Master Thesis

May 3, 2017

Workflow Optimization

Optimal Job Assignment in a Discrete Event Simulation Environment

Filip Kočovski

of Lugano, Switzerland (10-932-994)

supervised by

Prof. Dr. Daning Hu

Dr. Markus Uhr

University of Zurich[UZH]

ifi

Master Thesis

Workflow Optimization

Optimal Job Assignment in a Discrete Event Simulation Environment

Filip Kočovski

University of Zurich[UZH]

ifi

Master Thesis

Author: Filip Kočovski, filip.kocovski@uzh.ch

Project period: November 15, 2016 - May 15, 2017

Business Intelligence Research Group
Department of Informatics, University of Zurich

Acknowledgements

I would like to thank Prof. Dr. Daning Hu for giving me the marvelous opportunity of pursuing this topic in collaboration with immopac ag and supporting me all along the path with his invaluable flexibility.

I am deeply grateful and eternally indebted to Dr. Markus Uhr for his terrific knowledge, availability, patience, comprehension and guidance during the course of this thesis. His keen eye for details, both technical and not are the foundations on which this thesis is based. His tremendous patience and capability to convey difficult concepts in a pragmatic fashion were key factors for the successful completion of this work. Lastly, I am also grateful to Dr. Markus Uhr for proofreading this thesis.

I am enormously grateful to both Dr. Robert Höhener and Dr. Thomas Höhener for advocating for this collaborative project and for allowing me to conduct this research at immopac ag. Moreover I want to thank Dr. Robert Höhener for his invaluable feedback during the initial phase of this thesis and administrative consultation.

I am grateful to Dr. Andreas Horni for his technical suggestions and feedback on the LaTeX structure and for proofreading this thesis.

My deepest gratitude goes towards my parents, Dr. med. Tatjana Zafirovska and Ljupco Kočovski for believing in me as a person and for always motivating me to pursue my dreams and push my boundaries.

My warmest thankfulness goes to Martina Lardi for her academic feedback, patience, comprehension during the whole duration of this project and for proofreading this thesis.

I want to thank Dustin Kerner for proofreading the German parts of this work.

Ultimately I want to thank the whole staff of immopac ag for their continuous support during the course of my work and for their interest in this thesis.

Abstract

Efficiently assigning human resources in Workflow Management Systems (WfMS) is a vital aspect when implementing them in corporate environments (Cheng, 2000; Mentzas et al., 2001).

The traditional approach formulates a deterministic planning model using Mixed Integer Linear Programming (MILP) and solves role resolution to optimality which has lead to promising results (Zeng and Zhao, 2005). However the deterministic approach is a limiting factor when approaching role resolution with MILP (Zeng and Zhao, 2005).

This thesis expands on existing work in the field of effective role resolution in WfMS by extending the already researched MILP methodologies and proposes a novel approach by introducing Reinforcement Learning (RL) based approaches (Sutton and Barto, 2017). These modern approaches help overcoming the deterministic limitations of MILP by approaching role resolution in a stochastic fashion (Sutton and Barto, 2017).

For evaluation a subset of Business Process Model and Notation (BPMN) elements have been implemented in a discrete event simulation environment, in which existing policies for role resolution in WfMS, such as Shared Queue (SQ), Least Loaded Qualified Person (LLQP), K-Batch and 1-Batch-1 are tested.

Both the extended MILP as well as the RL based methods outperform the traditional approaches up to a 1.3-fold speedup. Even though promising results have been obtained, precautions have to be taken when interpreting the results: on one hand the MILP based optimizations obtained are coupled with higher computational complexity which raise business trade-offs and ,on the other hand, RL based optimizations overcome the computational complexity problem of MILP based methods but require lengthy training sessions in order to assert optimal convergence.

RL based methods lay the foundations for extensions by using alternative methods such as Inverse RL (IRL) (Ng and Russell, 2000) and Apprenticeship Learning (AL). AL promises to overcome the limitations of RL methods by eliminating the requirement of internal reward functions and learn optimal behaviors by “observing” experts executing tasks (Abbeel and Ng, 2004). Future work could reconcile traditional MILP and AL based methods by using the former as the “expert” agent performing role resolution and the latter observing its behavior in order to learn from it.

Zusammenfassung

Die effiziente Zuteilung menschlicher Ressourcen in Workflow Management Systems (WfMS) ist ein zentraler Aspekt bei der Umsetzung in Unternehmensumfelder (Cheng, 2000; Mentzas et al., 2001).

Die traditionelle Methode formuliert ein deterministisches Planungsmodell mittels Mixed Integer Linear Programming (MILP) und löst das Problem der Rollenauflösung nach Optimalität auf, welche zu vielversprechenden Resultate geführt hat (Zeng and Zhao, 2005). Allerdings ist das deterministische Vorgehen einen begrenzenden Faktor wenn man sich die Rollenauflösung mit MILP nähert (Zeng and Zhao, 2005).

Diese Masterarbeit erweitert die bestehende Forschung im Bereich der effektvollen Rollenauflösung in WfMS indem MILP-Methoden ausgebaut wurden und einen neuartigen Ansatz mit Reinforcement Learning (RL) vorgestellt wurde (Sutton and Barto, 2017). Diese moderne Massnahmen mit RL helfen bei der Überwindung der deterministischen Begrenzungen von MILP indem die Rollenauflösung stochastisch gelöst wird (Sutton and Barto, 2017).

Es wurde eine Teilmenge von Business Process Model and Notation (BPMN)-Elemente in einem diskreten Ereignissimulationsumfeld für die Bewertung implementiert, wobei bestehende Policen für die Rollenauflösung in WfMS *z.B.*, Shared Queue (SQ), Least Loaded Qualified Person (LLQP), K-Batch und 1-Batch-1 getestet wurden.

Sowohl die erweiterten MILP als auch die RL basierten Policen übertreffen die traditionellen Methoden bis zu einer Beschleunigung von 1.3. Obwohl vielversprechende Resultate erreicht wurden, muss man die Interpretation der Resultate mit Vorsicht geniessen: die MILP basierten Methoden weisen einerseits höhere rechnerische Komplexität auf, welche Businesskompromisse aufwirft und anderseits überwinden die RL basierten Methoden die rechnerische Komplexität der MILP-Methoden, diese verlangen jedoch langwierige Trainingseinheiten um die optimale Konvergenz sicherzustellen.

Die RL basierten Methoden legen die Grundlagen für Erweiterungen mittels alternativer Methoden *z.B.*, Inverse RL (IRL) (Ng and Russell, 2000) und Apprenticeship Learning (AL) (Abbeel and Ng, 2004) vor. AL verspricht die Einschränkungen der RL basierten Methoden durch die Auflösung interner Entlohnungsfunktionen zu überwinden und optimale Verhalten durch die "Beobachtung" von Sachverständigen zu lernen (Abbeel and Ng, 2004). Künftige Arbeit konnte MILP und AL basierten Methoden abgleichen indem die erstere sich als Sachverständige verhält und die letztere das Verhalten der ersteren beobachtet und lernt.

Contents

1 Introduction **1**
1.1 Objectives . . . 1
1.2 Thesis Structure . . . 1

2 Related Work **3**
2.1 Queueing . . . 3
2.2 Workflow Management Systems . . . 3
2.3 Reinforcement Learning . . . 5
2.4 Mixed Integer Linear Programming . . . 6
2.5 Simulation . . . 7

3 Theoretical Foundations **9**
3.1 Business Process Model and Notation . . . 9
3.1.1 Start Event . . . 10
3.1.2 User Task . . . 10
3.1.3 Gateway . . . 10
3.1.4 End Event . . . 11
3.2 Business Process Model and Notation Limitations . . . 11
3.3 Policy Based Role Resolution . . . 11

4 Discrete Event Simulation **13**
4.1 Business Process Model and Notation Implementation . . . 13
4.1.1 Start Event . . . 13
4.1.2 User Task . . . 16
4.1.3 Gateway . . . 16
4.1.4 End Event . . . 18
4.2 Policy Based Role Resolution Implementation . . . 18

5 Role Resolution Policies **21**
5.1 Mixed Integer Linear Programming Policies . . . 21
5.2 Reinforcement Learning Theory . . . 26
5.2.1 Reinforcement Learning Definition . . . 26
5.2.2 Finite Markov Decision Processes . . . 26
5.2.3 Dynamic Programming . . . 27
5.2.4 Monte Carlo Methods . . . 27
5.2.5 Temporal Difference Learning . . . 28
5.2.6 On Policy Prediction with Approximation . . . 28

5.2.7 On Policy Control with Approximation . . . 30
5.2.8 Off Policy Methods with Approximation . . . 30
5.2.9 Policy Gradient Methods . . . 31
5.3 Reinforcement Learning Policies . . . 32
5.3.1 Prediction and Control Methods . . . 33
5.3.2 Update Methods . . . 36
5.3.3 Batch Size Emulation . . . 37

6 Empirical Analysis **39**
6.1 Simulation Framework . . . 39
6.2 Business Process Modeling . . . 40
6.3 Key Performance Indicators and Data Visualization . . . 41

7 Results **45**
7.1 Mixed Integer Linear Programming . . . 45
7.2 Reinforcement Learning . . . 46
7.2.1 Batch . . . 48
7.2.2 Least Loaded Qualified Person . . . 48
7.2.3 Others . . . 49

8 Conclusion **51**
8.1 Mixed Integer Linear Programming Results Discussion . . . 51
8.2 Reinforcement Learning Results Discussion . . . 52
8.3 Summary . . . 53
8.4 Consequences . . . 54
8.5 Outlook . . . 54

A Tools Used **57**

B Mixed Integer Linear Programming Results **59**
B.1 Least Loaded Qualified Person . . . 59
B.1.1 KPIs . . . 59
B.2 Shared Queue . . . 60
B.2.1 KPIs . . . 60
B.3 K-Batch . . . 61
B.3.1 KPIs . . . 61
B.3.2 Batch Sizes Comparison . . . 63
B.4 K-Batch-1 . . . 64
B.4.1 KPIs . . . 64
B.4.2 Batch Sizes Comparison . . . 66
B.5 1-Batch-1 . . . 67
B.5.1 KPIs . . . 67

C Reinforcement Learning Results **69**
C.1 1-Batch . . . 69
C.1.1 KPIs . . . 69
C.1.2 Comparison with MSA . . . 71
C.2 1-Batch-1 . . . 72
C.2.1 KPIs . . . 72
C.2.2 Comparison with MSA . . . 73
C.3 Least Loaded Qualified Person . . . 74

C.3.1 KPIs 74
C.3.2 Comparison with MSA 75
C.3.3 Additional LLQP Policies 76
C.3.4 ANN Implementation in TF for LLQP 77
C.4 Others 78
C.4.1 KPIs 78
C.4.2 Comparison with MSA 80

D CD Content **83**

List of Figures

3.1 BPMN process showing a usual process starting with a start event (green) on the leftmost side, followed by different user tasks (purple) and gateways (orange) and eventually two different end events (red). 10
3.2 Single BPMN elements with their corresponding labeling according to BPMN's notation. 10
3.3 Cyclic diagram with repeat loop. Adapted from Sörensen (2005, p. 12). 11
3.4 Cyclic diagram with while loop. Adapted from Sörensen (2005, p. 12). 11

4.1 Class diagram of BPMN elements implementation for the discrete event simulation environment. 14
4.2 Simulation run with MSA showing token times at the first directly connected user task to the start event. 16
4.3 Simulation run with ST showing token times at the first directly connected user task to the start event. 16
4.4 Class diagram showing policies and policy jobs classes as independent objects. . . 18
4.5 Class diagram showing policy methods and attributes. 18
4.6 Class diagram showing policy job methods and attributes. 19

5.1 Artificial Neural Network with one hidden layer. 30
5.2 Artificial Neural Network with two hidden layers. 30
5.3 MC and TD proposed updates comparison. MC approaches have to wait until the end of an episode in order to update since they require a full episode to effectively discount rewards, compared to TD approaches that are able to update at each following time step $t + 1$. Adapted from Sutton and Barto (2017, p. 130). 36
5.4 User 1 receives job 1 while user 2 receives jobs 2 and 3. 38
5.5 User 1 receives jobs 2 and 1 while user 2 receives job 3. 38

6.1 Unitary process consisting of one start event, one user task and one end event. . . 40
6.2 Demonstration acquisition process consisting of one start event, multiple user tasks, multiple decision gateways and two end events. 41
6.3 KPIs summary plot for a 3-Batch policy using MSA, two users, generation interval set to 3 and simulation time set to 50. The left subgraph displays box plots for the KPIs lateness, wait time and service time and the orange box in the upper right corner shows the mean values. The right subgraph the histograms indicate the per user load while the dashed horizontal orange line indicates the average system load. 42
6.4 Evolution plot for a 3-Batch policy using MSA, two users, generation interval set to 3 and simulation time set to 50. Each colored box corresponds to the quanity of tokens in the global respectively user queues. 42

7.1 KPIs comparison for different MILP based policies using MSA. Lower is better. . . 46
7.2 KPIs comparison for different MILP based policies using ST. Lower is better. 47
7.3 Speedup comparison between KPIs for MSA and ST for 5-Batch, 5-Batch-1 and 1-Batch-1. Values bigger than 1.00 indicate a speedup while values smaller than 1.00 indicate a speeddown. Higher is better. 47

8.1 KPIs speedup comparison between MSA and ST for 1-Batch-1. Higher is better. . . 51
8.2 User loads distribution for 1-Batch-1 using MSA. More equally distributed histograms indicate fairer user treatement by the policy as seen in Figure 8.3. 52

8.3 User loads distribution for 1-Batch-1 using ST. More equally distributed histograms indicate fairer user treatement by the policy as seen here. 52
8.4 KPIs comparison between OP and ONP approaches. Values bigger than 1.00 indicate speedup while values smaller than 1.00 indicate speeddown. 53

B.1 LLQP KPIs. 59
B.2 SQ KPIs. 60
B.3 K-Batch with MSA KPIs. 61
B.4 K-Batch with DMF KPIs. 61
B.5 K-Batch with SDMF KPIs. 61
B.6 K-Batch with ESDMF KPIs. 62
B.7 K-Batch with ST KPIs. 62
B.8 K-Batch with MSA batch size comparison. 63
B.9 K-Batch with ST batch size comparison. 63
B.10 K-Batch-1 with MSA KPIs. 64
B.11 K-Batch-1 with DMF KPIs. 64
B.12 K-Batch-1 with SDMF KPIs. 64
B.13 K-Batch-1 with ESDMF KPIs. 65
B.14 K-Batch-1 with ST KPIs. 65
B.15 K-Batch-1 with MSA batch size comparison. 66
B.16 K-Batch-1 with ST batch size comparison. 66
B.17 1-Batch-1 with MSA KPIs. 67
B.18 1-Batch-1 with DMF KPIs. 67
B.19 1-Batch-1 with SDMF KPIs. 67
B.20 1-Batch-1 with ESDMF KPIs. 68
B.21 1-Batch-1 with ST KPIs. 68

C.1 1-Batch with MC and VFA KPIs. 69
C.2 1-Batch with MC, VFA and OP KPIs. 69
C.3 1-Batch with MC, VFA, OP and ϵ-Greedy KPIs. 70
C.4 1-Batch with TD, VFA and OP KPIs. 70
C.5 1-Batch with MC and VFA MSA comparison. 71
C.6 1-Batch with MC, VFA and OP MSA comparison. 71
C.7 1-Batch with MC, VFA, OP and ϵ-Greedy MSA comparison. 71
C.8 1-Batch with TD, VFA and OP MSA comparison. 71
C.9 1-Batch-1 with TD, VFA and OP KPIs. 72
C.10 1-Batch-1 with TD, VFA and OP MSA comparison. 73
C.11 LLQP with MC, VFA and OP KPIs. 74
C.12 LLQP with TD, VFA and OP KPIs. 74
C.13 LLQP with TD, TF and OP KPIs. 74
C.14 LLQP with MC, VFA and OP MSA comparison. 75
C.15 LLQP with TD, VFA and OP MSA comparison. 75
C.16 LLQP with TD, TF and OP MSA comparison. 75
C.17 1L implementation in TF for LLQP policy. 77
C.18 WZ with TD, VFA and OP KPIs. 78
C.19 WZO with TD, VFA and OP KPIs. 78
C.20 BI with MC, TF and 1L KPIs. 78
C.21 BI with MC, TF and 2L KPIs. 79
C.22 BI with MC, TF and 3L KPIs. 79
C.23 BI with MC, TF and 4L KPIs. 79
C.24 WZ with TD, VFA and OP MSA comparison. 80

C.25 WZO with TD, VFA and OP MSA comparison. 80
C.26 BI with MC, TF and 1L MSA comparison. 80
C.27 BI with MC, TF and 2L MSA comparison. 80
C.28 BI with MC, TF and 3L MSA comparison. 81
C.29 BI with MC, TF and 4L MSA comparison. 81

List of Tables

5.1 Description of used varaibles for all MILP formulations. 22
5.2 Comparison of formulation complexities expressed in number of binary variables for different problem formulations where m is the number of users and n is the batch size 25
5.3 Qualitative comparison between MC and TD update methods. Adapted from (Sutton and Barto, 2017, p. 130). 37
5.4 Sample service times of both users for all three jobs. 37

7.1 Global parameters definition for simulation. Column “default” defines the standard values used for the results obtained in this chapter. Column “valid range” defines the range of values that are accepted by each parameter. 45
7.2 Speedup across all KPIs for ST vs MSA. Values bigger than 1.00 indicate speedup. 47
7.3 Global RL parameters. These values complement those outlined in Table 7.1. . . . 48
7.4 Overview of developed batch policies with RL. 48
7.5 Speedup across all KPIs for batch policies with RL vs MSA. Values bigger than 1.00 indicate speedup. 48
7.6 Overview of developed LLQP policies with RL. 49
7.7 Speedup across all KPIs of LLQP policies with RL vs MSA. Values bigger than 1.00 indicate speedup while values smaller than 1.00 indicate speeddown. 49
7.8 Comparison between KPIs for OP and ONP approaches. Both approaches exhibit very similar results. Values bigger than 1.00 indicate speedup while values smaller than 1.00 indicate speedown. 49
7.9 Overview of additional developed policies with RL. These policies are used to effectively emulate arbitrary batch sizes for K-Batch and K-Batch-1 policies. 49
7.10 Speedup across all KPIs of the additional policies with RL against the MSA formulation. Values bigger than 1.00 indicate speedup while values smaller than 1.00 indicate speeddown. 50

C.1 Overview of additional LLQP policies with RL. 76

Chapter 1

Introduction

Workflow Management Systems (WfMS) empower business process automation (Zeng and Zhao, 2005). One crucial aspect of this empowerment consists in effectively assigning jobs to users in WfMS (Zeng and Zhao, 2005). This concept is termed role resolution in the literature (Cheng, 2000). Solving the role resolution problem in WfMS can lead to fully realize the potential gains such as: **I.** Cost savings **II.** Fairness in workload assignment **III.** Optimal resources usage.

Currently most WfMS implement elementary role resolution policies (Zeng and Zhao, 2005). Expanding on Zeng and Zhao (2005)'s work, this thesis focuses on further developing traditional Mixed Integer Linear Programming (MILP) based approaches and proposes novel Reinforcement Learning (RL) based methods for solving the role resolution problem in WfMS.

1.1 Objectives

The objectives of this thesis build upon the work of Zeng and Zhao (2005) in which they describe policies for optimal role resolution in WfMS and extends these capabilities from a twofold perspective: **I.** Further develops the MILP based approaches proposed by Zeng and Zhao (2005) **II.** Explores the capabilities offered by RL in order to overcome the deterministic limitations of traditional MILP based methods by approaching role resolution in WfMS from a stochastic perspective. Formally, this thesis tries to answer the following research questions:

Q. I Do traditional MILP based methods for role resolution in WfMS exhibit further optimization potential?

Q. II Do cutting edge alternative approaches for role resolution in WfMS exist?

1.2 Thesis Structure

This thesis is subdivided in seven main chapters:

- Chapter 2 gives an overview on the state of the art literature.
- Chapter 3 outlines the theoretical foundations in Business Process Model and Notation (BPMN), WfMS, their limitations and role resolution.
- Chapter 4 maps the theoretical concepts outlined in Chapter 3 with the actual discrete event simulation environment used for evaluation. It explains in detail the governing logic of the subset of BPMN elements implemented and the role resolution approach used.

- Chapter 5 initially describes in detail the MILP based approach and how traditional methods have been extended, followed by a theoretical introduction in RL which is eventually used in the culminating part where RL based approaches are explained.
- Chapter 6 describes the evaluation framework implemented, how both role resolution approaches are simulated and defines the Key Performance Indicators (KPIs) used to measure them.
- Chapter 7 initially exposes the results obtained by the MILP based methods and then compares them with the traditional methods of Zeng and Zhao (2005). Subsequently, the results of the RL based methods are outlined.
- Chapter 8 is the culminating chapter in which the results obtained in Chapter 7 are critically discussed and interpreted, a summary of the key findings is outlined and the research questions posed in Section 1.1 are answered. Finally consequences and outlooks about future trends and how the empirical results can be extended by prospective researchers are given.

Chapter 2

Related Work

This chapter servers as an overview of the state of the art literature that exists and has been used as a foundation basis.

2.1 Queueing

Queuing is the act of how people, or more general agents, are to be served while waiting *i.e.*, queueing, inside a system (Kendall, 1953). Starting with the seminal contribution of Kendall (1953) on Markov chains in queuing theory, where he formally defines different types of queues, upon which many researches have been based.

Pinedo (2008) outlines the most prominent key metrics that can be used in order to assert and measure queues performance.

Adan and Resing (2015)'s statistical modeling techniques for randomized generation rates, such as the Erlang's distribution, are used in the discrete event simulation environment.

2.2 Workflow Management Systems

Baker (1974) formally defines the Key Performance Indicator (KPI) used by Zeng and Zhao (2005) for evaluating policies, called task flowtime.

Macintosh (1993) gives an overview of the five levels of process maturity: **I.** Initial, the process has to be set up **II.** Repeatable, the process has to be repeatable **III.** Defined, documentation and standardization of processes **IV.** Managed, measurement and control of processes **V.** Optimizing, continuous process improvement.

Georgakopoulos et al. (1995) give a comprehensive business oriented overview of the different Workflow Management Systems (WfMS) technologies present on the market used as a sound foundation for analyzing today technologies.

Cheng (2000) formally defines the act of assigning jobs to users *i.e.*, role resolution in WfMS.

Following Georgakopoulos et al. (1995)'s business oriented overview, Giaglis (2001) lays out four different process perspectives: **I.** Functional **II.** Behavioral **III.** Organizational **IV.** Informational. Giaglis (2001)'s framework focuses on three dimensions: **I.** Breadth, where modeling goals are typically addressed by technique **II.** Depth, where modeling perspectives are covered **III.** Fit, where typical project to which techniques can be fit. The presented framework is used to combine the three different dimensions in order to assert a possible best fit of a specific modeling technique based on which approach to be used under the constraints of a modeling perspective to cover (Giaglis, 2001).

Mentzas et al. (2001) focus on a qualitative level on how WfMS can facilitate implementation of business processes by describing the pros and cons of adopting alternative Business Process Modeling (BPM) techniques. Moreover, Mentzas et al. (2001) formally define what a WfMS is and subdivide it in three main categories: **I.** Process modeling **II.** Process re-engineering **III.** WfMS implementation and automation. Each level of maturity as defined by Macintosh (1993) requires a different model, such as the first three levels might require more descriptive models whereas levels four and five require decision support keen models in order to monitor and control processes (Mentzas et al., 2001).

Aguilar-Savén (2004) reviews BPM literature and describes the main BPM techniques.

Interestingly enough, WfMS implementation in real world cases is not always only coupled with directly measurable effects, sometimes even unexpected results happen (Reijers and van der Aalst, 2005). What is called the "workflow paradox" according to Reijers and van der Aalst (2005) is the concept that the very fact of companies accepting requests for WfMS introduction might actually be the most promising way that leads to potentially better and more suitable alternatives.

Sörensen (2005) focuses on problematics when modeling WfMS with Business Process Model and Notation (BPMN) such as the presence of cycles and the consequences that these have in respect to termination and progress capability of WfMS.

Effective role resolution *i.e.*, the mechanism of assigning tasks to individual workers at runtime according to the role qualification defined in WfMS as defined by Zeng and Zhao (2005) is the main focus of this thesis. Zeng and Zhao (2005) differentiate between staffing decisions and role resolution, with the former being the assignment of one or more roles to each user and the latter being the assignment of a specific task to an appropriate worker at runtime. Staffing decisions are usually made off-line and periodically, thus being more of a strategic nature (Zeng and Zhao, 2005). If role resolution were to be made on-line it could translate to a major operational level decision *i.e.*, the differentiation between strategic vs operational playing role (Zeng and Zhao, 2005). Zeng and Zhao (2005) moreover define three roles a WfMS can fulfill: **I.** System built-in policies **II.** User customizable policies **III.** Rule based policies. Considering capacities of resources restrictions under the role resolution problem is NP-hard and Zeng and Zhao (2005) focus on how to solve it when accounting for worker's preferences. For this purpose they define five WfMS resolution policies: **I.** Least Loaded Qualified Person (LLQP) **II.** Shared Queue (SQ) **III.** K-Batch **IV.** K-Batch-1 **V.** 1-Batch-1. For all batch policies a simplified version of Dynamic Minimization of Maximum Task Flowtime (DMF) has to be solved (Zeng and Zhao, 2005). Zeng and Zhao (2005)'s key findings are outlined as follows: **I.** Batch policies are to be used when system load is medium to high **II.** Processing time variation has major impact on system performance *i.e.*, higher variation favors optimization based policies **III.** Average workload and workload variation can be significantly reduced by online optimization **IV.** 1-Batch-1 online optimization policy yields best results in operational conditions.

Data flow inside WfMS has to consider possible anomalies that might happen and this aspect has been extensively studies by Sun et al. (2006) where they formally define data flow methodologies for detecting such anomalies. Sun et al. (2006)'s framework is divided in two components: **I.** Data flow specification **II.** Data flow analysis. They moreover discuss aspects such that data requirements have been analyzed but the required methodologies on discovering data flow errors have not been extensively researched (Sun et al., 2006).

A more recent taxonomy of different BPM applications is given by a collaboration between SAP and accenture (Evolved Technologist, 2009).

An analysis of the Critical Success Factors (CSFs) for BPM is required in order to assert product validity and this has been done by Trkman (2010) where he defines CSFs from three perspectives: **I.** Contingency theory **II.** Dynamic capabilities **III.** Task-technology fit theory.

The domain of WfMS is permeated by BPMN and Silver (2011)'s guidelines are excellent formal foundations.

Change management in WfMS is yet another interesting aspect that should be considered and this has been broadly studied by Wang and Zhao (2011) where they developed an analytical framework for WfMS change management through formal modeling of constraints.

In companies different types of WfMS can exist and Fan et al. (2012) focus on two of these, namely: **I.** Conceptual **II.** Logical. Conceptual models serve as documentation for generic process requirements whereas logical models are used as definitions for technology oriented requirements (Fan et al., 2012). One difficult aspect is the transition from the former to the latter and Fan et al. (2012) propose a formal approach to efficiently support such transitions.

Sun and Zhao (2013) cover the aspect of formal analysis for WfMS and they claim that it should help “alleviating the intellectual challenge faced by business analysts.” (Sun and Zhao, 2013, p. 2).

2.3 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning that promises to overcome the drawbacks posed by the latter by not requiring a training set for efficient machine decisions (Sutton and Barto, 2017).

One of the first Monte Carlo (MC) based Policy Gradient (PG) methods is the algorithm proposed by Williams (1992) called `REINFORCE`.

Vanishing Gradient Problem (VGP) states that even very large changes in partial derivatives on initial layers have imperceptible effects on subsequent layers, as outlined by Bengio et al. (1994), which is a problem that affects deep Artificial Neural Networks (ANNs).

Haykin (1998)’s comprehensive work on ANNs is an excellent theoretical foundation which serves as basis for critical concepts regarding the matter.

Theoretical definitions on deep ANNs are also given by Lecun et al. (1998) which were used to better understand more recent developments in this domain.

PG methods with Value Function Approximation (VFA) and their convergence is of vital importance and according to Sutton et al. (1999) this can be achieved by representing the policy by an own function approximation which is independent of the value function and it is updated according to gradient of the expected rewards with respect to the afore mentioned policy.

Another branch originated from RL is Inverse RL (IRL): Ng and Russell (2000) outline the required algorithms for this domain.

Discretization of the state action space is not always feasible and different techniques have to be used for tractability and Smith (2002) proposes such an approach which he calls “self-organizing map”.

Abbeel and Ng (2004) collaboration lays the basis for a bleeding edge branch of IRL called Apprenticeship Learning (AL), which trains policies without rewards function by merely observing “expert” agents performing a task in a specific domain.

Bengio (2009) further develops the foundations laid by Lecun et al. (1998) and expands the theoretical basis of deep ANNs.

Statistical identifiability in RL is a crucial aspect that has to be ensured for effective learning, as were it not the case RL update methods might remain stuck in suboptimal solutions and never converge (Zhang and Hyvärinen, 2011).

Yet another problem in which deep ANNs might incur in addition to VGP is the so called Exploding Gradient Problem (EGP) as defined by Pascanu et al. (2012).

Huge state space requirement is a clear limitation to lookup tables (Sutton and Barto, 2017). Even if memory would not be a constraint, the actual learning from such tables would be infeasible (Sutton and Barto, 2017). In order to pragmatically learn by reinforcement on such huge problems, VFA in the domain of RL proves to be a viable solution (Sutton and Barto, 2017). For different types of RL approaches *i.e.*, MC or Temporal Difference (TD) methods exist different types of VFA, ranging from simple linear combinations of features for MC to ANNs for TD learning (Sutton and Barto, 2017). All these different methodologies are outlined in the tutorial by Geramifard et al. (2013) and complement the theoretical foundations laid by Sutton and Barto (2017).

Overfitting for deep ANNs is a common problem (Sutton and Barto, 2017, p. 218). Srivastava et al. (2014) propose the dropout method as solution which proves to be a domain standard in regards to this problem.

As Markov Decision Processes (MDPs) grow in size, so does the required computational memory to solve possible discrete lookup tables modeling the state-actions spaces that characterizes them (Sutton and Barto, 2017). Sutton and Barto (2017) show notable examples that demonstrate how large some of the most common problems can be: **I.** The game of backgammon has a total of 10^{20} states **II.** The traditional Chinese abstract board game Go has an estimated total of 10^{170} states **III.** Flying a helicopter or having a robot move in space all require a continuous state space.

When working with on-line algorithms such as TD it is important to choose correct parameters for an effective learning process, otherwise the learning algorithm put in place might never converge towards an optimal solution (Sutton and Barto, 2017). This aspect is being discussed by Korda and Prashanth (2014) in which they depict different non-asymptotic bounds for the TD learning algorithms.

There are two main fields in RL, one is using VFA for either the state value function or for using control mechanisms with the state Action Value (AV) function, while the other one is using PG methods for policy optimization (Sutton and Barto, 2017). The latter offers different methods such as the naive finite difference methods, MC based PG methods and finally Actor Critic (AC) PG methods as defined by Silver et al. (2014).

Clevert et al. (2015) discuss a special class of activation functions for ANNs called Exponential Linear Units (ELUs) which are improvements over traditional Rectified Linear Units (ReLUs). ELUs activation functions are used for the modeling of the ANNs layers later on.

Gershman (2016) proposes further advancements to Zhang and Hyvärinen (2011)'s work on statistical identifiability by analyzing prior distributions of the parameters. He argues that his approach helps to some extent to overcome the identifiability problem (Gershman, 2016).

Notable works in the field of RL and its application include DeepMind Technologies Limited's work on novel algorithms for tackling fields previously barely scratched, as mentioned by Mnih et al. (2015) and Silver et al. (2016).

Sutton and Barto (2017) started working on the RL topic in the early nineties and are now planning their third edition of the famous book on RL, which is due in 2017.

2.4 Mixed Integer Linear Programming

The NP-hardness of DMF is formally proved by Garey and Johnson (1990).

For role resolution in WfMS a Mixed Integer Linear Programming (MILP) problem must be solved in order to optimally assign jobs to users. The generalized assignment problem is a very well known problem in combinatorial mathematics and Cattrysse and Wassenhove (1992) give an overview of different algorithms for solving it. Heuristics are also a viable solution for solving such adaptations of the generalized assignment problem, as Racer and Amini (1994) state. Moreover a global perspective of optimization from a mathematical perspective is given in Boyd and Vandenberghe (2004)'s work on convex optimization.

According to the AIMMS guidelines, there are different linear programming maneuvers that can be used to shape such problems in solvable outlines (Bisschop, 2016). Zeng and Zhao (2005) propose that by adding auxiliary variables, their formulation of DMF can be transformed in a

MILP problem. For this purpose, the either-or constraints was used to transform DMF into Extended DMF (EDMF) (Bisschop, 2016, p. 77).

2.5 Simulation

Bahouth et al. (2007) concentrate on algorithmic analysis of discrete event simulation supplemented with focus on factors such as compiler efficiency, code interpretation and caching memory issues. According to their findings, a significant speedup can be achieved if one addresses the afore mentioned facets (Bahouth et al., 2007).

Simulating queues can prove to be arduous (Matloff, 2008). The main differentiation needed here is the one between continuous and step functions: the former is the result when the events being simulated yield values that if plotted against the simulation time give a continuous function (Matloff, 2008). On the other hand, if we simulate events that yield discrete values, such as inventory changes in a storage facility and plot the results against the simulation time we would get so called step functions (Matloff, 2008). According to Matloff (2008), different world views for discrete event programing exist *i.e.*, paradigms: **I.** Activity oriented **II.** Event oriented **III.** Process oriented. Activity oriented can be summarized as simulation events where time is being subdivided in tiny intervals at which the program checks the status for all simulated entities (Matloff, 2008). Since petite subdivisions of time are possible in such types of simulations, it is clear that the program might prove tedious, since most of the time there will not be any change in state for the simulated entities (Matloff, 2008). Event oriented paradigms circumnavigate this issue by advancing the simulation time directly to the next event to be simulated (Matloff, 2008). By filling these gaps, a dramatical increase in computation can be observed (Matloff, 2008). Last but not least, the process oriented simulation models each simulation activity as a process or thread (Matloff, 2008). Management of threads difficulty has steadily decreased in todays computation since many different packages for governing such tasks exist (Matloff, 2008).

Milo (2012) defines speedup as a metric for evaluating comparisons in computer architecture which is used when analyzing the results of this thesis.

Chapter 3

Theoretical Foundations

3.1 Business Process Model and Notation

Business Process Modeling (BPM) is the act of representing a business process (Silver, 2011). Modeling requires a standardized notation that allows for consistent model representation and this is achieved in the BPM field by the Business Process Model and Notation (BPMN) which is a graphical representation of a semantic process (Silver, 2011). BPMN is thus to be understood as a language used to draw diagrams *i.e.*, a diagramming language (Silver, 2011). Its most favorable point is that it is a standard maintained by the Object Management Group (OMG) and that it is widely adopted worldwide (Silver, 2011). Silver (2011) outlines two fundamental principles that BPMN should adhere to:

P. I "A given BPMN diagram should have one and only one interpretation. The process logic should be completely and unambiguously described by the diagram alone." (Silver, 2011, p. v).

P. II "A given BPMN diagram should have one and only one Extensible Markup Language (XML) serialization. Otherwise model interchange between tools cannot be achieved." (Silver, 2011, p. v).

Principle I is more modelers-oriented with no necessary implementation knowledge that should be able to fully understand a diagram by looking at it, while Principle II applies more to implementers and process interpreters (Silver, 2011).

Silver (2011) differentiates between two important concepts: **I.** Activities **II.** Processes. Activities are to be understood as units of work performed, or as Silver (2011) defines them: "discrete actions with well-defined start and end that are performed repeatedly in the course of business, whose instances can be arbitrarily repeated." (Silver, 2011, p. 10). More informally one can see each activity instance as a token that is being generated at a starting state and during its lifetime follows a path along the process flow. On the other hand, processes are sequence of actions that lead from a start state to and end state (Silver, 2011, p. 11).

Modeling such processes is done by relying as previously mentioned on BPMN in which different building blocks are defined (Silver, 2011). A traditional model designed following the BPMN guidelines can be found in Figure 3.1.

In the following paragraphs the different elements outlined are explained and Figure 3.2 gives an overview of the graphical representation in BPMN of them.

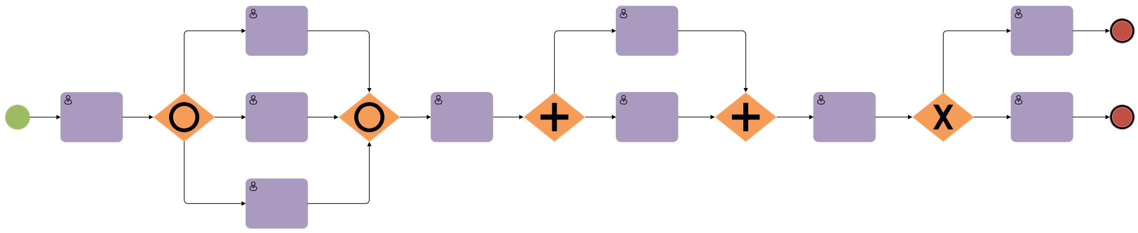

Figure 3.1: BPMN process showing a usual process starting with a start event (green) on the leftmost side, followed by different user tasks (purple) and gateways (orange) and eventually two different end events (red).

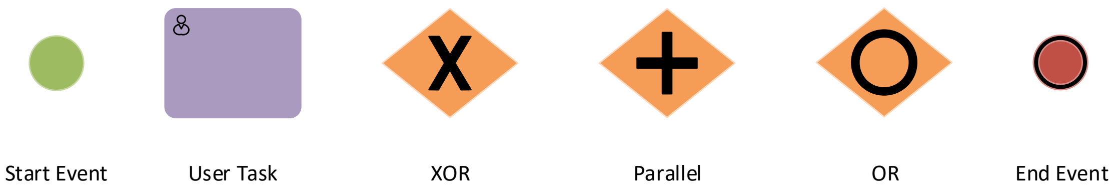

Figure 3.2: Single BPMN elements with their corresponding labeling according to BPMN's notation.

3.1.1 Start Event

The main purpose of start events is to indicate where a process starts and it is a strict requirement set by BPMN that each process must have such one and at most one (Silver, 2011). Technically, start events are those process elements that generate tokens that flow through a process which when they reach a user task get worked by the assigned user.

3.1.2 User Task

Activities are defined by Silver (2011) as atomic units of work that are performed in a process. They are the only elements that assign a performer to them (Silver, 2011). For the purpose of this thesis, the only type of activities considered is the user task. They get a user assigned which will work the token that has reached this task.

3.1.3 Gateway

Diamond shaped gateways are elements that control the tokens flow inside a process by effectively splitting its path into alternative ones (Silver, 2011). There are three main types of gateways: **I.** Exclusive OR (XOR) **II.** Parallel **III.** Inclusive OR (OR).

XOR gateways define exclusivity among the alternative paths a token can flow through, meaning that only one path can be pursued per instance (Silver, 2011). XOR gateways do not make decisions, they merely test conditions to activate a specific path (Silver, 2011). The decision making process is left to tasks (Silver, 2011).

Parallel gateways, in contrast to XOR gateways, unconditionally split and duplicate the flow of a token inside a process (Silver, 2011).

Last but not least, OR gateways are in a way similar to XOR gateways since they also have conditional flows but each condition is independent *i.e.*, more than one can be true and if that is the case, then the paths are split following the parallel gateway logic (Silver, 2011). Normally, OR gateways always have a default path to be followed (Silver, 2011).

3.1.4 End Event

End events indicate the end of the path for a token inside the process (Silver, 2011). In contrast to start events, it is possible to have multiple end events (Silver, 2011). As soon as a token reaches an end event, the corresponding activity instance is completed.

3.2 Business Process Model and Notation Limitations

Even though BPMN advocates for a formal semantic definition, there are formal gaps in more complex semantic constructs which are under-specified that should be accounted for when modeling processes with BPMN (Sörensen, 2005). Sörensen (2005) mentions the most critical specification areas lacking formal semantics: **I.** OR gateways with arbitrary number of cycles **II.** Cyclic processes in general **III.** Processes presenting deadlocks that might not ensure termination or progress.

Both Figure 3.3 and Figure 3.4 show the representation of two different types of cyclic diagrams that can cause problems, as outlined by Sörensen (2005).

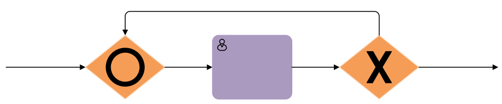

Figure 3.3: Cyclic diagram with repeat loop. Adapted from Sörensen (2005, p. 12).

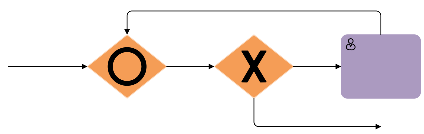

Figure 3.4: Cyclic diagram with while loop. Adapted from Sörensen (2005, p. 12).

As it is out of scope for this thesis to reiterate over Sörensen (2005)'s work on cyclic processes, we only model and subsequently simulate acyclic processes.

In a real world adoption of Workflow Management Systems (WfMS) bottlenecks can be encountered on each stage along a process. In the discrete event simulation environment proposed in this thesis, however, the focus is put only on user tasks and all other process elements are considered to be executed instantly.

3.3 Policy Based Role Resolution

As mentioned in Section 3.1, user task elements get users assigned which then work tokens that reach this specific task. The concept of assigning a user to a token (or task) is referred to in the literature as role resolution (Cheng, 2000; Zeng and Zhao, 2005). When referring to Figure 3.1, we see that tokens are being generated by the left most start event and immediately reach a user task. Let us moreover assume that we have a pool of different users that all qualify to process the token at this specific user task, however they all exhibit different skill levels and at the time of arrival they are also variously occupied.

Formally, applying role resolution is done by following predefined policies which govern how users are being effectively assigned to work tokens (Zeng and Zhao, 2005). As already mentioned, this thesis focuses on extending the foundations laid by Zeng and Zhao (2005) in which they define five different types of policies for effective role resolution.

In order to effectively test different polices governing role resolution in WfMS, in this thesis a discrete event simulation environment has been prepared. By means of simulating processes with different policies acting as supervisors, different metrics can be evaluated to assess policies' efficiency. Role resolution in this simulation environment happens on a per user task basis *i.e.*, when a token reaches a user task, a policy follows its internal definitions on how to optimally assign available users to this token. A simulation environment has been implemented since on one hand it allows to generate the required amount of data relatively fast (which in a real world situation not only could prove to be hard to obtain, but extremely costly as well) and on the other hand, it allows to equally test all role resolution policies by ensuring that they all undergo the exact same processes with the same conditions.

Chapter 4

Discrete Event Simulation

As previously outlined in Section 3.3, a discrete event simulation has been implemented in which role resolution policies have been tested. What follows is the description of how this environment has been implemented, then in Section 4.1 the Workflow Management System (WfMS) environment's implementation based on the outlined foundations in Section 3.1 is described and eventually in Section 4.2 the role resolution's implementation based on the outlined foundations in Section 3.3 is also explained.

`SimPy` is a `Python` process-based discrete-event simulation framework. It exploits `Python`'s generators according to which it models its processes.

Active components such as agents in a WfMS are modeled as processes which live inside an environment and the interaction between them happens via events.

As previously mentioned, processes in `SimPy` are described by `Python` generators. During their lifetime they create events, yield (note that with the term `yield` here it is to be understood as `Python`'s yield statements)[1] them to the environment, which then wait until they are triggered. The important logic to understand is how `SimPy` treats yielded events: when a process yields an event it gets suspended. From the suspended state a process gets resumed when the event actually occurs (or in `SimPy`'s notation when it gets triggered).

`SimPy` offers a built-in event type called `Timeout`: events of this type are automatically triggered after a determined simulation time step. Consistency is asserted since timeout events are created and called by the appropriate method of the passed `Environment`.

4.1 Business Process Model and Notation Implementation

The analysis environment consists in an object-oriented implementation of WfMS elements such as start events, user tasks, gateways and end events which have been developed to allow the simulation framework to effectively run. They are built upon the formal foundations outlined in Section 3.1. This object-oriented exoskeleton implementation of the WfMS elements can be seen depicted in Figure 4.1.

4.1.1 Start Event

Start events require a simulation environment, a generation interval, an actions to follow array and its corresponding weights. The generation interval is a plain scalar value in contrast to Zeng

[1] http://stackoverflow.com/questions/231767/what-does-the-yield-keyword-do-in-python (accessed 26.04.2017)

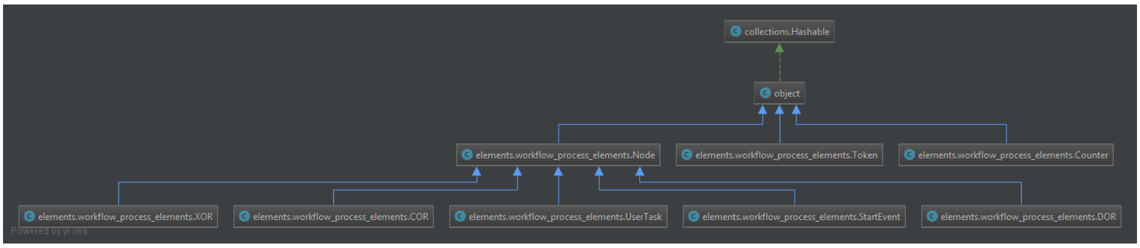

Figure 4.1: Class diagram of BPMN elements implementation for the discrete event simulation environment.

and Zhao (2005)'s work, where the generation λ interval follows a Poisson distribution and is defined as shown in Equation 4.1 with a fixed service interval time unit s, number of users n and an average system load l.

$$\lambda = \frac{ln}{s} \tag{4.1}$$

The path flow to be followed by the tokens generated is defined in a three step process: **I.** A per process action pool is defined a priori in order to assert that tokens navigate the process in a "semantically correct" fashion. This is achieved by creating an array containing an arbitrary number of dictionaries *i.e.*, hash-maps, where the key is the gateway node id and the value is which path *i.e.*, child, should be chosen next (see Listing 4.1) **II.** A weights vector is defined which assigns a probability to each possible path flow to the actions pool (see Listing 4.2) **III.** Start events sample a path by accounting for the weighted probability and assign it to each newly generated token (see Listing 4.3).

```
actions_pool = [{xor.node_id: 0, xor_a.node_id: 0, dor_c.node_id: (1, 2), xor_d
    .node_id: 1, xor_f.node_id: 0, cor_g.node_id: 2, xor_h.node_id: 0}]
```

Listing 4.1: Fixed actions pool consisting of different hashmaps each corresponding to a full path a token must follow during an instance. Keys correspond to node ids and values define which direction to take at gateways.

```
weights = [1 / len(actions_pool) for _ in range(len(actions_pool))]
```

Listing 4.2: Probability weights vector defining how often a path will be chosen. In this case equal probability is assigned to all paths in the action pool.

```
def generate_tokens(self):
    while True:
        if self.actions is not None:
            path = token.random_state.choice(self.actions, p=self.weights)
            token.actions = path
```

Listing 4.3: Start event token generation method in which an action is being sampled from the action pool by accounting for its weighted probability.

Such an approach allows to fine tune how often tokens will follow a predefined path flow along the process in order to efficiently simulate and put under stress specific path flows of the process.

The length of the simulation can either run indefinitely or be constrained to an upper bound in time steps t as it can be seen from Listing 4.4. This means that the simulation will persist for

100 time steps and it will then stop when the internal clock reaches 100[2]. The logic is similar to a new environment where the clock is zero and no events have been processed yet.

```
# "global" variables
SIM_TIME = 100
# runs simulation
env.run(until=SIM_TIME)
```

Listing 4.4: Discrete event simulation environment start by calling its corresponding run method with a discrete simulation time step t set. In this case the simulation will run for 100 time steps.

Master Random State

In order to assert fairness among all simulation runs a master random state is assigned to the start event. This master random state is generated from the PCG[3] family of random generators which exhibit peculiar characteristics, one amongst all is the possibility of "jumping ahead" in the state. The novelty of this approach allows to assign a fixed number of random yet consistent choices among all runs, since each generated token receives from the start event a "jumped" copy from the master state. The implementation of this recursive ahead jumps in random states is depicted in Listing 4.5.

```
def generate_tokens(self):
    while True:
        random_state = self.advance_master_state()
        token = Token(random_state)

def advance_master_state(self):
    current_state = self.master_state.get_state()
    random_state = pcg.RandomState()
    random_state.set_state(current_state)
    self.master_state.advance(int(1e9))
    return random_state
```

Listing 4.5: Random state advancement method which initially copies the current state of the master random state. It then creates a new random state and sets its state to the former copy then eventually advances the master random state by 1×10^9 units.

This approach greatly simplifies the analysis of the results since no matter the simulation conformation, it permits to ensure consistency and reproducibility among all configurations. Specifically, since each token gets a "jumped" copy of the master random state, it is ensured that under any circumstances the generation time during the simulation is consistent.

Figure 4.2 and Figure 4.3 are snippets of the statistical data dumps for two arbitrary simulation runs that clearly display the randomness consistency concept outlined before. When comparing the arrival column on both figures, it is possible to see that even though different number of jobs are being generated, they are consistently generated across different simulation, each arriving at the same simulation time t[4].

[2] Note that events that have been scheduled for time step 100 will not be processed.
[3] http://www.pcg-random.org/ (accessed 25.04.2017)
[4] Note that both figures only show data filtered corresponding to the first direct user task connected to the start event.

job	arrival	assigned	started	finished
0	5.311109689	5.311109689	5.311109689	5.932433185
4	9.712642076	9.712642076	9.712642076	9.830042582
6	19.58479207	19.58479207	19.58479207	20.27429297
11	22.38095224	22.38095224	22.38095224	22.77909589
15	26.638781	26.638781	26.638781	26.69025169
18	33.52744551	33.52744551	33.52744551	33.55946552
23	36.31998313	36.31998313	36.31998313	37.47852927
24	36.50819015	36.50819015	36.50819015	37.38701089
25	36.94657375	36.94657375	36.94657375	37.00416155
31	37.6882068	37.6882068	37.6882068	39.14904432
38	39.38223763	39.38223763	40.84913047	42.13016635
39	40.19008378	40.19008378	42.13016635	42.15046548
41	41.75883248	41.75883248	41.75883248	42.75968576
52	48.83897106	48.83897106	48.83897106	48.91014329

Figure 4.2: Simulation run with MSA showing token times at the first directly connected user task to the start event.

job	arrival	assigned	started	finished
0	5.311109689	5.311109689	5.311109689	5.932433185
4	9.712642076	9.712642076	9.712642076	9.830042582
6	19.58479207	19.58479207	19.58479207	20.27429297
11	22.38095224	22.38095224	22.38095224	22.77909589
15	26.638781	26.638781	26.638781	26.69025169
18	33.52744551	33.52744551	33.52744551	33.55946552
23	36.31998313	36.31998313	36.31998313	37.47852927
24	36.50819015	36.50819015	36.50819015	36.76911785
27	36.94657375	36.94657375	36.94657375	37.30895996
33	37.6882068	38.2358957	38.2358957	39.69673321
37	39.38223763	39.38223763	39.38223763	40.12551887
41	40.19008378	40.19008378	40.19008378	40.21038291
47	41.75883248	41.75883248	41.75883248	43.45140794
52	48.83897106	48.83897106	48.83897106	48.91014329

Figure 4.3: Simulation run with ST showing token times at the first directly connected user task to the start event.

4.1.2 User Task

User tasks also require a simulation environment, a policy, a descriptive name, a service interval and task variability. Each user task has a unique `child` field which is being set prior to starting the simulation.

Each user task has a claim token method, which takes tokens as input parameters and finally makes a call to its designed policy, passing the token. The logic is straightforward: start events generate tokens, user tasks that are direct children of start events claim the newly generated tokens, ask the designated policies to assign a user to the token and finally, after a service interval timeout which corresponds to the user's specific service interval has passed, they release the token. The logic can be seen in Listing 4.6.

```
def claim_token(self, token):
    token.worked_by(self)
    policy_job = self.policy.request(self, token)
    service_time = yield policy_job.request_event
    yield self.env.timeout(service_time)
    self.policy.release(policy_job)
```

Listing 4.6: User task claim method where a call to the corresponding policy request method is made. The policy performs role resolution effectively mapping a token to a user and returns the user's service time which is then yielded as a timeout in the discrete event simulation environment. When this time has elapsed, the policy releases the user.

4.1.3 Gateway

Gateways are those process nodes where conditions are tested (see Section 3.1). As previously mentioned, in the discrete event simulation environment each newly generated token holds an array with all conditions assigned to it, which are then tested at decision nodes in order to correctly forward or split the token along the path flow. For the purpose of this thesis two types of gateways have been implemented: **I.** Exclusive OR (XOR) **II.** Inclusive OR (OR) whose logic has been split between its converging and diverging part.

XOR

XOR's logic is straightforward: each gateway has a forward method which is used to correctly move the token along the process. As soon as a token reaches a XOR gateway, the token's condition is tested and then forwarded to the next element. The implementation can be seen in Listing 4.7.

```
class XOR(Node):
    def forward(self, token):
        token.worked_by(self)
        action = token.get_action(self)
        child = self.children[action]
        self.child_forward(child, self.env, token)
```

Listing 4.7: XOR's forward method in which the to be forwarded to node is retrieved from the token path by looking for the corresponding value with the node id. Eventually a call the XOR's forward method is made.

OR

OR gateways, in contrast to XOR gateways, require a synchronization between number of split path flows being fired at their diverging state which eventually reach a converging state. In other words, the number of parallel tokens fired at the diverging node must all be accounted for at the converging node. This is achieved by a counter that each token holds: when a token reaches a divergent OR gateway, the token's conditions are read and for each path flow that the token has to follow its counter is incremented by 1. The logic is depicted in Listing 4.8.

```
class DOR(Node):
    def choose_child(self, action, token):
        child = self.children[action]
        token.counter.increment()
        self.child_forward(child, self.env, token)
```

Listing 4.8: Token's counter increment logic at a divergent OR gateway in which the corresponding counter object is incremeted by 1 at each method's call.

Each diverging OR gateway has a corresponding converging OR gateway which is used to catch all parallel tokens fired by the former. When a token reaches a converging OR gateway, its counter is decremented each time by 1 and only when the counter reaches 0 the token is actually forwarded along. This logic can be seen in Listing 4.9.

```
class COR(Node):
    def forward(self, token):
        token.worked_by(self)
        token.counter.decrement()
        if token.counter.count == 0:
            action = token.get_action(self)
            child = self.children[action]
            self.child_forward(child, self.env, token)
```

Listing 4.9: Token's counter decrement logic at a convergent OR gateway in which the corresponding counter object is decremented by 1 at each method's call.

4.1.4 End Event

In `Python` one does not have to preallocate or deallocate memory by hand as when using pointer based programming languages such as `C` or `C++` since this is being taken care of internally. According to Silver (2011)'s definition of activities, processes and end events (refer to Section 3.1), when a token reaches an end event, the corresponding process' activity instance is deleted and to be never again used. This is exactly what can be implicitly achieved by `Python`'s internal garbage collection strategies: tokens are being generated by start events, when they reach the last element in the process, all references to the token are extinguished thus effectively engaging the internal garbage collection strategies which deallocate any memory previously allocated for it. This translates into the fact that there is no need to explicitly model end events.

4.2 Policy Based Role Resolution Implementation

Role resolution according to policies acting as supervisors (as explained in Section 3.3) in the discrete event simulation environment is achieved by means of a particular classes *i.e.*, policies and policy jobs. The implementation can be seen in Figure 4.4.

Policies do not directly participate in the WfMS, it serves a role as a general supervisor that has the whole overview of the process allowing it to operate on an abstract level.

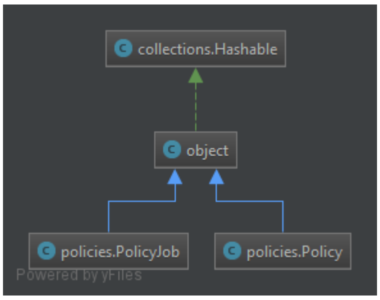

Figure 4.4: Class diagram showing policies and policy jobs classes as independent objects.

Each policy is a blueprint for the actual implementation of the policy itself. It holds minimal information such as a simulation environment, number of users and worker variability. As a blueprint, each policy defines two abstract methods for requesting an optimal assignment for a specific token and for later releasing that token and effectively freeing the user that was busy working on it. Refer to Figure 4.5 for its implementation overview.

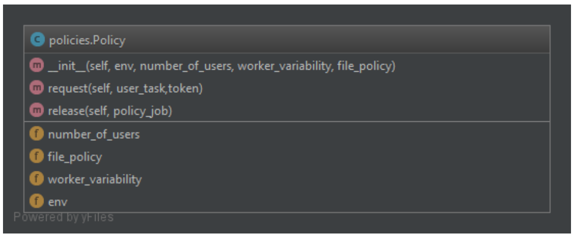

Figure 4.5: Class diagram showing policy methods and attributes.

In its request method, each policy generates a policy job, which is again an abstract implementation of a job that the policy will work in order to return an optimized assignment to a user task. Each policy job requires a user task and a token as initialization parameters in order to be uniquely identifiable inside the whole process. Moreover, each policy job serves as a bookkeeping agent by storing and dumping useful information every time its status changes, such as arrival, assigned, started and finished times, assigned user and a list of service times for all available users. Refer to Figure 4.6 for its implementation overview.

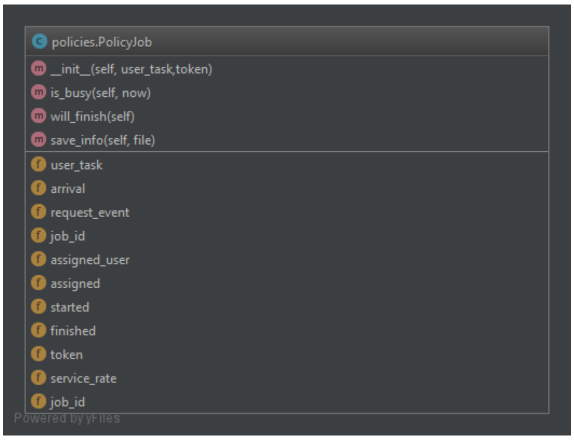

Figure 4.6: Class diagram showing policy job methods and attributes.

In regards to parameters service interval and task variability defined in Subsection 4.1.2 a detailed explanation is required. Both are used to randomly sample service rate intervals for each user active during the simulation. Zeng and Zhao (2005, p. 8) follow a two way process to generate such intervals. However a refined version of this process is used in this case: **I.** At initialization time, each user task receives a service rate s and a task variability t values **II.** Inside the policy request method for each user task a sample of an average processing time is made, following an Erlang distribution which takes as input parameters a shape k and a scale θ. In this case both values k and θ are dynamically evaluated at runtime as $k = s/t$ and $\theta = t$. This concept is depicted in Listing 4.10 **III.** The average processing time becomes a unique value of each user task and is used by each policy to sample each user's service time, again from an Erlang sampled pool as depicted in Listing 4.10 which we shall call p_j.

For each user eligible to work the assigned token, its service rate is sampled following the Erlang distribution. This time, the Erlang distribution takes as parameters the unique average processing time p_j of user task j and a worker variability w, which is a unique property of each policy.

In order to sample a service rate p_{ij} following the Erlang distribution for each user i working tokens at user task j, shape k is evaluated as $k = p_j/w$ and scale θ as $\theta = w$ as it can be seen in Listing 4.10

```
def request(self, user_task,token):
    average_processing_time = token.random_state.gamma(user_task.
        service_interval ** 2 / user_task.task_variability, user_task.
        task_variability / user_task.service_interval)
    policy_job.service_rate = [token.random_state.gamma(average_processing_time
        ** 2 / self.worker_variability, self.worker_variability /
```

```
    average_processing_time) for _ in range(self.number_of_users)]
```

Listing 4.10: User service rate sampling following an Erlang distribution where initially the average processing time is sampled. Afterwards, for each user a service time is sampled yet again relying on the average processing time and is assigned as an array to the policy job.

The Erlang distribution is better suited to model service rates since with an appropriate k one can approximate a normal distribution without incurring in the aspect of having to manually reset negative values to one. This is ensured by the formal definition of Erlang's validity range $x \in [0, \infty)$.

NumPy's implementation of the Erlang distribution is used[5]. Equation 4.2 defines the probability density function of the Erlang distribution with the alternative parametrization that uses μ instead of λ as scale parameter, which is its reciprocal.

$$f(x; k, \mu) = \frac{x^{k-1} e^{-\frac{x}{\mu}}}{\mu^k (k-1)!} \quad \text{for} \quad x, \mu \geq 0 \tag{4.2}$$

[5] https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.gamma.html (accessed 06.01.2017)

Chapter 5

Role Resolution Policies

Zeng and Zhao (2005, p. 7) investigate five role resolution policies used for optimal job assignment in Workflow Management Systems (WfMS). Following a brief description of the five aforementioned policies:

P. I A load balancing policy consists in assigning a task as soon as it arrives to a qualified worker with the shortest task queue at that moment. In this policy workers execute tasks assigned to them on a First In First Out (FIFO) basis. The authors call this policy Least Loaded Qualified Person (LLQP).

P. II A policy that maintains a single queue being shared among all users is referred to as Shared Queue (SQ).

P. III Another policy that maintains both a SQ among all users and each user having an own queue and transfers tasks from the former to the latter is called K-Batch policy. Transfer of tasks from the SQ to users is done using an optimal task assignment procedure as soon as the SQ reaches a critical batch size K.

P. IV The next policy takes the K-Batch policy but reduces the individual queue size of each user to one. This means that the optimization problem is still being solved as soon as the SQ reaches the critical size K, however actual movement of tasks from the SQ to the individual user queue happens only when user i is not busy *i.e.*, his individual queue is empty at simulation time t. This policy is called K-Batch-1.

P. V The last policy further simplifies K-Batch-1 by weakening the batch size constraint by reducing it to one. This means that the optimal task assignment procedure is executed immediately. This policy is referred to as 1-Batch-1.

Section 5.1 describes in detail the Mixed Integer Linear Programming (MILP) approach used in order to solve the role resolution procedure required for K-Batch, K-Batch-1 and 1-Batch-1.

5.1 Mixed Integer Linear Programming Policies

All batch policies require the solution of an optimization problem as explained in Chapter 5. Table 5.1 gives an overview of all variables used in this section for all MILP formulations.

Zeng and Zhao (2005) define this optimization as “minimizing the maximum flowtime given the dynamic availability of the workers” and call it Minimizing Sequential Assignment (MSA) (Zeng and Zhao, 2005, p. 7). Zeng and Zhao (2005) define the task flowtime as the elapsed

Variable	Description
i	User
j	Job
k	Job assignment order
x	Job to user assignment
p	User service time for job
a	User busy time
w	Job waiting time in queue
z	Maximum flowtime

Table 5.1: Description of used varaibles for all MILP formulations.

simulation time between task generation and its completion (Baker, 1974; Zeng and Zhao, 2005). Formally MSA is formulated as follows:

$$\begin{aligned} \min \quad & z \\ \text{subject to:} \quad & \\ \sum_{i \in W} x_{ij} &= 1 \quad \forall j \in T \\ a_i + \sum_{j \in T} x_{ij} p_{ij} &\leq z \quad \forall i \in W \\ x_{ij} = 0 \quad \text{or} \quad x_{ij} &= 1 \quad \forall i \in W, \forall j \in T \end{aligned} \tag{5.1}$$

Zeng and Zhao (2005)' definition of MSA is however a simplified version of "minimizing the maximum task flowtime with consideration of the dynamic arrival of tasks", defined as Dynamic Minimization of Maximum Task Flowtime (DMF) (Baker, 1974; Zeng and Zhao, 2005). DMF is formally defined by Zeng and Zhao (2005) as follows:

$$\begin{aligned} \min \quad & z \\ \text{subject to:} \quad & \\ \sum_{i \in W} \sum_{k \in T} x_{ijk} &= 1 \quad \forall j \in T \\ s_j &\geq r_j \quad \forall j \in T \\ (x_{ijk} + x_{ij'(k+1)} - 1)(s_j + p_{ij}) &\leq s_{j'} \quad \forall i \in W, \forall k \in T, \forall j \in T, \forall j' \in T \\ s_j + \sum_{i \in W} \sum_{k \in T} p_{ij} x_{ijk} - r_j &\leq z \quad \forall j \in T \\ x_{ijk} = 0 \quad \text{or} \quad x_{ijk} &= 1 \quad \forall i \in W, \forall j \in T, \forall k \in T \\ s_j &\geq 0 \end{aligned} \tag{5.2}$$

As Zeng and Zhao (2005) note in their work, Equation 5.2 contains nonlinear constraints and mention, that by adding auxiliary variables the aforementioned DMF formulation can be effectively converted into a MILP problem and thus solve it (Zeng and Zhao, 2005, p. 6). On this note they argue that the application of DMF in practice poses some problems (Zeng and Zhao, 2005). In this thesis a conversion of the DMF formulation proposed by Zeng and Zhao (2005) by adding the required auxiliary variables is introduced in order to adequately solve the optimization problem. The formulation is called Extended DMF (EDMF) and is devised as follows:

$$
\begin{aligned}
\min \quad & z_{\text{max}} \\
\text{subject to:} & \\
\sum_{i \in W} \sum_{k \in T} x_{ijk} &= 1 \quad \forall j \in T \\
a_i + \sum_{j \in T} p_{ij} x_{ijk} &\leq z_{i*k} \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k = 0 \\
z_{i*k-1} + \sum_{j \in T} p_{ij} x_{ijk} &\leq z_{i*k} \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k > 0 \\
z_{i*k} + \sum_{j \in T} w_j x_{ijk} &\leq z_{\text{max}} \quad \forall i \in W, \forall k \in T \\
\sum_{j \in T} x_{ijk} &\leq 1 \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k = 0 \\
\sum_{j \in T} x_{ijk} &\leq \sum_{j \in T} x_{ijk-1} \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k > 0 \\
z_{i*k} &\geq 0 \quad \forall i \in W, \forall k \in T
\end{aligned}
\tag{5.3}
$$

This formulation clearly gets rid of the nonlinear constraints while still accounting for dynamical arrival of tasks, making thus DMF as defined by Zeng and Zhao (2005) effectively solvable.

When considering the minimization of the maximum flowtime of a task inside a process, EDMF can be further simplified by adopting some assumptions about the order and sequence of tasks. Based on how batch policies are implemented, queued policy jobs are implicitly stored in a sorted fashion. This means that the z helper variable defined for EDMF is not strictly necessary and thus can be compressed by Equation 5.4:

$$
a_i + \sum_{t=1}^{k} \sum_{j} (p_{ij} + w_j I(t = k)) x_{ijt} \tag{5.4}
$$

The whole concept consists in the introduction of an identity variable I which is true if and only if task j is currently being assigned as the kth task to user i, meaning that for this specific case also the waiting time for task j has to be accounted for. For all other cases *i.e.*, $j < k$, the identity variable I will not hold thus effectively zeroing the w_j variable.

In order to calculate z_{ijk}, one has to consider when user i will actually be available to process his first task as well. This is depicted by the variable a_i, which summed together with the respective service times of user i for task j gives the complete work time user i will require in order to process all tasks assigned to him.

Without further ado, the simplified formulation of EDMF *i.e.*, Simplified DMF (SDMF), is the following:

$$
\begin{aligned}
&\min \quad z_{\max} \\
&\text{subject to:} \\
&\sum_{i \in W} \sum_{k \in T} x_{ijk} = 1 \quad \forall j \in T \\
a_i + \sum_{t=1}^{k} \sum_{j} (p_{ij} + w_j I(t = k)) x_{ijt} \leq z_{\max} \\
&\sum_{j \in T} x_{ijk} \leq 1 \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k = 0 \\
&\sum_{j \in T} x_{ijk} \leq \sum_{j \in T} x_{ijk-1} \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k > 0
\end{aligned}
\tag{5.5}
$$

By comparing both formulations it is clear that SDMF manages to simplify the mathematical formulation and relaxing the required amount of constraints while still attaining the same level of effectiveness[1].

Based on this approach and by further exploiting the implicit order implementation of task arrival, it is possible to argue that the k sequence indexing can be relaxed as well.

The formulation of DMF by relaxing both the z variables and k indexes is the following:

$$
\begin{aligned}
&\min \quad z_{\max} \\
&\text{subject to:} \\
&\sum_{i \in W} x_{ij} = 1 \quad \forall j \in T \\
a_i + \sum_{k=1}^{j} (p_{ik} + w_k I(k = j)) x_{ik} \leq z_{\max}
\end{aligned}
\tag{5.6}
$$

and is colloquially called Extremely Simplified DMF (ESDMF).

This version is however only possible by the nature of its implementation. Since both the global as well as the local queues are implemented as FIFO queues, it is possible to relax the ordering constraint from the mathematical formulation since it is already implicitly defined by the implementation.

The culminating "flagship" improvement of the optimization method proposed by Zeng and Zhao (2005) done in this thesis is a method that aims to optimally solve the assignment problem by changing its goal: minimize the service times by setting an upper bound on the maximum flowtime and is called Service Time Minimization with ESDMF as Upper Bound (ST). This method uses a two step process in order to optimally apply role resolution: **I.** Solve to optimality by using ESDMF. This yields an upper bound for the maximum flowtime **II.** Use this upper bound as a constraint for the actual optimization in order to effectively optimize the problem for the minimal service time amongst users, jobs and their corresponding service time.

The formulation of ST is the following:

[1] Note, however, that this simplification is only possible because of the nature of the implementation.

$$
\begin{aligned}
\min \quad & \sum_{i \in W} \sum_{k \in T} z_{ik} \\
& \text{subject to:} \\
& \sum_{i \in W} \sum_{k \in T} x_{ijk} = 1 \quad \forall j \in T \\
a_i + \sum_{j \in T} p_{ij} x_{ijk} - M(1 - \sum_{j \in T} x_{ijk}) \leq z_{i*k} & \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k = 0 \\
z_{i*k-1} + \sum_{j \in T} p_{ij} x_{ijk} - M(1 - \sum_{j \in T} x_{ijk}) \leq z_{i*k} & \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k > 0 \\
z_{i*k} + \sum_{j \in T} w_j x_{ijk} \leq z_{\text{max}} + \epsilon & \quad \forall i \in W, \forall k \in T \\
\sum_{j \in T} x_{ijk} \leq 1 & \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k = 0 \\
\sum_{j \in T} x_{ijk} \leq \sum_{j \in T} x_{ijk-1} & \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k > 0 \\
z_{i*k} \geq 0 & \quad \forall i \in W, \forall k \in T \\
M = \max a_i + \max \sum_{i \in W} \sum_{j \in T} p_{ij} & \\
\epsilon = 1 \times 10^{-4} &
\end{aligned}
\tag{5.7}
$$

Table 5.2 shows the formulation complexity for the methods outlined in this chapter where m is the number of users and n is the batch size. The formulation complexity is expressed in number of binary variables that each problem formulation has. The MSA method is the simplest formulation and exhibits a linear complexity compared to the DMF method proposed by Zeng and Zhao (2005). As it can be seen the methods implemented in this thesis, specifically the ESDMF method, all solve the DMF method and do it by keeping the same linear complexity as the MSA method. The ST method proposed in this thesis exhibits the same formulation complexity as the traditional DMF, as shown in Equation 5.8

$$
\mathcal{O}(mn + mn^2) \subseteq \mathcal{O}(mn^2) \tag{5.8}
$$

and yet achieves a better optimization. This trade-off however requires a more in depth explanation which will follow in Section 8.1.

Problem Formulation	Formulation Complexity
MSA	$\mathcal{O}(mn)$
DMF	$\mathcal{O}(mn^2)$
SDMF	$\mathcal{O}(mn^2)$
ESDMF	$\mathcal{O}(mn)$
ST	$\mathcal{O}(mn^2)$

Table 5.2: Comparison of formulation complexities expressed in number of binary variables for different problem formulations where m is the number of users and n is the batch size

5.2 Reinforcement Learning Theory

In this section the Reinforcement Learning (RL) approach used to solve the different role resolution problems is depicted.

5.2.1 Reinforcement Learning Definition

RL is a novel approach originated as a branch from the broader field of machine learning (Sutton and Barto, 2017). It is an automated approach to understanding and automating learning and decision-making (Sutton and Barto, 2017, p. 15). It distinguishes itself from other approaches by its novel focus on learning thanks to an agent which directly interacts with its environment, without the necessity of relying on training sets (Sutton and Barto, 2017, p. 15).

The formal framework used by RL defines the interaction between the so called learning agent and its environment by means of states, actions and rewards (Sutton and Barto, 2017, p. 15).

Key concepts in the field of RL are those of values and value functions which help distinguish RL methods from evolutionary methods which have to undergo scalar evaluations of entire policies (Sutton and Barto, 2017, p. 15).

5.2.2 Finite Markov Decision Processes

RL approaches learn by interacting with the environment in order to achieve a goal (Sutton and Barto, 2017). The agent interacting with the environment does this in a sequence of discrete time steps, it performs actions[2], reaches then states[3] and eventually receives rewards[4] (Sutton and Barto, 2017, p. 73). Moreover, a policy is a stochastic rule that the agent relies upon to choose actions as a function of states (Sutton and Barto, 2017, p. 73). Ultimately, the sole goal of the agent is to maximize the reward that it receives over time (Sutton and Barto, 2017, p. 73).

Returns are modeled as functions of future rewards that an agents must maximize (Sutton and Barto, 2017, p. 73). There exist two types of return functions which depend on the nature of the tasks and a discounting preference (Sutton and Barto, 2017, p. 73): **I.** For episodic tasks a non discontinued approach is preferred **II.** For continuous tasks a discounted approach is better suited.

Equation 5.9 defines the sum of the rewards received over time step t (Sutton and Barto, 2017, p. 73):

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots R_T \tag{5.9}$$

If we account for discounting, Equation 5.9 has to be slightly adapted by introducing a discounting factor γ and can be found in Equation 5.10:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{5.10}$$

where $0 \leq \gamma \leq 1$ (Sutton and Barto, 2017, p. 73).

An environment with which one agent interacts, can satisfy a Markov property if the information contained at present summarizes the past without affecting the capability of effectively predicting the future (Sutton and Barto, 2017, p. 73). If the Markov property is satisfied, then this environment is called a Markov Decision Process (MDP) (Sutton and Barto, 2017, p. 73).

[2]Choices made by the agent.
[3]Basis for making decisions.
[4]Basis for evaluating the choices.

Last but not least, value functions are used to assign each state or state-action pair an expected return based on the policy used by the agent (Sutton and Barto, 2017, p. 74). Optimal value functions assign the highest achievable return by any policy to a state or state-action pair and such policies, whose values are optimal, are called optimal policies (Sutton and Barto, 2017, p. 74).

Optimal State Value (SV) functions v_* are formally defined as follows (Sutton and Barto, 2017, p. 74):

$$v_*(s) \doteq \max_{\pi} v_{\pi}(s) \tag{5.11}$$

whereas optimal Action Value (AV) functions q_* are formally defined as follows (Sutton and Barto, 2017, p. 74):

$$q_*(s,a) \doteq \max_{\pi} q_{\pi}(s,a) \tag{5.12}$$

5.2.3 Dynamic Programming

Dynamic Programming (DP) is a set of ideas and algorithms that can be used to solve MDPs (Sutton and Barto, 2017, p. 95). There are two approaches in DP for solving MDPs (Sutton and Barto, 2017, p. 95): **I.** Policy evaluations is the iterative computation of value functions of a given policy **II.** Policy improvement is the idea of computing an improved policy under the conditions of its given value functions.

By combining these two approaches we obtain the two most notable DP methods *i.e.*, policy iteration and value iteration (Sutton and Barto, 2017, p. 95).

One captivating property of DP methods is the concept of bootstrapping: updating estimates of values of states by approximating the values of future states (Sutton and Barto, 2017, p. 96).

5.2.4 Monte Carlo Methods

Monte Carlo (MC) methods use experience in form of sample episodes in order to learn value functions and optimal policies (Sutton and Barto, 2017, p. 123). This approach yields different advantages over the DP methods seen in Subsection 5.2.3 (Sutton and Barto, 2017, p. 123): **I.** They do not need a model of the environment's dynamics as they learn the optimal solutions by merely interacting with the environment **II.** Since they learn from sample episodes, they are very well suited for simulation environments **III.** It is efficient and surprisingly easy to use MC methods to focus on smaller regions or subsets of a problem **IV.** MC methods are more robust when it comes to violations of the Markov property since they do not bootstrap for updating their values.

One of the drawbacks that MC methods bring along is the concept of maintaining sufficient exploration: by always acting greedily, alternative states will never yield their returns thus potentially never learning that they might prove to be better (Sutton and Barto, 2017, p. 123).

A MC simplified method can be formally defined as follows:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{5.13}$$

where G_t is the discounted return function defined by Equation 5.10 and α is a constant step-size parameter (Sutton and Barto, 2017, p. 127). MC methods must wait until the end of one episode in order to evaluate the incremental value of $V(S_t)$ since only at that point in time G_t is known (Sutton and Barto, 2017, p. 128).

5.2.5 Temporal Difference Learning

Temporal Difference (TD) are yet another set of learning methods for RL (Sutton and Barto, 2017). Compared to the MC methods explained in Subsection 5.2.4, TD methods do not need to wait all the way up to the end of an episode to actually learn, they only must wait until the next step *i.e.*, they can bootstrap (Sutton and Barto, 2017, p. 128). When they reach time step $t+1$, they observe a reward R_{t+1} which then use to estimate $V(S_{t+1})$ (Sutton and Barto, 2017, p. 128). The simplest TD method is defined as follows:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{5.14}$$

TD methods, same as MC methods, are not exempt from sufficient exploration (Sutton and Barto, 2017, p. 147). TD methods deal with this complication in two different ways (Sutton and Barto, 2017, p. 128): **I.** On Policy (ONP) learning by using an algorithm called State-Action-Reward-State-Action (SARSA) **II.** Off Policy (OP) learning by using an algorithm called Q-learning.

5.2.6 On Policy Prediction with Approximation

Up until now, the different methods presented are not suited for arbitrarily large state spaces (Sutton and Barto, 2017). There exist solutions to tackle such large state spaces: approximate solution methods (Sutton and Barto, 2017). Under the assumption that one must always account for finite and limited computational resources, it is not feasible to find an optimal policy or value function, instead we have to settle for a good approximation of the solution (Sutton and Barto, 2017, p. 189).

An essential characteristic for RL algorithms venturing in the area of approximation is being able of generalization *i.e.*, using experience from a limited subset of the state space to effectively generalize and produce a valid approximation of a much larger subset (Sutton and Barto, 2017, p. 189). RL methods are capable of achieving this by relying on supervised-learning function approximation which essentially uses backups as training examples (Sutton and Barto, 2017, p. 222). Specifically, one brilliant set of methods are those using parametrized function approximation *i.e.*, the policy is parametrized by a weight vector θ (Sutton and Barto, 2017).

The parametrized functional form with weight vector θ can be used to write $\hat{v}(s,\theta) \approx v_\pi(s)$, which is the approximated value of state s given weight vector θ (Sutton and Barto, 2017, p. 191).

It is then clear that the weight vector θ has to be chosen wisely: this can be done by using variations of Stochastic Gradient Descent (SGD) methods (Sutton and Barto, 2017, p. 223). SGD methods adjust the weight vector after each step by a tiny amount following the direction that would reduce the error the most:

$$\begin{aligned} \theta_{t+1} &\doteq \theta_t - \frac{1}{2}\alpha\nabla[v_\pi(S_t) - \hat{v}(S_t,\theta_t)]^2 \\ &= \theta_t + \alpha[v_\pi(S_t) - \hat{v}(S_t,\theta_t)]\nabla\hat{v}(S_t,\theta_t) \end{aligned} \tag{5.15}$$

where α is a positive step size parameter and $\nabla f(\theta)$:

$$\nabla f(\theta) \doteq \left(\frac{\partial f(\theta)}{\partial\theta_1}, \frac{\partial f(\theta)}{\partial\theta_2}, \ldots, \frac{\partial f(\theta)}{\partial\theta_n}\right)^\top \tag{5.16}$$

is the vector of partial derivatives with respect to θ (Sutton and Barto, 2017, p. 195).

An exceptional case is linear methods for function approximation, where the approximate function $\hat{v}(\cdot,\theta)$ is a linear function of the weight vector θ (Sutton and Barto, 2017, p. 198). This means that for each state s there is a corresponding vector of features

$$\phi(s) \doteq (\phi_1(s), \phi_2(s), \ldots, \phi_n(s))^\top \tag{5.17}$$

which has the same number of components as θ (Sutton and Barto, 2017, p. 198). With this definition in mind, we can now formally define the State VFA (SVFA) as the inner product between θ and $\phi(s)$ (Sutton and Barto, 2017, p. 198):

$$\hat{v}(s, \theta) \doteq \theta^\top \phi(s) \doteq \sum_{i=1}^{n} \theta_i \phi_i(s) \tag{5.18}$$

This simplified case of linear function approximation for SV functions finally brings us to how we can use the SGD:

$$\nabla \hat{v}(s, \theta) = \phi(s) \tag{5.19}$$

Equation 5.19 tells us that for the simple linear case the SGD is nothing more than the corresponding features vector (Sutton and Barto, 2017, p. 199).

Artificial Neural Networks

Artificial Neural Networks (ANNs) can be used for nonlinear function approximation (Sutton and Barto, 2017, p. 199). The simplest case of an ANN is a single feedforward perceptron, meaning that it has only one hidden layer *i.e.*, a layer that is neither an input nor an output layer and that the ANN at hand has no loops between its neurons, meaning that the output cannot influence the input[5] (Sutton and Barto, 2017, p. 216).

The connections between neurons inside an ANN are called weights and the analogy to the human counterpart is how strong synaptic connections are (Sutton and Barto, 2017, p. 216). Refer to Figure 5.1 for a depiction of a single layer ANN.

The key about solving nonlinearity with ANNs is how they apply nonlinear functions to the sum of their weights and this process is done by means of activation functions (Sutton and Barto, 2017, p. 216). There are different types of activation functions that can be used but each one of them usually exhibits an S shape *i.e.*, sigmoid, such as the sigmoid $\sigma(x) = \frac{1}{(1+e^{-x})}$, the $\tanh(x) = 2\sigma(2x) - 1$ or the different classes of Rectified Linear Units (ReLUs) such as $f(x) = \max(0, x)$, which have become captivating in the last few years due to their peculiar characteristics (Sutton and Barto, 2017, p. 216).

Even though single layer perceptrons are powerful enough to approximate nonlinearity, in the past years a development towards more complex ANNs with multiple hidden layers *i.e.*, multi-layer perceptrons, has been on the rise (Sutton and Barto, 2017, p. 217). These complex ANNs allow to solve many artificial intelligence tasks in a much more efficient way (Bengio, 2009). This area is called deep RL and it has shed light on solutions that were never though possible before (Bengio, 2009). Refer to Figure 5.2 for a depiction of a multi layer ANNs.

Despite appearing more complex, ANNs rely on a similar approach for learning *i.e.*, updating their internal parameters, or in this case the whole network's synaptic connections, based on the SGD method outlined in Subsection 5.2.6 (Sutton and Barto, 2017, p. 217). This algorithm is called backpropagation and consists of doing a forward pass in which the activation function of each neuron is computed and then a backward pass computes the partial derivatives for each synaptic connection (Sutton and Barto, 2017, p. 218).

As any other function approximation method, overfitting can be a problem for ANNs as well and it is particularly present for deep ANNs (Sutton and Barto, 2017, p. 218). There are different techniques that can be used in order to mitigate this effect, with the most prominent one being the dropout method outlined by Srivastava et al. (2014).

[5] Compared to recurrent ANNs, in which the output indeed can influence the input.

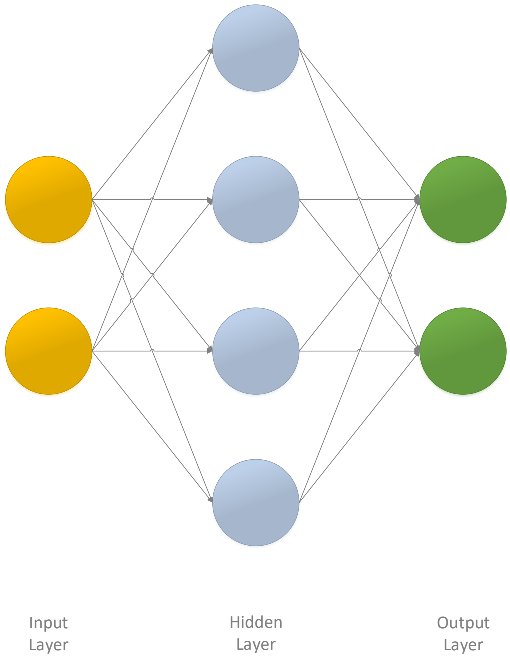

Figure 5.1: Artificial Neural Network with one hidden layer.

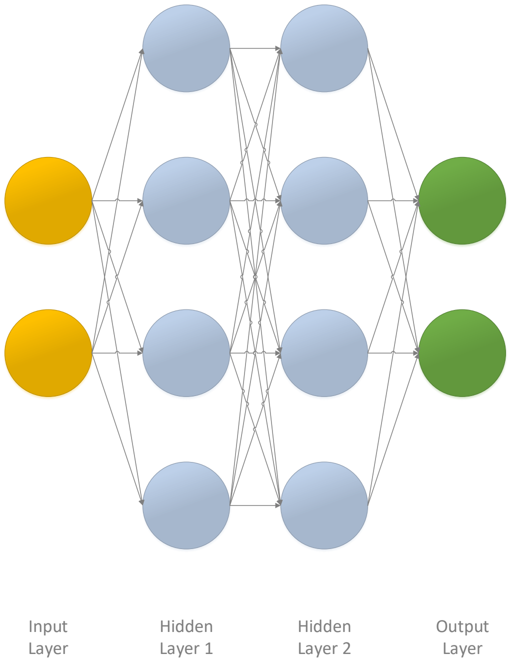

Figure 5.2: Artificial Neural Network with two hidden layers.

5.2.7 On Policy Control with Approximation

Moving towards control with Value Function Approximation (VFA), we now focus on the approximation of the AV function $\hat{q}(s, a, \theta) \approx q_*(s, a)$ (Sutton and Barto, 2017, p. 229).

For the special case of the so called one-step SARSA method, its SGD update for the AV function is defined as follows:

$$\theta_{t+1} \doteq \theta_t + \alpha[R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \theta_t) - \hat{q}(S_t, A_t, \theta_t)]\nabla\hat{q}(S_t, A_t, \theta_t) \tag{5.20}$$

and this method has excellent convergence properties towards optimality (Sutton and Barto, 2017, p. 230).

5.2.8 Off Policy Methods with Approximation

When moving towards the field of OP learning, one of the biggest problems that one might incur in is the convergence problem: OP learning with approximation is considerably harder compared to its tabular counterpart (Sutton and Barto, 2017, p. 243). OP learning defines two policies, π and μ, where the former is the value function we seek to learn based on the latter (Sutton and Barto, 2017, p. 243).

A new aspect being introduced in OP learning is the importance sampling concept, formally defined as follow:

$$\rho_t \doteq \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \tag{5.21}$$

which can be used to "warp the update distribution back to the ONP distribution, so that semi-gradient methods are guaranteed to converge." (Sutton and Barto, 2017, p. 243).

During OP learning π is defined as full greedy and μ is somewhat more exploratory ϵ-Greedy (EP) (Sutton and Barto, 2017, p. 243).

For the purpose of this thesis, the focus has been put upon the episodic AV update algorithm, defined as follows:

$$\theta_{t+1} \doteq \theta_t + \alpha \delta_t \nabla \hat{q}(S_t, A_t, \theta_t) \tag{5.22}$$

where δ_t is defined as:

$$\delta_t \doteq R_{t+1} + \gamma \overbrace{\sum_a \pi(a|S_{t+1}) \hat{q}(S_{t+1}, a, \theta_t)}^{\max_a \hat{q}(S_{t+1}, a, \theta_t)} - \hat{q}(S_t, A_t, \theta_t) \tag{5.23}$$

what is to note here, is that the episodic OP algorithm does not use importance sampling as defined by Equation 5.21 (Sutton and Barto, 2017, p. 244). Sutton and Barto (2017, p. 244) state that this approach is clear for tabular methods but it is rather a "judgment call" for methods using approximation functions and deeper understanding of the theory of function approximation is needed.

5.2.9 Policy Gradient Methods

Up until now all methods were based on the concept of learning values of actions and subsequently choosing the correct actions based on estimates, however, we now move our focus towards methods that actually learn a parametrized policy without needing value functions at all[6] (Sutton and Barto, 2017, p. 265). Parametrized policies work with probabilities that a specific action a will be chosen at time t if the agent finds itself in state s at time t with a weight vector θ (Sutton and Barto, 2017, p. 265). For PG methods it is crucial to learn the weight vector based on a performance measure $\eta(\theta)$ by trying to maximize and thus approximating the Stochastic Gradient Ascent (SGA) of η as follows:

$$\theta_{t+1} \doteq \theta_t + \alpha \widehat{\nabla \eta(\theta_t)} \tag{5.24}$$

where $\widehat{\nabla \eta(\theta_t)}$ is nothing else than a stochastic estimate that approximates the gradient of $\eta(\theta)$ (Sutton and Barto, 2017, p. 265).

For discrete action spaces, a suitable solution consists in forming parametrized numerical preferences $h(s, a, \theta) \in \mathbb{R}$ (Sutton and Barto, 2017, p. 266). This means that the best action is given the highest probability according to a softmax distribution:

$$\pi(a|s, \theta) \doteq \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}} \tag{5.25}$$

where $e \approx 2.71828$ (Sutton and Barto, 2017, p. 266). Moreover, the preferences can be, as previously mentioned:

$$h(s, a, \theta) \doteq \theta^\top \phi(s, a) \tag{5.26}$$

simply linear in features (Sutton and Barto, 2017, p. 266).

[6] Actor Critic (AC) methods are an exception, where a learned value function is used in combination with Policy Gradient (PG) as a baseline in order to lower variance (Sutton and Barto, 2017).

With these definitions in mind, one can formally define one of the very first MC based PG methods: `REINFORCE` (Williams, 1992). Williams (1992) defines his `REINFORCE` algorithm by the following update function:

$$\theta_{t+1} \doteq \theta_t + \alpha\gamma^t G_t \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \tag{5.27}$$

note that the vector $\frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$ is called eligibility vector and it is usually written in a more compact form of $\nabla \log \pi(A_t|S_t,\theta)$ by relying on the mathematical identity $\nabla \log x = \frac{\nabla x}{x}$ (Sutton and Barto, 2017, p. 271). For the `REINFORCE` algorithm, its eligibility vector is defined as follows:

$$\nabla_\theta \log \pi(a|s,\theta) = \phi(s,a) - \sum_b \pi(b|s,\theta)\phi(s,b) \tag{5.28}$$

and this method has solid convergence properties (Sutton and Barto, 2017, p. 271).

Policy Gradient with Baseline

`REINFORCE`, however, being a method based on MC it might exhibit high variance and prove relatively slow in its learning rate (Sutton and Barto, 2017, p. 271). By introducing a baseline $b(s)$ to compare the AV:

$$\theta_{t+1} \doteq \theta_t + \alpha(G_t - \overbrace{b(S_t)}^{\text{baseline}}) \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \tag{5.29}$$

one can achieve a positive effect towards diminishing variance of the update rule (Sutton and Barto, 2017, p. 271).

Actor Critic Policy Gradient

By introducing a base line, we have seen that variance can be lowered, however, the `REINFORCE` algorithm with a baseline is not a proper AC method as its SV function is only used as baseline and not as a critic *i.e.*, it is not used for bootstrapping (Sutton and Barto, 2017, p. 273). By introducing bootstrapping we introduce bias and dependence of the quality of the approximated function, which in turn help to reduce variance and learn faster (Sutton and Barto, 2017, p. 273).

The only negative aspect still remaining is that PG methods are still based on a full MC update trajectory: this can be also mitigated by replacing the update function by TD learning approaches, such as those defined in Subsection 5.2.5 (Sutton and Barto, 2017, p. 273). The formal definition of a one-step AC update method is depicted as follows:

$$\theta_{t+1} \doteq \theta_t + \alpha(R_{t+1} + \overbrace{\gamma\hat{v}(S_{t+1},w) - \hat{v}(S_t,w)}^{\text{TD update}}) \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \tag{5.30}$$

and this is now a fully online implementation that executes updates after each newly visited state (Sutton and Barto, 2017, p. 274).

5.3 Reinforcement Learning Policies

Analog to Section 5.1, the same policies are considered but now the RL methods and techniques outlined in Section 5.2 are used to solve role resolution.

Different subsections are used in order to separate better the different approaches used for each type of policy: **I.** Batch policy methods **II.** LLQP policy methods **III.** Other policy methods that do not fit in any of the previous categories.

The three key concepts required in order to effectively apply RL techniques for role resolution in WfMS are: **I.** Correctly defined states and actions spaces **II.** Precise rewards definition **III.** Effective update method for the policy's internal parameters.

5.3.1 Prediction and Control Methods

As previously outlined in Subsection 5.2.6, Subsection 5.2.7, Subsection 5.2.8 and Subsection 5.2.9 there are different prediction and control methods that can be applied.

Value Function Approximation

As mentioned in Section 5.3, it is crucial to correctly define the states and actions space for each problem. Each request that the policy receives generates a policy job which is then passed in the internal evaluate method of the policy. Inside this method, for each policy job the state space S is defined as a $m \times n+1$ matrix which contains all busy times of the potential candidates (*i.e.*, users) concatenated to the user's current service time for the job. Formally the state space is defined as depicted in Equation 5.31:

$$S_{m,n+1} = \begin{bmatrix} a_1 & \cdots & a_1 \\ a_2 & \cdots & a_2 \\ \vdots & \ddots & \vdots \\ p_{1,j} & \cdots & p_{i,j} \end{bmatrix} \tag{5.31}$$

Since the possible actions are represented by the number of users, the state space is modeled such that for each possible actions a 1-D vector containing all busy times plus the service time of the user are present.

During the evaluation phase of a job the policy has to choose an action *i.e.*, a user, by taking into account the current state space and its internal θ parameters. By using a SVFA as defined in Equation 5.18, the policy evaluates the highest score for each possible user.

As an example, let us consider a snippet of how a K-Batch policy using a linear SVFA performs its choices during its greedy phase: it iterates over all possible actions and performs the dot product between the state space and the corresponding θ vector and then maximizes the returned Q value. This approach can be seen in Listing 5.1 and its respective SVFA in Listing 5.2.

```
if self.greedy:
    action = max(range(self.number_of_users), key=lambda action: self.q(
        state_space, action))
else:
    rnd = self.EPSILON_GREEDY_RANDOM_STATE.rand()
    if rnd < self.epsilon:
        action = self.EPSILON_GREEDY_RANDOM_STATE.randint(0, self.number_of_users
            )
    else:
        action = max(range(self.number_of_users), key=lambda action: self.q(
            state_space, action))
```

Listing 5.1: ϵ-Greedy approach where if the policy is defined as greedy, actions are chosen by maximing the Q values otherwise, with probability ϵ actions are randomly sampled.

```
def q(self, states, action):
    features = self.features(states, action)
    q = np.dot(features[action], self.theta[action])
    return q
```

Listing 5.2: SVFA which performs the dot product between features ϕ and weight parameters θ.

Q values are however only one part of the requirements set by RL methods, the next crucial aspect is defining the reward function. Since RL agents are able to back-propagate what they have learned from one episode and thus update their internal factors, correctly defining a reward is a must. Since the goal for our domain is minimizing the maximum flowtime (from now on this metric will be referred to as lateness) of a job, the reward itself corresponds to the lateness of a job during a specific task. This can be evaluated a priori since for each policy job we know its internal parameters required to calculate the lateness *i.e.*, busy time of user i plus the service time of user i for job j, or formally $a_i + p_{ij}$.

The last definition required in order to effectively apply the update on the policy's internal parameters θ is defining the SGD method as outlined by Equation 5.19. This method will give us the direction in which we have to update our internal θ parameters during our chosen update method and it is nothing more than the features themselves. As an example, refer to Listing 5.3 for the concrete implementation.

```
def features(self, states, action):
    features = np.zeros((self.number_of_users, self.number_of_users + 1))
    features[action] = states[action]
    return features
```

Listing 5.3: Features definition which initializes a null matrix and fills the column corresponding to the chosen action by the policy.

The features method outputs a matrix which has its values populated only for the actual chosen action. Let us assume our policy has chosen user 1 out of two possible users, then the state space looks as defined by Equation 5.32:

$$S_{2,3} = \begin{bmatrix} a_1 & a_1 \\ a_2 & a_2 \\ p_{1,j} & p_{2,j} \end{bmatrix} \tag{5.32}$$

and its features vector looks as defined by Equation 5.33:

$$\phi_{2,3} = \begin{bmatrix} a_1 & 0 \\ a_2 & 0 \\ p_{1,j} & 0 \end{bmatrix} \tag{5.33}$$

Policy Gradient

With PG methods the approach on how an action is chosen is shifted. Instead of maximizing a Q value through internal θ parameters in order to choose the "best greedy" action, we now have probabilistic choices. As already outlined in Subsection 5.2.9, having a probabilistic policy π means that the best action is now chosen according to the highest probability which follows a softmax distribution as defined in Equation 5.25 and its implementation can be seen in Listing 5.4.

```
def policy_probabilities(self, busy_times):
    probabilities = [None] * self.number_of_users
    for action in range(self.number_of_users):
```

```
        probabilities[action] = np.exp(np.dot(self.features(busy_times, action),
            self.theta)) / sum(np.exp(np.dot(self.features(busy_times, a), self.
            theta)) for a in range(self.number_of_users))
    return probabilities
```

Listing 5.4: Softmax distribution of preferences probabilities.

The policy probabilities method takes as input parameter the current state space and computes for each user its probability according to the current internal θ parameter as defined in Equation 5.26. The result of this method is a 1-D probabilities vector corresponding to a preference to assign a job to a specific user, where the index of the vector corresponds to the user and the value to its preference. Based on this preferences vector, the policy then computes a weighted random choice among all users, as can be seen in Listing 5.5.

```
chosen_action = self.RANDOM_STATE_PROBABILITIES.choice(self.number_of_users, p=
    probabilities)
```

Listing 5.5: Probabilistic user choice by accounting for preferences weights.

Artificial Neural Networks as Function Approximation

Up to this point we have used linear functions for the approximation of the Q value for the different policies. As mentioned in Subsection 5.2.6, ANNs can be used for nonlinear function approximation. The assignment problem poses itself very well for this kind of application, in which we model our input layer as a 1-D vector containing all required information such as waiting time w of job j, service time p_{ij} of user i for job j and busy time a_i of user i. By following a PG approach, we can categorize the output layer of our ANNs using a softmax categorization function, mapping the preferences of job j to user i assignment as probabilities. Listing 5.6 show the modeling of a single layer perceptron in Tensorflow (TF): one hidden layer connects the state space *i.e.*, input to the ANNs, together with its weights and biases, creates an activation function and maps the prediction layer *i.e.*, output, with a softmax classification.

```
with tf.name_scope("neural_network"):
    layer_1 = tf.add(tf.matmul(state_space_input, weights['h1']), biases['b1'])
    layer_1 = tf.nn.elu(layer_1)
    pred = [tf.add(tf.matmul(layer_1, weights['out'][b]), biases['out'][b]) for
        b in range(batch_input)]
    probabilities = [tf.nn.softmax(pred[b]) for b in range(batch_input)]
```

Listing 5.6: Modeling of a single perceptron in TF, where the initial hidden layer is composed as the product between the input layer and its corresponding weights plus the corresponding biases by using an ELU activation function.

In order to update the ANNs, a backpropagation has to take place. Such an update can both be made following a MC or TD approach (refer to Subsection 5.3.2 for a detailed distinction between these two update methods). As outlined in Subsection 5.2.6, we follow a SGD approach in which we update all the synaptic connections by computing the partial derivatives of all the weights. Listing 5.7 shows how the backpropagation for an ANNs following a MC update method is done.

```
def train(self):
    for t, (state, output, choices) in enumerate(self.history):
        disc_rewards = self.discount_rewards(t)
        tmp_choices = [choice for choice in choices if choice is not None]
```

```
for job_index, chosen_user in enumerate(tmp_choices):
    prob_value = output[job_index].flatten()[chosen_user]
    reward = disc_rewards[job_index]
    factor = reward / prob_value
    grad_input = np.zeros((self.number_of_users, 1))
    grad_input[chosen_user] = 1.0
    self.sess.run(self.apply[job_index], {self.state_space_input: state,
        self.gradient_input: grad_input, self.factor_input: factor})
```

Listing 5.7: Backpropagation algorithm following a MC update approach. Initially rewards are discounted which are eventually used to calculate the multiplying factor for the partial derivatives.

5.3.2 Update Methods

As outlined in Subsection 5.2.4 and Subsection 5.2.5, there are mainly two different methods to update the policy's internal θ parameters *i.e.*, MC and TD. Let us take the example outlined by Sutton and Barto (2017, p. 130) of leaving the office and getting home and the respective updates proposed by the two update methods. Figure 5.3 shows the graphical updates proposed by the two update methods.

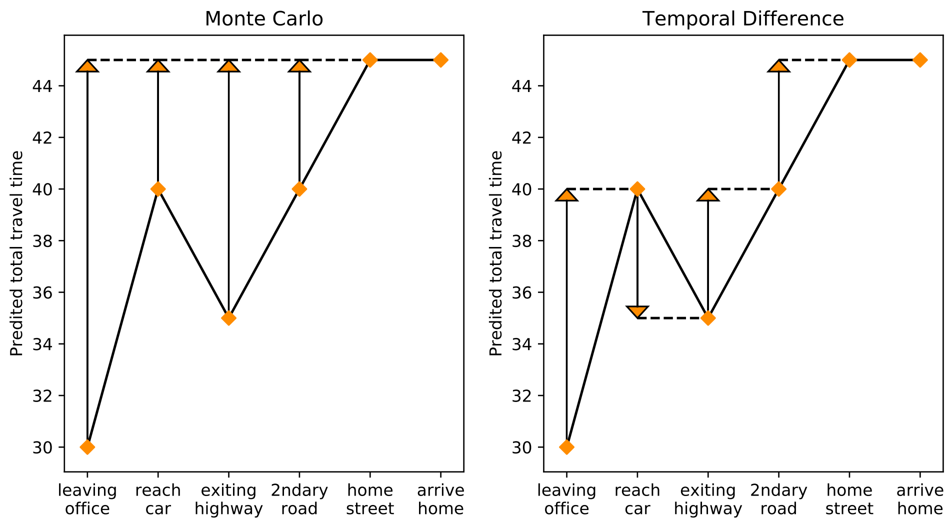

Figure 5.3: MC and TD proposed updates comparison. MC approaches have to wait until the end of an episode in order to update since they require a full episode to effectively discount rewards, compared to TD approaches that are able to update at each following time step $t+1$. Adapted from Sutton and Barto (2017, p. 130).

As it can be clearly seen, the main deference lays in when the actual updating takes place. On one hand, the MC method needs to reach the end of an entire episode *e.g.*, here it consists of actually arriving home, in order to fully back-propagate its learned value and update the θ parameters. On the other hand, TD is much more flexible and robust since it executes its updates at each time step, hence its name: TD.

For a formal overview of the difference between the two update methods, refer to Equation 5.13 for the MC update and to Equation 5.14 for the TD update.

For the case at hand, this means that training the policies has to be done in a different fashion for the two update methods: while TD based policies can be updated "on-the-fly", MC methods require batch training sessions *i.e.*, episodes, at the end of which they can effectively learn and update their internal θ parameters to be used for the next episode. Not only the training approach is different, but the logic of the policy itself is also different: for TD based policies, the update method is being called internally since the policy knows its temporal steps, while for MC based policies the policy itself cannot know a priori when an episode will finish and thus must relay on an "artificial" definition of such. The overall overhead is also different since MC based policies have to keep track of their whole episode history which usually is composed of the state space, chosen action and reward at time step t.

Sutton and Barto (2017) outline a qualitative comparison between both update methods which can be found summarized in table Table 5.3.

Characteristic	MC	TD
Bootstrap	No	Yes
Update Time	End of episode	Each time step
Discount	Required	Not required
Convergance	Good	Very Good
Learning Rate	Slow for long episodes	Very fast even for long episodes

Table 5.3: Qualitative comparison between MC and TD update methods. Adapted from (Sutton and Barto, 2017, p. 130).

5.3.3 Batch Size Emulation

Correctly defined state spaces is a crucial requirement for effective RL methods. LLQP policies are relatively easy to be modeled, on the other hand policies with batch sizes require a more meticulous consideration. By introducing a batch size that retains jobs in its global queue *i.e.*, all batch sizes $K > 1$, not only the role resolution plays a role, but the ordering of the assignment influences greatly the final outcome as well. Let us consider a simple case with number of jobs $m = 3$, number of users $n = 2$ and at time step t. Table 5.4 summarizes the service times p_{ij} in time units t of both users for all three jobs.

User	Job 1	Job 2	Job 3
User 1	1	2	3
User 2	4	5	6

Table 5.4: Sample service times of both users for all three jobs.

It is clear that the ordering of the jobs assigned has an impact on the final outcome. Figure 5.4 outlines a possible conformation where user 1 receives job 1 while user 2 gets assigned to jobs 2 and 3 respectively. In this case, job 1 is started at t and is finished at $t + 1$, job 2 is started at t and is finished at $t + 5$ and job 3 is started at $t + 5$ and is finished at $t + 11$. The respective lateness per job is: 1 for job 1, 5 for job 2 and 6 for job 3.

By changing the assignment order (refer to Figure 5.5 for a graphical representation), the final outcome changes as well: In this case, job 1 is started at $t+2$ and is finished at $t+3$, job 2 is started at t and is finished at $t+2$ and job 3 is started at t and is finished at $t+6$. The respective lateness per job is: 1 for job 1, 2 for job 2 and 6 for job 3. By merely changing the assignment order a 2.5 speedup factor in lateness is observed for job 2.

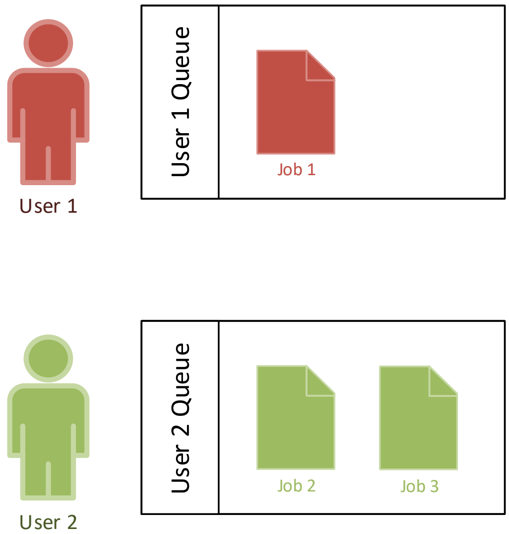

Figure 5.4: User 1 receives job 1 while user 2 receives jobs 2 and 3.

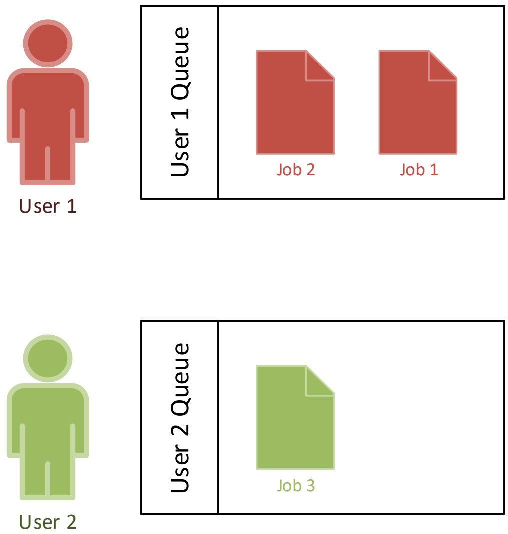

Figure 5.5: User 1 receives jobs 2 and 1 while user 2 receives job 3.

Subsection 7.2.3 outlines three different types of policies (refer to Table 7.9 for a detailed explanation of the policies) that take into account the previously outlined job assignment ordering principle and exploit it in their state space modeling. This is done by means of integrating an additional parameter B that defines how many jobs have to be considered when trying to optimally assign jobs queuing in the global queue to users. This is done by listing all possible combinations by accounting for the job order as well. Refer to Listing 5.8 for the actual implementation.

```
combinations = list(itertools.product(range(self.number_of_users), repeat=self.
    wait_size))
for i, combination in enumerate(combinations):
   state_space[i] = a + [p[user_index][job_index] for job_index, user_index in
       enumerate(combination)]
```

Listing 5.8: State space modeling by considering B jobs from the global queue and integrating all possible combinations.

This approach effectively simulates batch policies with batch sizes $K > 1$.

Chapter 6

Empirical Analysis

In order to consistently and fairly evaluate all policies with the methods defined in Chapter 5, the following methodology was put in place: **I.** Each policy has its own simulation script that initializes a process that uses the predefined policy as means to solve role resolution **II.** Parameters are centrally defined **III.** Different Key Performance Indicators (KPIs) have been defined which are used to assert the efficiency of one policy against one another.

6.1 Simulation Framework

Each policy is simulated by means of simulation scripts. It imports all required dependencies, initializes the `SimPy` discrete event simulation environment, the statistics file into which the policy job dumps all data during runtime, the policy itself to be used for role resolution and the process to be used.

In order to assert comparability, parameters are centrally defined. Listing 6.1 shows the central parameters defined as global variables.

```
NUMBER_OF_USERS = 3
SERVICE_INTERVAL = 1
GENERATION_INTERVAL = 3
SIM_TIME = 1000
BATCH_SIZE = 5
TASK_VARIABILITY = 0.2 * SERVICE_INTERVAL
WORKER_VARIABILITY = 0.2 * SERVICE_INTERVAL
SEED = 2
```

Listing 6.1: Global parameters definition which ensure comparability across simulation runs.

The script initializes the chosen process and then calls the tokens generation method of the start event. Eventually the whole simulation is started by calling `SimPy`'s run method. A snippet of a simulation script can be found in Listing 6.2.

```
import simpy
from evaluation.statistics import calculate_statistics
from evaluation.subplot_evolution import evolution
from policies.optimization.batch.k_batch import K_BATCH
from simulations import *
from solvers.dmf_solver import dmf
```

```
policy_name = "{}_BATCH_DMF_NU{}_GI{}_SIM{}".format(BATCH_SIZE,NUMBER_OF_USERS,
     GENERATION_INTERVAL, SEED, SIM_TIME)
env = simpy.Environment()
file_policy = create_files("{}.csv".format(policy_name))
policy = K_BATCH(env, NUMBER_OF_USERS, WORKER_VARIABILITY, file_policy,
    BATCH_SIZE, dmf)
start_event = acquisition_process(env,policy,1,GENERATION_INTERVAL,False,None,
    None,None)
env.process(start_event.generate_tokens())
env.run(until=SIM_TIME)
file_policy.close()
calculate_statistics(file_policy.name, outfile=True)
evolution(file_policy.name, outfile=True)
```

Listing 6.2: Structure example of simulation framework using a K-Batch policy with DMF. Initially dependencies are imported. Afterwards the discrete event simulation environment, a policy file for storing statistical data, a policy and a process returning its start event are initialized. Finally, by calling the environment's run method the simulation is started.

6.2 Business Process Modeling

Two different types of processes have been defined: **I.** A unitary process consisting of one start event, directly followed by one user task and eventually an end event (see Figure 6.1) **II.** A compound process modeled against a demonstration acquisition process used in real estate for properties acquisition (see Figure 6.2).

Figure 6.1: Unitary process consisting of one start event, one user task and one end event.

As described in Section 4.1, user tasks are the heart of each simulation. In each user task the policy has to solve the role resolution problem for each token reaching it. By focusing on a larger model, such as the one outlined in Figure 6.2 it is possible to simulate the policies in a "real-world-like" scenario. Furthermore, as explained in Subsection 4.1.1, the logical path flow for each token is manually prepared following a weighted probabilistic assignment and it gets designated to it. This approach permits to simulate only intrinsically correct paths.

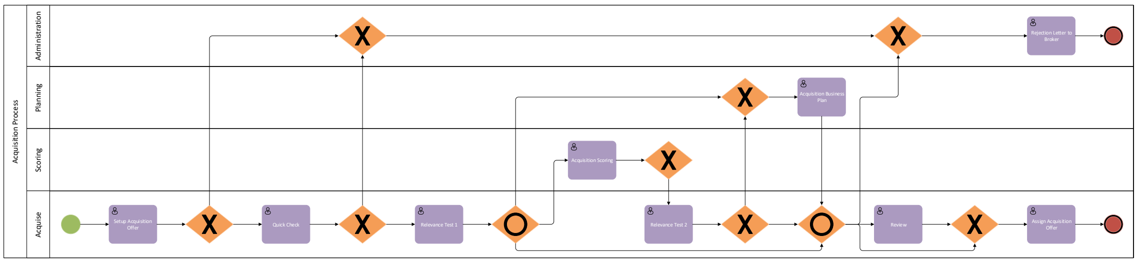

Figure 6.2: Demonstration acquisition process consisting of one start event, multiple user tasks, multiple decision gateways and two end events.

6.3 Key Performance Indicators and Data Visualization

Based on Pinedo (2008)'s and Zeng and Zhao (2005)'s definitions, different KPIs have been defined to assert the efficacy of a policy, such as lateness, waiting time, service time, number of tokens completed, user loads and system load. Following the formal definitions of the per token j KPIs in respect to lateness L_j, wait time w_j, service time p_{ij} of assigned user i to token j, arrival time A_j, assignment time a_j, start time S_j and finish time F_j.

$$L_j = F_j - A_j \tag{6.1}$$

$$w_j = S_j - A_j \tag{6.2}$$

$$p_{ij} = F_j - S_j \tag{6.3}$$

Moreover, if we account for simulation time T, load l_i of user i is defined as the sum of all service times of tokens that have been assigned to him during the simulation divided by the total simulation time T, or formally:

$$l_i = \frac{\sum_j p_{ij}}{T} \tag{6.4}$$

and thus the average system load $\bar{l}$ over all n users participating is defined as the average across all user's loads *i.e.*,

$$\bar{l} = \frac{\sum_i l_i}{n} \tag{6.5}$$

A summary plot with all KPIs is done for each simulation script. Figure 6.3 shows an example of how this summary looks like.

The left subplot in Figure 6.3 shows box plots for the three most importans KPIs *i.e.*, L_j, w_j and p_{ij}, and the right subplot shows the per user i load l_i and the average system load $\bar{l}$.

Additionally, for a more in depth visualization of a policy's performance, an evolution plot is also necessary. All types of policies share a common queues configuration, with a single global queue and a user specific queue. Each policy defines the maximal threshold a specific queue can reach. For a detailed explanation of the queues conformation refer to Chapter 5.

The evolution plot shows the state change for the policy being analyzed by plotting the flow of a token across different user tasks. Figure 6.4 shows such an example. Starting from the top, the initial box represents the state of the global queue over time t. We can see that for this specific

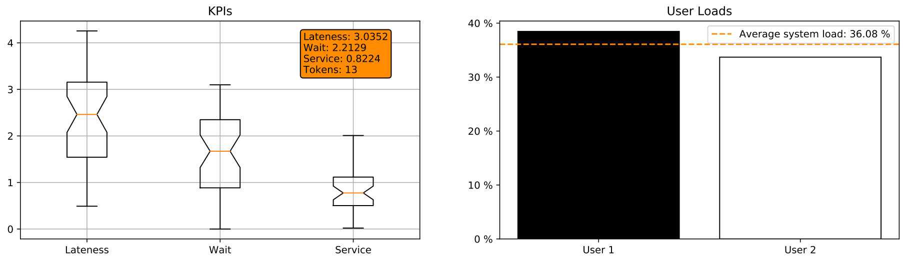

Figure 6.3: KPIs summary plot for a 3-Batch policy using MSA, two users, generation interval set to 3 and simulation time set to 50. The left subgraph displays box plots for the KPIs lateness, wait time and service time and the orange box in the upper right corner shows the mean values. The right subgraph the histograms indicate the per user load while the dashed horizontal orange line indicates the average system load.

simulation, the first token reaches the first user task called "Setup Acquisition Offer" around $t = 5$. Around $t = 10$ a second token reaches the same user task, which we can see from the fact that the size of the queue has grown up to 2. Around $t = 20$ we see a drop in the global queue size to 0. If we compare the queues sizes underneath, which correspond to user 1 respectively user 2, we note that all three tokens have been successfully assigned: user 1 received 2 tokens while user 2 only 1. The change in color helps to identify at which user task a token is being worked.

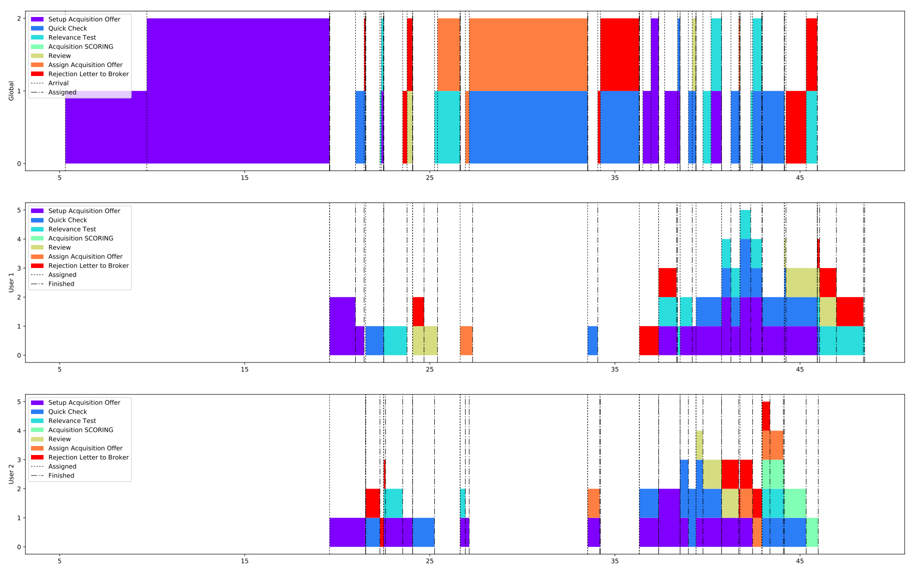

Figure 6.4: Evolution plot for a 3-Batch policy using MSA, two users, generation interval set to 3 and simulation time set to 50. Each colored box corresponds to the quanity of tokens in the global respectively user queues.

Finally, when comparing different policies between each other, Milo (2012)'s definition of speedup S between two quantities q_1 and q_2 is adopted, which he formally defines as the quotient:

$$S = \frac{q_1}{q_2} \tag{6.6}$$

Chapter 7

Results

This chapter outlines the results obtained by the Mixed Integer Linear Programming (MILP) approach in Section 7.1 respectively the Reinforcement Learning (RL) approach in Section 7.2 following the guidelines defined in Chapter 6.

All simulations have been tested with different combinations of global variables *i.e.*, number of users, service interval, generation interval, length of simulation time, batch size (where it applies), task variability, worker variability and random state seed (where it applies). For ease of reading purposes, the global variables have been set to the following parameters according to the default column in Table 7.1. The rightmost column in Table 7.1 indicates the validity range for each variable.

Variable	Default	Valid Range
Number of Users	5	$1 - \infty$
Service Interval	1	$1 - \infty$
Generation Interval	3	$1 - \infty$
Simulation Time	1000	$1 - \infty$
Batch Size	5 (1 for 1-Batch-1)	$1 - \infty$
Task Varaibility	20% of service interval	$0\% - 100\%$
Worker Variability	20% of service interval	$0\% - 100\%$
Random State Seed	2	$\emptyset - \infty$
Process	Acquisition	Acquisition, Simple

Table 7.1: Global parameters definition for simulation. Column "default" defines the standard values used for the results obtained in this chapter. Column "valid range" defines the range of values that are accepted by each parameter.

7.1 Mixed Integer Linear Programming

Zeng and Zhao (2005, pp. 18–22) outline how different global parameters configurations and policy usage can affect Key Performance Indicators (KPIs). They summarize their key findings as follows: **I.** Usage of batch optimization should be done only with medium to high system load (Zeng and Zhao, 2005, p. 24) **II.** Batch optimization policies without a fixed batch size, such as 1-Batch-1 yield, best results (Zeng and Zhao, 2005, p. 24).

In order to assert the validity of the interpretation of Zeng and Zhao (2005)'s works and all subsequent derivative policies a comparison with similar configurations has been made for all five MILP based policies. Zeng and Zhao (2005)'s main efficiency parameter is defined as the maximum flowtime or in their own words: "In business terms, maximum flowtime represents the guaranteed response time across tasks, indicating the quality of services" (Zeng and Zhao, 2005, p. 17). In this study, the comparable parameter used to evaluate a policy's efficiency is called lateness and has been previously defined in Equation 6.1. In regards to lateness, Figure 7.1 shows that akin results to Zeng and Zhao (2005) are obtained.

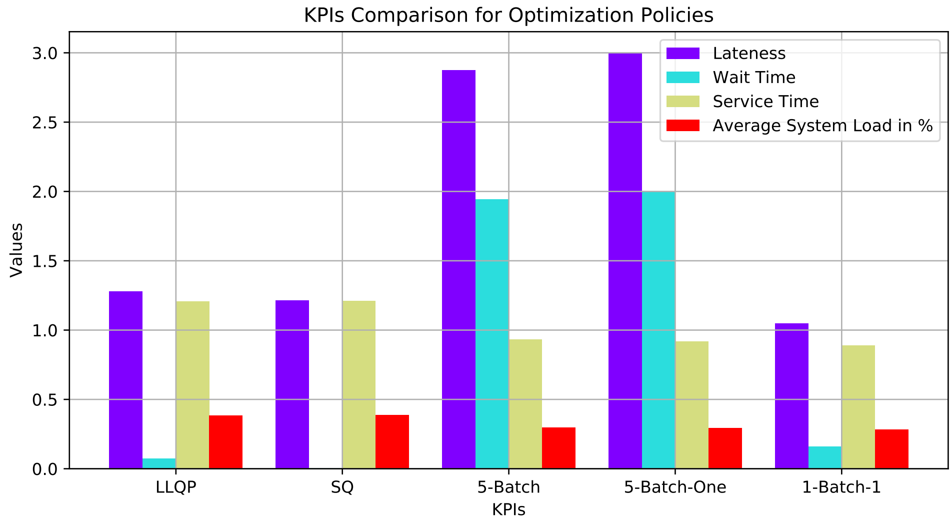

Figure 7.1: KPIs comparison for different MILP based policies using MSA. Lower is better.

The simulations have been run with the parameters outlined in Table 7.1 by using the same formulation used by Zeng and Zhao (2005): Minimizing Sequential Assignment (MSA).

By running the same simulations with the MILP based formulation implemented for this thesis *i.e.*, Service Time Minimization with ESDMF as Upper Bound (ST), ceteris paribus, the summarized KPIs among all MILP based policies can be seen in Figure 7.2.

The speedup for both batch policies with a higher batch size is tiny, but when considering the speedup between the 1-Batch-1 policy with MSA and ST, a wealthy speedup is present for all KPIs. For a detailed overview of the overall speedups of ST against MSA refer to Figure 7.3.

Astonishing speedups have been observed for the 1-Batch-1 policy[1], which is indeed the most efficient policy as mentioned by Zeng and Zhao (2005, p. 24). Table 7.2 summarizes these values.

7.2 Reinforcement Learning

This section focuses on the results obtained with the RL methods outlined in Section 5.3. A more in-depth review of the different policies is required, thus a finer subdivision has been made in different subsections per policy type: Subsection 7.2.1 focuses on batch policies, Subsection 7.2.2

[1]For a detailed comparison of how different batch sizes affect the policy's KPIs refer to Subsection B.3.2 and Subsection B.4.2.

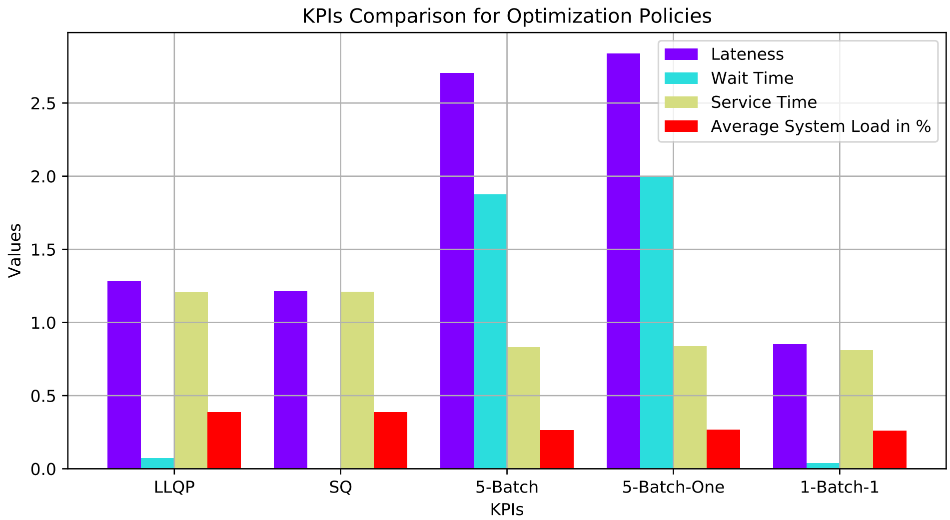

Figure 7.2: KPIs comparison for different MILP based policies using ST. Lower is better.

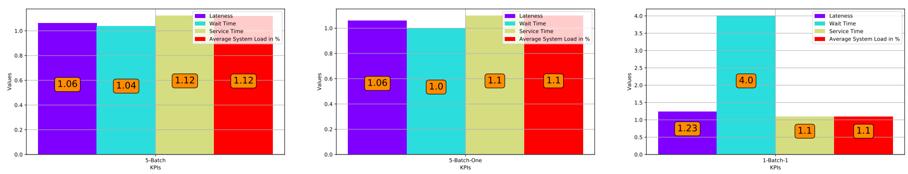

Figure 7.3: Speedup comparison between KPIs for MSA and ST for 5-Batch, 5-Batch-1 and 1-Batch-1. Values bigger than 1.00 indicate a speedup while values smaller than 1.00 indicate a speeddown. Higher is better.

KPI	Speedup
Lateness	1.23
Wait Time	4.0
Service Time	1.1
Average System Load	1.1

Table 7.2: Speedup across all KPIs for ST vs MSA. Values bigger than 1.00 indicate speedup.

focuses on Least Loaded Qualified Person (LLQP) policies and Subsection 7.2.3 focuses on all remaining policies that do not fit in either of the previous categories.

In order to maintain fairness amongst RL training methods, all required parameters are globally set and equal across all simulation scripts and can be found summarized in Table 7.3 which complement the global simulation parameters depicted in Table 7.1.

Comparisons will be made, where not otherwise stated, with the corresponding MILP based policy under the same conditions.

Parameter	Value
Discount factor γ	0.5
Step size parameter α	0.0001
ϵ-Greedy threshold ϵ	0.1
MC epochs	1000
ANNs MC epochs	10000
MC and ANNs MC epoch training time	100
TD training time	100000

Table 7.3: Global RL parameters. These values complement those outlined in Table 7.1.

7.2.1 Batch

Five different batch policies with RL have been developed. Table 7.4 gives an overview.

Technical Name	Policy Type	Update Method	Q Value Method	Other Characteristics
k_batch_mc_vfa	1-Batch	MC	VFA	None
k_batch_mc_vfa_op	1-Batch	MC	VFA	OP
k_batch_mc_vfa_opep	1-Batch	MC	VFA	ϵ-Greedy, OP
k_batch_td_vfa_op	1-Batch	TD	VFA	OP
k_batchone_td_vfa_op	1-Batch-1	TD	VFA	OP

Table 7.4: Overview of developed batch policies with RL.

Table 7.5 summarizes the results. For the detailed results refer to Subsection C.1.2 for 1-Batch respectively to Subsection C.2.2 for 1-Batch-1.

KPI	k_batch_mc_vfa	k_batch_mc_vfa_op	k_batch_mc_vfa_opep	k_batch_td_vfa_op	k_batchone_td_vfa_op
Lateness	1.22	1.24	1.22	1.23	1.23
Wait Time	2.63	3.08	2.39	3.62	3.51
Service Time	1.11	1.11	1.12	1.1	1.1
Average System Load	1.11	1.11	1.12	1.1	1.1

Table 7.5: Speedup across all KPIs for batch policies with RL vs MSA. Values bigger than 1.00 indicate speedup.

7.2.2 Least Loaded Qualified Person

Three different LLQP policies with RL have been developed. Table 7.6 gives an overview. Other policies have been implemented for evaluating different RL methods, however they are not considered for the final evaluation. These policies can be seen in Table C.1.

Table 7.7 shows the summarized results. For the detailed results refer to Subsection C.3.2.

Table 7.8 shows the comparison between Off Policy (OP) and On Policy (ONP) approaches.

Technical Name	Policy Type	Update Method	Q Value Method	Other Characteristics
llqp_mc_vfa_op	LLQP	MC	VFA	OP
llqp_td_vfa_op	LLQP	TD	VFA	OP
llqp_td_tf_op	LLQP	TD	ANNs	OP, 1L

Table 7.6: Overview of developed LLQP policies with RL.

KPI	llqp_mc_vfa_op	llqp_td_vfa_op	llqp_td_tf_op
Lateness	1.0	1.0	1.0
Wait Time	0.92	0.99	1.02
Service Time	1.01	1.0	1.0
Average System Load	1.01	1.0	1.0

Table 7.7: Speedup across all KPIs of LLQP policies with RL vs MSA. Values bigger than 1.00 indicate speedup while values smaller than 1.00 indicate speeddown.

KPI	llqp_mc_vfa_op	llqp_mc_vfa	Speedup
Lateness	1.2756	1.2914	0.99
Wait Time	0.0796	0.0711	1.12
Service Time	1.1960	1.2203	0.98
Average System Load	38.22%	39.00%	0.98

Table 7.8: Comparison between KPIs for OP and ONP approaches. Both approaches exhibit very similar results. Values bigger than 1.00 indicate speedup while values smaller than 1.00 indicate speedown.

7.2.3 Others

Three different additional policies with RL have been developed which have been used to fully emulate the behavior of K-Batch and 1-Batch-1 (as explained in Subsection 5.3.3). Table 7.9 gives an overview.

Technical Name	Policy Type	Update Method	Q Value Method	Other Characteristics
wz_td_vfa_op	WZ	TD	VFA	OP
wz_one_td_vfa_op	WZO	TD	VFA	OP
bi_one_mc_tf	BI	MC	ANNs	PG

Table 7.9: Overview of additional developed policies with RL. These policies are used to effectively emulate arbitrary batch sizes for K-Batch and K-Batch-1 policies.

Table 7.10 shows the summarized results. For the detailed results refer to Subsection C.4.2.

KPI	wz_td_vfa_op	wz_one_td_vfa_op	bi_one_mc_tf_1l	bi_one_mc_tf_2l	bi_one_mc_tf_3l	bi_one_mc_tf_4l
Lateness	0.97	1.2	1.07	0.9	0.84	0.84
Wait Time	0.89	2.33	2.46	1.11	0.85	0.86
Service Time	1.21	1.1	0.98	0.87	0.84	0.84
Average System Load	1.22	1.1	0.98	0.87	0.84	0.84

Table 7.10: Speedup across all KPIs of the additional policies with RL against the MSA formulation. Values bigger than 1.00 indicate speedup while values smaller than 1.00 indicate speeddown.

Chapter 8

Conclusion

This chapter interprets the results obtained in Chapter 7 by discussing the Mixed Integer Linear Programming (MILP) results in Section 8.1, the Reinforcement Learning (RL) results in Section 8.2, summarizing them in Section 8.3, exposing the consequences in Section 8.4 and eventually outlining future work in Section 8.5.

8.1 Mixed Integer Linear Programming Results Discussion

Using MILP to solve role resolution proves to be an efficient measure, however different formulations yield different solutions and formulation complexities. Zeng and Zhao (2005, p. 15) mention that Minimizing Sequential Assignment (MSA) greatly simplifies Dynamic Minimization of Maximum Task Flowtime (DMF) since only those tasks that are immediately available are considered. Garey and Johnson (1990) state that DMF proves to be computationally expensive to solve. Zeng and Zhao (2005, p. 13) propose however that by introducing auxiliary variables one can reduce the complexity of DMF and effectively solving it. This is what has been done in this thesis, as outlined in Section 5.1 by introducing new types of formulations. The Service Time Minimization with ESDMF as Upper Bound (ST) “flagship” formulation significantly outperforms MSA as can be seen from Figure 8.1). Having said that, the higher formulation complexity of ST compared to MSA (see Table 5.2) questions the practical use of this formulation over the other methods. A 1.23-fold speedup in respect to lateness having quadratic higher formulation complexity and requiring a double optimization pose a dubious trade-off from a business perspective.

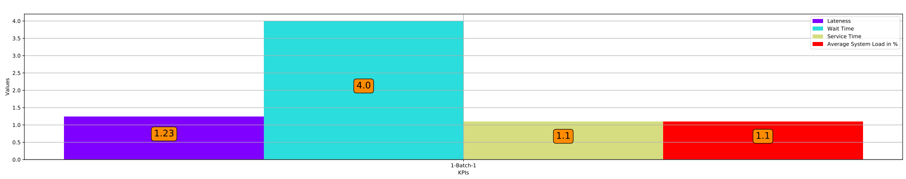

Figure 8.1: KPIs speedup comparison between MSA and ST for 1-Batch-1. Higher is better.

Yet another aspect mentioned by Zeng and Zhao (2005, pp. 17–18) is a more “social” aspect: the fairness of a policy *i.e.*, how fairly are single users treated by a policy during job assignment. Figure 8.2 and Figure 8.3 both show how fairly are users treated in the same scenario by the two formulations. ST achieves a more equally distributed user load, making it "fairer" at balancing loads across users compared to MSA.

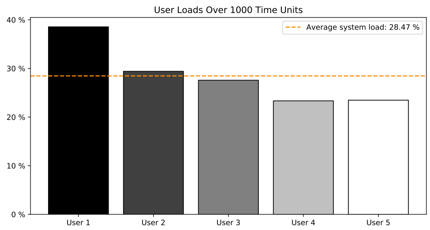

Figure 8.2: User loads distribution for 1-Batch-1 using MSA. More equally distributed histograms indicate fairer user treatement by the policy as seen in Figure 8.3.

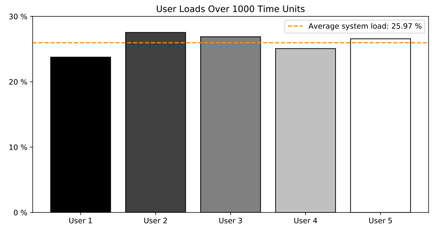

Figure 8.3: User loads distribution for 1-Batch-1 using ST. More equally distributed histograms indicate fairer user treatement by the policy as seen here.

Lastly, MILP based approaches solve role resolution in Workflow Management Systems (WfMS) in a deterministic way *i.e.*, they "blindly" optimize only at a specific point in time without accounting for the process dynamics. This kind of solution is done by the RL methods outlined in Section 8.2.

8.2 Reinforcement Learning Results Discussion

Let us first focus on Least Loaded Qualified Person (LLQP), which, as explained in Section 5.1, focuses on assigning a job to the least loaded qualified person. By comparing the results obtained with RL against the MILP based methods, we clearly see that speedups across all Key Performance Indicators (KPIs) are imperceptible (refer to Subsection C.3.2 for more details). This sheds light on two key aspects: **I.** LLQP policies are intrinsically optimized by their nature of implementation **II.** RL methods converge really well[1] and, even if only slightly, exploit internal mechanisms of LLQP policies to extract better results.

On the other hand, batch policies exhibit a much bigger optimization potential for which RL methods do really adapt well. A 1.24-fold speedup for 1-Batch (see Subsection C.1.2) respectively 1.23 for 1-Batch-1 (see Subsection C.2.2) confirm the previous claim.

Lastly, when accounting for job order during assignment (refer to Subsection 5.3.3 for the detailed explanation), improvements are observed only under specific conditions: **I.** By using Waiting Zone One for K-Batch-1 Emulation (WZO) (see Figure C.25) **II.** Artificial Neural Networks (ANNs) with One Hidden Layer for ANN (1L) (see Figure C.26 for KPIs comparison against MSA and Figure C.17 for the graphical representation of the ANN implemented with TF).

Having said that, deep ANNs exhibit worse results compared to their equivalent emulated MILP based methods. This can be explained from a twofold perspective: **I.** Vanishing Gradient Problem (VGP) which states that even very large changes in partial derivatives on initial layers have imperceptible effects on subsequent layers (Bengio et al., 1994) **II.** Exploding Gradient Problem (EGP) which states that huge spikes in the norm of changes in partial derivatives which could potentially grow exponentially can happen under training, thus influencing internal parameters (Bengio et al., 1994; Pascanu et al., 2012).

[1] For a 1000 time steps LLQP simulation, LLQP with Temporal Difference (TD) and Tensorflow (TF) only needs twenty times the simulation time as training in order to perfectly converge to actual LLQP.

Yet another crucial aspect to consider when undergoing RL methods is usage between On Policy (ONP) or Off Policy (OP) learning (refer to Subsection 5.2.6 and Subsection 5.2.7 for ONP respectively to Subsection 5.2.8 for OP). As shown in Table 7.8 (and Figure 8.4), OP methods converge slightly better compared to ONP, however the change is imperceptible and can be found in different factors *e.g.*, random state seed choice or length of simulation. Having said that, it is important to note that these different approaches are being currently heavily studied by pioneers and the debate is still open, as Sutton and Barto (2017, pp. 245–249) explain. The current key takeaways from this outgoing debate can be summarized as follows (Sutton and Barto, 2017): **I.** Learning OP **II.** Usage of scalable function approximation methods like linear semi-gradient **III.** Usage of bootstrapping which is used in TD methods, whose combination is referred to as "the deadly triad" (Sutton and Barto, 2017, p. 249). Sutton and Barto (2017, p. 249) argue that dangers can arise only when all three aspects are present, if used singularly convergence properties are safe.

On a more general note, different comparisons have shown huge spikes in speedup of waiting time *e.g.*, as seen in Subsection C.1.2. Even though compelling, the practical usage is limited: when considering WfMS with fixed available resources *i.e.*, users participating, huge speedup in job waiting times does not always correlate with better system performance, since it might be the case that even if a job is ready to be assigned there are no available resources to complete such job.

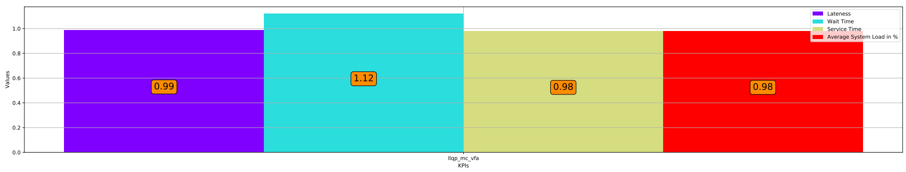

Figure 8.4: KPIs comparison between OP and ONP approaches. Values bigger than 1.00 indicate speedup while values smaller than 1.00 indicate speeddown.

8.3 Summary

The focus of this thesis was analyzing different approaches for optimal role resolution in WfMS by means of MILP and RL based methods.

Initially, a through literature review has been made in order to establish the existing optimization solution *i.e.*, foundations set by Zeng and Zhao (2005), then a roadmap on how to further develop such methods has been laid out. The roadmap encloses a twofold approach: **I.** Further develop MILP based methods **II.** Use RL methods as novel solution for role resolution in WfMS.

Subsequently, a discrete event simulation environment as outlined in Chapter 4 has been implemented which served as foundation for evaluating all future optimization policies.

For Approach I five existing optimization policies as outlined in Section 5.1 have been implemented and evaluated against the existing policies defined by Zeng and Zhao (2005, pp. 13–14). Furthermore, for Approach II a profound literature review about RL had to be made and following on the theoretical foundations laid (see Section 5.2), novel policies for solving role resolution in WfMS based on RL have been implemented (see Section 5.3). Lastly, all RL policies have been also evaluated and a comparison between them and the corresponding MILP methods has been made.

MILP based methods approach role resolution in a deterministic way *i.e.*, they optimize the assignment problem on fixed times without considering the dynamics of the system. The "flagship" formulation developed for this thesis, knows as ST, is a twofold optimization approach that outperforms traditional MILP based methods up to a 1.3-fold speedup, but exhibits higher formulation complexity.

RL methods achieve the same results in term of speedup as ST. They additionally approach role resolution in a stochastic way, overcoming the deterministic drawbacks posed by traditional MILP based methods. Nonetheless, there are aspects that have to be accounted for when solving role resolution in WfMS with RL such as susceptibility of ANNs to VGP and EGP when used as Value Function Approximation (VFA), lengthy training sessions required for convergence towards optimality and sensibility when using either ONP or OP learning.

8.4 Consequences

MILP methods are viable solutions for role resolution in WfMS, as already argued by Zeng and Zhao (2005). Further developed policies relying on MILP done for this thesis proved indeed to yield better results (see Section 7.1), however the higher complexity required (refer to Section 8.1) poses a critical trade-off. Research Question I can be affirmatively answer: there is definitely further development and optimization potential for traditional MILP based methods but further questions arise: **I.** Does a slight increase in performance justify the higher formulation complexity requirements? **II.** Can only one KPI be chosen as central parameter to evaluate a policy's performance or is a combination of different KPIs more suitable? **III.** Is it always the best solution to use only one type of MILP based method or is it better to follow a more flexible approach and chose different policies on a per case basis?

All these questions must be accounted for when considering MILP based optimization methods to solve role resolution in WfMS.

On the other hand, RL methods demonstrate to overcome some of the problematics laid by optimization very well: **I.** Generally less computationally expensive **II.** Can be adopted relatively fast **III.** Are generally more "dynamic" and can adapt and exploit case specific characteristics *i.e.*, overcome determinism of MILP based approached in favor of stochastic approaches.

Then again RL methods are not exempt from disadvantages: **I.** They require long training sessions in order to equal (or even outperform) optimization methods **II.** When using ANNs, overfitting might lead to suboptimal solution in which policies get stuck (for viable solutions refer to Srivastava et al. (2014)) **III.** Multilayer ANNs *i.e.*, deep ANNs, are very sensitive to VGP and EGP (Bengio et al., 1994; Pascanu et al., 2012).

In general RL based approaches are a refreshing and novel methodology for solving optimization problems but require further development and sound domain knowledge.

8.5 Outlook

This thesis sets the foundations for viable alternatives to existing MILP based techniques for role resolution in WfMS. A direct follow up consists in testing in operative environments the introduction feasibility and efficiency of the methods measured in the discrete event simulation environment outlined in this thesis. This includes testing the robustness, efficacy and viability of the proposed policies by putting them under "real-world" stress situation in order to assert the speedup claims. For RL methods executing lengthy training sessions might prove impractical: thousands of training sessions required by RL methods in order to comply with convergence properties can prove infeasible for companies that are not able to generate such amounts of data, thus directly limiting applicability of these methods.

RL is a novel field that is currently still being actively researched and pursued, as it has yielded promising results (Mnih et al., 2015; Silver et al., 2016). By using ANNs one can effectively approximate nonlinear functions as it has been outlined in Subsection 5.2.6. Even though in this thesis only feedforward ANNs have been used, the problem domain is very well suited for the recurrent variant as well.

Moreover, bleeding edge domains have originated from RL such as Inverse RL (IRL) (Ng and Russell, 2000) and Apprenticeship Learning (AL) which is based on the former (Abbeel and Ng, 2004). AL could prove to be yet another captivating approach to solve the assignment problem in which the reward function is not explicitly modeled, instead an "expert" of the domain *i.e.*, MILP based agents such as ST or MSA, demonstrates a task and by means of AL the policy is trained (Abbeel and Ng, 2004).

Having said that, Research Question II can be answered positively as well: there do indeed exist cutting edge approaches that can be used as alternatives or complements to role resolution in WfMS.

Appendix A

Tools Used

Different tools were used in the analysis environment in order to efficiently simulate and analyze the work of this thesis:

I. The simulation environment is based on `Python 3.5.2`[1] using the `Anaconda`[2] platform.

II. The discrete event simulation environment is implemented by using the `SimPy 3.0.10`[3] package.

III. The resulting data is plotted using `Matplotlib 2.0.0`[4].

IV. `Tensorflow (TF) 1.0`[5] is the library used for the Artificial Neural Networks (ANNs) modeling.

V. Coding was done using `PyCharm 2017.1`[6] as Integrated Development Environment (IDE) for `Python`.

VI. For solving the Mixed Integer Linear Programming (MILP) problems for batch policies `Gurobi 7.0.1`[7] was used.

[1]https://www.python.org (accessed 06.01.2017)
[2]https://www.continuum.io/anaconda-overview (accessed 03.04.2017)
[3]https://simpy.readthedocs.io/en/latest/ (accessed 06.01.2017)
[4]http://matplotlib.org/ (accessed 03.04.2017)
[5]https://www.tensorflow.org/ (accessed 03.04.2017)
[6]https://www.jetbrains.com/pycharm/ (accessed 03.04.2017)
[7]http://www.gurobi.com (accessed 06.01.2017)

Appendix B

Mixed Integer Linear Programming Results

B.1 Least Loaded Qualified Person

B.1.1 KPIs

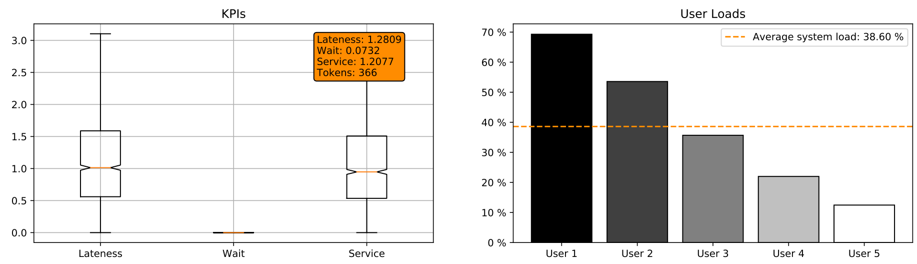

Figure B.1: LLQP KPIs.

B.2 Shared Queue

B.2.1 KPIs

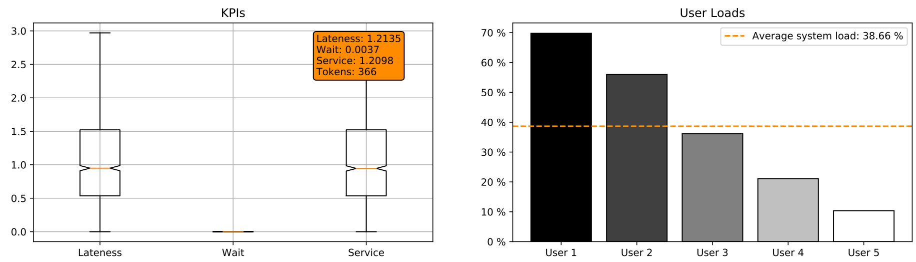

Figure B.2: SQ KPIs.

B.3 K-Batch

B.3.1 KPIs

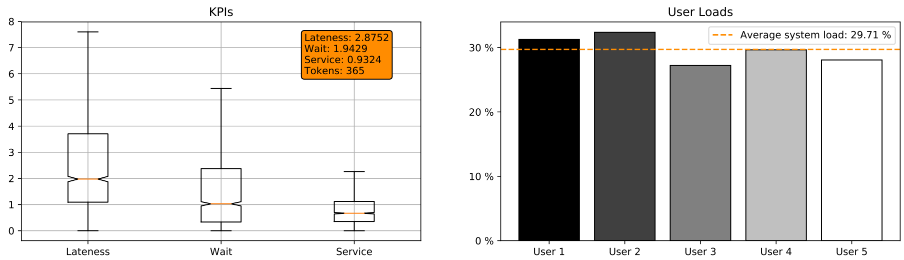

Figure B.3: K-Batch with MSA KPIs.

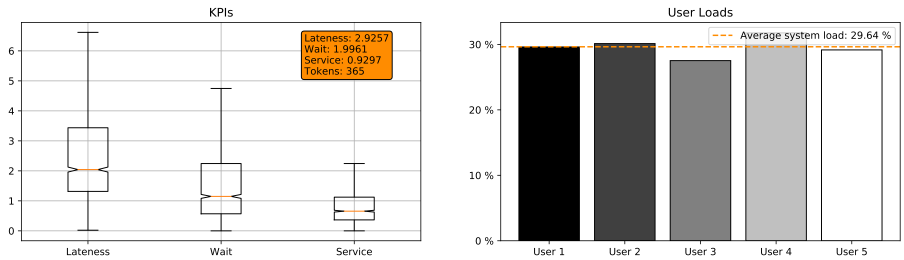

Figure B.4: K-Batch with DMF KPIs.

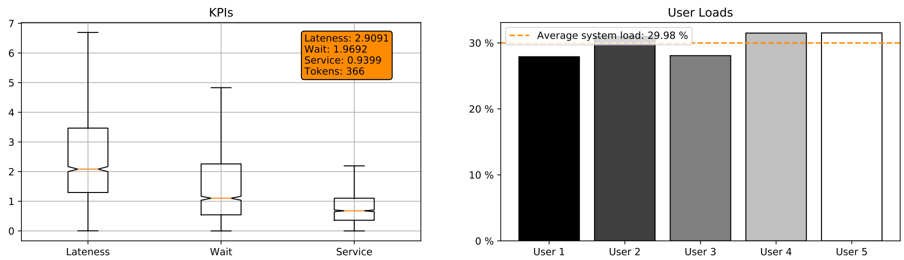

Figure B.5: K-Batch with SDMF KPIs.

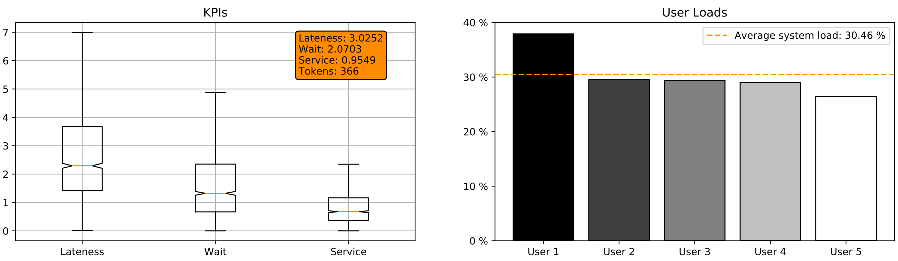

Figure B.6: K-Batch with ESDMF KPIs.

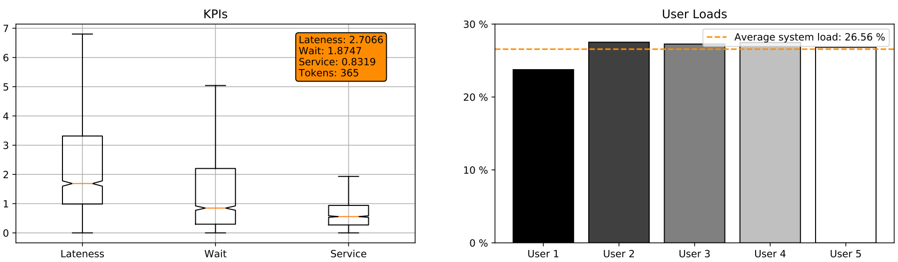

Figure B.7: K-Batch with ST KPIs.

B.3.2 Batch Sizes Comparison

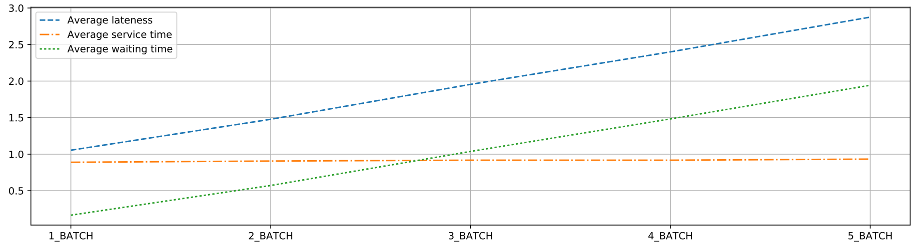

Figure B.8: K-Batch with MSA batch size comparison.

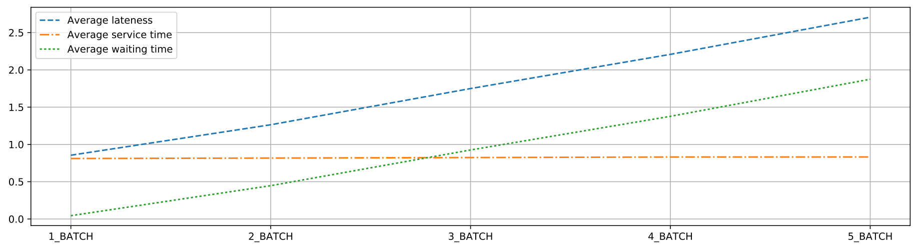

Figure B.9: K-Batch with ST batch size comparison.

B.4 K-Batch-1

B.4.1 KPIs

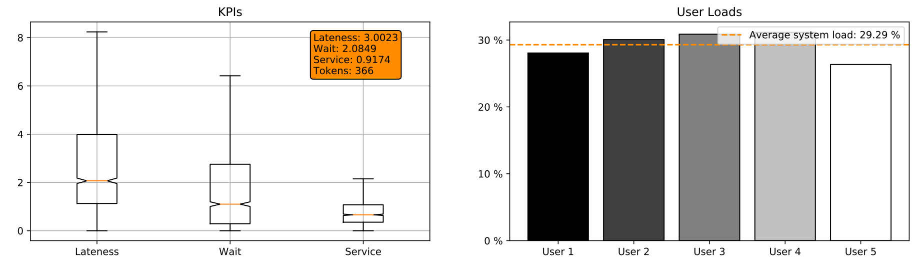

Figure B.10: K-Batch-1 with MSA KPIs.

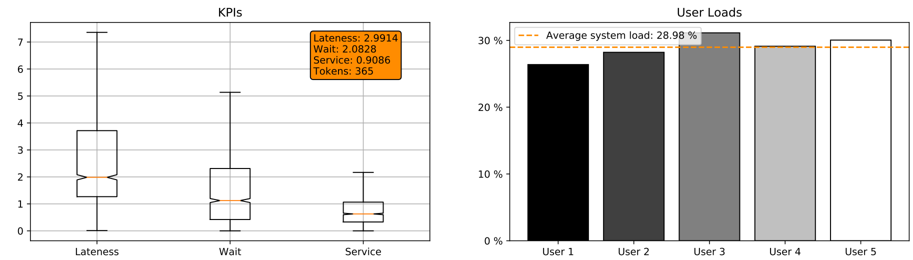

Figure B.11: K-Batch-1 with DMF KPIs.

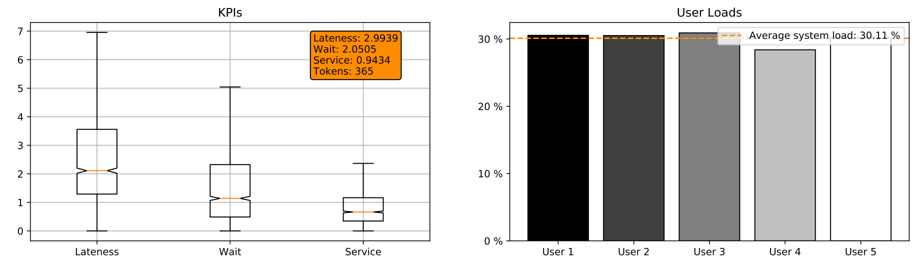

Figure B.12: K-Batch-1 with SDMF KPIs.

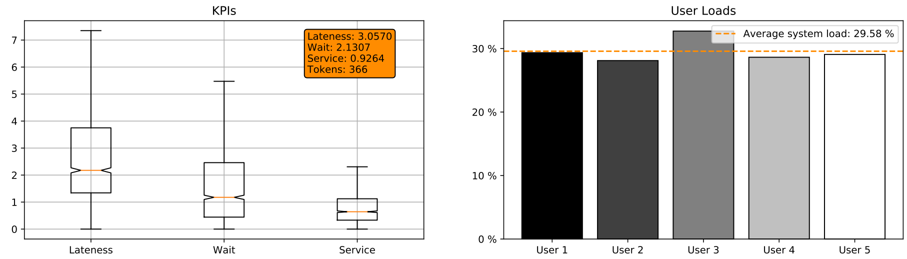

Figure B.13: K-Batch-1 with ESDMF KPIs.

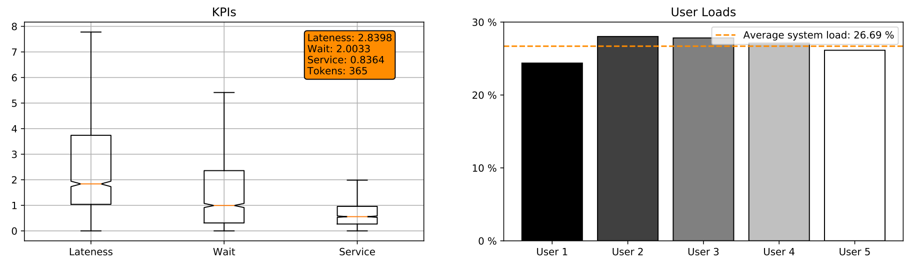

Figure B.14: K-Batch-1 with ST KPIs.

B.4.2 Batch Sizes Comparison

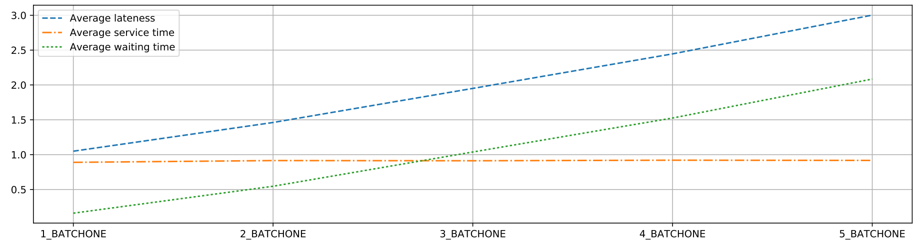

Figure B.15: K-Batch-1 with MSA batch size comparison.

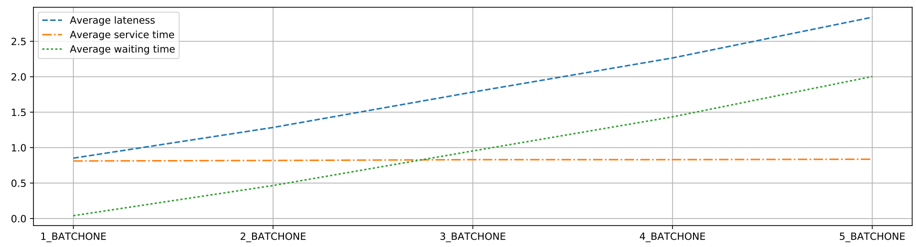

Figure B.16: K-Batch-1 with ST batch size comparison.

B.5 1-Batch-1

B.5.1 KPIs

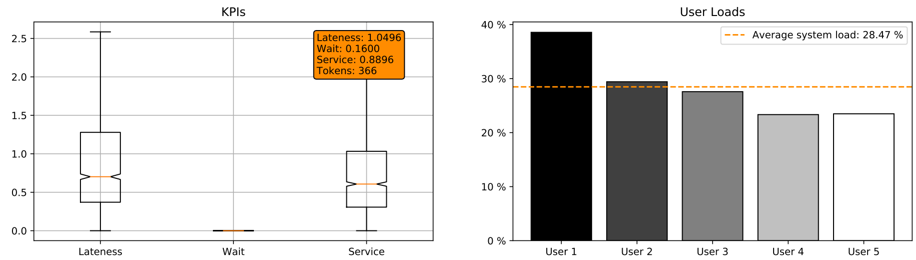

Figure B.17: 1-Batch-1 with MSA KPIs.

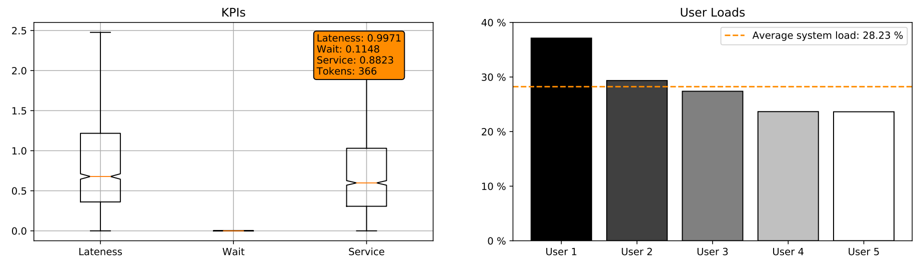

Figure B.18: 1-Batch-1 with DMF KPIs.

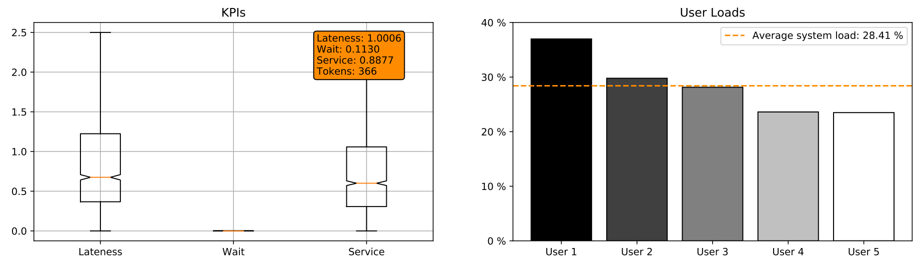

Figure B.19: 1-Batch-1 with SDMF KPIs.

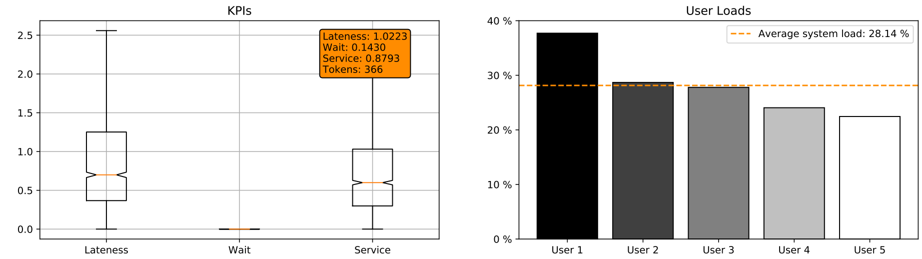

Figure B.20: 1-Batch-1 with ESDMF KPIs.

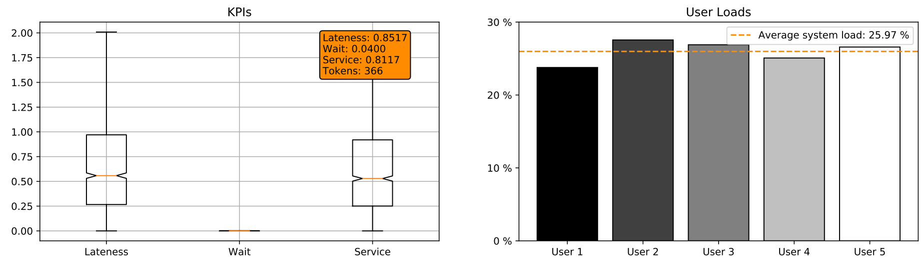

Figure B.21: 1-Batch-1 with ST KPIs.

Appendix C

Reinforcement Learning Results

C.1 1-Batch

C.1.1 KPIs

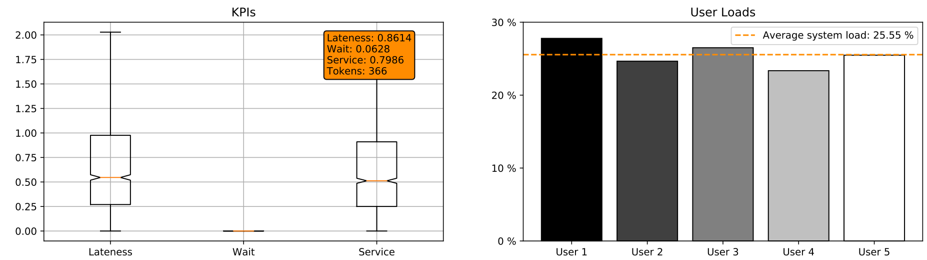

Figure C.1: 1-Batch with MC and VFA KPIs.

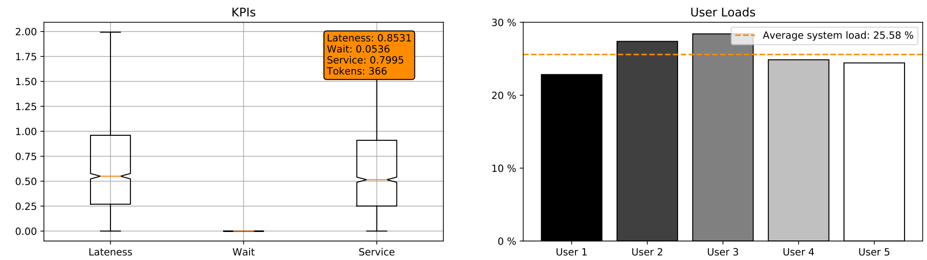

Figure C.2: 1-Batch with MC, VFA and OP KPIs.

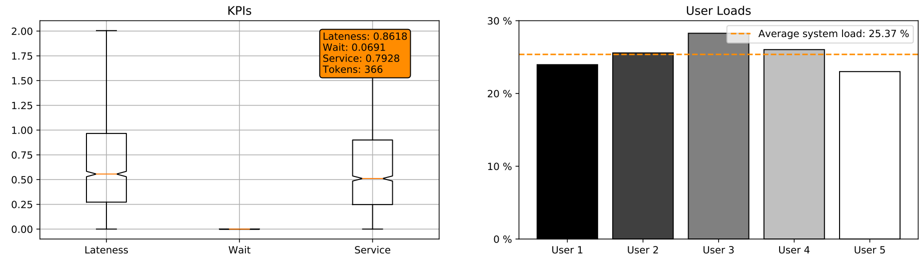

Figure C.3: 1-Batch with MC, VFA, OP and ϵ-Greedy KPIs.

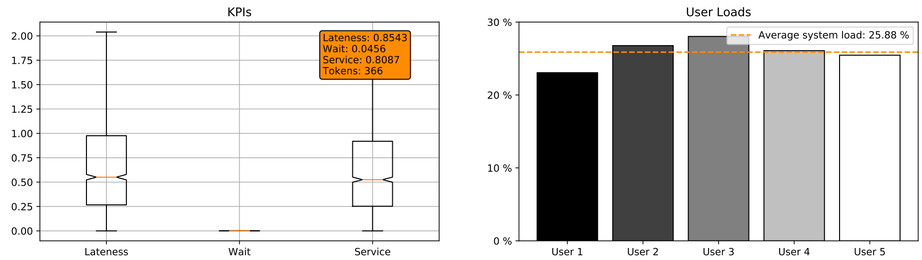

Figure C.4: 1-Batch with TD, VFA and OP KPIs.

C.1.2 Comparison with MSA

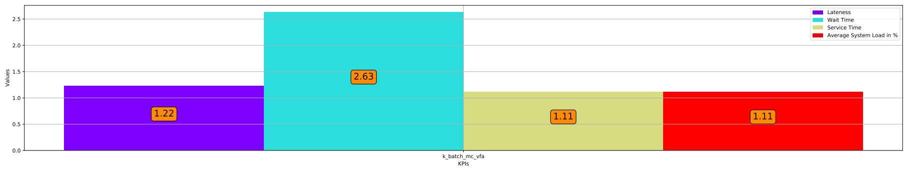

Figure C.5: 1-Batch with MC and VFA MSA comparison.

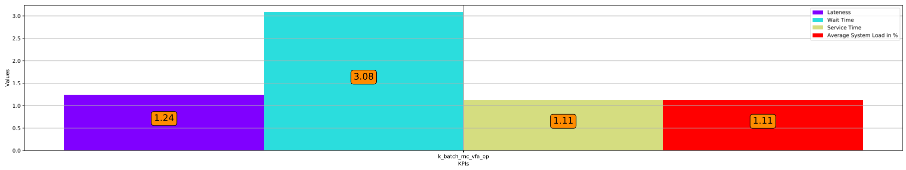

Figure C.6: 1-Batch with MC, VFA and OP MSA comparison.

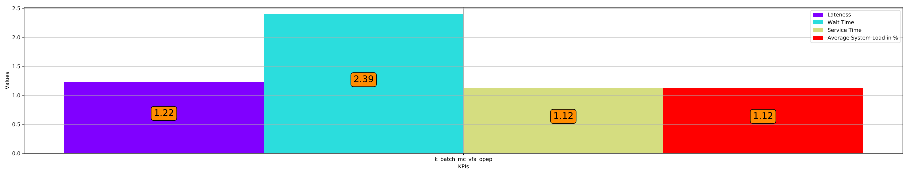

Figure C.7: 1-Batch with MC, VFA, OP and ϵ-Greedy MSA comparison.

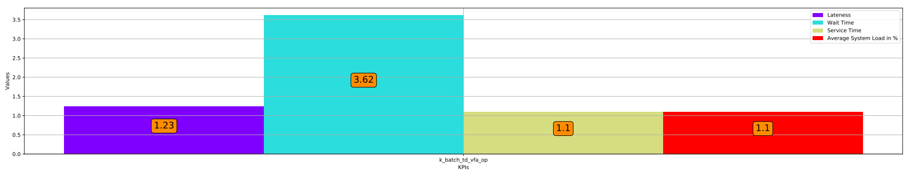

Figure C.8: 1-Batch with TD, VFA and OP MSA comparison.

C.2 1-Batch-1

C.2.1 KPIs

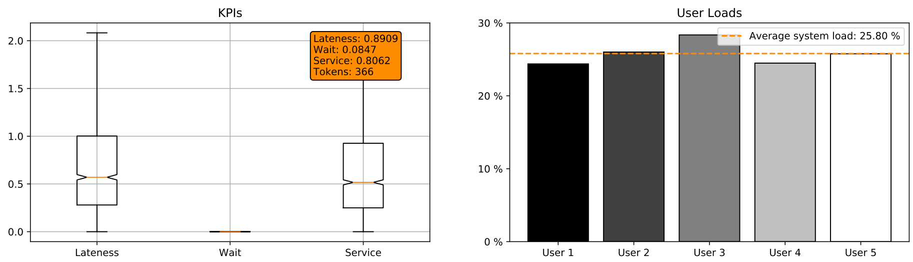

Figure C.9: 1-Batch-1 with TD, VFA and OP KPIs.

C.2.2 Comparison with MSA

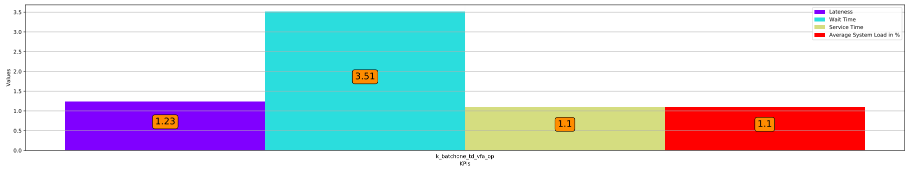

Figure C.10: 1-Batch-1 with TD, VFA and OP MSA comparison.

C.3 Least Loaded Qualified Person

C.3.1 KPIs

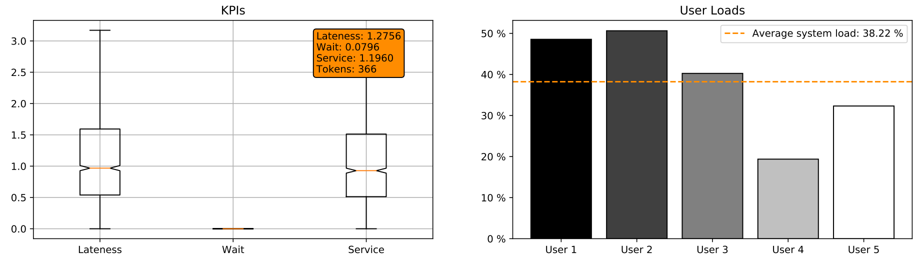

Figure C.11: LLQP with MC, VFA and OP KPIs.

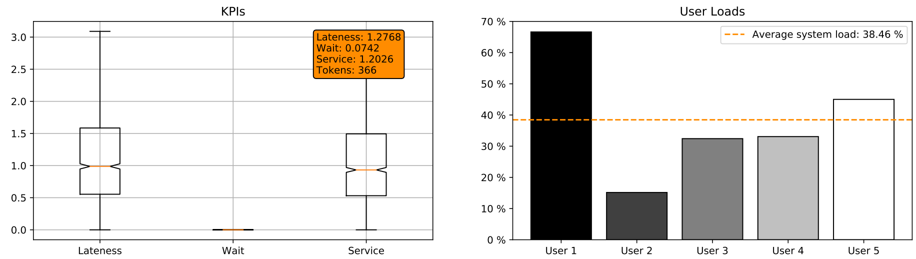

Figure C.12: LLQP with TD, VFA and OP KPIs.

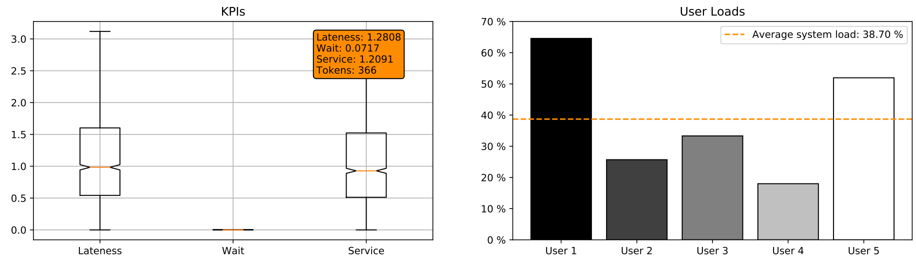

Figure C.13: LLQP with TD, TF and OP KPIs.

C.3.2 Comparison with MSA

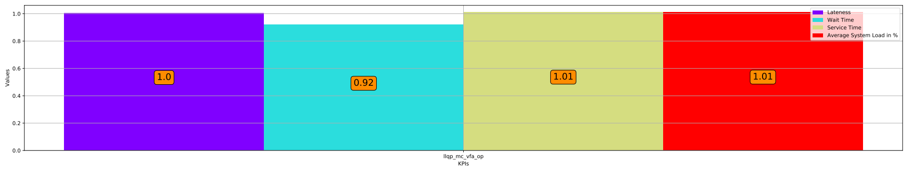

Figure C.14: LLQP with MC, VFA and OP MSA comparison.

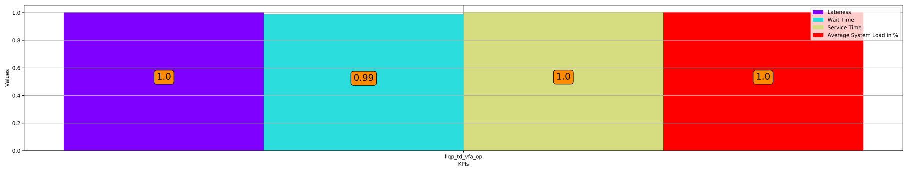

Figure C.15: LLQP with TD, VFA and OP MSA comparison.

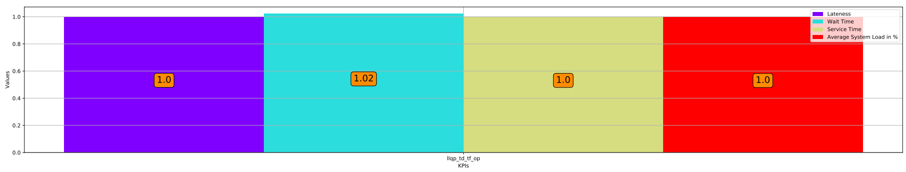

Figure C.16: LLQP with TD, TF and OP MSA comparison.

C.3.3 Additional LLQP Policies

Technical Name	Policy Type	Update Method	Q Value Method	Other Characteristics
llqp_mc_vfa	LLQP	MC	VFA	None
llqp_td_vfa	LLQP	TD	VFA	None
llqp_mc_pg	LLQP	MC	PG	None
llqp_mc_pg_wb	LLQP	MC	PG	With Baseline
llqp_td_pg_ac	LLQP	TD	PG	AC
llqp_td_pg_avac	LLQP	TD	PG	AV, AC

Table C.1: Overview of additional LLQP policies with RL.

C.3.4 ANN Implementation in TF for LLQP

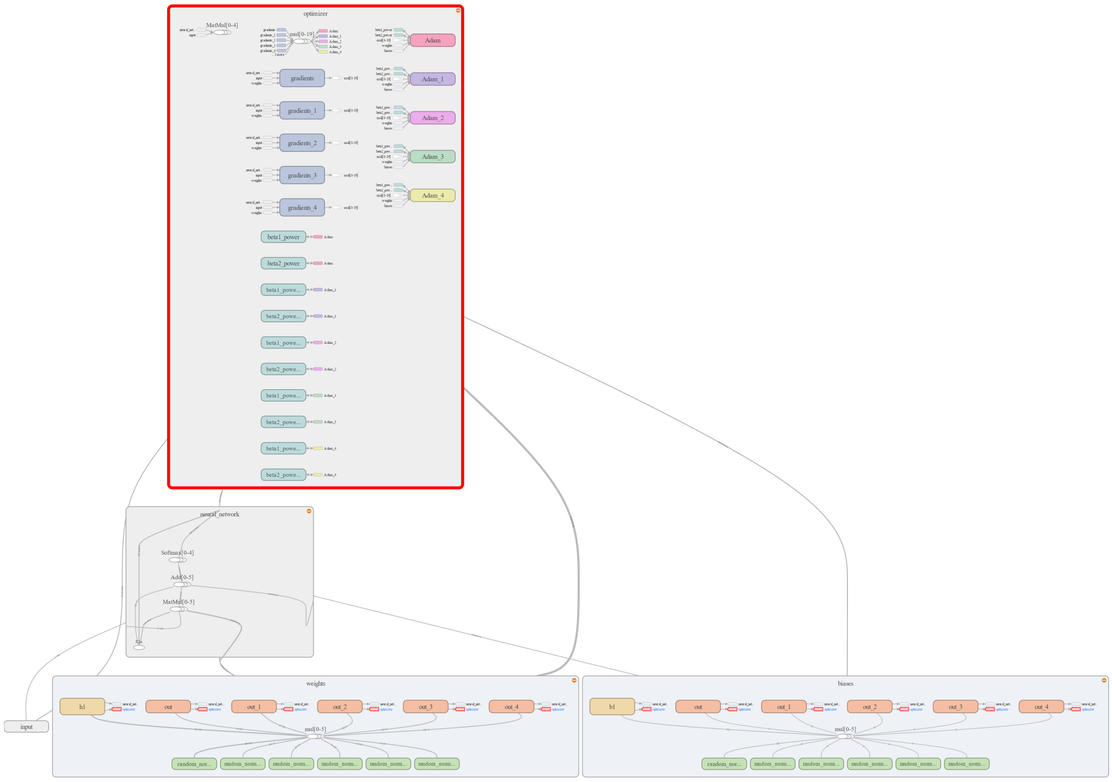

Figure C.17: 1L implementation in TF for LLQP policy.

C.4 Others

C.4.1 KPIs

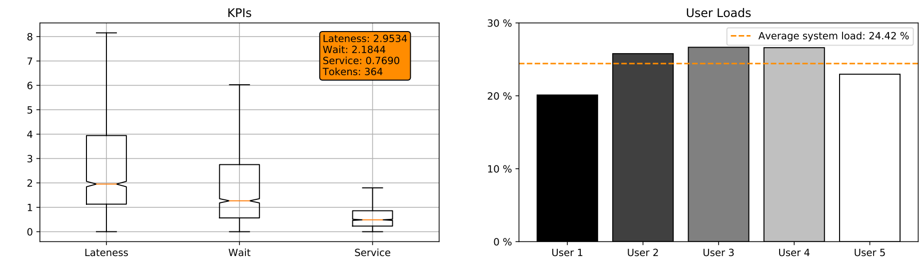

Figure C.18: WZ with TD, VFA and OP KPIs.

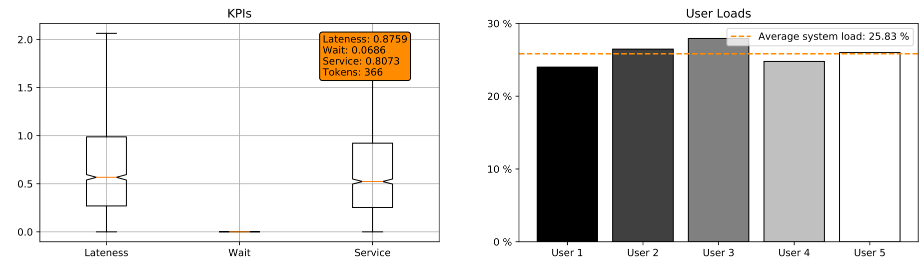

Figure C.19: WZO with TD, VFA and OP KPIs.

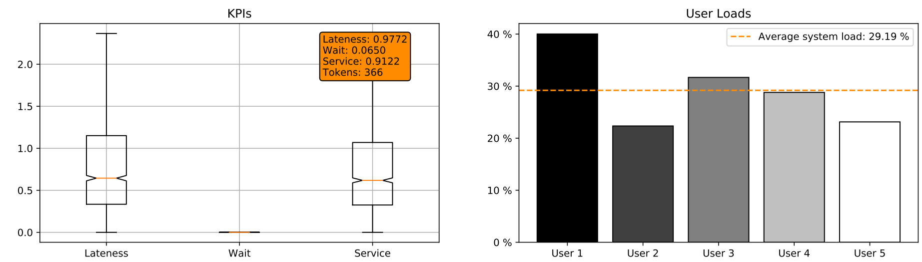

Figure C.20: BI with MC, TF and 1L KPIs.

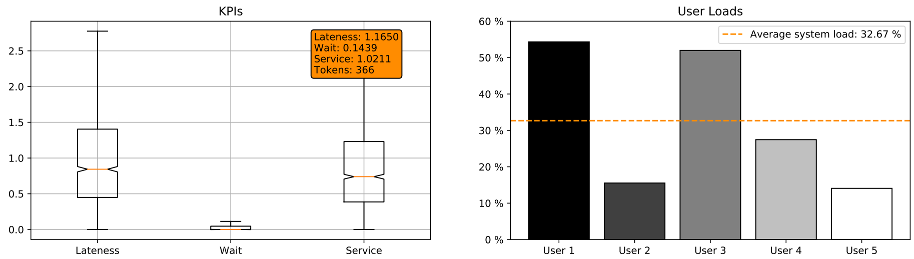

Figure C.21: BI with MC, TF and 2L KPIs.

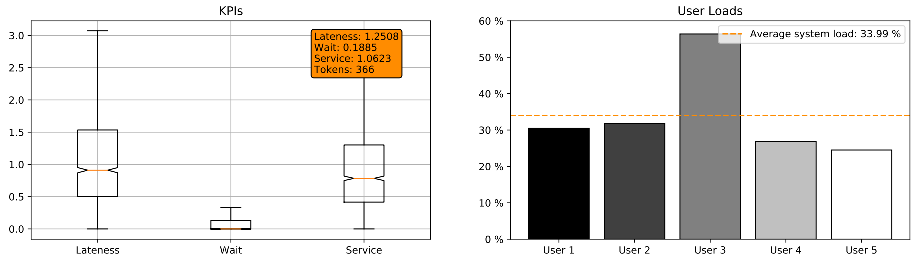

Figure C.22: BI with MC, TF and 3L KPIs.

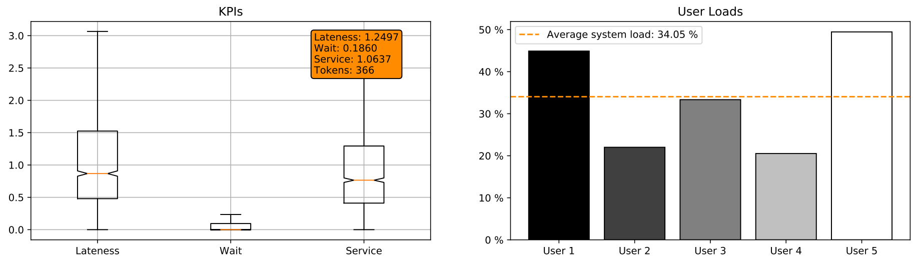

Figure C.23: BI with MC, TF and 4L KPIs.

C.4.2 Comparison with MSA

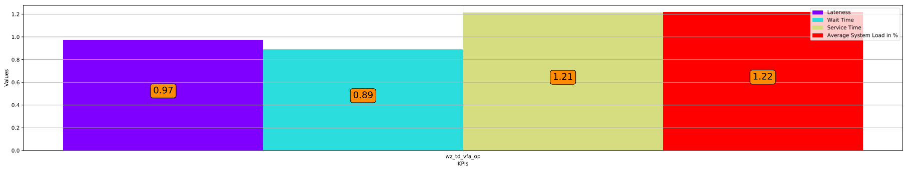

Figure C.24: WZ with TD, VFA and OP MSA comparison.

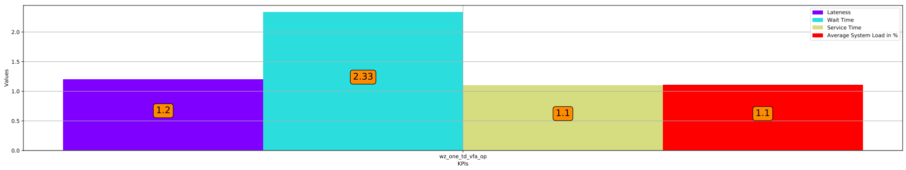

Figure C.25: WZO with TD, VFA and OP MSA comparison.

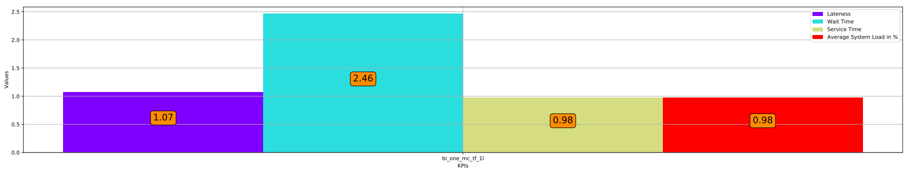

Figure C.26: BI with MC, TF and 1L MSA comparison.

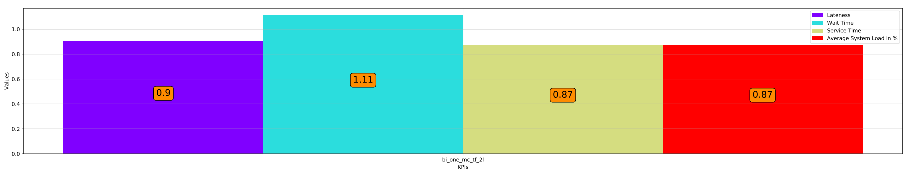

Figure C.27: BI with MC, TF and 2L MSA comparison.

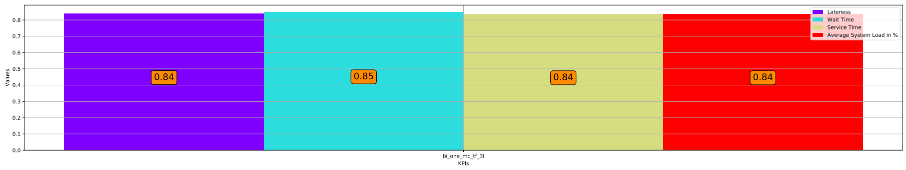

Figure C.28: BI with MC, TF and 3L MSA comparison.

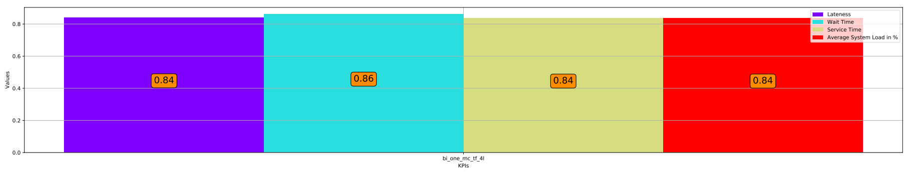

Figure C.29: BI with MC, TF and 4L MSA comparison.

Appendix D

CD Content

- A `Zusfsg.txt` file which contains a German summary of the thesis
- An `Abstract.txt` file which contains an English summary of the thesis
- A `Masterarbeit.pdf` file which corresponds to this report
- A `Code.zip` file which contains the whole source code of this thesis

Bibliography

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 1–, New York, NY, USA. ACM.

Adan, I. and Resing, J. (2015). Queueing Systems.

Aguilar-Savén, R. S. (2004). Business process modelling: Review and framework. *International Journal of Production Economics*, 90(2):129 – 149. Production Planning and Control.

Bahouth, A., Crites, S., Matloff, N., and Williamson, T. (2007). Revisiting the issue of performance enhancement of discrete event simulation software. In *Simulation Symposium, 2007. ANSS '07. 40th Annual*, pages 114–122.

Baker, K. R. (1974). *Introduction to Sequencing and Scheduling*. John Wiley & Sons.

Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, 2(1):1–127.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166.

Bisschop, J. (2016). AIMMS Optimization Modeling.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Cattrysse, D. G. and Wassenhove, L. N. V. (1992). A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60(3):260 – 272.

Cheng, E. C. (2000). An object-oriented organizational model to support dynamic role-based access control in electronic commerce. *Decision Support Systems*, 29(4):357 – 369.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *ArXiv e-prints*.

Evolved Technologist (2009). BPM Technology Taxonomy: A Guided Tour to the Application of BPM.

Fan, S., Zhao, J. L., Dou, W., and Liu, M. (2012). A framework for transformation from conceptual to logical workflow models. *Decision Support Systems*, 54(1):781 – 794.

Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

Georgakopoulos, D., Hornick, M., and Sheth, A. (1995). An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153.

Geramifard, A., Walsh, T. J., Tellex, S., Chowdhary, G., Roy, N., and How, J. P. (2013). A tutorial on linear function approximators for dynamic programming and reinforcement learning. *Found. Trends Mach. Learn.*, 6(4):375–451.

Gershman, S. J. (2016). Empirical priors for reinforcement learning models. *Journal of Mathematical Psychology*, 71:1 – 6.

Giaglis, G. M. (2001). A taxonomy of business process modeling and information systems modeling techniques. *International Journal of Flexible Manufacturing Systems*, 13(2):209–228.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition.

Kendall, D. G. (1953). Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain. *The Annals of Mathematical Statistics*, 24(3):338–354.

Korda, N. and Prashanth, L. A. (2014). On TD(0) with function approximation: Concentration bounds and a centered variant with exponential convergence. *ArXiv e-prints*.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Macintosh, A. L. (1993). The need for enriched knowledge representation for enterprise modelling. In *IEE Colloquium on AI (Artificial Intelligence) in Enterprise Modelling*, pages 3/1–3/3.

Matloff, N. (2008). Introduction to discrete-event simulation and the simpy language.

Mentzas, G., Halaris, C., and Kavadias, S. (2001). Modelling business processes with workflow systems: an evaluation of alternative approaches. *International Journal of Information Management*, 21(2):123 – 135.

Milo, M. (2012). Performance and benchmarking.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533. Letter.

Ng, A. Y. and Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Pascanu, R., Mikolov, T., and Bengio, Y. (2012). Understanding the exploding gradient problem. *CoRR*, abs/1211.5063.

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition.

Racer, M. and Amini, M. M. (1994). A robust heuristic for the generalized assignment problem. *Annals of Operations Research*, 50(1):487–503.

Reijers, H. A. and van der Aalst, W. M. (2005). The effectiveness of workflow management systems: Predictions and lessons learned. *International Journal of Information Management*, 25(5):458 – 472.

Silver, B. (2011). *BPMN Method and Style: With BPMN Implementer's Guide*. Cody-Cassidy Press.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489. Article.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. *Journal of Machine Learning Research*.

Smith, A. J. (2002). Applications of the self-organising map to reinforcement learning. *Neural Networks*, 15(8–9):1107 – 1124.

Sörensen, O. (2005). Semantics of Joins in Cyclic BPMN Workflows. Master's thesis, University of Kiel.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Sun, S. X. and Zhao, J. L. (2013). Formal workflow design analytics using data flow modeling. *Decision Support Systems*, 55(1):270 – 283.

Sun, S. X., Zhao, J. L., Nunamaker, J. F., and Sheng, O. R. L. (2006). Formulating the data-flow perspective for business process management. *Info. Sys. Research*, 17(4):374–391.

Sutton, R. S. and Barto, A. G. (2017). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 2 edition.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA. MIT Press.

Trkman, P. (2010). The critical success factors of business process management. *International Journal of Information Management*, 30(2):125 – 134.

Wang, H. J. and Zhao, J. L. (2011). Constraint-centric workflow change analytics. *Decision Support Systems*, 51(3):562 – 575.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3-4):229–256.

Zeng, D. D. and Zhao, J. L. (2005). Effective Role Resolution in Workflow Management. 17(3):374–387.

Zhang, K. and Hyvärinen, A. (2011). A general linear non-gaussian state-space model: Identifiability, identification, and applications. In *Asian Conference on Machine Learning*, volume 20, pages 113–128. JMLR: Workshop and Conference Proceedings.

Acronyms

1L One Hidden Layer for ANN. 52

AC Actor Critic. 6, 31, 32

AL Apprenticeship Learning. iii, v, 5, 55

ANN Artificial Neural Network. 5, 6, 29, 35, 52, 54, 55, 57

AV Action Value. 6, 27, 30–32

BPM Business Process Modeling. 4, 9

BPMN Business Process Model and Notation. iii, v, 1, 4, 9–11

CSF Critical Success Factor. 4

DMF Dynamic Minimization of Maximum Task Flowtime. 4, 6, 7, 22–25, 40, 51

DP Dynamic Programming. 27

EDMF Extended DMF. 7, 22, 23

EGP Exploding Gradient Problem. 5, 52, 54

ELU Exponential Linear Unit. 6

EP ϵ-Greedy. 31

ESDMF Extremely Simplified DMF. 24, 25

FIFO First In First Out. 21, 24

IDE Integrated Development Environment. 57

IRL Inverse RL. iii, v, 5, 55

KPI Key Performance Indicator. 2, 3, 39, 41, 45, 46, 52, 54

LLQP Least Loaded Qualified Person. iii, v, 4, 21, 33, 37, 47, 48, 52

MC Monte Carlo. 5, 6, 27, 28, 32, 35–37

MDP Markov Decision Process. 6, 26, 27

MILP Mixed Integer Linear Programming. iii, v, x, 1, 2, 6, 7, 21, 22, 45–47, 51–55, 57

MSA Minimizing Sequential Assignment. 21, 22, 25, 46, 51, 52, 55

OMG Object Management Group. 9

ONP On Policy. 28, 31, 48, 53, 54

OP Off Policy. 28, 30, 31, 48, 53, 54

OR Inclusive OR. 10, 11, 16, 17

PG Policy Gradient. 5, 6, 31, 32, 34, 35

ReLU Rectified Linear Unit. 6, 29

RL Reinforcement Learning. iii, v, 1, 2, 5, 6, 26, 28, 29, 32–34, 37, 45–49, 51–55

SARSA State-Action-Reward-State-Action. 28, 30

SDMF Simplified DMF. 23, 24

SGA Stochastic Gradient Ascent. 31

SGD Stochastic Gradient Descent. 28–30, 34, 35

SQ Shared Queue. iii, v, 4, 21

ST Service Time Minimization with ESDMF as Upper Bound. 24, 25, 46, 51, 52, 54, 55

SV State Value. 27, 29, 32

SVFA State VFA. 29, 33

TD Temporal Difference. 6, 28, 32, 35–37, 52, 53

TF Tensorflow. 35, 52, 57

VFA Value Function Approximation. 5, 6, 30, 54

VGP Vanishing Gradient Problem. 5, 52, 54

WfMS Workflow Management System. iii, v, 1, 3–6, 11–13, 18, 21, 33, 52–55

WZO Waiting Zone One for K-Batch-1 Emulation. 52

XML Extensible Markup Language. 9

XOR Exclusive OR. 10, 16, 17