Section 2, we find the SMART indicators relevant for disk replacements via changepoint analysis.

In Figure 6.2, we illustrate the evolution of the time series for a parameter, namely, SMART_187_raw (reported uncorrectable errors) over 80 days for SgtA disk. Note that after 50 days of usage the disk starts to accumulate uncorrectable errors, up to the point where a replacement is necessary. Since there is a significant difference between the time series observed on days 1 to 50 and the one observed on days 50 to 80, our algorithm detects a changepoint 30 days before the disk has been replaced.

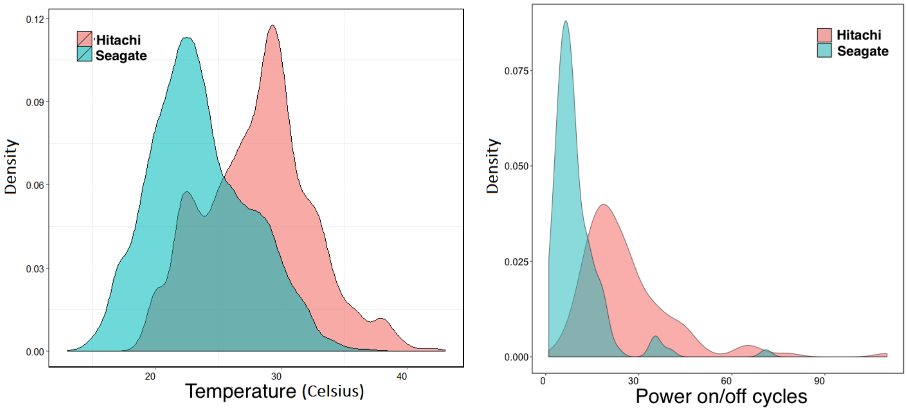


Figure 6.3: Distribution of the temperature and of the power on/off cycles across the replaced disks for Hitachi and Seagate.

We perform the changepoint analysis for both Seagate and Hitachi disks and present the results in Table 6.2. For each of the considered SMART parameters we report the percentage of drives for which a correlation with disk is observed.

For Seagate, 63% of the replaced drives correlate with an increase in SMART_193_raw

	SgtA		HitA	
	Ratio	Inp.	Ratio	Inp.
SMART_1_norm	23%	✓	28%	✓
SMART_1_raw	2%	✓	15%	✓
SMART_3_norm	–	×	13%	✓
SMART_3_raw	–	×	15%	✓
SMART_5_norm	2%	✓	22%	✓
SMART_5_raw	19%	✓	31%	✓
SMART_7_norm	14%	✓	–	×
SMART_7_raw	26%	✓	–	×
SMART_183_norm	0.5%	×	–	×
SMART_183_raw	0.5%	×	–	×
SMART_184_norm	1%	✓	–	×
SMART_184_raw	1%	✓	–	×
SMART_187_norm	21%	✓	–	×
SMART_187_raw	21%	✓	–	×
SMART_188_norm	0%	×	–	×
SMART_188_raw	10%	✓	–	×
SMART_189_norm	1%	✓	–	×
SMART_189_raw	1%	✓	–	×
SMART_190_norm	2%	✓	–	×
SMART_190_raw	2%	✓	–	×
SMART_193_norm	10%	✓	–	×
SMART_193_raw	63%	✓	–	×
SMART_194_norm	2%	✓	31%	✓
SMART_194_raw	2%	✓	2%	✓
SMART_196_norm	–	×	20%	✓
SMART_196_raw	–	×	26%	✓
SMART_197_norm	5%	✓	4%	✓
SMART_197_raw	27%	✓	22%	✓
SMART_198_norm	6%	✓	–	×
SMART_198_raw	27%	✓	–	×
SMART_199_norm	0%	×	–	×
SMART_199_raw	0.5%	×	–	×
SMART_240_norm	0.5%	×	–	×
SMART_240_raw	21%	✓	–	×
SMART_241_norm	0%	–	–	×
SMART_241_raw	15%	✓	–	×
SMART_242_norm	0%	×	–	×
SMART_242_raw	19%	✓	–	×

Table 6.2: SMART correlation frequencies for SgtA and HitA. A ✓ indicates the predictor is included in the classification task.

(the load cycle count), and between 19 and 26% of them also correlate with SMART_7_raw (seek error count), SMART_1_normalized (read error rate), SMART_240_raw (transfer error rate), SMART_197_raw (nr. of pending sectors), SMART_198_raw (uncorrectable sector count), SMART_187_raw (number of uncorrectable errors), as well as SMART_5_raw (reallocated sector count).

For Hitachi, only some of these SMART parameters are indicative of drive replacements. Among the top correlated indicators for Hitachi (30-47%), we note SMART_196_raw (reallocation event count), SMART_194_normalized (internal temperature), SMART_5_raw

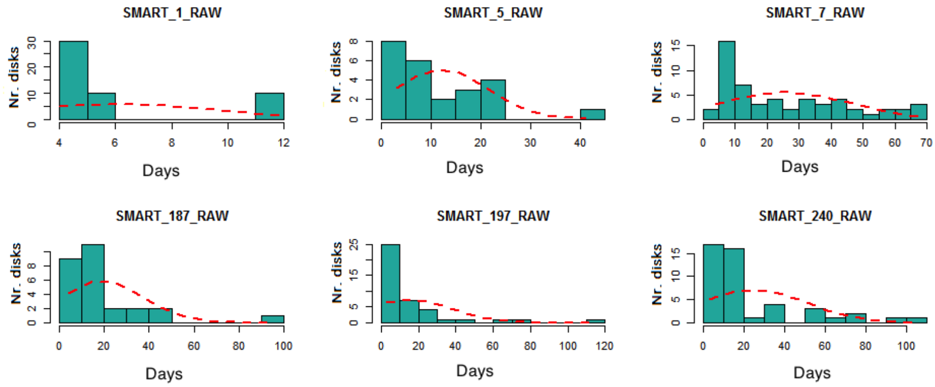


Figure 6.4: Distribution of the number of days before replacement when the changepoint was observed.

and SMART_197_raw.

We also note that it's mostly the raw values of SMART indicators that correlate with impending replacements. This is expected, since, the normalized values are computed based on generous thresholds, where a replacement can also occur *before* the normalized value changes at all.

The changepoint analysis also shows that some SMART indicators correlate stronger with the replacements of the Seagate model than with those of the Hitachi model, and vice versa. We discuss these differences in the context of SMART_194 (the disk internal temperature). Relative to temperature, as we can see in Table 6.2 31% of the Hitachi replaced disks correlate to disk replacement, compared to only 2% for Seagate. By correlation we denote that a significant change in a given attribute occurred before the replacement occurred. We attribute this to the overall higher temperatures that characterize the Hitachi disks, as shown through the comparative plots in Fig. 6.3. Although the distributions are similar, there is a clear shift towards higher temperature for Hitachi, by 5 to 10 degrees Celsius.

6.3.3 Compact time series representations

Fig. 6.4 shows the distribution of the number of days before replacement when the changepoint was observed for six SMART indicators (SMART_1_raw – read error rate , SMART_5_raw – the number of reallocated sectors, SMART_197_ raw – the number of pending sectors, SMART_187_ raw – the reported uncorrectable errors, SMART_7_ raw – the seek error count and SMART_240_ raw – the transfer error rate). Note how the median values are different from one predictor to another. We use these median values to select the length of the window of the time series when creating the compact representation.

We notice that on the one hand, for the number of reallocated sectors and of pending sectors (SMART_5_raw and SMART_197_raw), considering the last 12 and 10 days, respectively, before the disk replacement in the predictive model is sufficient. This is because an increase in either of these two parameters indicates that a remapping operation is necessary (i.e., data from the defective sector is transferred to a spare area). As shown

		RGF		**GBDT**		**RF**		**SVM**		**LR**		**DT**	
		SgtA	**HitA**	**SgtA**	**HitA**	**SgtA**	**HitA**	**SgtA**	**HitA**	**SgtA**	**HitA**	**SgtA**	**HitA**
Replaced	P	0.98	0.84	0.97	0.82	0.93	0.82	0.93	0.72	0.73	0.72	0.89	0.74
	R	0.98	0.79	0.96	0.78	0.94	0.76	0.95	0.65	0.81	0.59	0.87	0.61
	F	**0.98**	**0.81**	**0.96**	**0.80**	**0.94**	**0.79**	**0.94**	**0.68**	**0.77**	**0.65**	**0.88**	**0.67**
	Sd	**0.01**	**0.02**	0.01	0.04	0.05	0.08	0.02	0.05	0.07	0.1	0.04	0.03
Healthy	P	0.99	0.93	0.98	0.92	0.97	0.92	0.97	0.87	0.89	0.85	0.94	0.86
	R	0.98	0.95	0.98	0.94	0.96	0.93	0.96	0.90	0.85	0.90	0.95	0.91
	F	**0.98**	**0.94**	0.98	0.93	0.97	0.92	0.96	0.88	0.87	0.87	0.94	0.88
	Sd	**0.01**	**0.02**	0.02	0.03	0.04	0.05	0.02	0.04	0.08	0.05	0.02	0.02

Table 6.3: Precision, Recall, F-score, Deviation of different classifiers - median on 100 runs , each of which using randomly-drawn training and test data points

in [54], a drive which has had any reallocated or pending sectors at all is significantly more likely to fail in the nearest future. An even stronger indicator for replacement is the read error rate (SMART_1_raw), which represents the rate of hardware read errors while reading data from the disk surface. It indicates a critical problem with the read/write heads (e.g., head resonance or contamination, broken head, etc.) and in most cases an imminent failure. From our dataset, we find that for this predictor, looking back only 4 days in the past is sufficient for the predictive model. Similarly, a drive with uncorrectable errors as indicated by SMART_187_raw (i.e., cannot be recovered using hardware ECC) may need to replaced about 15 days after the event.

On the other hand, for the seek error rate and for the transfer error rate, SMART_7_raw and SMART_240_raw respectively, the algorithm considers the past 25 days in the aggregation process. Both are indicative of malfunctions of the magnetic heads, but hinder performance primarily and lead to failures only in a second phase. For instance, seek errors hint at the drive overshooting or undershooting the correct track when it moves the heads. This implies it will need to perform another seek to acquire the track before it can read or write data.

6.3.4 Classification for disk replacements

Since only 2.5 to 3% of the disks for both SgtA and HitA models are replaced, the classifier will be biased towards the healthy drives. Thus, we downsample the healthy class to an amount that is close to the size of the replaced class. We choose to downsample to 1000 for SgtA and to 500 for HitA. These values are chosen based on the error estimate of the Regularized Greedy Forest (RGF) classifier. Consequently, we run K-means with 100 and 50 clusters as inputs and subsequently for each cluster we select the top 10 data points closest to the centroid of each cluster.

The SMART attributes used to build the predictive model correspond to the rows that have non-null entries and values higher than 1% in Table 6.2. In essence, for the Seagate model we use 26 SMART predictors and for the Hitachi model only 12. This discrepancy in the amount of predictors we feed to the Seagate model versus the Hitachi one will be reflected in the difference in performance of the classifiers.

To evaluate the classifier's performance, we measure precision, recall and F-score as defined below, for both replaced and healthy classes. Precision (P) is used to measure the ability of the classifier to correctly identify disks at risk. Recall (R) measures the classifier's sensitivity, i.e the the ability of the classifier to capture all replaced disks. A higher recall is equivalent to minimizing the number of false negatives (i.e., the number of disks labeled as healthy when they were actually replaced). The F-score is the combined score between precision and recall, or the weighted harmonic mean.

$$P = \frac{tp}{tp + fp} \qquad R = \frac{tp}{tp + fn} \qquad \text{F-score} = \frac{2PR}{P + R}$$

In the definitions above, *tp, tn* stand for true positives and negatives respectively, whereas *fp, fn* stand for false positives and negatives respectively. We perform a systematic comparison with different classifiers. In order to assess the goodness of each classifier we run the following experiment. We generate 100 random splits of the dataset into training (80%) and test (20%). The split is done through sampling without replacement of a disk model from the population of disks. For each such split, we train the model on the training set and evaluate it on the test set and compare the performance of RGF with that of other classifiers such as Random Forests (RF), Gradient Boosted Decision Trees (GBDT), Support Vector Machines (SVM), Logistic Regression (LR) and decision trees (DT).

For a fair comparison, we have performed parameter tuning (grid search on parameter space to maximize accuracy) for all the parametric classifiers. For RGF, we have obtained the best performance when using the L2 regularizer. There were two L2 regularization parameters to be tuned: one for weight optimization – which was set to 1 and and the other for tree learning which was set to 0.005 following the results obtained in the grid search. The model size in terms of the number of leaf nodes in the forest was set to 10000 leafs. The results are given in Table 6.3.

In case of the replaced disks, the model exhibits better prediction quality for Seagate, where we have 4x more data points and 2x more non-null SMART indicators, with 98% accuracy and 1-2% error over 100 runs. The precision, recall and F-score for Hitachi are lower by 14-19% and the error is higher – 2%, due to a smaller number of drives in the set and 60% less predictors.

For the healthy class, the model achieves similar performance for Seagate as on the replaced class, with 99% precision and 98% recall and F-score. In the case of Hitachi, the model achieves better performance in discriminating the healthy drives as compared to the faulty ones, by 15% on average. We attribute this boost in accuracy to the fact that healthy disks are easier to identify due to the lower variability in the values of the SMART parameters recorded for them.

6.3.5 Transfer learning

Figure 5 illustrates the covariate shift for various relevant predictors between different disk models from the same manufacturer. This demonstrates that if we want to reuse the data from one model to build a predictive model for another one, we need to employ appropriate transfer learning.

To illustrate the usefulness of our transfer learning approach we compare the models trained and evaluated on SgtA and HitA with the models built with transfer learning and tested on SgtB and HitB, respectively. The results are given in Table 6.4.

The gain in predictive performance achieved from using transfer learning when building the new disk models (SgtB and HitB, respectively) compared to directly evaluating the base

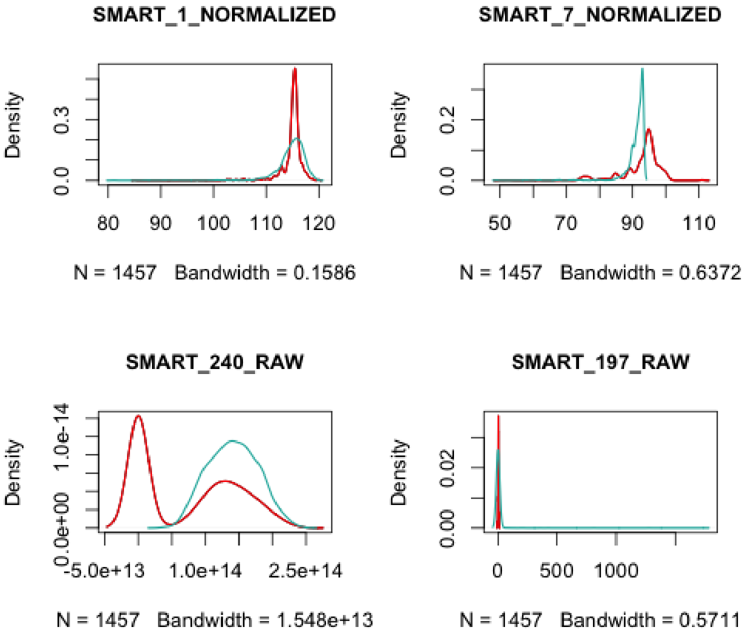


Figure 6.5: Covariate shift for the two Seagate models

model (trained on SgtA and HitA, respectively) on the new disk models is shown in Table 6.4. We obtain 50% increase in the accuracy of the model with transfer learning compared to the accuracy of the base for model SgtA. Also for Hitachi, transfer learning boosts the accuracy of the model by 20%.

6.3.6 Comparison with human designed replacement policies

As mentioned in Section 6.1, a drive is labeled as failed when it stopped working, it is non-responsive to commands, the RAID system reports that the drive cannot be written or read, or it shows evidence of failing soon [9]. Currently, datacenter administrators at Backblaze only focus on a very small set of SMART indicators (5, 187, 188, 197, 198) [9]. However, we illustrate that if one were to do proactive replacement using only this small subset of indicators, the number of disks one could correctly identify drops by almost 50%.

In order to mimic a set of such replacement rules we train a decision tree on the aforementioned subset of SMART indicators. We report the results in Table 6.5.

Note the differences in recall for SgtA and HitA for our model (98% and 81% respectively) compared to a simple rules based model (53% and 44% respectively) – see Table 6.5 vs. Table 6.3. Our solution employs powerful learning methods, leverages a larger set of relevant SMART attributes and hence has numerous advantages: captures a higher amount of the failure patterns of disks (high recall), it has low false alarm rate, early detection of disks that need to be replaced, and enables transferring the knowledge acquired by ex-

		SgtB		HitB	
		Base	Tr. Learn.	Base	Tr. Learn.
Replaced	P	0.65	**0.90**	0.53	**0.76**
	R	0.52	**0.82**	0.84	**0.78**
	F	0.58	**0.86**	0.65	**0.77**
Healthy	P	0.89	0.96	0.92	0.83
	R	0.93	0.98	0.73	0.82
	F	0.91	0.97	0.81	0.83

Table 6.4: Precision, recall and F-score to illustrate the importance of transfer learning

	DT on the reduced subset		
		SgtA	HitA
Replaced	Precision	0.95	0.66
	Recall	**0.53**	**0.44**
	F-score	0.68	0.51
	Sd	0.06	0.15
Healthy	Precision	0.70	0.84
	Recall	0.98	0.96
	F-score	0.81	0.92
	Sd	0.02	**0.12**

Table 6.5: Simple decision tree with (insufficient but commonly used) subset of SMART indicators

pert data center administrators on specific disk models to new disk models from the same manufacturer.

6.3.7 Early vs. late replacement detection

While one would prefer to use as much as possible from the lifespan of a disk, being able to detect an impending failure early on allows administrators to plan properly for replacements. Therefore, we evaluate how many of the replaced disks our model correctly captures based on snapshots of the SMART indicators taken 1, 3, 10 and 30 days prior to the actual replacement. We expect this amount to be higher when using the snapshots closer to the failure event, since SMART attributes would become more indicative of the impeding replacement, thus making the model more accurate. Figure 6.6 shows the results obtained for

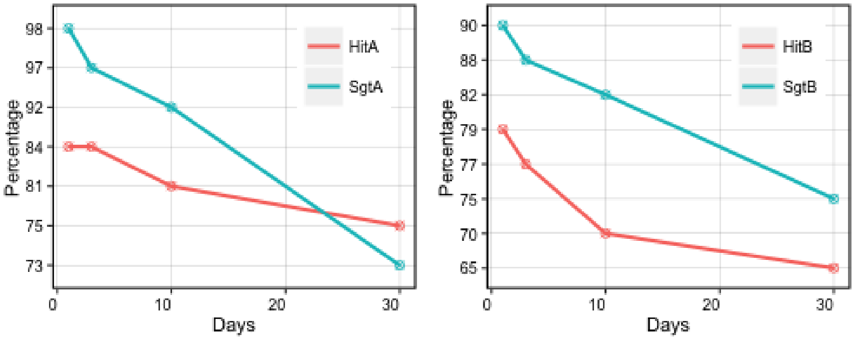


Figure 6.6: Percentage of disks correctly predicted as replaced on snapshots taken 1,3,10 and 30 days before the actual replacement event.

Line	Model	Rule	Output	Conf
1	Seagate	If $SMART\_197\_raw < 2$ and $SMART\_188\_raw > 0$ and $SMART\_1\_normalized \in [0, 117)$	Healthy	100%
2	Seagate	If $SMART\_197\_raw \geq 2$	Replace	100%
3	Seagate	If $SMART\_197\_raw < 2$ and $SMART\_188\_raw > 0$ and $SMART\_1\_normalized > 117$	Replace	80%
4	Seagate	If $SMART\_197\_raw < 2$ and $SMART\_188\_raw = 0$ and $SMART\_187\_normalized < 100$ and $SMART\_240\_raw < 14780$ billion	Replace	97%
5	Hitachi	If $SMART\_197\_raw > 1$ and $SMART\_3\_raw > 626$	Replace	100%
6	Hitachi	If $SMART\_197\_raw > 5$ and $SMART\_3\_raw < 626$ and $SMART\_5\_raw > 17$	Replace	92%
7	Hitachi	If $SMART\_197\_raw > 1$ and $SMART\_3\_raw < 626$ and $SMART\_5\_raw < 17$	Replace	100%
8	Hitachi	If $SMART\_197\_raw < 1$ and $SMART\_5\_raw < 7200$ and $SMART\_3\_raw > 629$ and $SMART\_1\_raw \in [0, 109]$	Healthy	97%

Table 6.6: Examples of rules extracted from a decision tree model trained on the Seagate and Hitachi.

both Seagate and Hitachi. On the one hand, for SgtA, the model correctly identifies 97% of the faulty disks 3 days prior, 92% of them 10 days in advance and, then, sees a more significant decrease to 73% for up to 30 days in the past. On the other hand, for HitA, the decrease in percentage is less dramatic, as 84% of replaced disks are predicted 3 days before the event and 75% of them 30 days prior. The ratio late vs. early detection is similar for the model built with transfer learning for SgtB and HitB. Compared to the current predictive model implemented in SMART, which mostly warns about a disk failure in the last minute, our model has a major advantage. For both Seagate and Hitachi, an administrator can identify 73 to 75% of the disks to replace a month in advance, which provides her/him with the possibility of planning the replacement in advance, while still using the drives for another 25-30 days.

6.3.8 SMART indicator rules

Once we have the balanced training dataset comprising the informative SMART attributes in a compact form (this is obtained by running steps 1 through 3 in Algorithm 1) we can use it to fit a decision tree and extract a set of rules that can be used to predict disk replacements. We present rules that are of the form of the underlying learner, more specifically:

$$Rule(\mathbf{x}) = \prod_j \mathscr{I}(\mathbf{x}[i_j] \leq t_j) \prod_k \mathscr{I}(\mathbf{x}[i_k] > t_k) \tag{6.5}$$

where $\{(i_j, t_j), (i_k, t_k)\}$ represent a set of (smart attribute index, threshold) pairs and $\mathscr{I}(z) = 1$ if z is true and 0 otherwise. The rules provide a detailed insight into the information on the relation among the relevant SMART attributes and the disk replacements available in our training dataset.

Examples of such rules for both SgtA and HitA are provided in Table 6.6, together with the predicted outcome for the disks adhering to these rules and the prediction confidence. Each rule is composed of one or more single SMART parameter conditions. The fewer conditions, the higher the correlation of the corresponding SMART indicators to the healthy or faulty state of the disks. For instance, in the case of Seagate, the second rule states with 100% confidence that if SMART_197_raw is at least 2, that is the disk has at least two pending sectors, it should be replaced. On the other hand, if its value is below 2, the outcome of the prediction can go both ways, depending on other parameters' values. As an example, consider rules 1 and 3, in which the one indicator that changes the prediction output is the normalized read error rate (SMART_1_normalized). As predicted by our decision tree model, if the number of read errors exceeds 800 thousand, then the disk status is unhealthy.

For Hitachi, the majority contain at most three conditions. As seen in lines 5–8, an indicative combination of attributes is (SMART_197_raw, SMART_3_raw). Line 5 shows that if the number of pending sectors is higher than 1 and the average time spent during a spin up operation exceeds 626 milliseconds, the model predicts an impending faulty state of the disk with 100% confidence. However, if the spin up time is lower, it also considers the number of reallocated sectors in its decision and determines that even with less than 17 such sectors, the disk should be replaced (Line 7). A healthy state is predicted with 97% confidence when the disk has no pending and at most 7200 reallocated sectors, a slow spin up time (i.e., higher than 629 milliseconds) and less than 109 read errors.

Comparing Seagate and Hitachi, we make the following remarks. First, the primarily important SMART indicators are sowewhat different. The pending sector count and the read error rate seem to be model and even manufacturer agnostic, while the command timeout (SMART_188), the average spin up time and the reallocated sectors count are disk model-specific. Second, we note a very large difference in the number of read errors that determine a faulty disk state. For Seagate, this threshold is in hundreds of millions, while for Hitachi they are 6 orders of magnitude lower. We attribute this gap to the fact that this indicator is vendor specific, and therefore a comparison across manufacturers is not feasible.

6.4 Deployment

Our predictive model has been designed foresee when a disk replacement is needed and allow for more efficient, scheduled maintenance processes in place of the inefficient, reactive repairs procedures. Especially for enterprise workloads, where at least three nines (99.9%) data availability needs to be guaranteed, current storage systems (e.g., IBM DS8000/8800) use an incorporated Predictive Failure Analysis (PFA) component jointly with RAID (Redundant Array of Independent Disks) to anticipate certain forms of disk failures. Such a model is threshold-based and uses only read and write error statistics to nominate disks for replacement. As shown in Table 6.5, thresholds lead to less accurate replacement decisions, therefore integrating our approach in the PFA enables more accurate replacement strategies. RAID on the other side is an implementation of the idea of combining several (often cheap) drives into a single volume with higher capacity. It was thought for redundancy such that the data remains available in the presence of a failure. There are multiple types of RAID implementations [12], the most common of which being RAID 5 and RAID 6.

We exemplify how our component could be integrated for rebuilding a RAID 5 array when a disk is signaled as likely to fail, through smart rebuild [13]. An early signal enables the disk to still be available for I/O operations, and thus kept in the array, rather than being rejected because of a standard rebuild. A spare disk can be either used from the array or brought in if none is available. The signaled drive and the spare are put in a temporary RAID 1 (full mirroring), therefore allowing the duplication of the faulty drive onto the spare, rather than performing full RAID reconstruction which slows down the entire array's performance. The spare becomes a regular member of the array and the signaled-to-be-faulty disk can be safely removed from the disk, without any risk of data loss.

By using such models that can detect failures early in advance and have low ratios of false positives, the array would never go through a time consuming n-1 stage where it would be exposed to complete RAID failure if another drive fails in the meantime. Thus, the benefits – time saving and increased availability – are substantial.

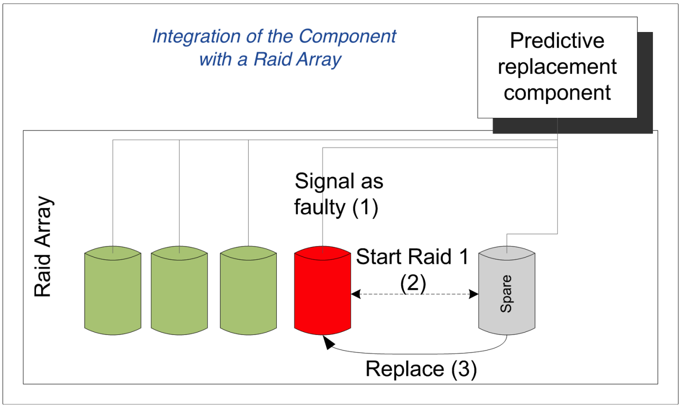

Integration of the Component with a Raid Array
Predictive replacement component
Raid Array
Signal as faulty (1)
Start Raid 1 (2)
Spare
Replace (3)

Figure 6.7: Integration of the predictive replacement component with storage arrays

In Figure 7, we show how our predictive component could be used in interaction with the RAID array and the steps necessary for the automatic rebuild: the predictive component signals a disk with impending failure (1), mirroring is started on the spare (2), unhealthy disk is replaced by its healthy mirror (3). The process falls back to RAID rebuild only if necessary.

Our failure model can be deployed in large scale environments (e.g data centers), provided that two conditions are fulfilled. First, the SMART parameters identified as relevant (see Table 6.2) should be continuously measured by the manufacturer. Second in order to learn such model for different manufacturers, even though failed disks represent only a small fraction compared to the healthy ones, in absolute terms, these need to be in the order of hundreds for the model to achieve precision and recall higher than 80%.

6.5 Related work

Researchers have performed a couple of large-scale studies on disk failures, the most notable ones pertaining to the authors of [28, 54]. They observed that the field replacement rates of drives are significantly higher than those in the technical datasheets – 2-10 times higher for disks aged less than 5 years and up to 30 times higher for disks between 5 and 8 years. They also demonstrate a significant overestimation of MTTF by the manufacturer. The authors also observed a continuous increase in the replacement rates, starting already in the second year of operation and a high correlation between the first error and a later disk failure. Our analysis also confirms these findings.

This line of research is complemented by works of [145, 62, 60, 92, 163] where the focus is on building a predictive model for the timely discovery of impending disk failures. In [62], the authors employ Bayesian methods to model disk drive failures based on SMART data. First, they solve an anomaly detection problem (i.e., by looking back in the life-span of the drive and establishing if any of the previous observations is an anomaly). They achieve this by applying a mixture model based on naive Bayes clusters trained using expectation-maximization. Second, they train a naive Bayes classifier which predicts that a drive will fail if any of its snapshots are identified as anomalous or as failures. They evaluate the approach on a smaller dataset, consisting of 1936 drives, out of which only 9 were marked as failed, with a detection rate of up to 55% only.

The authors of [60] explore the capabilities of statistical tests such as the multivariate rank sum test to improve failure warning accuracy and lower false alarms. Their dataset is also fairly small with only 3744 drives (out of which 36 failures), coming from two different models and with each set containing at most 3 months of reliability design test data. The highest accuracies achieved were modest (40%-60%) at 5% anual failure rate. A different model for predicting failures is proposed in [92], comprising of an algorithm based on the multiple-instance learning framework and the naive Bayes classifier. The dataset used was again very small - only data from 369 drives.

There are several key differences between the aforementioned studies and ours. First, the number of disks we consider is significantly larger, with over 23000 drives. Second, our approach focuses on selecting the SMART indicators that correlate with disk replacements and proposes stable representations of the time series data for each disk as input to the predictive model. Last, but not least, some studies are based on monitoring data from drives used in accelerated life tests, whereas we rely only on field data collected when the disks where in actual use. The problem with data collected during testing in uniform controlled environments is that although it can be insightful in understanding the role of certain environmental factors, it has been shown to be not informative enough with respect to actual failure rates observed in the field [93].

Finally, we note that some manufacturers deploy the disks with embedded failure predictive models. However, these models are based on simple methods, such as threshold-based normalizations which according to field observations these models are built such that they avoid false alarms at the expense of a weak predictive power [28, 54, 74].

The computational complexity of the proposed approach is rather low, as most time is spent in the training phase (which may take a few seconds) while the prediction is very fast - within less than one second.

6.6 Conclusions

In this chapter, we presented a machine learning-based pipeline for predicting disk replacements, built and evaluated on real data from a large disk population from two different manufacturers. We demonstrate the ability of our model built using SMART data to predict disk replacements with high accuracy. A changepoint based selection and a compact representation of the time series data for the SMART indicators plugged into a RGF classifier achieves up to 98% accuracy in predicting replacements, 10-15 days in advance. As expected, such models are sensitive to the number of SMART attributes they learn from and the size of the available training data. Given that in our original dataset there were considerably less indicators with non-null values for Seagate, we were able to build a model with 24 attributes for Seagate contrary to 12 only for Hitachi. This together with the dataset size explains the 17% difference in accuracy for the two disk models – 98% and 81%, respectively. We also demonstrated how transfer learning can be used to reuse the information available in the labeled dataset for a disk model from a specific manufacturer to build a high quality predictive model for a new disk model from the same manufacturer with no available labeled data.

We believe that such high quality models have many practical benefits. First of all, they can be easily applied to any disk model or manufacturer as long as SMART data is collected. Second, they provide an automatic tool for the disk replacement problem that can be a valuable asset enabling the administrators to identify faulty disks in due time. Last but not least, the predictive models mitigate the reliability issues of storage service providers by allowing administrators to backup the data and plan the actual replacement in advance. Also note that all these benefits are achievable based only on data that is automatically collected from the disk and no extra effort is necessary.

6.6.1 Instantiation of the predictive pipeline to workflows

Since the work we described in this chapter was driven by a specific use case – prediction of disk replacement, we have highlighted its potential mostly with regard to the use case. However, its scope can be extended to other use cases, especially in the space of workflows.

Monitoring and analyzing process execution is important for understanding and anticipating the evolution of the process, as this can increase the effectiveness of the business operation and help in dealing with operational risk [136]. Broadly speaking, if we want to predict the next state (given there is a choice) or the outcome of a process instance, our pipeline requires the existence of a set of parameters that are considered to be indicative for the purpose and whose values can be fetched periodically. The particular characteristics of the data and of the problem at hand might make some of the *modules* in our pipeline redundant (e.g., assume we have a balanced dataset and downsampling is not necessary anymore) or it might be the case that additional modules might be necessary such as an *imputation* [26] module for the cases where we are confronted with the problem of missing data.

[1] Seagate is a trademark of Seagate Technology LLC.

[2] Hitachi is a registered trademark of Hitachi, Ltd., and/or its affiliates in the United States and other countries.

Attribute code	**Definition**
SMART_1	Read error rate – o the rate of hardware read errors that occurred when reading data from a disk surface
SMART_3	Spin-Up Time – average time of spindle spin up
SMART_5	Reallocated sectors count – raw value represents a count of the bad sectors found and remapped
SMART_7	Seek error count – rate of seek errors of the magnetic heads
SMART_183	SATA downshift error count or runtime bad block –nr. of data blocks with detected, uncorrectable errors
SMART_184	Count of parity errors which occur in the data path to the media via the drive's cache RAM
SMART_187	Reported uncorrectable errors – count of errors that could not be recovered using hardware ECC
SMART_188	Count of aborted operations due to HDD timeout
SMART_189	Count of times when a recording head is flying outside its normal range
SMART_190	Airflow temperature
SMART_193	Count of load/unload cycles into head landing zone position
SMART_194	Current internal temperature
SMART_196	Reallocation event count – count of attempts to transfer data from reallocated sectors to a spare area
SMART_197	Count of unstable sectors because of unrecoverable read errors
SMART_198	Uncorrectable Sector – count of uncorrectable errors when reading/writing a sector
SMART_199	UltraDMA CRC error count –count of errors in data transfer via the interface cable as given by ICRC
SMART_240	Transfer error rate – count of times the link is reset during a data transfer
SMART_241	Total LBAs written
SMART_242	Total LBAs read

Table 6.7: SMART attributes and their definitions.

Chapter 7

Concluding Remarks

Business processes management is comprised of several parts such as the process specification, the modeling, the analysis and the process execution. Observations collected on process execution might lead to iterations on the aforementioned steps.

In our thesis we focused on the analysis of processes from different perspectives. The static perspective when we are given the process, duration estimates and resource count assumptions and a dynamic perspective when the focus is on the data inputs and process execution logs.

We have devised new polynomial-time algorithms that extend the understanding of the timing behavior of business processes. First, we assumed that we have control over the choices made in the process at runtime and we asked whether there exists an execution of the process which meets a given deadline. This boils down to computing the minimum duration of an execution. We have shown that the minimum duration of an execution of a workflow graph executed by a single resource or by an unbounded number of resources can be computed in polynomial time.

For workflow graphs with probabilistic choice we studied different problems such as the probability of a deadline transgression to occur and the expected duration of an execution. We have proven that computing whether the probability of an execution with a single resource terminating before the deadline exceeds a given threshold is NP-hard. The hardness results carries over for the case when we assume an unbounded number of resources.

We showed that computing the expected duration of a workflow graph executed by a single resource can be achieved in polynomial time. In contrast to this result, when we assume an unbounded number of resources, we have proven that computing the expected duration is NP-hard even for regular graphs.

Often, there exists a gap in between what is sought in practice for business process management and the theoretical accomplishments in the Petri nets field. This motivated us to also performed a case study, on data obtained from a Dutch financial institution. In this case study, we mined the logs for the underlying business process, then translated it to the workflow graph representation, we computed the event durations and transition probabilities and finally employed our algorithms on this data. We showed that our algorithms are feasible to use in practice, as their runtime is negligible they can provide additional insights on the process.

In the second part of this dissertation we analyzed the processes taking a data-centric approach. We tapped into the field of process mining by unveiling root causes of workflow execution issues which were not visible at design-time but were apparent in practice.

We focused on historic execution logs and sought opportunities for improving the performance of the process, by making better decisions at run-time or by automating certain tasks. To this end, we have shown in two different case studies, that process execution data is valuable in developing tools for process automation that can save costs and/or decrease the execution time of business processes.

We demonstrated on real data that considering empirical evidence on which agent is the most efficient on resolving incident tickets in the dispatching process significantly reduces service time. The main observation and contribution was the discovery of incident ticket categories which are both topically similar but also homogeneous in terms of the execution time for a particular agent.

Lastly, we looked into the hard disk's life cycle process and the possibility of predicting when a disk needs to be replaced. We presented a machine learning-based pipeline for predicting disk replacements, built and evaluated on real data from a large disk population from two different manufacturers. We demonstrated the ability of our model to predict disk replacements with high accuracy.

Bibliography

[1] Automatic ticket dispatching. https://www.manageengine.com/products/service-desk-msp/automatic-ticket-dispatch.html.

[2] Backblaze dataset. https://www.backblaze.com/hard-drive-test-data.html.

[3] Best practice incident resolution workflow. http://wiki.servicenow.com/index.php?title=Best_Practice_Incident_Resolution_Workflow.

[4] Best practice incident resolution workflow. http://www.datacenterknowledge.com/archives/2008/05/30/failure-rates-in-google-data-centers/.

[5] Bpmn specification. http://www.omg.org/spec/BPMN/2.0/.

[6] Business process simulation with jbpm. http://camunda.com/download/20080201-bernd-ruecker-business-process-simulation-with-jbpm.pdf.

[7] Comparing and aligning process representations. http://hanvanderaa.com/wp-content/uploads/2017/11/PhDThesis2018-Comparing-and-Aligning-Process-Representations-prepr
pdf.

[8] Data center downtime costs. http://www.emerson.com/en-us/News/Pages/Net-Power-Study-Data-Center.aspx.

[9] Hard drive smart stats. https://www.backblaze.com/blog/hard-drive-smart-stats/.

[10] Process mining on the loan application process of a dutch financial institute. https://www.win.tue.nl/bpi/lib/exe/fetch.php?media=2017:bpi2017_winner_professional.pdf. Accessed: 2018-06-30.

[11] Signavio - bpm and simulation white paper. `https://www.signavio.com/wp-content/uploads/2012/09/BPM-and-Simulation-white-paper.pdf`.

[12] Standard raid levels. `https://en.wikipedia.org/wiki/Standard_RAID_levels`.

[13] IBM system storage DS8000 architecture and implementation. `http://www.redbooks.ibm.com/redbooks/pdfs/sg248886.pdf`.

[14] S.M.A.R.T. `https://en.wikipedia.org/wiki/S.M.A.R.T.`

[15] The ticket dispatching process. `https://www.researchgate.net/figure/259551586_fig1_FIGURE-1-Ticket-dispatching-process`.

[16] Using auto dispatching. `http://wiki.servicenow.com/index.php?title=Using_Auto-Dispatch#gsc.tab=0`.

[17] Adriansyah A. and Buijs J. Mining process performance from event logs. In Marcello La Rosa and Pnina Soffer, editors, *Business Process Management Workshops*, pages 217–218, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[18] Gionis A., Mannila H., and Tsaparas P. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data*, 1(1), March 2007.

[19] Jardine A., Lin D., and Banjevic D. A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical systems and signal processing*, 20(7):1483–1510, 2006.

[20] Kumar A., Dijkman R.M., and Song M. Optimal resource assignment in workflows for maximizing cooperation. In *BPM*, pages 235–250, 2013.

[21] Ray A. and Tangirala S. Stochastic modeling of fatigue crack dynamics for on-line failure prognostics. *IEEE Transactions on Control Systems Technology*, 4(4):443–451, 1996.

[22] Rozinat A. and Van der Aalst W.M.P. *Decision mining in business processes*. Beta, Research School for Operations Management and Logistics, 2006.

[23] Rozinat A. and Van der Aalst W.M.P. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.

[24] Voisin A., Levrat E., P. Cocheteux, and Iung B. Generic prognosis model for proactive maintenance decision support: application to pre-industrial e-maintenance test bed. *Journal of Intelligent Manufacturing*, 21(2):177–193, 2010.

[25] Ter Hofstede A.H.M., Van der Aalst W.M.P., Adams M., and Russell N., editors. *Modern Business Process Automation - YAWL and its Support Environment*. Springer, 2010.

[26] Efron B. Missing data, imputation, and the bootstrap. *Journal of the American Statistical Association*, 89(426):463–475, 1994.

[27] Gaujal B., Haar S., and Mairesse J. Blocking a transition in a free choice net and what it tells about its throughput. *Journal of Computer and System Sciences*, 66(3):515 – 548, 2003.

[28] Schroeder B. and Gibson G.A. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? FAST 2007, Berkeley, CA, USA, 2007. USENIX Association.

[29] Van Dongen B.F. Event log for the bpi challenge 2012. http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.

[30] Ha B.H., Reijers H.A., Bae J., and Bae H. *An Approximate Analysis of Expected Cycle Time in Business Process Execution*, pages 65–74. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[31] Byington C., Roemer M., and Galie T. Prognostic enhancements to diagnostic systems for improved condition-based maintenance [military aircraft]. In *Aerospace Conference Proceedings, 2002. IEEE*, volume 6, pages 6–6. IEEE, 2002.

[32] Favre C., Fahland D., and Völzer H. The relationship between workflow graphs and free-choice workflow nets. *Inf. Syst.*, 47:197–219, 2015.

[33] Favre C. and Völzer H. *The Difficulty of Replacing an Inclusive OR-Join*, pages 156–171. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[34] Favre C., Völzer H., and Müller P. Diagnostic information for control-flow analysis of workflow graphs a.k.a. free-choice workflow nets. In *Proceedings of the 22Nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 9636*, pages 463–479, New York, NY, USA, 2016. Springer-Verlag New York, Inc.

[35] Haiyan C. and Fuji Z. The expected hitting times for finite markov chains. *Linear Algebra and its Applications*, 428(1112):2730 – 2749, 2008.

[36] Oppenheimer C. and Loparo K. Physically based diagnosis and prognosis of cracked rotor shafts. In *AeroSense 2002*, pages 122–132. International Society for Optics and Photonics, 2002.

[37] Yang C.-Q. and Miller B.P. Critical path analysis for the execution of parallel and distributed programs. In *Distributed Computing Systems, 1988., 8th International Conference on*, pages 366–373, Jun 1988.

[38] Ramamoorthy C. V. and Ho G. S. Performance evaluation of asynchronous concurrent systems using petri nets. *IEEE Transactions on Software Engineering*, SE-6(5):440–449, Sept 1980.

[39] Ellis C.A. and Nutt G.J. *Modeling and enactment of workflow systems*, pages 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[40] Petri C.A. Introduction to general net theory. In *Net theory and applications*, pages 1–19. Springer, 1980.

[41] Aggarwal C.C. and Wang H. *Managing and Mining Graph Data*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[42] Burges C.J.C. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, June 1998.

[43] Defays D. An efficient algorithm for a complete link method. *Comput. J.*, 20(4):364–366, 1977.

[44] Fahland D., Favre C., Koehler J., Lohmann N., Völzer H., and Wolf K. Analysis on demand: Instantaneous soundness checking of industrial business process models. *Data Knowl. Eng.*, 70(5):448–466, 2011.

[45] Levin D., Peres Y., and Wilmer E. *Markov chains and mixing times*. American Mathematical Society, 2006.

[46] Lin D., Raghu R., Ramamurthy V., Yu J., Radhakrishnan R., and Fernandez J. Unveiling clusters of events for alert and incident management in large-scale enterprise it. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1630–1639, New York, NY, USA, 2014. ACM.

[47] Metzler D., Dumais S., and Meek C. Similarity measures for short segments of text. In *Proceedings of the 29th European Conference on IR Research*, ECIR'07, pages 16–27, Berlin, Heidelberg, 2007. Springer-Verlag.

[48] Zhou D. and Burges C.J.C. Spectral clustering and transductive learning with multiple views. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 1159–1166, New York, NY, USA, 2007. ACM.

[49] Blei D. M., Ng A. Y., and Jordan M.I. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[50] Levin D.A., Peres Y., and Wilmer E.L. *Markov chains and mixing times*. American Mathematical Society, 2006.

[51] Cox D.R. The regression analysis of binary sequences (with discussion). *J Roy Stat Soc B*, 20:215–242, 1958.

[52] Dijkstra E. A note on two problems in connexion with graphs. *Numer. Math.*, 1:269–271, 1959.

[53] Kindler E. On the semantics of epcs: Resolving the vicious circle. *Data & Knowledge Engineering*, 56(1):23–40, 2006.

[54] Pinheiro E., Weber W.D., and Barroso L.A. Failure trends in a large disk drive population. FAST 2007, Berkeley, CA, USA, 2007. USENIX Association.

[55] Papalexakis E.E., Akoglu L., and Ience D. Do more views of a graph help? community detection and clustering in multi-graphs. In *FUSION*, pages 899–905. IEEE, 2013.

[56] R. El-Yaniv and D. Yanay. Supervised learning of semantic relatedness. In PeterA. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 7523 of *Lecture Notes in Computer Science*, pages 744–759. Springer Berlin Heidelberg, 2012.

[57] Juan E.Y.T., Tsai J.P., Murata T., and Zhou Y. Reduction methods for real-time systems using delay time petri nets. *IEEE Trans. Softw. Eng.*, 27(5):422–448, May 2001.

[58] Ciarapica F. and Giacchetta G. Managing the condition-based maintenance of a combined-cycle power plant: an approach using soft computing techniques. *Journal of Loss Prevention in the Process Industries*, 19(4):316–325, 2006.

[59] Bowden F.D.J. A brief survey and synthesis of the roles of time in petri nets. *Mathematical and Computer Modelling*, 31(10):55 – 68, 2000.

[60] Hughes F.G., Murray J.F., Kreutz-Delgado K., and Elkan C. Improved disk-drive failure warnings. *IEEE Transactions on Reliability*, 51(3):350–357, 2002.

[61] Berthelot G. Transformations and decompositions of nets. In *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986*, pages 359–376, 1986.

[62] Hamerly G. and Elkan C. Bayesian approaches to failure prediction for disk drives. ICML '01, pages 202–209, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[63] Kacprzynski G., Sarlashkar A., Roemer M., Hess A., and Hardman B. Predicting remaining life by fusing the physics of failure modeling with diagnostics. *JOM Journal of the Minerals, Metals and Materials Society*, 56(3):29–35, 2004.

[64] Lange G. and Williams W.T. A general theory of classificatory sorting strategies. i. hierarchical systems. *Computer Journal*, 9:373–380, 1967.

[65] Recchia G. and Jones M. More data trumps smarter algorithms: Comparing pointwise mutual information with latent semantic analysis. *Behavior Research Methods*, 41(3):647–656, 2009.

[66] Weiss G.M. and Hirsh H. Learning to predict rare events in event sequences. In *KDD 1998*, pages 359–363. AAAI Press, 1998.

[67] Furnas G.W., Landauer T.K., Gomez L.M., and Dumais S.T. The vocabulary problem in human-system communication. *Commun. ACM*, 30(11):964–971, November 1987.

[68] Hansson H. and Jonsson B. A framework for reasoning about time and reliability. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 102–111, Dec 1989.

[69] Junzhou H., Tong Z., and Dimitris M. Learning with structured sparsity. *Journal of Machine Learning Research*, 12(Nov):3371–3412, 2011.

[70] Mili H., Tremblay G., Jaoude G., Lefebvre É., Elabed L., and El Boussaidi G. Business process modeling languages: Sorting through the alphabet soup. *ACM Comput. Surv.*, 43(1):4:1–4:56, December 2010.

[71] Reijers H., Jansen-Vullers M., Zur Muehlen M., and Appl W. Workflow management systems + swarm intelligence = dynamic task assignment for emergency management applications. In *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 125–140. Springer Berlin Heidelberg, 2007.

[72] Van der Aa H., Leopold H., and Reijers H.A. Checking process compliance on the basis of uncertain event-to-activity mappings. In *CAiSE*, volume 10253 of *Lecture Notes in Computer Science*, pages 79–93. Springer, 2017.

[73] Van der Aa H., Leopold H., and Reijers H.A. Comparing textual descriptions to process models - the automatic detection of inconsistencies. *Inf. Syst.*, 64:447–460, 2017.

[74] How does S.M.A.R.T. function of hard disks work? www.hdsentinel.com/smart/index.php.

[75] Kriegel H.P., P. Kröger, and Zimek A. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. Knowl. Discov. Data*, 3(1):1:1–1:58, March 2009.

[76] Camerzan I. Determining best-case and worst-case times of unknown paths in time workflow nets. *The Computer Science Journal of Moldova*, 18(1):59–69, 2010.

[77] Becker J., Weiss B., and Winkelman A. Developing a business process modeling language for the banking sector-a design science approach. *AMCIS 2009 Proceedings*, page 709, 2009.

[78] Desel J. and Esparza J. *Free Choice Petri Nets*. Cambridge University Press, New York, NY, USA, 1995.

[79] Eder J., Panagos E., and Rabinovich M. *Time Constraints in Workflow Systems*, pages 286–300. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[80] Eder J., Pozewaunig H., and Liebhart W. epert: Extending pert for workflow management systems. In *First East-European Symposium on Advances in Database and Information Systems (ADBIS 1997)*, pages 217–224. Nevsky Dialect, September 1997.

[81] Klayman J. and Ha Y.W. Confirmation, disconfirmation, and information in hypothesis testing. *Psychological review*, 94(2):211, 1987.

[82] Kleinberg J. An impossibility theorem for clustering. pages 446–453. MIT Press, 2002.

[83] Leskovec J., Lang K.J., and Mahoney M. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 631–640, New York, NY, USA, 2010. ACM.

[84] Lim J., Salakhutdinov R., and Torralbo A. Transfer learning by borrowing examples for multiclass object detection. In *Advances in neural information processing systems*, pages 118–126, 2011.

[85] Vanhatalo J., Völzer H., and Leymann F. Faster and more focused control-flow analysis for business process models through sese decomposition. In Kraemer B., Lin K.J., and Narasimhan P., editors, *Service-Oriented Computing – ICSOC 2007*, volume 4749 of *Lecture Notes in Computer Science*, pages 43–55. Springer Berlin Heidelberg, 2007.

[86] Vanhatalo J., Völzer H., and Koehler J. The refined process structure tree. *Data Knowl. Eng.*, 68(9):793–818, 2009.

[87] William J. and Charles S. Estimation with quadratic loss. In *Proceedings of the fourth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 361–379, 1961.

[88] Xu J. and Parnas D.L. On satisfying timing constraints in hard-real-time systems. *IEEE transactions on software engineering*, 19(1):70–84, 1993.

[89] Ye J., Chow J.H., Chen J., and Zheng Z. Stochastic gradient boosted distributed decision trees. CIKM 2009, pages 2061–2064. ACM, 2009.

[90] Bezdek J.C. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1981.

[91] Kleinsorge J.C., Falk H., and Marwedel P. Simple analysis of partial worst-case execution paths on general control flow graphs. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–10, Sept 2013.

[92] Murray J.F., Hughes G.F., and Schuurmans D. Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning research*, 6:816, 2005.

[93] Elerath J.G. and Shah S. Server class disk drives: how reliable are they? In *Reliability and Maintainability, 2004 Annual Symposium - RAMS*, pages 151–156.

[94] Li J.Q., Fan Y., and Zhou M.C. Performance modeling and analysis of workflow. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 34(2):229–242, March 2004.

[95] Sankaranarayanan K. *Thermal Modeling and Management of Microprocessors*. PhD thesis, Charlottesville, VA, USA, 2009. AAI3353832.

[96] Toutanova K., Klein D., Manning C.D., and Singer Y. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 173–180, Stroudsburg, PA, USA, 2003.

[97] Van Hee K. and Reijers H.A. Using formal analysis techniques in business process redesign. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 142–160, London, UK, UK, 2000. Springer-Verlag.

[98] Brodersen K.H., Gallusser F., Koehler J., Remy N., and Scott S.L. Inferring causal impact using bayesian structural time-series models. *Annals of Applied Statistics*, 9:247–274, 2015.

[99] Braghetto K.R, Ferreira J.E., and Vincent J.M. From Business Process Model and Notation to Stochastic Automata Network. Research report, 2011.

[100] Breiman L. Random forests. *Machine Learning*, 45(1):5–32.

[101] Kaufman L. and Rousseeuw P.J. *Finding Groups in Data: An Introduction to Cluster Analysis*, pages 1–67. John Wiley and Sons, Inc., 2008.

[102] Popova-Zeugmann L. and Heiner M. Worst-case analysis of concurrent systems with duration interval petri nets. In *BTU COTTBUS*, pages 162–179, 1996.

[103] Alizadeh M., Lu X., Fahland D., Zannone N., and Van der Aalst W.M.P. Linking data and process perspectives for conformance analysis. *Computers & Security*, 73:172–193, 2018.

[104] Beaudouin-Lafon M., Mackay W.E., Andersen P., Janecek P., Jensen M., Lassen M., Lund K., Mortensen K., Munck S., Ratzer A., Ravn K., Christensen S., and Jensen K. Cpn/tools: A post-wimp interface for editing and simulating coloured petri nets. In José-Manuel Colom and Maciej Koutny, editors, *Applications and Theory of Petri Nets 2001*, pages 71–80, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[105] De Leoni M., Maggi F.M., and Van der Aalst W.M.P. An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Information Systems*, 47(1):258–277, 2015.

[106] Indulska M., Green P., Recker J., and Rosemann M. *Business Process Modeling: Perceived Benefits*, pages 458–471. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[107] Kwiatkowska M. and Parker D. Advances in probabilistic model checking. In T. Nipkow, O. Grumberg, and B. Hauptmann, editors, *Software Safety and Security - Tools for Analysis and Verification*, volume 33 of *NATO Science for Peace and Security*

Series - D: Information and Communication Security, pages 126–151. IOS Press, 2012.

[108] Kwiatkowska M., Norman G., and Parker D. Prism 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, CAV'11, pages 585–591, Berlin, Heidelberg, 2011. Springer-Verlag.

[109] Porter M. The Porter Stemming Algorithm.

[110] Wan M. and Ciardo G. Symbolic reachability analysis of integer timed petri nets. In *SOFSEM 2009: Theory and Practice of Computer Science*, volume 5404 of *Lecture Notes in Computer Science*, pages 595–608. Springer Berlin Heidelberg, 2009.

[111] Garey M. R. and Johnson D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[112] J. Macqueen. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[113] Wynn M.T., Verbeek M., Van der Aalst W.M.P., Ter Hofstede A., and Edmond D. Business process verification–finally a reality! *Business Process Management Journal*, 15(1):74–92, 2009.

[114] Russell N., Van der Aalst W.M.P., Hofstede A., and Edmond D. Workflow resource patterns: Identification, representation and tool support. In *Advanced Information Systems Engineering*, volume 3520 of *Lecture Notes in Computer Science*, pages 216–232. Springer Berlin Heidelberg, 2005.

[115] Dragomir O.E., Gouriveau R., Dragomir F., Minca E., and Zerhouni N. Review of prognostic problem in condition-based maintenance. In *Control Conference (ECC), 2009 European*, pages 1587–1592. IEEE, 2009.

[116] Chrétienne P. Timed event graphs: A complete study of their controlled executions. In *International Workshop on Timed Petri Nets, Torino, Italy, July 1-3, 1985*, pages 47–54, 1985.

[117] Wang P. and Vachtsevanos G. Fault prognostics using dynamic wavelet neural networks. *AI EDAM*, 15(4):349–365, 2001.

[118] Turney P. D. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *Proceedings of the 12th European Conference on Machine Learning*, EMCL '01, pages 491–502, London, UK, UK, 2001. Springer-Verlag.

[119] Shao Q., Chen Y., Tao S., Yan X., and Anerousis N. Efficient ticket routing by resolution sequence mining. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 605–613, New York, NY, USA, 2008. ACM.

[120] Bellman R. On a Routing Problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[121] Chinnam R. and Baruah P. Autonomous diagnostics and prognostics through competitive learning driven hmm-based clustering. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 4, pages 2466–2471. IEEE, 2003.

[122] Dunkl R., Rinderle-Ma S., Grossmann W., and Fröschl K.A. Decision point analysis of time series data in process-aware information systems. In *CAiSE (Forum/Doctoral Consortium)*, volume 1164 of *CEUR Workshop Proceedings*, pages 33–40. CEUR-WS.org, 2014.

[123] Johnson R. and Tong Z. Learning nonlinear functions using regularized greedy forest. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(5):942–954, May 2014.

[124] Mans R., Russell N., Van der Aalst W.M.P., Moleman A., and Bakker P. Schedule-aware workflow management systems. *T. Petri Nets and Other Models of Concurrency*, 4:121–143, 01 2010.

[125] Mihalcea R., Corley C., and Strapparava C. Corpus-based and knowledge-based measures of text semantic similarity. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pages 775–780. AAAI Press, 2006.

[126] Protasi R., Battiti M. *Approximate Algorithms and Heuristics for MAX-SAT*, pages 77–148. Springer US, Boston, MA, 1999.

[127] Wallace R.B. and Whitt W. A staffing algorithm for call centers with skill-based routing. *Manufacturing & Service Operations Management*, 7(4):276–294, October 2005.

[128] Sloan R.H. and Buy U. Reduction rules for time petri nets. *Acta Informatica*, 33(7):687–706, 1996.

[129] Cilibrasi R.L. and Vitanyi P.M.B. The google similarity distance. *Knowledge and Data Engineering, IEEE Transactions on*, 19(3):370–383, March 2007.

[130] Agarwal S., Sindhgatta R., and Sengupta B. Smartdispatch: Enabling efficient ticket dispatch in an it service environment. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 1393–1401, New York, NY, USA, 2012. ACM.

[131] Boyd S. and Vandenberghe L. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[132] Fenz S., Ekelhart A., and Neubauer T. Business process-based resource importance determination. In Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo A. Reijers, editors, *Business Process Management*, pages 113–127, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[133] Leemans S., Fahland D., and Van der Aalst W.M.P. Scalable process discovery and conformance checking. *Software & Systems Modeling*, pages 1–33, 2016.

[134] Mani S., Sankaranarayanan K., Sinha V.S., and Devanbu P. Panning requirement nuggets in stream of software maintenance tickets. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 678–688, New York, NY, USA, 2014. ACM.

[135] Pan S. and Yang Q. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, October 2010.

[136] Rozsnyai S., Lakshmanan G., Muthusamy V., Khalaf R., and Duftler M. Business process insight: An approach and platform for the discovery and analysis of end-to-end business processes. In *SRII Global Conference (SRII), 2012 Annual*, pages 80–89. IEEE, 2012.

[137] Visa S. and Ralescu A. Issues in mining imbalanced data sets-a review paper. In *Proceedings of the sixteen midwest artificial intelligence and cognitive science conference*, volume 2005, pages 67–73. sn, 2005.

[138] Zhang S. and Ganesan R. Multivariable trend analysis using neural networks for intelligent diagnostics of rotating machinery. *TRANSACTIONS-AMERICAN SOCIETY OF MECHANICAL ENGINEERS JOURNAL OF ENGINEERING FOR GAS TURBINES AND POWER*, 119:378–384, 1997.

[139] Cormen T., Stein C., Rivest R., and Leiserson C.E. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[140] Murata T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989.

[141] Wei T., Zhengdong L., and Dhillon I.S. Clustering with multiple graphs. In *Data Mining, 2009. ICDM '09. Ninth IEEE International Conference on*, pages 1016–1021, Dec 2009.

[142] Landauer T. K., Foltz P. W., and Laham D. An introduction to latent semantic analysis. *Discourse Processes*, 25:259–284, 1998.

[143] Haveliwala T.H., Gionis A., Klein D., and Indyk P. Evaluating strategies for similarity search on the web. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 432–442, New York, NY, USA, 2002. ACM.

[144] Zwick U. Exact and approximate distances in graphs a survey. In *Algorithms ESA 2001*, volume 2161 of *Lecture Notes in Computer Science*, pages 33–48. Springer Berlin Heidelberg, 2001.

[145] Agarwal V., Bhattacharyya C., Niranjan T., and Susarla S. Discovering rules from disk events for predicting hard drive failures. ICMLA '09, pages 782–786, Dec 2009.

[146] Denisov V., Fahland D., and Van der Aalst W.M.P. Unbiased, fine-grained description of processes performance from event data. In *BPM*, volume 11080 of *Lecture Notes in Computer Science*, pages 139–157. Springer, 2018.

[147] Derguech W., Gao F., and Bhiri S. *Configurable Process Models for Logistics Case Study for Customs Clearance Processes*, pages 119–130. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[148] Jiang W., Hu C., Zhou Y., and Kanevsky A. Are disks the dominant contributor for storage failures?: A comprehensive study of storage subsystem failure characteristics. *Trans. Storage*, 4(3):7:1–7:25, November 2008.

[149] Reisig W. and Rozenberg G., editors. *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the Volumes Are Based on the Advanced Course on Petri Nets*, London, UK, UK, 1998. Springer-Verlag.

[150] Van der Aalst W.M.P. van der (2011). process mining: Discovery, conformance and enhancement of business processes.

[151] Van der Aalst W.M.P. Using interval timed coloured petri nets to calculate performance bounds. In *Proceedings of the 7th International Conference on Computer Performance Evaluation : Modelling Techniques and Tools: Modelling Techniques and Tools*, pages 425–444, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[152] Van der Aalst W.M.P. *Verification of workflow nets*, pages 407–426. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[153] Van der Aalst W.M.P. *Three Good Reasons for Using a Petri-Net-Based Workflow Management System*, pages 161–182. Springer US, Boston, MA, 1998.

[154] Van der Aalst W.M.P. Workflow verification: Finding control-flow errors using petri-net-based techniques. In *Business Process Management*, pages 161–183. Springer, 2000.

[155] Van der Aalst W.M.P. Trends in business process analysis - from verification to process mining. In *ICEIS 2007 - Proceedings of the Ninth International Conference on Enterprise Information Systems, Volume DISI, Funchal, Madeira, Portugal, June 12-16, 2007*, pages 5–9, 2007.

[156] Van der Aalst W.M.P. Process discovery: capturing the invisible. *IEEE Computational Intelligence Magazine*, 5(1):28–41, 2010.

[157] Van der Aalst W.M.P. *Process Mining: Data Science in Action*. Springer Publishing Company, Incorporated, 2nd edition, 2016.

[158] Van der Aalst W.M.P., Hirnschall A., and Verbeek H. M. W. *Advanced Information Systems Engineering: 14th International Conference, CAiSE 2002 Toronto, Canada, May 27–31, 2002 Proceedings*, chapter An Alternative Way to Analyze Workflow Graphs, pages 535–552. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[159] Van der Aalst W.M.P., Nakatumba J., Rozinat A., and Russell N. Business process simulation. In *Handbook on Business Process Management 1*, pages 313–338. Springer, 2010.

[160] Van der Aalst W.M.P. and Van Hee K.M. Framework for business process redesign. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 1995., Proceedings of the Fourth Workshop on*, pages 36–45. IEEE, 1995.

[161] Van der Aalst W.M.P., Van Hee K.M., Ter Hofstede A.H.M., Sidorova N., Verbeek H.M.W., Voorhoeve M., and Wynn M.T. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.

[162] Chen Y., Sanghavi S., and Xu H. Clustering sparse graphs. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2204–2212. Curran Associates, Inc., 2012.

[163] Tan Y. and Gu X. On predictability of system anomalies in real world. In *IEEE MASCOTS, 2010*, pages 133–140, Aug 2010.

[164] Ng Y. A., Jordan M. I., and Weiss Y. On spectral clustering: Analysis and an algorithm. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 849–856. MIT Press, 2002.

Curriculum Vitae

Mirela Madalina Botezatu

Date of Birth: 04.10.1987

Nationality: Romanian

Address: Zürich, Switzerland

Email: mirela.botezatu@gmail.com

EDUCATION

2013 - p.t.	**Eidgenössische Technische Hochschule Zürich (ETH Zurich)** Ph.D. Candidate at Department of Information Technology and Electrical Engineering Thesis: Deadline Analysis of Workflow Graphs and Workflow Performance Optimization
2010 - 2012	**Universite Pierre et Marie Curie** Master in Data Mining Thesis: Predicting Compiler Optimizations from Performance Events
2006 - 2010	**Polytechnic University of Bucharest** Bachelor in Computer Science Thesis: Gateway Personal Firewall

WORK EXPERIENCE

2017 - p.t	**Google – Zürich, Switzerland** Software Engineer
2013 - 2017	**IBM Research – Zürich, Switzerland** Predoctoral Researcher at Computer Science Department
2012 - 2013	**CERN, collaborating with Intel – Geneva, Switzerland** Technical Student
Summer, 2011	**Laboratoire d'Informatique Paris 6 – Paris, France** Research Internship
Summer, 2010	**BitDefender – Bucharest, Romania** Internship Software Developer
Summer, 2009	**Avira Antivirus – Bucharest, Romania** Internship Software Developer
Summer, 2008	**Jinny Software – Bucharest, Romania** Internship Software Developer

PUBLICATIONS

7. Mirela Botezatu, Hagen Völzer and Lothar Thiele. The Complexity of Deadline Analysis for Workflow Graphs with Multiple Resources. In BPM 2016:252-268.
6. Mirela Botezatu, Ioana Giurgiu, Jasmina Bogojeska and Dorothea Wiesmann. Predicting Disk Replacement towards Reliable Data Centers. In ACM SIGKDD 2016:39-48.
5. Mirela Botezatu, Hagen Völzer and Lothar Thiele. The Complexity of Deadline Analysis for Workflow Graphs with a Single Resource. In IEEE ICECCS 2015:110-119.
4. Mirela Botezatu, Jasmina Bogojeska, Ioana Giurgiu, Hagen Völzer and Dorothea Wiesmann. Multi-View Incident Ticket Clustering for Optimal Ticket Dispatching. In ACM SIGKDD 2015:1711-1720.
3. Mirela Botezatu, Hagen Völzer and Remco M. Dijkman. A Case Study in Workflow Scheduling Driven by Log Data. In BPI Workshop 2014:251-263.
2. Anne Baumgrass, Mirela Botezatu, Claudio Di Ciccio, Remco Dijkman, Paul Grefen, Marcin Hewelt, Jan Mendling, Andreas Meyer, Shaya Pourmirza and Hagen Völzer. Towards a Methodology for the Engineering of Event-driven Process Applications. In BPI Workshop 2015:501-514.
1. Ioana Giurgiu, Mirela Botezatu and Dorothea Wiesmann. Comprehensible Models for Reconfiguring Enterprise Relational Databases. In ACM CIKM 2015:1371-1380.

AWARDS

2003, 2004, 2005	Participation in the National Olympiad in Mathematics – Romania. Won Honorable Mention.
2010 – 2012	Full scholarship awarded by the European Commission for MSc. in Data Mining.
2015	Won 3rd place in Swiss semi-finals for the International Competition for Mathematical and Logical games.