Master Thesis

February 13, 2017

BPM

Discrete Event Simulation for Optimal Role Resolution in Workflow Processes

Filip Kočovski

of Lugano, Switzerland (10-932-994)

supervised by

Prof. Dr. Daning Hu

Dr. Markus Uhr

University of Zurich[UZH]

ifi

Master Thesis

BPM

Discrete Event Simulation for Optimal Role Resolution in Workflow Processes

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

Abstract

Zusammenfassung

Contents

List of Figures

List of Tables

List of Listings

Structure for the thesis adapted from https://wwz.unibas.ch/fileadmin/wwz/redaktion/fmgt/Images/FinanzmanagementLeitfadenfuerArbeiten.pdf

Chapter 1

Introduction

1.1 Problem Definition

Workflows are IT solutions that can help increase efficiency and get tasks done better and faster. However a key element of each workflow process still remains the human aspect. This human aspect can take many facets, such as humans analyzing a process, humans designing a process and humans executing the latter. This thesis focuses on the latter *i.e.*, where human agents interact with the workflow process in order to work on tasks. A business process that has been efficiently analyzed and subsequently optimally implemented still cannot ensure optimal execution, or no optimal execution can be achieved while a human intervention for task execution is present. It is here that optimal role resolution comes in play: optimally choosing and assigning a specific task inside the workflow process to the best possible actor is a non trivial task that has to be solved in order to close the "optimization" circle that workflow engines advertise.

This field is relevant since an optimal role resolution can bring optimization from many sides: 1. cost savings, 2. fairness in workload assignment 3. optimal resources usage.

Currently many different workflow engines exist, ranging from complete fully functional suites and down to extensible frameworks that allow the implementer to adapt it to its own needs. However all these solutions lack optimality in the task assignment sector.

1.2 Objectives

The objectives of these thesis build upon the work of Zeng and Zhao [ZZ], in which they depicted preliminary policies for optimal role resolution, and extends these capabilities from a threefold perspective: 1. further develops the mathematical premises and extends the capabilities of the batching policies proposed by Zeng and Zhao 2. explores the capabilities offered by reinforcement learning as addition and improvement for even precises, faster and better task assignment 3. deployment of the aforementioned optimization techniques in an operative environment of a real estate company using a workflow engine.

Formally, this thesis tries to answer the following research questions:

1. Are there better optimization techniques for optimal role resolution techniques inside workflow processes?
2. Is the deployment of optimization policies in a working environment for a workflow engine a critical success factor?
3. How is optimization in the field of task assignment perceived by the workflow users (actors)?

1.3 Thesis Structure

This thesis is subdivided in five main chapters:

- Chapter 1 gives an overview of why the chosen topic is relevant, what is the current context of the work and how this work fits in. It moreover articulates the central research questions that permeate this thesis and gives an overview of this essay
- Chapter 2 gives an overview of the most important conceptual definitions and the state of the art literature review in the touched thematic topics of this work. Conclusively this chapter critically reflects upon the existing literature and exposes the deficits that this thesis aims filling
- Chapter 3 gives an overview of the approach used for the research *e.g.*, the analysis environment and the used tools, states the hypothesis that wants to be proved and eventually describes statistically and qualitatively the data sets upon which the methodology is applied
- Chapter 4 builds upon Chapter 3 and makes its way into the hypothesis test field and the respective analysis results. Furthermore looks introspectively on the data correlation and gives an interpretation of the latter. Eventually in this section a statement about the contribution that the results bring into this field is given
- Chapter 5 is the culminating chapter in which a summary of the key findings of the thesis are outlined, the research questions posed in Section 1.2 are answered by looking at the actual usability, limitations and to whom the results are most applicable. Finally outlooks about the future trends and how the empirical results of this thesis can be extended by prospective researchers.

Chapter 2

Theoretical Foundations

2.1 Definitions

2.1.1 Queueing Theory

2.1.2 Workflow Processes

2.1.3 Reinforcement Learning

2.1.4 Mixed Integer Linear Optimization

2.1.5 Discrete Event Simulation

2.2 Literature Overview

This section servers as an overview of the state of the art literature that exists and has been used as a foundation basis for this work. Section 2.2 is divided in different thematic subsections.

2.2.1 Queueing

Queuing is a topic that talks about how people or more general agents are to be server while waiting.

Starting with one of the most notable contributions to this field done by Kendall in 1953 and his work on the Markov chains in queuing theory, where he formally defines different types of queues [Ken].

In 2002, Adan describes the necessary basic concepts for queuing theory and an important topic here is the statistical foundation outlined in his work about different modeling techniques for randomized generation rates, such as the Erlang's distributions [AR].

Pinedo outlines in his work in 2008 the most prominent key metrics that can be used in order to assert and measure queues performance [Pin].

Sun and Zhao in their work cover the aspect of formal analysis for workflow models and they claim that it should help "...alleviating the intellectual challenge faced by business analysts when creating workflow models" [SZ].

2.2.2 Workflow

A good starting point in the workflow thematic is Macintosh's work in which he gives an overview of the five levels of process maturity [Mac93]:

1. Initial, the process has to be set up
2. Repeatable, the process has to be repeatable
3. Defined, documentation standardization of processes
4. Managed, measurement and control of processes
5. Optimizing, continuous process improvement

Even though Georgakopoulos' work dates back to 1995, he still gives a comprehensive business oriented overview of the different workflow technologies present on the market [GHS].

On this note, Giaglis lays out four different process perspectives: 1. Functional 2. Behavioral 3. Organizational 4. Informational

His framework focuses on three dimensions: 1. Breadth, where modeling goals are typically addressed by technique 2. Depth, where modeling perspectives are covered 3. 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 [Gia].

Mentzas focuses on a qualitative level on how workflow technologies can facilitate implementation of business processes by focusing on the pros and cons of adopting alternative workflow modeling techniques [MHK]. Moreover he formally defines what a workflow management system is and subdivides it in three main categories: 1. Process modeling 2. Process re-engineering 3. Workflow implementation and automation

Each level of maturity as defined by Macintosh 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 [MHK].

Aguilar describes the main modeling techniques existing with workflow being one of them [AS].

The key core topics on which this thesis lays its foundations upon is the work done by Zeng in 2005. Effective role resolution *i.e.*, the mechanism of assigning tasks to individual workers at runtime according to the role qualification defined in the workflow model [ZZ], is the core aspect that is being extended during this thesis work.

Zeng differentiates between staffing decisions and role resolution, with the former being the assignment one or more role to each user and the latter being the assignment of a specific task to an appropriate worker at runtime [ZZ]. Staffing decisions are usually made off-line and periodically, thus being more of a strategic nature [ZZ]. 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 [ZZ].

He moreover defines three roles a workflow can fulfill: 1. System built-in policies 2. User customizable policies 3. Rule based policies

Considering capacities of resources restrictions under the assignment problem is an NP-hard computational problem and in his work Zeng focuses on how to solve the assignment problem and scheduling decisions with consideration of worker's preferences [ZZ]. For this purpose he defines five workflow resolution policies:

1. Load balanced approach (LLQP)

2. Shared queue (SQ)
3. K-Batch
4. K-Batch-1
5. 1-Batch-1

For all batching policies a simplified version of the dynamic minimization of the maximum flowtime (DMF) has to be solved [ZZ].

Zeng's key findings are outlined as follows: 1. Batching policies to be used when system load is medium to high 2. Processing time variation has major impact on system performance *i.e.*, higher variation favors optimization based policies 3. Average workload and workload variation can be significantly reduced by online optimization 4. 1-Batch-1 online optimization policy yields best results in operational conditions

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

Specifically speaking on the data flow inside workflow processes, one has to consider possible anomalies that might happens. This has been extensively studies by Sun *et al.* where they formally define data flow methodologies for detecting such anomalies [SZNS]. Their framework is divided in two components: 1. Data flow specification 2. Data flow analysis

Yet again we stumble upon mentioning that simulation for workflow management systems is usually inefficient and inaccurate [SZNS]. They moreover discuss aspects that data requirements have been analyzed but the required methodologies on discovering data flow errors have not been extensively researched [SZNS].

A more recent taxonomy of different BPM application is given by a collaboration between SAP and accenture in 2009 [Evo].

In the realm of workflow processes and engines BPMN's notation permeates the field and the work of Silver summarizes these foundations very well [SR09].

An analysis of the critical success factors (CSF) for BPM is required in order to assert a product validity and this has been done by Trkman where he defines CSF from three perspectives [Trk]: 1. Contingency theory 2. Dynamic capabilities 3. Task-technology fit theory

Change management in workflow is yet another interesting aspect that should be considered and this has been broadly studied by Wang where he developed an analytical framework for workflow change management through formal modeling of workflow constraints [WZ].

In companies different types of workflow models can exist and Fan focuses on two of these, namely: 1. Conceptual 2. Logical

Conceptual models serve as documentation for generic process requirements whereas logical models are used as definitions for technology oriented requirements [FZDL]. One difficult aspect is the transition from the former to the latter and Fan proposes a formal approach to efficiently support such transitions [FZDL].

2.2.3 Reinforcement Learning

Reinforcement learning 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.

One of the first Monte Carlo based policy gradient methods is the algorithm proposed by Williams called REINFORCE [Wil92].

Policy gradient methods with value function approximation and their convergence is of vital importance and 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 [SMS+99].

Discretization of the state action space is not always feasible and different techniques have to be used for tractability. Smith proposes such an approach which he calls "self-organizing map" [Smi02].

In their work, Schaul *et al.* have developed a modular machine learning library for python that contains different algorithm implementations such as Q-learning, SARSA and REINFORCE and yet also natural actor-critic and neural-fitted implementations such as Q-iteration, recurrent policy gradients, state-dependent exploration and reward-weighted regressions [SBW+10].

As Markov Decision Processes grow in size, so does the required computational memory to solve possible discrete lookup tables modeling the state-actions spaces that characterizes them. Notable examples that show how large some of the most common problems can be: 1. the game of backgammon has a total of 10^{20} states 2. the traditional Chinese abstract board game Go has an estimated total of 10^{170} states 3. flying a helicopter or having a robot move in space all require a continuous state space.

This very large state space requirement is a clear limitation to lookup tables. Even if memory would not be a constraint, the actual learning from such tables would be infeasible. In order to pragmatically learn by reinforcement on such huge problems, value function approximation in the domain of reinforcement learning proves to be a viable solution. For different types of reinforcement learning approaches *i.e.*, Monte-Carlo (MC) or Temporal Difference (TD) methods exist different types of value function approximation, ranging from simple linear combinations of features for MC to neural networks for TD learning. All these different methodologies are outlined in the tutorial by Geramifard [GWT+13].

When working with on-line algorithms such as TD(0) 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. This aspect is being discussed by Korda in which he depicts different non-asymptotic bounds for the temporal difference learning algorithms [KP14].

There are two main fields in reinforcement learning, one is using value function approximation for either the state value function or for using control mechanisms with the state action value function, while the other one is using policy gradient methods for policy optimization. The latter offers different methods such as the naive finite difference methods, monte carlo based policy gradient methods and finally actor critic policy gradient methods [SLH+14].

Notable works in the field of reinforcement learning and its application include Google DeepMind work on novel algorithms for tackling fields previously barely scratched, as mentioned by Mnih *et al.* and Silver *et al.* [MKS+15, SHM+16].

Sutton started working on the reinforcement learning topic in the early nineties and is now planning his third edition of his famous book on machine learning, which is due in 2017 [SB]. In our case reinforcement learning is used in order for the policies to be able to alone get better by continuously analyzing their own decision models and optimize upon them.

2.2.4 Optimization

For all batching policies implemented in this work, a mixed integer optimization was solved in order to optimally assign jobs to users in the workflow processes. The generalized assignment problem is a very well known problem in combinatorial mathematics. Cattrysse gives an overview of different algorithms for solving the generalized assignment problem [CW]. Heuristics are also a viable solution for solving such adaptation of the generalized assignment problem, as Racer states [RA]. Moreover a global perspective of optimization from a mathematical perspective is

given in Boyd's work on convex optimization [BV04].

Last but not least, according to the AIMMS guidelines, there are different linear programming tricks that can be used to shape such problems in solvable outlines [Bis16]. In this thesis, a specific linear programming trick, called either-or constraints, was used by adding so called auxiliary variables to the evaluation method presented in order to efficiently solve an otherwise non solvable equation [Bis16, p. 77].

2.2.5 Simulation

Simulating queues can prove to be extremely difficult. The main differentiation needed here is that 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. 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 [Mat08].

According to Matloff, there exist different world views for discrete event programing, as he calls them paradigms [Mat08]:

1. Activity oriented
2. Event oriented
3. 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. Since very small subdivisions of time are possible in such types of simulations, it is clear that the program might prove extremely inefficient, since most of the time there won't be any change in state for the simulated entities [Mat08]. Event oriented circumnavigate this issue by advancing the simulation time directly to the next event to be simulated. By filling these gaps, a dramatical increase in computation can be observed [Mat08]. Last but not least, the process oriented simulation models each simulation activity as a process or thread. Management of threads has steadily decreased in todays computation since many different packages for governing such tasks.

On another note, Bahouth focuses in work on algorithmic analysis of discrete event simulation supplemented with focus on factors such as compiler efficiency, code interpretation and caching memory issues [BCMW07]. According to his findings, a significant speedup can be achieved if one addresses the afore mentioned facets.

2.3 Research Deficit

Chapter 3

Methodology

3.1 Analysis Structure

3.1.1 Tools

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

The whole architecture is subdivided as follows:

1. The simulation environment is based on `Python 3.5.2`[1] and as a discrete event simulation the `SimPy`[2] package is used.

2. The resulting data are interpreted and analyzed using `R`[3].

3. The workflow engine itself is `Java`[4] based and uses the `jBPM`[5] suite.

4. `PyBrain`[6] is the library used for reinforcement learning.

5. Coding IDE used were `PyCharm 2016.3`[7] for `Python` respectively `IntelliJ IDEA 2016.3` for `Java`[8].

6. For solving the mixed integer problems for batching policies `Gurobi 7.0.1`[9] was used.

7. In order to allow a seamless integration between the optimal resolution policies implemented in `Python` and the workflow engine developed in `Java`, `Jython 2.7.0`[10] was used.

[1] https://www.python.org (accessed: 06.01.2017)
[2] https://simpy.readthedocs.io/en/latest/ (accessed: 06.01.2017)
[3] https://www.r-project.org (accessed: 06.01.2017)
[4] https://www.java.com/en/ (accessed: 06.01.2017)
[5] https://www.jbpm.org (accessed: 06.01.2017)
[6] http://pybrain.org (accessed: 04.01.2017)
[7] https://www.jetbrains.com/pycharm/ (accessed: 06.01.2017)
[8] https://www.jetbrains.com/idea/ (accessed: 06.01.2017)
[9] http://www.gurobi.com (accessed: 06.01.2017)
[10] http://www.jython.org (accessed: 06.01.2017)

3.1.2 Discrete event simulation using SimPy

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

Active components such as agents in a workflow 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)[11] them to the environment, which then wait until they are triggered. The important logic to understand here 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 a timeout event are created and called by called the appropriate method of the passed `Environment`.

3.1.3 Analysis Environment

The analysis environment consists in an object-oriented implementations of workflow process elements such as user task, starting, decision and end nodes which have been developed to allow the simulation framework to effectively run. This object-oriented exoskeleton implementation of the workflow elements can be seen depicted in Figure 3.1.

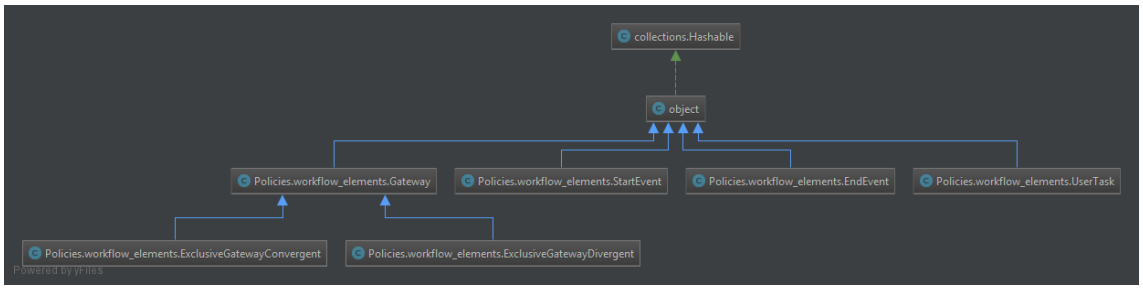

Figure 3.1: Workflow elements

An implicit object that is not part of the workflow elements is the token: a token is to be understood as an object that travels through the whole process and all its elements and gets worked by in different ways by them. In this implementation the token object is directly a policy element as it can be seen in Figure 3.2.

The core elements of a workflow process (relevant for the simulation environment) are start nodes, user tasks, decision nodes and end nodes. Start events are used to indicate where and how a process starts and usually each process has only one such event [SR09, p. 42]. No distinction between trigger types is being made.

[11] https://docs.python.org/3.5/reference/simple_stmts.html#the-yield-statement (accessed: 06.01.2017)

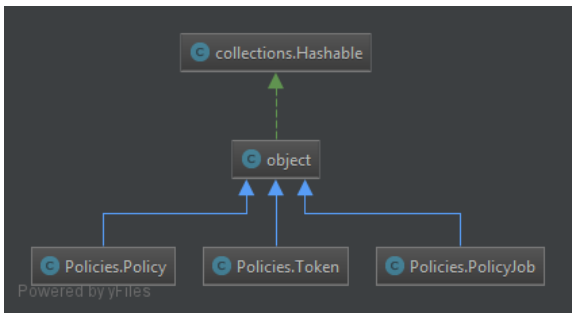

Figure 3.2: Policies initialization objects

Start event

Start event objects require a simulation environment, a policy to be used, a generation interval and the number of tokens to be generated. The simulation environment is generated with SimPy, the policy object is initialized prior to the simulation and the tokens to be generated is a plain scalar value. The generation interval is generated in a three step process: 1. before the simulation starts, a fixed service interval time unit s, number of users n and an average system load l are set. Analog to Zeng's and Zhao's work, the generation λ interval follows a Poisson distribution [ZZ] and is defined in Equation 3.1 2. for a Poisson random exponential sampling of the generation rate, `NumPy`'s implementation of its exponential distribution is used[12]. The scale parameter β, which is the inverse of the rate parameter $\lambda = 1/\beta$ is used, meaning that the generation rate λ defined in Equation 3.1 is passed to the start event inverted.

$$\lambda = \frac{ln}{s} \tag{3.1}$$

Listing 3.1 shows the token generation method for the start event. As shown, this method generates infinitely many tokens and for each token a random exponential simulation time is being drawn. Note that a random state with a fixed seed is used in order to preserve generality across multiple simulation runs. Furthermore the aforementioned timeout event of the SimPy framework can be seen in action. For each token a timeout event is being automatically triggered after its sampled arrival time has elapsed.

```
def generate_tokens(self):
    while True:
        exp_arrival = round(RANDOM_STATE.exponential(self.generation_interval)
            , 1)
        yield self.env.timeout(exp_arrival)
        token = Token()
```

Listing 3.1: Token generation method for start event object

Even though tokens are generated infinitely, this process is controlled from the simulation environment where a discrete simulation time steps have to be set, as it can be seen from Listing 3.2.

This can be interpreted as that the whole simulation will persist for 100 time steps and it will then stop when the internal clock reaches 100. Please note that events that have been scheduled for time step 100 will not be processed. The logic is similar to a new environment where the clock is zero and no event have been processed yet.

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

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

Listing 3.2: Starting the simulation with discrete time steps

User task

User task objects also require a simulation environment, a policy, a descriptive name, a knowledge level, a service interval and task variability. Each user task has a unique `child` field which is being set prior to starting the simulation by the method depicted in Listing 3.3.

```
def connect(source, destination):
if isinstance(source, ExclusiveGatewayDivergent):
   for child in destination:
      source.children.append(child)
elif isinstance(source, (StartEvent, UserTask, EndEvent,
    ExclusiveGatewayConvergent)):
   source.child = destination
```

Listing 3.3: Method used to connect workflow elements

In regards to parameters service interval and task variability a detailed explanation is required. Both are used to randomly sample service rate intervals for each user active during the simulation. Zeng and Zhao in their work follow a two way process to generate such intervals [ZZ, p. 8]. However in this thesis' implementation an refined version of this process is used: 1. at initialization time, each user task receives a service rate s and a task variability t value 2. in the class initialization method, each user task samples an average processing time following an Erlang distribution (a special case of the gamma distribution) which takes as input parameters a shape k and a scale θ. The shape value k, as the name suggests, defines the curve shape that the Erlang distribution will follow. 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 3.5 3. the average processing time becomes a unique value of each user task object and is used by each policy to sample each user's service time, again from an Erlang sampled pool as depicted in Listing 3.4 and we shall call this value p_j

Listing 3.4 gives a glimpse of the inner logic of how policies work. It is however out of scope for this section to cover this aspect and it is provided "as is". 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 value worker variability, which is a unique property of each policy, which we shall call w.

In order to sample a service rate p_{ij} following the Erlang distribution for each user i, shape k is evaluated as $k = p_j/w$ and scale as $\theta = w$ as it can be seen in Listing 3.4

```
user_service_rate = [round(
      RANDOM_STATE.gamma(kbatchone_request_job.user_task.
          average_processing_time / self.worker_variability,
                    self.worker_variability),
      1) for _ in range(self.number_of_users)]
```

Listing 3.4: User service rate sampling following an Erlang distribution

As previously mentioned, the Erlang distribution is a special case of the Gamma distribution where k defines the shape of the curve. This 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 (thus loosing statistical generality). This is asserted by the formal definition of Erlang's support with $x \in [0, \infty)$.

`NumPy`'s implementation of its Erlang distribution is used[13]. Equation 3.2 defines the probability density function of the Erlang's distribution with the alternative parametrization that uses μ instead of λ as scale parameter, which is its reciprocal. This corresponds to the NumPy's implementation.

$$f(x; k, \mu) = \frac{x^{k-1} e^{-\frac{x}{\mu}}}{\mu^k (k-1)!} \text{ for } x, \mu \geq 0 \tag{3.2}$$

Each user task object has a claim token method, which takes tokens as input parameters and finally makes a call to its designed policy, passing the token. On this top level, without stepping into the single policies implementations, the logic is straightforward: start events generate tokens, user tasks that are direct children of start events claim the newly generated tokens, ask to the designated policies to work the token assigned to them and finally, after a service interval timeout which corresponds to the user's specific service interval, they release the token. The logic can be seen in Listing 3.5.

```
class UserTask(object):
def __init__(self, env, policy, name, knowledge, service_interval,
    task_variability):
   self.env = env
   self.policy = policy
   self.name = name
   self.knowledge = knowledge
   self.child = None
   self.average_processing_time = round(RANDOM_STATE.gamma(service_interval
       / task_variability, task_variability),
                                1)

def claim_token(self, token):
   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 3.5: User task initialization

Policy

Different types of policies have been implemented following the foundations laid by Zeng and Zhao as outlined in Subsection 2.2.2. In their work the authors investigate five "role-resolution" policies used for optimal task to user assignment [ZZ, p. 7]. Following a brief description of the five aforementioned policies:

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

1. 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 FIFO fashion. The authors call this policy the "least loaded qualified person" or LLQP.

2. A policy that maintains a single queue being shared among all users is referred to the authors as "shared queue" or SQ policy.

3. Another policy that maintains both a shared queue 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 shared queue to users is done using an optimal task assignment procedure as soon as the shared queue reaches a critical batch size K.

4. The following policies 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 shared queue reaches the critical size K, however actual movement of tasks from the shared queue 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 according to the authors as "K-Batch-1"

5. The last policy further simplifies the fourth by weakening the batch size constraint and reduces it to one. This means that the optimal task assignment procedure is executed immediately. This policy is referred by the authors as "1-Batch-1".

All batching policies require the solution of an optimization problem. The authors define this problem as "minimizing the maximum flowtime given the dynamic availability of the workers" and call it "minimizing sequential assignment (MSA)" [ZZ, p. 7]. The authors define the task flowtime as the elapsed simulation time between task generation and its completion [ZZ, Bak74]. Formally MSA is formulated as follows:

$$\min_{z} \quad z \tag{3.3}$$

subject to:

$$\sum_{i \in W} x_{ij} = 1 \quad \forall j \in T \tag{3.4}$$

$$a_i + \sum_{j \in T} x_{ij} p_{ij} \leq z \quad \forall i \in W \tag{3.5}$$

$$x_{ij} \quad \text{or} \quad x_{ij} = 1 \quad \forall i \in W, \forall j \in T \tag{3.6}$$

All variables definition still hold without loss of generality as defined by the authors [ZZ, pp. 5-7].

The class inheritance structure of the policies implementation with the corresponding fields and methods can be seen in Figure 3.3.

The authors definition of the MSA problem is however a simplified version of the actual problem of "minimizing the maximum task flowtime" (MF) as defined by Baker [Bak74] with consideration of the dynamic arrival of tasks problem, defined by the authors as the DMF problem [ZZ]. The DMF problem is formally defined by Zeng as follows:

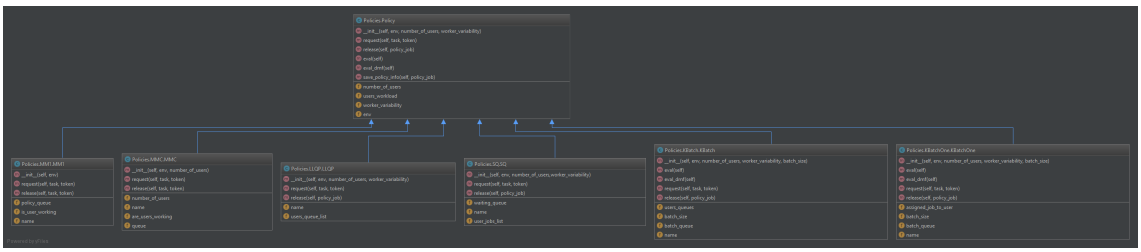

Figure 3.3: Policies class structure

$$\min_{z} \quad z \tag{3.7}$$

subject to:

$$\sum_{i \in W} \sum_{k \in T} x_{ijk} = 1 \quad \forall j \in T \tag{3.8}$$

$$s_j \geq r_j \quad \forall j \in T \tag{3.9}$$

$$(x_{ijk} + x_{ij'(k+1)} - 1)(s_j + p_{ij}) \leq s_{j'} \quad \forall i \in W, \forall k \in T, \forall j \in T, \forall j' \in T \tag{3.10}$$

$$s_j + \sum_{i \in W} \sum_{k \in T} p_{ij} x_{ijk} - r_j \leq z \quad \forall j \in T \tag{3.11}$$

$$x_{ijk} = 0 \quad \text{or} \quad x_{ijk} = 1 \quad \forall i \in W, \forall j \in T, \forall k \in T \tag{3.12}$$

$$s_j \geq 0 \tag{3.13}$$

Again, all variables definition still hold without loss of generality as described by the authors [ZZ, p. 6]. As Zeng notes in his work, Equation 3.10 contains nonlinear constraints but mentions that by adding auxiliary variables the aforementioned DMF formulation can be effectively converted into a mixed integer program and thus solve it [ZZ, p. 6]. On this note Zeng argues that the application of the DMF problem in practice poses some problems [ZZ]. In this thesis however a conversion of the DMF formulation proposed by Zeng is formulated in order to adequately solve the optimization problem. The formal definition of such optimization problem is called EDMF (which stands for extended DMF) and is devised as follows:

$$\min_{z_{\max}} \quad z_{\max} \tag{3.14}$$

subject to:

$$\sum_{i \in W} \sum_{k \in T} x_{ijk} = 1 \quad \forall j \in T \tag{3.15}$$

$$a_i + \sum_{j \in T} p_{ij} x_{ijk} \leq z_{i*k} \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k = 0 \tag{3.16}$$

$$z_{i*k-1} + \sum_{j \in T} p_{ij} x_{ijk} \leq z_{i*k} \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k > 0 \tag{3.17}$$

$$z_{i*k} + \sum_{j \in T} w_j x_{ijk} \leq z_{\max} \quad \forall i \in W, \forall k \in T \tag{3.18}$$

$$\sum_{j \in T} x_{ijk} \leq 1 \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k = 0 \tag{3.19}$$

$$\sum_{j \in T} x_{ijk} \leq \sum_{j \in T} x_{ijk-1} \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k > 0 \tag{3.20}$$

$$z_{i*k} \geq 0 \quad \forall i \in W, \forall k \in T \tag{3.21}$$

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

When considering the minimization of the maximum flowtime of a task inside a process, the EDMF formulation can be further simplified by adopting some assumptions about the order and sequence of tasks. Based on how the batching policies are implemented, the policy job objects to be worked by users are implicitly stored in a sorted fashion. This means that the z helper variables defined for EDMF are not strictly necessary and thus can be compressed by Equation 3.22:

$$a_i + \sum_{t=1}^{k} \sum_{j} (p_{ij} + w_j I(t = k)) x_{ijt} \tag{3.22}$$

The whole concept consists in the introduction of an identity variable I which is true 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.

Figure 3.4 depicts the potential scenario where three tasks are assigned to a specific user i following a sequence where task 2, 3 and j are assigned respectively as first, second and third tasks (thus respecting the k notation outlined above them).

In order to calculate z_{ijk}, one has to consider also when user i will actually be available to process his first task. 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 to process all tasks assigned to him.

Without further ado, the simplified formulation of the extended DMF variant (called SDMF, which stands for simplified DMF) is the following:

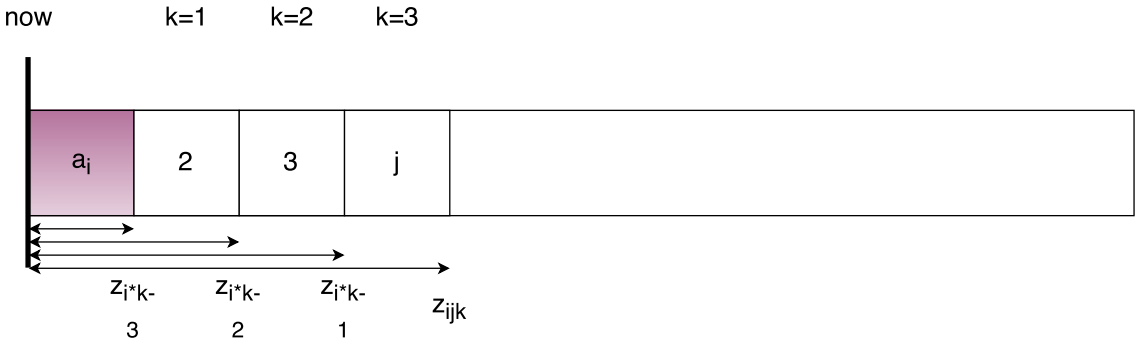

Figure 3.4: EDMF Task Assignment

$$\min_{z_{\max}} \quad z_{\max} \tag{3.23}$$

subject to:

$$\sum_{i \in W} \sum_{k \in T} x_{ijk} = 1 \quad \forall j \in T \tag{3.24}$$

$$a_i + \sum_{t=1}^{k} \sum_{j} (p_{ij} + w_j I(t = k)) x_{ijt} \leq z_{\max} \tag{3.25}$$

$$\sum_{j \in T} x_{ijk} \leq 1 \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k = 0 \tag{3.26}$$

$$\sum_{j \in T} x_{ijk} \leq \sum_{j \in T} x_{ijk-1} \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k > 0 \tag{3.27}$$

By comparing both formulation 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. Please note, however, that this simplification is only possible because of the nature of the implementation.

Based on this approach and by further exploiting the implicit order implementation of task arrival in the global queues for both batching policies, it is possible to argue that the k sequence indexing can be relaxed as well, thus even further simplifying the mathematical formulation and respectively the optimization problem size and computation costs.

The formulation of the DMF problem by relaxing both the z variables and k indexes, it is possible to formulate the same DMF problem as follows:

$$\min_{z_{\max}} \quad z_{\max} \tag{3.28}$$

subject to:

$$\sum_{i \in W} x_{ij} = 1 \quad \forall j \in T \tag{3.29}$$

$$a_i + \sum_{k=1}^{j} (p_{ik} + w_k I(k = j)) x_{ik} \leq z_{\max} \tag{3.30}$$

This formulation is colloquially called the ESDMF method (extremely simplified DMF).

TODO ▷*add explanation that by exploiting how the arrival order is implemented one can solve the same DMF problem without increasing the computability costs.*◁

Solver	Computation Costs
MSA	$O(mn)$
EDMF	$O(mn^2)$
SDMF	$O(mn^2)$
ESDMF	$O(mn)$

Table 3.1: Comparison of computational costs for different solvers

Sum of service times minimization with DMF as upper bound:

$$\min_{z} \quad \sum_{i \in W} \sum_{k \in T} z_{ik} \tag{3.31}$$

subject to:

$$\sum_{i \in W} \sum_{k \in T} x_{ijk} = 1 \quad \forall j \in T \tag{3.32}$$

$$a_i + \sum_{j \in T} p_{ij} x_{ijk} - M(1 - \sum_{j \in T} x_{ijk}) \leq z_{i*k} \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k = 0 \tag{3.33}$$

$$z_{i*k-1} + \sum_{j \in T} p_{ij} x_{ijk} - M(1 - \sum_{j \in T} x_{ijk}) \leq z_{i*k} \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k > 0 \tag{3.34}$$

$$z_{i*k} + \sum_{j \in T} w_j x_{ijk} \leq z_{\max} + \epsilon \quad \forall i \in W, \forall k \in T \tag{3.35}$$

$$\sum_{j \in T} x_{ijk} \leq 1 \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k = 0 \tag{3.36}$$

$$\sum_{j \in T} x_{ijk} \leq \sum_{j \in T} x_{ijk-1} \quad \forall i \in W, \forall k \in T \quad \text{for} \quad k > 0 \tag{3.37}$$

$$z_{i*k} \geq 0 \quad \forall i \in W, \forall k \in T \tag{3.38}$$

$$M = \max_{a} a_i + \max_{p} \sum_{i \in W} \sum_{j \in T} p_{ij} \tag{3.39}$$

$$\epsilon = 1 \times 10^{-4} \tag{3.40}$$

3.2 Reinforcement Learning Analysis Environment

In this section the reinforcement learning approach used to solve the different role resolution problems is depicted. Initially a foundation basis in the required knowledge is depicted and afterwards the description of the analysis environment implementation is presented.

3.2.1 Reinforcement Learning

Reinforcement learning is a novel approach originated as a branch from the broader field of machine learning. It is an automated approach to understanding and automating learning and

decision-making [SB, 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 [SB, p. 15].

The formal framework used by reinforcement learning defines the interaction between the so called learning agent and its environment by means of states, actions and rewards [SB, p. 15].

Key concepts in the field of reinforcement learning are those of values and value functions which helps distinguish reinforcement learning methods from evolutionary methods which have to undergo scalar evaluations of entire policies [SB, p. 15].

3.2.2 Finite Markov Decision Processes

Reinforcement learning approaches learn by interacting with the environment in order to achieve a goal. The agent interacting with the environment does this in a sequence of discrete time steps, it performs actions (choices made by the agent), reaches then states (basis for making decisions) and eventually receives rewards (basis for evaluating the choices) [SB, p. 73]. Moreover, a policy is a stochastic rule that the agent relies upon to choose actions as a function of states [SB, p. 73]. Ultimately, the sole goal of the agent is to maximize the reward that it receives over time [SB, p. 73].

Returns are modeled as functions of future rewards that an agents must maximize [SB, p. 73]. There exist two types of return functions which depend on the nature of the tasks and a discounting preference: 1. for episodic tasks a non discontinued approach is preferred while 2. for continuous tasks, however, a discounted approach is better suited [SB, p. 73].

Equation 3.41 defines the sum of the rewards received over time step t:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots R_T \tag{3.41}$$

If we account for discounting, Equation 3.41 has to be slightly adapted by introducing a discounting factor γ and can be found in Equation 3.42:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.42}$$

where $0 \leq \gamma \leq 1$.

An environment with which one agent interacts, can satisfy a Markov property if the information contained at present effectively summarizes the past without affecting the capability of effectively predicting the future [SB, p. 73]. If the Markov property is satisfied, then this environment is called a Markov decision process (MDP) [SB, p. 73].

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 [SB, 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 [SB, p. 74].

Optimal state-value functions v_* are formally defined as follows:

$$v_*(s) \doteq \max_{\pi} v_\pi(s) \tag{3.43}$$

whereas optimal action-value functions q_* are formally defined as follows:

$$q_*(s, a) \doteq \max_{\pi} q_\pi(s, a) \tag{3.44}$$

3.2.3 Dynamic Programming

Dynamic programming (DP) is a set of ideas and algorithms that can be used to solve MDPs [SB, p. 95]. There are two approaches in dynamic programming for solving MDPs: 1. policy evaluations is the iterative computation of value functions of a given policy and 2. policy improvement is the idea of computing an improved policy under the conditions of its given value functions [SB, p. 95].

By combining these two approaches we obtain the two most notable DP methods *i.e.*, policy and value iteration [SB, p. 95].

One very interesting property of DP methods is the concept of bootstrapping: updating estimates of values of states by approximating the values of future states [SB, p. 96].

3.2.4 Monte Carlo Methods

Monte Carlo (MC) methods use experience in form of sample episodes in order to learn value functions and optimal policies [SB, p. 123]. This approach yields different advantages over the DP methods seen in Section 3.2.3: 1. they do not need a model of the environment's dynamics as they learn the optimal solutions by merely interacting with the environment, 2. since the learn from sample episodes, they are very well suited for simulation environments, 3. it is efficient and surprisingly easy to use MC methods to focus on smaller regions or subsets of a problem and 4. MC methods are more robust when it comes to violations of the Markov property since they do not bootstrap for updating their values [SB, p. 123].

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 [SB, p. 123].

A MC simplified method can be formally defined as follows:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{3.45}$$

where G_t is the discounted return function defined by Equation 3.42 and α is a constant step-size parameter [SB, 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 [SB, p. 128].

3.2.5 Temporal-Difference Learning

Temporal-difference (TD) are yet another set of learning methods for reinforcement learning. Compared to the MC methods explained in Subsection 3.2.4, 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 [SB, p. 128]. When they reach time step $t + 1$, they observe a reward R_{t+1} which then use to estimate $V(S_{t+1})$ [SB, p. 128]. The simplest TD method, which is called $TD(0)$, is defined as follows:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{3.46}$$

TD methods, same as MC methods, are not excluded from the sufficient exploration methods [SB, p. 147]. TD methods deal with this complication in two different ways: 1. on policy by using an algorithm called Sarsa and 2. off policy by using an algorithm called Q-learning [SB, p. 128].

3.2.6 On-policy Prediction with Approximation

Up until now, the different methods presented are not suited for arbitrarily large state spaces. However, there exist solution to tackle such large state spaces: approximate solution methods. 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 [SB, p. 189].

An essential characteristic for reinforcement learning 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 [SB, p. 189]. Reinforcement learning methods are capable of achieving this by relying on supervised-learning function approximation which essentially use backups as training example [SB, p. 222]. Specifically, one very efficient set of methods are those using parametrized function approximation *i.e.*, the policy is parametrized by a weight vector θ.

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 [SB, 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 [SB, 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:

$$\theta_{t+1} \doteq \theta_t - \frac{1}{2}\alpha\nabla[v_\pi(S_t) - \hat{v}(S_t,\theta_t)]^2 \tag{3.47}$$

$$= \theta_t + \alpha[v_\pi(S_t) - \hat{v}(S_t,\theta_t)]\nabla\hat{v}(S_t,\theta_t) \tag{3.48}$$

where α is a positive step size parameter and $\nabla f(\theta)$ is the vector of partial derivatives with respect to θ:

$$\nabla f(\theta) \doteq \left(\frac{\partial f(\theta)}{\partial\theta_1}, \frac{\partial f(\theta)}{\partial\theta_2}, \ldots, \frac{\partial f(\theta)}{\partial\theta_n}\right)^\top \tag{3.49}$$

CITE ▷*add book citation*◁

A very special case (and also very simple) is linear methods for function approximation, where the approximate function $\hat{v}(\cdot,\theta)$ is a linear function of the weight vector θ [SB, p. 198]. This means that for each state s there is a corresponding vector of features $\phi(s) \doteq (\phi_1(s), \phi_2(s), \ldots, \phi_n(s))^\top$ which has the same number of components as θ [SB, p. 198]. With this definition in mind, we can now formally define the state-value function approximation as the inner product between θ and $\phi(s)$:

$$\hat{v}(s,\theta) \doteq \theta^\top\phi(s) \doteq \sum_{i=1}^{n}\theta_i\phi_i(s) \tag{3.50}$$

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

$$\nabla\hat{v}(s,\theta) = \phi(s) \tag{3.51}$$

Equation 3.51 tells us that for the simple linear case the SGD is nothing more than the corresponding features value [SB, p. 199].

3.2.7 On-policy Control with Approximation

Moving towards control with value function approximation, we now focus on the approximation of the action-value function $\hat{q}(s, a, \theta) \approx q_*(s, a)$ [SB, p. 229].

For the special case of the so called one-step Sarsa method, its gradient-descent update for the action-value function is defined as follows:

$$\theta_{t+1} \doteq \theta_t + \alpha[R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \theta_t) - \hat{q}(S_t, A_t, \theta_t)]\nabla\hat{q}(S_t, A_t, \theta_t) \tag{3.52}$$

and this method has also very good convergence properties towards optimality [SB, p. 230].

3.2.8 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[14] [SB, 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 [SB, p. 265]. For policy gradient methods it is crucial to learn the weight vector based on a performance measure $\eta(\theta)$ by trying to maximize and thus approximating the gradient ascent of η as follows:

$$\theta_{t+1} \doteq \theta_t + \alpha\widehat{\nabla\eta(\theta_t)} \tag{3.53}$$

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

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

$$\pi(a|s, \theta) \doteq \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}} \tag{3.54}$$

where $e \approx 2.71828$ [SB, p. 266].

Moreover, the preferences can be, as previously mentioned:

$$h(s, a, \theta) \doteq \theta^\top\phi(s, a) \tag{3.55}$$

i.e., simply linear in features [SB, p. 266].

With these definitions in mind, one can formally define one of the very first Monte Carlo based policy gradient methods: REINFORCE [Wil92].

Williams defines his REINFORCE algorithm by the following update function:

$$\theta_{t+1} \doteq \theta_t + \alpha\gamma^t G_t \frac{\nabla_\theta\pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \tag{3.56}$$

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 $\nabla \log x = \frac{\nabla x}{x}$ [SB, p. 271].

For the REINFORCE algorithm, its eligibility vector is defined as follows:

[14] Actor-critic methods are an exception, where a learned value function is used in combination with policy gradient as a baseline in order to lower variance.

$$\nabla_\theta \log \pi(a|s,\theta) = \phi(s,a) - \sum_b \pi(b|s,\theta)\phi(s,b) \tag{3.57}$$

and this method has solid convergence properties [SB, p. 271].

Policy Gradient with Baseline

REINFORCE, however, being a method based on Monte Carlo it might exhibit high variance and prove relatively slow in its learning rate [SB, p. 271]. By introducing a baseline $b(s)$ to compare the action value:

$$\theta_{t+1} \doteq \theta_t + \alpha(G_t - \overbrace{b(S_t)}^{\text{baseline}})\frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \tag{3.58}$$

one can achieve a positive effect towards diminishing variance of the update rule [SB, 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 actor-critic method as its state-value function is only used as baseline and not as a critic *i.e.*, it is not used for bootstrapping [SB, 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 [SB, p. 273].

The only negative aspect still remaining is that policy gradient methods are still based on a full Monte Carlo update trajectory: this can be also mitigated by replacing the update function by temporal difference learning approaches, such as those defined in Section 3.2.5 [SB, p. 273].

The formal definition of a one-step actor-critic update method is depicted as follows:

$$\theta_{t+1} \doteq \theta_t + \alpha(R_{t+1} + \overbrace{\gamma\hat{v}(S_{t+1},w) - \hat{v}(S_t,w)}^{\text{TD update}})\frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \tag{3.59}$$

and this is now a fully online implementation that executes updated after each newly visited state [SB, p. 274].

3.3 Hypothesis

3.4 Data

Chapter 4

Empirical Analysis

4.1 Results

4.2 Discussion

4.3 Research Contribution

Chapter 5

Conclusion

Appendix A

First

Appendix B

Second

Bibliography

[AR] Ivo Adan and Jacques Resing. Queueing theory.

[AS] Ruth Sara Aguilar-Savén. Business process modelling: Review and framework. 90(2):129 – 149. Production Planning and Control.

[Bak74] Kenneth R Baker. *Introduction to sequencing and scheduling*. John Wiley & Sons, 1974.

[BCMW07] Alex Bahouth, Steven Crites, Norman Matloff, and Todd Williamson. Revisiting the issue of performance enhancement of discrete event simulation software. In *Annual Simulation Symposium*, volume 40, page 114. Citeseer, 2007.

[Bis16] Johannes Bisschop. AIMMS Optimization Modeling, December 2016.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[CW] Dirk G. Cattrysse and Luk N. Van Wassenhove. A survey of algorithms for the generalized assignment problem. 60(3):260 – 272.

[Evo] Evolved Technologist. Bpm technology taxonomy: A guided tour to the application of bpm.

[FZDL] Shaokun Fan, J. Leon Zhao, Wanchun Dou, and Manlu Liu. A framework for transformation from conceptual to logical workflow models. 54(1):781 – 794.

[GHS] Diimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. 3(2):119–153.

[Gia] George M. Giaglis. A taxonomy of business process modeling and information systems modeling techniques. 13(2):209–228.

[GWT+13] Alborz Geramifard, Thomas J Walsh, Stefanie Tellex, Girish Chowdhary, Nicholas Roy, Jonathan P How, et al. A tutorial on linear function approximators for dynamic programming and reinforcement learning. *Foundations and Trends® in Machine Learning*, 6(4):375–451, 2013.

[Ken] David G. Kendall. Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain. 24(3):338–354.

[KP14] Nathaniel Korda and Prashanth A. Prashanth. On TD(0) with function approximation: Concentration bounds and a centered variant with exponential convergence. *CoRR*, abs/1411.3224, 2014.

[Mac93] A. L. Macintosh. The need for enriched knowledge representation for enterprise modelling. In *IEE Colloquium on AI (Artificial Intelligence) in Enterprise Modelling*, pages 3/1–3/3, Apr 1993.

[Mat08] Norm Matloff. Introduction to discrete-event simulation and the simpy language. *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, 2:2009, 2008.

[MHK] Gregory Mentzas, Christos Halaris, and Stylianos Kavadias. Modelling business processes with workflow systems: an evaluation of alternative approaches. 21(2):123 – 135.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Pin] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition.

[RA] Michael Racer and Mohammad M. Amini. A robust heuristic for the generalized assignment problem. 50(1):487–503.

[RvdA] Hajo A. Reijers and Wil M.P. van der Aalst. The effectiveness of workflow management systems: Predictions and lessons learned. 25(5):458 – 472.

[SB] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

[SBW+10] Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 2010.

[SHM+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[SLH+14] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. *Journal of Machine Learning Research*, 2014.

[Smi02] Andrew James Smith. Applications of the self-organising map to reinforcement learning. *Neural Networks*, 15(8–9):1107 – 1124, 2002.

[SMS+99] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999.

[SR09] Bruce Silver and Bruce Richard. *BPMN method and style*, volume 2. Cody-Cassidy Press Aptos, 2009.

[SZ] Sherry X. Sun and J. Leon Zhao. Formal workflow design analytics using data flow modeling. 55(1):270 – 283.

[SZNS] Sherry X. Sun, J. Leon Zhao, Jay F. Nunamaker, and Olivia R. Liu Sheng. Formulating the data-flow perspective for business process management. 17(4):374–391.

[Trk] Peter Trkman. The critical success factors of business process management. 30(2):125 – 134.

[Wil92] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[WZ] Harry Jiannan Wang and J. Leon Zhao. Constraint-centric workflow change analytics. 51(3):562 – 575.

[ZZ] Daniel D. Zeng and J. Leon Zhao. Effective role resolution in workflow management. 17(3):374–387.