Vrije Universiteit Amsterdam

Master Project
Msc. Business Analytics

Reinforcement Learning Based Control for Wastewater Purification Facilities

Author:
Kevin Overbeek

Graduation supervisor:
dr. R. Bekker

Company supervisor
B. van der Lugt MSc

July 6, 2020

Reinforcement Learning Based Control for Wastewater Purification Facilities

Kevin Overbeek

Master Project Business Analytics

Vrije Universiteit Amsterdam
Faculty of Science
Business Analytics
De Boelelaan 1081a
1081 HV, Amsterdam

Ynformed B.V.
Stadsplateau 4
3521 AZ, Utrecht

Preface

The report before you is written as the graduation project for the Master in Business Analytics at the Vrije Universiteit in Amsterdam. Business Analytics is a multidisciplinary program with a focus on computational intelligence, business process optimization, and financial mathematics. This report is written during a six-month internship between January and June 2020 at Ynformed, a data science consulting company operating in the public sector.

In this thesis, I will investigate the application of planning and reinforcement learning methods to control the amount of water entering a wastewater purification facility. We aim to create an agent that makes this stream as constant as possible using methods like buffering, anticipation, and coordination. This problem is based on a case from waterschap Rijnland, who we thank for providing us the case.

This thesis would not have been possible without the help from a number of people. First of all, I would like to thank dr. René Bekker of the Vrije Universiteit for all the ideas we came up with during our monthly meetings and his extensive and detailed feedback on draft versions of this thesis. Furthermore, I would like to thank Bart van der Lugt, my supervisor at Ynformed, for all the insights and support he provided during the process of writing this thesis. This support continued even when meeting in person was not possible, for which I am grateful. Finally, I would like to thank all my other colleges at Ynformed both for their kind and welcoming attitude as well as their extensive domain knowledge on which I could rely for the past six months.

Management summary

The adaptation of AI and digitization are trends that are changing both the public and private sector at a rapid pace. Waterschappen are no exception to this and are looking into AI-driven solutions to increase the efficiency of their operations. One example of this is the use of planning and reinforcement learning methods to control the inflow of water at their wastewater purification facilities.

Currently, a set of simple rules manages the influent of wastewater into these facilities. However, with a desire to have the influent as constant as possible, there are several inefficiencies under the current system. The inefficiencies that we address in this thesis are a lack of buffering to counter the daily pattern in wastewater, a lack of anticipation on future high influent periods, and a lack of coordination between the different entry points these wastewater purification facilities have. By creating an environment that incentives these characteristics we are able to train agents that show behavior more in line with the desires of the waterschap.

Although the data available to us covers a large period, planning and reinforcement learning methods require an amount of samples that are near impossible to obtain in the real world. To cope with this we create a simulator that produces data with characteristics comparable to what we would observe in the real world. An additional advantage of using a simulator is that it allows us to start with a simple problem and add layers of complexity in every subsequent environment. Four environments are introduced, each with their specific purpose. The first environment focuses on creating a realistic wastewater and rainwater influent pattern. The parameters of this environment are estimated based on the observed data to ensure our simulator closely resembles reality. This environment allows us to establish buffering behavior in our agents, which is the process of countering the daily pattern we see in wastewater through the use of the storage tank. After this, we introduce environments that include a forecast of future influent or that consist of multiple agents. These two new environments allow us to observe anticipating behavior in the case of the influent forecast or coordination in the multi-agent environment. Finally, an environment is created that combines all of these features into a single realistic environment.

The agents that we create are the decision making entities that interact with our environment. Along with the currently used rule-based agent, two different types of agents are used. A planning algorithm called value iteration is used because of its guaranteed optimal performance in simple, low dimensional problems. Due to the high complexity of our environments, the necessary adaptations are made to this algorithm. Along with this, four different deep reinforcement learning algorithms are used. These algorithms are general-purpose algorithms, but require some tweaking of the parameters in order to function optimally.

When we test our agents we find that our adaptation of the value iteration algorithm outperforms the rule-based and reinforcement learning methods by a quite significant margin. The value iteration agent showcases all of the three desired characteristics and does so with the added benefits of high transparency of its policy and near deterministic training to guarantee high performance in every run. When we perform a sensitivity analysis on various environment parameters we conclude that this good performance is maintained in circumstances slightly different from what the agent was designed for.

The deep reinforcement learning agents perform at a level that is about halfway between the rule-based agent and the value iteration agent. The downside of this class of algorithms is that finding good parameter settings is a process that takes a lot of time and the outcome can differ quite substantially between successive runs. In the sensitivity analysis, the deep reinforcement learning algorithms showed mixed results with the drop in performance being quite substantial in certain scenarios. However, it has to be noted that these results were obtained using the optimal parameter settings from our main analysis, and that performance is likely better after a full parameter analysis is performed.

In future research, the environments can be made slightly more realistic in several ways. An example of this is to add a multivariate simulation model for multi-agent settings. Another improvement of our simulator is to incorporate the lag between rainfall and the arrival of influent at the wastewater purification facility.

Overall, we are confident to conclude that the adaptation of value iteration is the best algorithm to cope with this type of problem. The value iteration agent showed all the desired characteristics, has a transparent policy, and shows a very stable learning trajectory, making it the ideal agent for our problem. The main weakness of this method is its lack of scalability, which becomes a problem when more state components need to be added.

Table of Contents

List of Figures

List of Tables

List of Symbols

$\mathbb{E}(\cdot)$	Expected value
$\min_x$	Minimum over variable x
$\max_x$	Maximum over variable x
$B(\cdot, \cdot)$	Beta function
$\Gamma(\cdot)$	Gamma function
$L(\cdot)$	Likelihood function
s_t	Observed state at time t
a_t	Observed action at time t
r	Observed reward at time t
s'_t	Observed next state at time t
$p()$	Transition probability
$\pi(s\|a)$	Probability of selecting action a in state s
γ	Discount factor
$V_\pi(s)$	State value of state s under policy π
$Q_\pi(s, a)$	State-action value of state s and action a under policy π
$V_k(s)$	State values of state s at iteration k
$Q_k(s, a)$	State-action value of state s and action a at iteration k
R_t	Expected reward at time t
G_t	Expected total reward at time t
S_t	State at time t
A_t	Action at time t
$\mathcal{S}$	State space
$\mathcal{A}(s)$	Action space
$T(s, a, s')$	Transition function
$R(s, a, s')$	Reward function
$A(s, a)$	Advantage function

1 Introduction

In recent years organizations are starting to adopt artificial intelligence (AI) to automate processes and increase efficiency. The public sector is no exception to this trend and it is believed that AI can have a significant impact on public services and policies. The importance of this is acknowledged by institutions like the OECD (2019) who outline how AI can be of value in the public sector. In the Netherlands, one type of public institution that has embraced AI are the waterschappen ("water boards" would be the English term for "waterschappen", but due to the ambiguous nature of the term "water boards" we opt to go with the Dutch term instead).

In this thesis, we look into a specific problem these waterschappen face and try to improve on the current solution using planning and reinforcement learning methods. However, before we look into the problem we will first look at the business context of both the waterschap and Ynformed, the host company. This helps us in understanding the problem and some of the terminology that is used throughout the thesis.

1.1 Business context

Waterschappen are a Dutch type of public institution that are in charge of managing bodies of water, water protection measures, and the sewage system / purification facilities. There are 21 of these waterschappen throughout the Netherlands, all with their own challenges depending on the area that they serve. The problem that we discuss in this thesis is based on a case from waterschap Rijnland. This waterschap serves an area of approximately 1100 km^2 in North- and South Holland.

The case that we are working on revolves around a wastewater purification facility that is managed by waterschap Rijnland. These purification facilities receive water from the sewage system, which we call influent. This influent can come from one or multiple entry points, depending on the facility. In our case, the influent is made up of two main sources; influent from wastewater and influent from rainwater. In some newer neighborhoods, these two sources do not share the same sewage system, with wastewater going to the wastewater purification facility and rainwater flowing back into a natural source without purification.

Water that arrives at the facility first reaches a storage tank before it is pumped into the facility. This gives the purification facility some control over the amount of influent entering the facility. The pumps that transfer water from the storage tank into the wastewater purification facility have a minimum and maximum possible speed, with all speeds in between being available to choose from. Unfiltered water can be stored in the sewage system, albeit at a limited capacity. For simplicity reasons we use the term storage tank when talking about water that has not yet entered the facility, however, when we use this term we mean the combination of the storage tank and storage capacity of the sewage system. When the maximum capacity of the storage tank is reached

the unfiltered water flows directly into a natural water source, which is called overflow. This is of course undesirable and should be avoided at all costs.

The thesis is carried out on behalf of Ynformed, a data science consulting company located in Utrecht. Ynformed is specialized in data sciences projects in the public sector. Ynformed is part of Royal HaskoningDHV, a Dutch advisory and engineering company. Waterschappen are organizations that Ynformed works for regularly which is very useful since domain knowledge is often required when working for these clients. As mentioned in the preface, this domain knowledge was of considerable help during this thesis.

1.2 Problem statement and research objective

In the business context, we briefly explained the general setup of a wastewater purification facility. In this subsection, we discuss the specifics of our problem, which are broken down in a number of research questions.

It was already touched upon that the waterschap has some control over the amount of water that enters the wastewater purification facility. We can use this control to improve the efficiency of the wastewater purification plant, which is optimal if the stream of influent into the facility is as constant as possible. Under the current method, this is far from optimal, which is why we are looking into the use of planning and reinforcement learning algorithms to see if we can improve on this. Specifically, we can break this down into three research questions.

1. Can we learn the agent to spread out wastewater as evenly as possible over the day?
2. Can the agent learn to anticipate on future influent, mainly coming from rain events?
3. Can we achieve coordination between multiple agents?

Note that we use the term agent to refer to the decision making entity, something that is common in planning and reinforcement learning research. Each of the research questions highlights a characteristic that is lacking in the currently used method but would help to smooth out the influent into the water purification facility. We will see in Section 3.1 that wastewater shows a strong daily pattern with high influent during the day and low influent during the night. Countering this daily pattern using buffering in the storage tank can help to make influent into the facility smoother. Rain showers are events that affect a limited amount of time periods but can have a high impact when they occur. Increasing influent in the periods leading up to a rain shower reduces the need to overcompensate when the rain shower occurs, smoothing out influent. Coordination between agents can be of use to counter some of the variation in the actions of individual agents.

In order to find answers to these research questions, we create a number of simulation environments. Each of these environments adds an element of interest that we can use to answer the research questions that are stated above. For example, the first environment creates a realistic influent

pattern for wastewater and rainwater. This influent pattern allows us to look at the first research question in isolation since there is a single agent and no forecast for future influent, ruling out the ability to anticipate on future influent. We then proceed to create environments that have multiple agents or enable the single agent to anticipate on future influent. This allows us to test the second and third research questions independent of each other. Finally, we combine all environments into a single environment that closely resembles reality. In this final environment, we can assess if our agents showcase the three desired characteristics of buffering, anticipation, and coordination.

The remainder of this thesis is organized in the following way. In Section 2 we will cover all literature related to our planning and reinforcement learning methods, as well as literature related to the design of our environments. In Section 3 we will discuss how the four different environments are designed and which behavioral characteristics they stimulate. Section 4 covers the design of our agents. The results are discussed in Section 6. Finally, we will discuss the conclusions of our research in Section 7 and give recommendations for future research in Section 8.

2 Preliminaries and related literature

In this section, we take a look at literature related to our own research. The reason to do so is twofold. First of all, work by other authors is a great source of inspiration for models, techniques, and concepts that we could use in our own work. A lot of problems similar to ours have already been studied, so it only makes sense to draw inspiration from this. Secondly, diving into related literature gives us a good understanding of where we can make a contribution and what the significance is of our findings.

The organization of this section is as follows. First, we look into dynamic programming and planning methods. These relatively simple techniques are the foundation of reinforcement learning and can even be applied to simplified versions of our problem. After this, we move on to traditional reinforcement learning methods like Q-learning. We also discuss several concepts that were introduced in traditional reinforcement learning literature, but which have become key in deep reinforcement learning. This gives us a good basis for the next subsection, which is about deep reinforcement learning. Finally, we look at a completely different topic, which is the simulation of influent. We use this to create our own simulation model in Section 3.

2.1 Dynamic programming and planning methods

As already stated in the introduction of this section, dynamic programming and planning are the foundation of reinforcement learning. The concept of dynamic programming was introduced in the 1950s by Richard Bellman (1954). It was designed to analyze multi-stage decision processes and finding an optimal policy, where a policy is defined as a sequence of decisions.

The central idea in solving these multi-stage decision processes through the method of Bellman (1954) is that we do not require a full overview of the whole decision sequence. Rather, we base our decision at every point in time on a set of quantities called the state variables. These state variables contain all relevant information, taking away the need to know past or future actions. This greatly reduces the dimensionality of the problem, especially in cases where the effect of our actions is stochastic.

Although dynamic programming can be applied to a wide range of problems, we are mainly interested in its use for solving Markov Decision Processes (MDPs). In the definition given by Watkins (1989), the MDP has four main components; a state-space $\mathcal{S}$, an action-space $\mathcal{A}$, a transition function $T(s, a, s') = p(s'|s, a)$, and a reward function $R(s, a, s') = \mathbb{E}[r|s, a, s']$. The set of available actions can be dependent on the state. If this is the case, $\mathcal{A}(s)$ denotes the set of available actions in state s. The transitions and rewards may be stochastic, which is why $p(s'|s, a)$ denotes the probability of ending in state s', given our initial state s and action a. $\mathbb{E}[r|s, a, s']$ denotes the expected reward given that our initial state is s, we take action a and end up in a new state s'.

In an MDP, the Markov property should hold. This property implies that only the current state and action are relevant to the dynamics of the transition and reward function. In other words, we do not need to know the previous states and actions as these do not have any influence on the environment. We can represent this as

$$p(s_{t+1} \mid s_t, a_t) \;=\; p(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) \tag{2.1}$$

It can easily be seen that the Markov property matches well with the requirement that all information should be contained in the state from dynamic programming. A key distinction between the two is the type of objective function that we work with. When dynamic programming was first introduced by Bellman (1954), the goal was to maximize the expected total reward in a multi-stage decision process of finite length. If our decision process ends at time T, our target at the current time t is written as follows.

$$G_t \;=\; R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \;=\; \sum_{k=0}^{T-t-1} R_{t+k+1} \tag{2.2}$$

The problem with this expected total reward G_t is that for multi-stage decision problems of infinite length, G_t can be infinite as well, as Sutton and Barto (2018) argue. To adjust for this, discounting was introduced. Under discounting, we value immediate rewards more than rewards we receive in the future. Our expected total reward G_t changes to an expected discounted reward, being defined as

$$G_t \;=\; R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \;=\; \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.3}$$

where $\gamma \in [0, 1)$ denotes our discount factor. Under the formulation in Equation 2.3, we are certain that G_t is finite as long as all rewards are finite. What the definition of G_t does not capture is that the expected discounted reward depends on the current state we are in and the policy that we follow. For this reason, we need a state-value function. Sutton and Barto (2018) define the state-value function in the infinite horizon setting $V_\pi(s)$ as the expected discounted reward under policy π if we are currently in state s.

$$V_\pi(s) \;=\; \mathbb{E}_\pi\left[G_t \,\middle|\, S_t = s\right] \;=\; \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, S_t = s\right] \tag{2.4}$$

This can be easily extended to a state-action value function $Q_\pi(s, a)$

$$Q_\pi(s,a) \;=\; \mathbb{E}_\pi\left[G_t \,\middle|\, S_t = s, A_t = a\right] \;=\; \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, S_t = s, A_t = a\right] \tag{2.5}$$

What makes these state-value function and state-action value function so useful is that they can be written as recursive functions. After we have selected our action we end up in a new state with its own state-value. This takes away the need to estimate all future rewards, as long as we have estimates of the state-values. In Equation 2.6 we can see these recursive versions of the state-value function and state-action value function. Here, we assume a deterministic policy (i.e. we always select the same action a in state s).

$$\begin{aligned} V_\pi(s) &= \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V_\pi(s')\big] \\ Q_\pi(s,a) &= \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma Q_\pi(s', a')\big] \end{aligned} \tag{2.6}$$

Bellman (1957) showed that under an optimal policy π^* there is a specific relationship between these state-values and state-action values. This relationship is known as the Bellman Optimality Equations and can be seen in Equations 2.7 and 2.8 for state-values and state-action values respectively.

$$\begin{aligned} V_{\pi^*}(s) &= \max_a \mathbb{E}\left[r + \gamma V_{\pi^*}(S') \,\middle|\, S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + V_{\pi^*}(s')\big] \end{aligned} \tag{2.7}$$

$$\begin{aligned} Q_{\pi^*}(s,a) &= \mathbb{E}\left[r + \gamma \max_{a'} Q_{\pi^*}(s', a') \,\middle|\, S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma \max_{a'} Q_{\pi^*}(s', a')\big] \end{aligned} \tag{2.8}$$

We can use planning methods to derive the optimal policy from these Bellman Optimality Equations. Sutton and Barto (2018) describe the two most important planning methods that can be used. The first method is policy iteration. Under policy iteration, we start off with some random deterministic policy. By iteratively finding the state-values and improving the policy, we are guaranteed to converge to the optimal policy. The other popular planning method is value iteration. This method combines the policy evaluation and policy improvement into a single algorithm. Both methods converge to the same optimal policy and state-values, but value iteration is often more efficient in doing so. For that reason, we use value iteration in the remainder of this thesis.

2.1.1 Value iteration

Value iteration is a method that was first introduced by Bellman (1957). This method makes direct use of the Bellman optimality equations for state and state-action values, as shown in Equations 2.7 and 2.8. We convert these optimality equations into an update rule for state-values and state-action values. These equations are proven to converge to the exact state-values in the limit. In practice, we stop updating when the maximum difference of two subsequent sets of state-values falls below a threshold. The value iteration update rules can be seen in Equations 2.9 and 2.10.

$$\begin{aligned} V_{k+1}(s) &= \max_a \mathbb{E}\left[r_{t+1} + \gamma V_k(s') \,\middle|\, S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p(s', r|s, a)\big[r + V_k(s')\big] \end{aligned} \tag{2.9}$$

$$\begin{aligned} Q_{k+1}(s, a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q_k(S_{t+1}, a')|S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s', r|s, a)\big[r + \gamma \max_{a'} Q_k(s', a')\big] \end{aligned} \tag{2.10}$$

In these equations, we have added a subscript k to the state and state-action values to denote the current iteration number. Furthermore, we have dropped the policy subscript as we no longer follow a single policy. Rather, we constantly update the policies in order to find the best one.

Pseudocode for the value iteration algorithm can be seen below. In this algorithm, θ determines the threshold value for convergence of the state-values. We find the optimal policy after the algorithm has converged by checking which action attains the maximum state value for every state. This optimal policy is again denoted by π^*.

Algorithm 1: Value iteration

Initialize V arbitrarily, e.g. $V(s) = 0 \ \forall \ s \in \mathcal{S}$

while $\Delta > \theta$ **do**
    $\Delta \leftarrow 0$
    for *each* $s \in \mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

Choose a policy π^* such that:
$\pi^*(s) = \text{argmax}_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

In a situation where we know the exact transition and reward functions, this algorithm is guaranteed to give us the optimal policy. However, in reality, these transition and reward functions are often unknown. Reinforcement learning does not assume these functions to be known, which is one of the main differences it has with planning methods.

2.2 Traditional reinforcement learning methods

In this subsection of the literature study, we are going to look at Q-learning, one of the most popular traditional reinforcement learning methods. Furthermore, we discuss several concepts like experience replay and function approximation which are key in deep reinforcement learning.

In our discussion of dynamic programming and planning methods we already mentioned that reinforcement learning does not assume the transition and reward functions to be known. The idea behind these methods is that we directly learn the state-values or state-action values, capturing the transition and reward functions implicitly in our estimates.

The way the state-values or state-action values are estimated is through incremental updating. These methods (called temporal difference methods) make use of the recursive nature of the state-values, as they update values based on the observed reward and estimated next state-value. The first to formalize this idea was Sutton (1988), but the concept itself has been around for much longer. Samuel (1959) used a technique which he called "rote learning" to teach an algorithm to play checkers. This technique stored estimates of board positions and used these estimates to evaluate a move of sequences. Furthermore, in the Adaptive Heuristic Critic algorithm by Sutton (1984) a similar update rule was used for the predicted reward signal.

Under regular temporal difference learning, we follow some policy π to select our actions. For every point in time, we execute our action and observe the reward r and next state s'. Using these observations, we update our estimate of the state value $V_\pi(s)$ through the update rule in Equation 2.11. The parameter α is the learning rate, which determines the size of the adjustment we make to our state-value estimate. The term $\big[r+\gamma V_\pi(s')-V_\pi(s)\big]$ is called the temporal difference (TD) error.

$$V_\pi(s) \leftarrow V_\pi(s) + \alpha\big[r + \gamma V_\pi(s') - V_\pi(s)\big] \tag{2.11}$$

The idea behind this update rule is that if we make α small enough, the state-values contain information on a large number of experiences. This is similar to using the transition function as we did under planning methods, with the advantage that we do not need a specific representation of this transition function.

2.2.1 Q-learning

The temporal difference update rule gives us a method to estimate the state-values of a given policy. However, what is still lacking is a method to find the optimal policy. This is what was addressed by Watkins (1989), who introduced Q-learning. The idea behind Q-learning is similar to the idea behind value iteration. Instead of evaluating a static policy, we adapt our policy to the

current estimates of the Q-values. We update our estimates for the state-values and state-action values according to Equation 2.12.

$$\begin{aligned} V(s) &\leftarrow \max_a Q(s,a) \\ Q(s,a) &\leftarrow Q(s,a) + \alpha\big[r + \gamma V(s') - Q(s,a)\big] \end{aligned} \tag{2.12}$$

To find the optimal policy there are two requirements. The first requirement is that the policy that we are executing depends on the current state-action values. The second requirement is that all state-action combinations have a non-zero probability of being selected during training. The reason for this comes from the work by Watkins and Dayan (1992), who showed that this is a required condition to prove convergence to the true state-action values.

There are several popular methods to select actions during Q-learning. One of the most popular mechanisms to do this is ϵ-greedy. Under this method, we take a random action with probability ϵ and take the best action based on the current state-action value estimates with probability $1-\epsilon$. It can easily be seen that under this method all actions have a non-zero probability of being selected. Another popular method is the soft-max method, where the selection probability is proportional to the estimated Q-value. Under this method, we can calculate the probability of selecting action a in state s using Equation 2.13. This method has a parameter τ, which determines how strongly we prefer actions with high Q-values over actions with low Q-values.

$$\pi(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a'} e^{Q(s,a')/\tau}} \tag{2.13}$$

The action-selection methods described above are guaranteed to converge to the true state-action values in the limit. In practice, it can still take a long time before this convergence occurs. To boost exploration in the early stages of the algorithm optimistic initialization is often used. Optimistic initialization means that all state-action values are set at a value that is likely to be above their true value. During training, the state-action values are adjusted downwards, causing a change in what the greedy action is. When this greedy action is selected, its respective state-action value is adjusted downwards as well, causing a different action to become the greedy action. This process repeats itself until the state-value estimates start to approximate the true state-values, having a higher degree of exploration in the meantime.

The Q-learning algorithm as described above can be seen in Algorithm 2. The algorithm is again based on the version from Sutton and Barto (2018), but it has been adapted to work with infinite horizon problems.

Algorithm 2: Q-learning for infinite horizon problems

Initialize V arbitrarily, e.g. $V(s) = 0 \ \forall \ s \in \mathcal{S}$
Initialize Q arbitrarily, e.g. $Q(s, a) = 0 \ \forall \ s \in \mathcal{S}, \ a \in \mathcal{A}$

for $i = 1 : n$ **do**
  Choose a from s using policy derived from Q (e.g. ϵ-greedy)
  Take action a, observe r, s'
  $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma V(s') - Q(s, a)]$
  $V(s) \leftarrow \max_a Q(s, a)$
  $s \leftarrow s'$

Choose a policy π^* such that:
$\pi^*(s) = \text{argmax}_a \, Q(s, a)$

2.2.2 Sample efficiency

One of the main critiques of the basic Q-learning algorithm is that it is very sample inefficient. When an experience is generated (i.e. a sequence of (s, a, r, s')) we only use it once to update a state-value, after which we forget the experience. Lin (1992) noted this lack of sample efficiency in his review of the Adaptive Heuristic Critic algorithm by Sutton (1984) and Q-learning algorithm by Watkins (1989) and proposed three extensions to improve the sample efficiency. Of these three extensions, experience replay turned out to speed up learning a lot without the need to learn a model of the environment. This makes it an easy solution to speed up learning.

The way experience replay works in a tabular setting is explained by Adam et al. (2011). An experience buffer is set up of some fixed size. Observed experiences are stored in this buffer, where the oldest experiences are dropped if the buffer is full. After a pre-determined number of experiences, we sample experiences from our experience buffer and use these to update the state-action values. Since we can use the same experience multiple times to update the state-action values, this algorithm is generally more efficient than regular Q-learning.

The most obvious benefit of using experience replay is increased sample efficiency. However, Adam et al. (2011) found another benefit of using it. In environments where rewards are only received in a small number of states (e.g. maze problems), experience replay can help to propagate information from the goal states to all other states. Under normal Q-learning, only the state leading up to the goal state is adjusted the first time we reach this goal state. If we want to propagate this information further back, we need to sample the same path again. It can easily be seen that this method does not use all available information in states leading up to the goal states. Under experience replay, this path leading up to the goal state is in our experience buffer. If we then sample steps from this path during the replay stage, we help to propagate the reward from the goal state backward. This helps to give the agent an idea about the direction it should move to.

This exact idea is key in another sample efficiency-related concept called eligibility traces. This concept comes from traditional animal learning theory and was, for example, discussed by Sutton and Barto (1981). Furthermore, the concept was applied in a control problem setting by Barto et al. (1983). The concept was first applied in a temporal difference setting by Sutton (1988), who discussed its advantageous effect on learning speed. Under eligibility traces, we do not only update the last visited state but all recently visited states. This makes learning more stable and faster to converge, especially in environments with sparse rewards. [1]

The final sample efficiency concept we discuss is called Dyna. Just like the other sample efficiency concepts discussed here, Dyna was introduced very early in the development of reinforcement learning methods. Sutton (1991) came up with the idea to integrate planning and reinforcement learning into a single algorithm. After each interaction with the environment, the agent uses the gained experience for two causes. Firstly, it updates the state-values like it would under a normal Q-learning algorithm. Secondly, it uses the experience to update its action model. The action model captures the agent's belief about how he transitions between states and the expected reward given the selected action. This action model is used to sample hypothetical experiences, which are used to update the state-values or state-action values of the model after each recorded experience.

In a way, Dyna can be seen as an extended version of experience replay. Where experience replay stores the experiences and samples from these, Dyna uses the experiences to create a model of the environment and samples from this model. While this approach could be more efficient in terms of memory use, it does assume that we can find and represent a realistic model of the environment.

2.2.3 Function approximation

While most traditional planning and reinforcement learning methods assume a tabular form of the state-values (or state-action values), the idea to represent them by some function is far from new. Early reinforcement learning work often discussed how reinforcement learning was different from supervised learning, see for example Sutton (1988). Most supervised learning methods at the time made use of neural networks or other function approximators, so the idea to use function approximators to represent the state-values or policy was proposed almost instantaneously. Examples of this can be found in the aforementioned work by Sutton (1988) when he introduced the concept of temporal difference methods, the introduction of Q-learning by Watkins (1989), and the work by Lin (1992) when he discussed the concept of experience replay.

Generally, there are two ways in which we can use function approximators in reinforcement learning. The first way is to use a function approximator to map states to actions, which is a representation of the policy. If $\boldsymbol{\theta}$ is the vector of policy parameters, we can represent the probability of selecting action a in state s as $\pi(a \mid s, \boldsymbol{\theta})$. We update our parameter vector based on the derived gradient $\nabla\boldsymbol{\theta}$.

[1] For a full specification and explanation of the algorithm, we refer to Sutton and Barto (2018).

One of the first to use this type of function approximator is Williams (1992) when he introduced his REINFORCE algorithm. This algorithm first simulates a full episode given the current policy $\pi(a \mid s, \boldsymbol{\theta})$, after which updates are performed proportionally to their action-selection probability and observed return. This lets the network converge towards actions with higher observed rewards, hence finding a better policy. The downside of this method is that it is episode-based, which is due to the absence of a value function in this algorithm.

The second way we can use a function approximator is to estimate the value function instead of the policy. This is the type of function approximator that was used in the work by Sutton (1988). A similar idea was introduced by Watkins (1989) when he discussed function representation of the state-action values. Under tabular methods, having a large state or action space gives rise to what Bellman and Dreyfus (1962) calls the curse of dimensionality. This curse of dimensionality means that the number of state-action pairs is too large to represent in a tabular form due to the large number of combinations. Using a function, for example a neural network, to represent the relation between state-action pairs and their respective value could lead to a large decrease in the number of required parameters, overcoming this curse of dimensionality.

Finally, it is possible to combine these two types of function approximators into a single framework. In the literature, this is known as the actor-critic framework where the policy function approximator is the actor and the value function approximator is the critic. While it is a popular concept in (deep) reinforcement learning, actor-critic methods were introduced in the field of control theory by Witten (1977). Konda and Tsitsiklis (2000) explain that the idea behind the actor-critic architecture is that we can use the critic to bootstrap the action of the actor. This takes away the need to work with episodic problems like we had during the discussion of the REINFORCE algorithm. The temporal difference error is calculated using the observed reward, estimated state value, and estimated next state value. This error can be used to update both the actor and the critic, letting them converge to the optimal policy and value function respectively.

The advantage of actor-critic methods is that it allows for multiple agents training in parallel. These agents then share the same critic which is updated based on the results of all agents. Having multiple agents working on the same problem increases both the speed of convergence and stability during learning.

2.3 Deep reinforcement learning methods

Now that we have covered the foundations of reinforcement learning methods and have introduced some of the key concepts, we can dive deeper into deep reinforcement learning methods. This section starts with a general introduction on deep reinforcement learning, answering questions like how deep reinforcement learning is different from function approximation, in which cases deep reinforcement learning has been successfully applied, and what advantages deep reinforcement learning has over methods discussed in the previous subsection. After this, we look deeper into

three specific deep reinforcement learning models. These models are used in Section 4 to act as the agents in our environment, so it is key that we understand these models properly.

The first paper to introduce methods that we now know as deep reinforcement learning was the paper by Mnih et al. (2013). In this paper, the authors identified that *"learning to control agents directly from high-dimensional sensory inputs like vision and speech is one of the long-standing challenges of reinforcement learning"* (p. 1). By then, all successful reinforcement learning applications had a value function based on features created by the user. Failing to produce good features also meant poor performance of the reinforcement learning algorithm. To circumvent this problem, Mnih et al. (2013) created a model that directly mapped high-dimensional inputs (pixel values in this case) to actions and showed its performance on seven different video games. The absence of feature engineering is the main difference between what we call deep reinforcement learning and function approximation. It was found that the new method was state-of-the-art at that time on six out of seven tested games, beating human performance on three of those games.

To achieve this level of performance the authors combined a convolutional neural network with experience replay. The reason a convolutional neural network was used was to deal with the high-dimensional input data. This type of neural network can automatically extract features from this type of input data, taking away the need to do manual feature extraction. Experience replay was used to avoid having highly correlated samples, as subsequent states are often closely related. Furthermore, experience replay helps to overcome local optima by also remembering off-policy actions.

These results were extended in the 2015 paper by Mnih et al.. In this paper, the algorithm was tested on 49 different classic video games. In 29 of these games, the algorithm beat human-level performance or came close to human-level performance. The only change that was made to the algorithm compared to the 2013 version of the algorithm is that the next state-value is approximated based on an old version of the parameter vector that is updated periodically, helping to further reduce the correlation between samples.

To improve convergence and stability, van Hasselt et al. (2016) applied double Q-learning in a deep reinforcement learning setting. The central concept is the same as in standard double Q-learning; avoid over-estimations of the next state-action value through the use of a double estimator (van Hasselt, 2010). Applying double Q-learning in a deep reinforcement learning setting only requires small changes to be made. Instead of using a single parameter vector θ, we now use two parameter vectors denoted by θ_a and θ_b. Working with two parameter vectors takes away the need to use an old version of the parameter vector during updating. Rather, we update one of the parameter vectors based on the bootstrapped state-action value of the other parameter vector. This approach is tested on the same set of video games as used by Mnih et al. (2015), where it is shown that the double Q-learning approach consistently finds a better policy because it suffers less from overestimation.

Further improvements were made by adjusting the experience replay implementation. Schaul et al. (2016) noted that in neuroscience evidence for experience replay exists, but in a way different than is used in reinforcement learning. Experiences that have high rewards or are very different from what is expected (i.e. that have a high temporal difference error) are replayed more often. Under regular experience replay, this is not the case. Prioritizing certain events during learning was not new to reinforcement learning, exemplified by the concept of prioritized sweeping which was introduced by Moore and Atkeson (1993). What Schaul et al. (2016) did was to combine the concepts of experience replay and prioritized sweeping. Instead of giving every experience the same probability of being selected during the replay stage, experiences are sampled proportional to their observed TD error. This centers improvement of the policy around states where it performs poorly, causing a quicker convergence. The prioritized experience replay algorithm beat the original deep reinforcement learning algorithm of Mnih et al. (2015) on 41 out of the 49 tested games.

There are two main advantages that deep reinforcement learning has over planning and traditional reinforcement learning methods. The first advantage is that deep reinforcement learning methods can easily work with large state or action spaces. This is because the size of the state and action space only influences the size of the first and last layer of the neural network, adding more neurons when we increase the state or action space. The second advantage is that deep reinforcement learning methods can easily work with continuous state and action spaces. This is unlike tabular methods like value iteration or Q-learning which require a discrete number of actions and states.

In the next subsections, we discuss three different deep reinforcement learning models that are used as agents later in this thesis. We discuss these methods in the same order that they were introduced in, as they draw inspiration from one another.

2.3.1 Trust Region Policy Optimization (TRPO)

The first algorithm that we discuss is called Trust Region Policy Optimization and was introduced by Schulman et al. (2015). The main advantage of this method is that it theoretically guarantees improvement in every update of the policy as the algorithm trains itself. To do so, it makes use of the advantage function A. The advantage function is the difference between a state-action value Q and the state value V and can be written as

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{2.14}$$

where π represents the policy that is currently followed. The goal is to use the advantage function to find a new policy $\tilde{\pi}$ that has a non-negative advantage for all states. However, in practice, this is impossible in most cases. To circumvent this, Schulman et al. (2015) created a policy iteration algorithm that does not improve all states at the same time but is guaranteed to give an improvement in the overall policy in each update.

This new algorithm is tested on two classical reinforcement learning tasks; robot locomotion tasks and game playing. These games overlapped with the ones used by Mnih et al. (2013) to make comparisons easier between the newly introduced algorithm and the then state of the art. The results that the algorithm achieved were somewhat mixed. On the locomotion tasks, the method was generic enough to create a good controller for all tasks, something that had not been achieved by then. In the game playing it outperformed the algorithm by Mnih et al. (2013) in about half of the games, so no clear improvement was found there. Duan et al. (2016) confirms that TRPO is one of the best available algorithms on locomotion tasks at the time in a benchmark study using 28 different problems and 9 different algorithms to solve these problems.

2.3.2 Advantage Actor Critic (A2C)

The second deep reinforcement learning algorithm that we use is called Advantage Actor Critic (A2C). As the name already suggests, it makes use of the actor-critic concept that was introduced in the previous subsection. The concept was introduced by Mnih et al. (2016) and builds on the deep reinforcement learning paper by Mnih et al. (2015). The reason why the actor-critic concept is of added value is twofold. First of all, it improves the speed at which we can train the agents since this type of agent can be trained in parallel. Secondly, having multiple instances of the agents running at the same type improves training stability, potentially leading to better overall solutions.

The key idea in this paper is that there are two types of parameter vectors; the global parameter vector and the individual parameter vector. Each agent selects his actions based on the global parameter vector. These actions are evaluated using the individual parameter vector of the agent and gradients are calculated based on these evaluations. After every $n_{individual}$ time steps the agent updates his version of the global parameter vector through copying the global version. This is needed because this global parameter vector might have changed due to other agents performing updates. After every n_{global} time steps the agent uses the computed gradients to stepwise update the global parameter vector.

When we look at the results we can see improvements in both the training speed and performance of the algorithms, as was to be expected. The A2C algorithm was training in half the time it took to train all the benchmark models while using CPU's instead of GPU's. Furthermore, the new A2C algorithm was able to beat the state of the art in most of the games that it faced.

2.3.3 Proximal Policy Optimization (PPO)

The final algorithm that we consider was introduced by Schulman et al. (2017), the same author who introduced the TRPO algorithm. The idea behind this algorithm is fairly simple, as it combines the policy iteration algorithm from TRPO with the actor-critic element from A2C. The introduction of A2C already showed great improvement over the traditional deep reinforcement algorithm, so combining these two methods seems like a sensible next step.

When looking at the results, we can conclude that this concept works well, especially on the previously mentioned locomotion tasks. On these tasks, PPO is both successful in obtaining the highest average score and showing the fastest convergence speed. On the Atari games that were used in the original paper by Mnih et al. (2015) it beats the A2C algorithm in 41 of the 49 games.

2.4 Influent simulation

So far we have focused on the different optimization methods that serve as agents in our reinforcement learning setting. However, setting up the environment itself is a challenging task as well. In Section 3 we describe this task in-depth, but before we do so we first look into the relevant literature in this field. Specifically, we look into the literature on precipitation simulation, as this is one of the main sources of influent in our model.

Weather simulation is a tool commonly used in hydrological research. It can serve as an input for, among others, climate change scenarios (Kilsby et al., 2007; Semenov and Barrow, 1997), flood risk analysis (Blazkova and Beven, 1997) or soil erosion models (Favis-Mortlock and Boardman, 1995; Williams, 1990). Under a typical weather simulation model multiple variables are estimated which are dependent on each other and/or other locations. The relation between variables can be modeled as a full multi-variate process or using a two-stage model where the variables from the second stage are conditioned on the variables from the first stage. This two-stage model is, for example, used by Richardson (1981).

There are several different reasons why one would prefer simulated data over the observed data. The first reason is that we can generate an arbitrarily large number of observations with a simulation model, whereas we cannot using only the observed data. Certain applications like extreme-event analysis and training deep reinforcement learning models require more samples than the observed data can provide. Using simulation models is a way to work around this. Another advantage of simulation models is the control the user can exert over the properties of the data it generates. This is, for example, useful in scenario analysis. Making rainfall more intense, drought periods longer, or raising average temperatures produces data different from what is observed in reality. The main downside of using simulation models is that it may not be a perfect representation of reality, so bias could be introduced into the simulated data.

In this section, we look into the simulation of precipitation data as this affects the amount of influent at the purification facility. Variables like minimum- and maximum temperature or solar radiation are often included in the model as well, they are not relevant for our environment so we choose to neglect these.

One of the first weather models to be introduced was made by Richardson (1981). He recognized that hydrological models often need weather data as input and that using only the observed data gives an incomplete picture. In this weather model, four variables (precipitation, minimum

temperature, maximum temperature, and solar radiation) are modeled at a daily scale. Of these variables, precipitation is modeled independently of all other variables. The probability of rainfall on a given day is conditioned on yesterday's weather since days with rainfall often cluster together. The amount of rain given rainfall occurs on a given day is modeled by an exponential distribution and is independent of the amount of precipitation on the previous day. Seasonal effects are added to both the probability of rainfall and the amount of rainfall.

The model by Richardson (1981) works well for daily data, but extending it to higher frequency data causes a few problems. The first problem is that the authors assume that the probability of rainfall on a given day only depends on yesterday's rainfall, the so-called Markov property. This property is unlikely to hold when we work with 5-minute data, as observations from an hour ago can still hold information on whether more rain is expected. We could solve this by including more past observations in the state space, but it is far from ideal. The second, more severe problem is the assumed independence between the amount of rain in successive periods. When working on a 5-minute level, the amount of rain that has been registered in the current period is likely to be close to the amount of rain that is registered in the next period. Modeling these events as being independent underestimates the probability of long intense rain showers.

For this reason, we turn to the paper by Acreman (1990). In this paper, hourly data is used which is modeled in a different way than was done by Richardson (1981). Under this approach, rainfall is modeled in three different steps. The first step is to model the event durations (i.e. the length of a dry or rain spell). To do this, three different distributions are fitted on the observed event durations. These distributions are the exponential distribution, log-normal distribution, and gamma distribution. The one with the best fit (as measured by the χ^2 goodness of fit measure) is selected and is used to sample from during the simulation process.

The second step is to model the event depths. The event depth is defined as the amount of rain that is observed during the event. Similarly, we define the intensity of an event as the average rainfall during the event. Since there is a strong relationship between the depth and duration of an event, Acreman (1990) decided to model the depth within different duration buckets, each bucket containing 1 or multiple event lengths. Then within each bucket, the same fitting procedure as for the event durations can be used. This time, the distributions that were considered are the log-normal distribution and the gamma distribution. An alternative approach is to model the event intensities instead of the event depths, which is the approach we use.

The final step is to model the event profiles. The event profile is how the event depth is distributed over the duration of the event. The author observed that individual profiles differed a lot, but the average over all profiles follows a smooth shape. This observation holds over all different durations tested, so it seems that there is some underlying structure in the event profiles. The fraction of rain fallen within an event is bounded between 0 and 1, so the author chooses to model this with a beta

distribution. The density of the beta distribution can be seen in Equation 2.15. In this density, $B(\cdot,\cdot)$ is the beta function, $\Gamma(\cdot)$ is the gamma function and α and β are the model parameters.

$$\begin{aligned} f(x) &= \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} \\ B(\alpha,\beta) &= \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \end{aligned} \tag{2.15}$$

However, there is a problem with modeling the fraction of rain fallen using a beta distribution. The simulated fractions may show a decrease between subsequent periods, which would correspond to a negative amount of rain. To circumvent this problem, the author notes that it is possible to simulate the proportion of rain still to fall in the remaining time periods using the same beta distribution. This quantity decreases over time depending on how much rain has already been registered in previous time periods and the total rainfall of that specific rain shower. Under this approach, simulated rain quantities are always positive which is of course a desired feature to have. The parameters of the beta distribution are determined based on the observed events of equal length.

The approach by Acreman (1990) is in many aspects suitable to handle datasets with a higher frequency as well. Specifically, the approach to model event durations and intensities can be directly applied to such datasets. The approach to model event profiles is harder to extend to high-frequency data. During the simulation of an event, the approach by Acreman (1990) makes use of observed events with the same length. This approach implicitly assumes that such events are available in the dataset. When high-frequency data is used, it can occur that the length of a simulated event does not match with any observed event, meaning that it is not possible to extract the required information for the simulation.

To solve this problem, we have to look at approaches that are independent of the time scale of the event. Brigandi and Aronica (2019) applies one such approach through the use of dimensionless hyetographs, also called mass curves. The concept of mass curves has been studied for an extensive period of time, for example by Huff (1967) and Garcia-Guzman and Aranda-Oliver (1993). The idea here is that we model the normalized cumulative event depth versus the normalized time, which we can transform based on the simulated event duration and intensity. To do this, all events are transformed into the same number of periods, which is 25 in this case. For every time step, the α and β parameters are estimated based on all observed events. These parameters are used to simulate new mass curves, which are rescaled such that they match the simulated event duration and intensity.

These methods to simulate event durations, event intensities, and event profiles are the basis for our simulation model for rainfall. This model is discussed extensively in Section 3.2.

3 Environment Design

In this section, we start with the methodological aspects of our research. The first major aspect is the design of our environment, which is of course essential in every reinforcement learning task. In the introduction we have already discussed the context of our problem. Through the design of the environment, we aim to make a virtual representation of this context which closely matches reality.

As argued before, we take a step-wise approach in designing our environment such that we can see the effect of every added element. This idea is also reflected in the organization of this chapter. We start by introducing the two main influent streams at every water purification facility, wastewater and rainwater in Sections 3.1 and 3.2. Next, we show how we can create a forecast for these influent steams in Section 3.3, information that will improve the efficiency of our agents. Finally, in Section 3.4 we explain how we can scale our environments to a multi-agent setting and what the various components of the reward function are in Section 3.5.

3.1 Simulation of wastewater

The first influent stream we consider is wastewater coming from households and companies. Before we can start with setting up the simulation environment for wastewater, we first need to discuss the desired properties of a wastewater simulator according to domain experts.

The first property is that during the night the influent is below the minimum pump capacity. This means that the agent is not able to let the pump run at some fixed level. This creates the need to either turn off the pump in some time periods or to use the storage tank as some kind of buffer. This buffer would be filled during the day and be depleted during the night.

The second property is that there is a clear daily pattern visible in the influent of wastewater. According to domain experts from waterschap Rijnland, we see two peaks during the day. The largest peak occurs in the morning with a slightly smaller peak present during the early evening. In between those peaks, water consumption falls as most people are at work or at school. During the night we see a large dip in water consumption as most people are asleep. We extract this pattern from influent data later in this section.

The final property is that the noise of the wastewater influent is not independent between successive time periods. It takes a number of time periods for water to reach the purification facility, depending on the distance that it has to cover. This means that a peak in water consumption gets spread out over several time periods, leading to correlated influent values after taking into account the daily pattern. It is important to take this into account, as it influences the amount of information an agent can extract from the current observation.

Now that we have discussed the required properties, we can continue with the simulation model. However, before we do this, it seems like a good idea to first take a look at the dataset to see if this matches our expectations. The dataset that has been provided to us by waterschap Rijnland contains influent values at a 5-minute interval between the 16th of January 2017 and the 21st of June 2019. We can see an example of an influent series over a time span of 3 days in Figure 1. To eliminate the effect of rain, we have taken a sample of days on which no rain has occurred. Filtering out these days has been done using an open dataset from the Koninklijk Nederlands Meteorologisch Instituut (2020).

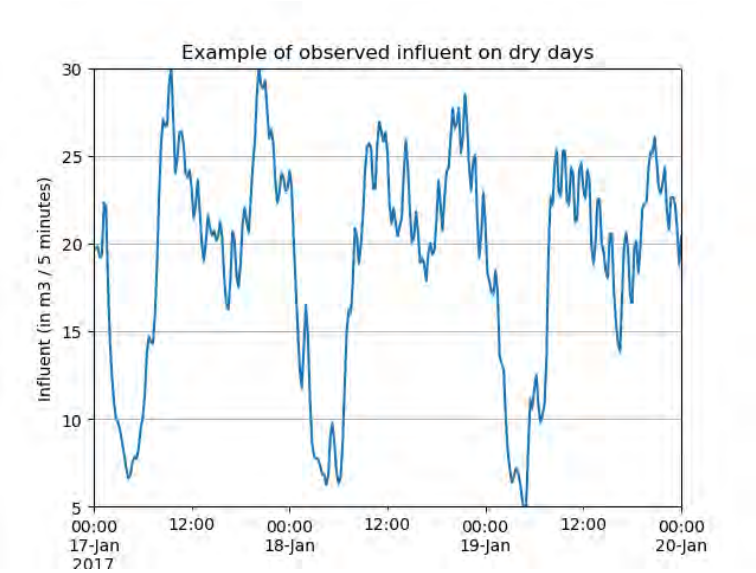

Figure 1: Observed influent on dry days

In the figure above, we can see the observed influent in m^3 per 5 minutes on three subsequent dry days, as observed at one of the water purification facilities of waterschap Rijnland.

In Figure 1 we can indeed observe the double-peaked pattern that was described by domain experts. There is some noise around this pattern, but from this plot it is not visible if this noise has any sequential dependence. We assess this later in this subsection.

The first step in modeling this influent pattern is to extract the underlying daily cycle on days without rainfall. To improve the data quality, we first take some simple preprocessing steps. The first step is to drop all observations without an influent value since we are certain that this is a measurement error. The next step is to get rid of the outliers. This is needed because the dataset contains values that are unrealistically high or low from the perspective of a domain expert. We get rid of these observations by dropping the top and bottom 1% of all influent values.

Using the filtered dataset, we can now extract the daily pattern on dry days. We do this by taking the mean influent at every time point. As expected, there is a clear daily pattern with a period of low influent during the night and two large peaks during the day. To filter out the noise that is still remaining in this pattern we perform a simple smoothing step. We can see the extracted and smoothed pattern in Figure 2.

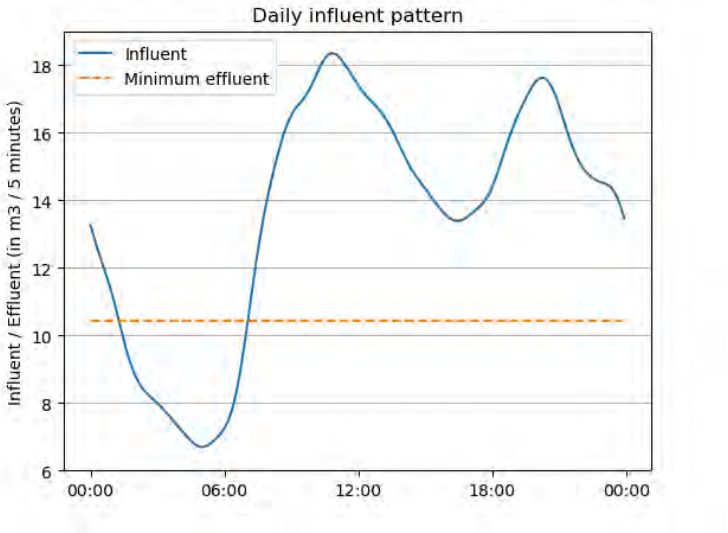

Figure 2: Daily influent pattern on dry days

In the figure above, we can see the average observed influent in m^3 per 5 minutes on dry days. This pattern is extracted on the dataset as provided to us by waterschap Rijnland, which contains influent data between the 16th of January 2017 and the 21st of June 2019.

One thing that we did not consider while extracting this daily pattern is the effect of rain on the previous day. If we have a large amount of rain just before midnight, this rain affects influent values just after midnight. If there is no more rain the next day, these influent values end up in our dataset while they are still affected by rain. The effect of this would be that the decrease in influent between 23:55 and 00:00 is much smaller than the decrease in influent in the time periods right before and after midnight. However, we find no evidence that this is the case, as the decrease in influent between 23:55 and 00:00 (-0.288) was almost similar to the average decrease in adjacent periods (-0.292 between 23:45 and 00:15).

The next step in modeling wastewater influent is to consider the relation between the noise in subsequent samples. To do this, we first have to normalize all observations, which can be done in two ways. The first option is to subtract the expectation from all observations, centering them around 0. The second option is to divide all observations by their expectation, centering all observations around 1. The option that we choose impacts the kind of noise we generate. If we use the subtraction method we create noise by adding a noise series to the baseline, assuming that the variance remains fixed over the day. Using the method where we divide observations by their expectation allows us to add noise that scales linearly with the expected influent. Since the data tells us that variance is generally higher for time periods where the influent is high, we choose to normalize all observations by dividing them by their expected value according to the daily pattern. We refer to this ratio as the scaling factor of an observation. We have plotted this for a single day in Figure 3.

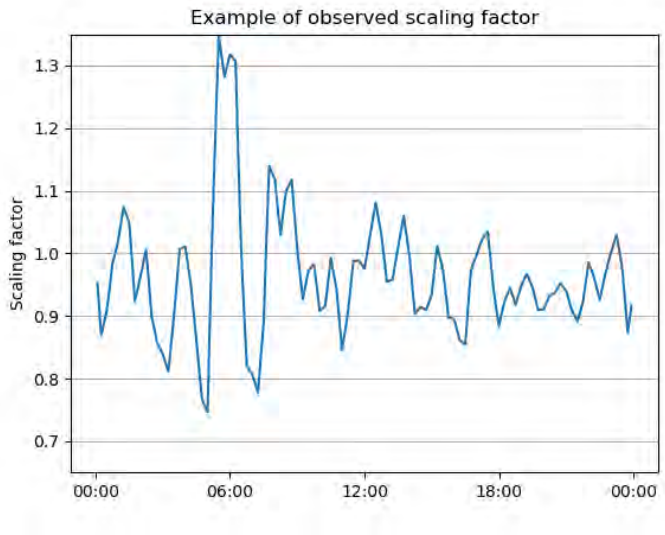

Figure 3: Observed scaling factor example

In the figure above we can observe the scaling factor on the 19th of January 2017. It is extracted based on the dataset as provided to us by waterschap Rijnland. Observations are centered around 1 to account for long-term trends present in the data.

In Figure 3 we can see the scaling factor that we have just described. We observe that the autocorrelation is high and that the process shows signs of mean-reversion. These are properties that we have to consider while modeling this process. Furthermore, it is important to note that the scaling factor can never become negative since this would imply that influent is negative as well. To model these properties, we make use of the Ornstein-Uhlenbeck process on the logarithm of the scaling factor. The process introduced by Uhlenbeck and Ornstein (1930) has a mean-reversion property and normally distributed noise. Fitting this process on the logarithm of the scaling factor ensures that influent cannot become negative. Furthermore, typical values of this scaling factor lie between 0.7 and 1.3, so taking the logarithm of this is a near-linear transformation of what is shown in Figure 3. The full model specification can be seen in Equation 3.1.

$$\begin{aligned} Y_t &= e^{X_t} D_t \\ dX_t &= -\theta X_t + \sigma dW_t \end{aligned} \tag{3.1}$$

In Equation 3.1, Y_t denotes the influent value at time t. This influent value is determined by the value of the daily influent pattern D_t and the scaling factor e^{X_t}. The logarithm of the scaling factor X_t is modeled as an Ornstein-Uhlenbeck process where dW_t denotes the increment of a Brownian motion. A Brownian motion is a simple stochastic process with mean 0 and standard deviation σ. The Ornstein-Uhlenbeck process has two parameters: θ and σ. The θ parameter determines the degree of mean-reversion per time step, whereas σ determines the standard deviation of the random noise.

To estimate the parameters of the Ornstein-Uhlenbeck process, we use the method described by Tang and Chen (2009). In this paper, a method to estimate the parameters for the Vasicek model is described using closed-form Maximum Likelihood Estimators. We can use this method since the Vasicek model is just the Ornstein-Uhlenbeck process with an additional parameter for the long-term mean. Since our long-term mean is 0 by definition, we can use this to simplify the formulas.

To find the closed-form Maximum Likelihood Estimators, we first have to define the likelihood function of the parameter vector (θ, σ). If we define $\phi(x)$ as the density function of a standard normal distribution $N(0,1)$, we have

$$L(\theta, \sigma^2) = \phi\left(\frac{1}{\sigma}\sqrt{2\theta}X_0\right)\prod_{t=1}^{n}\phi\left(\frac{1}{\sigma}\sqrt{2\theta(1-e^{-2\theta})^{-1}}\{X_t - X_{t-1}e^{-\theta}\}\right) \tag{3.2}$$

Using this density, we can derive the Maximum Likelihood Estimators. [2]

$$\begin{aligned}\hat{\theta} &= \ln(\hat{\beta}_1)\\ \hat{\sigma^2} &= \frac{2\hat{\theta}\hat{\beta}_2}{(1-\hat{\beta}_1^2)}\end{aligned} \tag{3.3}$$

Where we have

$$\begin{aligned}\hat{\beta}_1 &= \frac{\frac{1}{n}\sum_{i=1}^{n} X_i X_{i-1} - \frac{1}{n^2}\sum_{i=1}^{n} X_i \sum_{i=1}^{n} X_{i-1}}{\frac{1}{n}\sum_{i=1}^{n} X_{i-1}^2 - \frac{1}{n^2}(\sum_{i=1}^{n} X_{i-1})^2}\\ \hat{\beta}_2 &= \frac{1}{n}\sum_{i=1}^{n}\{X_i - \hat{\beta}_1 X_{i-1}\}^2\end{aligned} \tag{3.4}$$

Using these estimates, it is straightforward to first generate a time series of scaling factors, after which these are converted into influent values using Equation 3.1. Examples of the generated scaling factors and influent values can be found in Section 6.1.

3.2 Simulation of rainwater

The second influent stream that we consider is the simulation of rainwater. Of course, the effect of rainwater on influent is vastly different from the effect of wastewater. As we have seen in the previous subsection, wastewater causes a constant stream of influent to the purification facility. Rainwater on the other hand is only relevant for brief periods of time, as it only rains a small fraction of the time. However, when it rains, this has a large impact on influent, which leads us to the required property of rainwater modeling.

[2] For a full derivation, we refer to Tang and Chen (2009).

This rainwater property is that although rain does not occur very often, it can have a large impact on influent. Most rain showers lead to an average influent below or around the maximum pump capacity, so it is no problem to cope with these. However, some intense rain showers generate more water than the pump can handle, up to 10 times the maximum capacity in the most extreme case. This means that in some cases, overflow of the system is unavoidable, but we want to limit the number of occurrences. This can be achieved by emptying the storage tanks before a long and intense rain shower.

To make our rainfall simulator, we draw inspiration from Acreman (1990). However, there are a few key differences between our situation and the situation from the paper by Acreman (1990). The first and most prominent one is that we work with data at a 5-minute level, whereas the original paper works with hourly data. This affects how strong subsequent observations are related, the duration of dry and rain spells, and potentially which distributions are most suitable during modeling.

The second difference is that whereas the paper by Acreman (1990) had the aim to make a model as realistic as possible, our aim is to create a model with all the desired properties. This implies for example that we do not take into account seasonal effects, as this is not of importance in our desired outcome. The seasonal effects will for example shift the daily pattern on weekends since people wake up at a later time. However, the general shape of the daily pattern does not change, and it is this general shape that we are interested in. In our agents can cope with the daily pattern we observe on weekdays the same is true for the daily pattern on weekends, so there is no need to model them both.

What is similar to the paper by Acreman (1990) is the general structure of our rainfall simulation. We make use of the alternating renewal model, which sees the rainfall process as alternating dry and rainy periods of varying duration. When modeling event intensities we divert somewhat from the approach in the paper, as the approach used there does not extend well to high-frequency data. The same holds for modeling event profiles.

In the coming subsections, we look into these aspects in more detail to see how we can adequately model them. When these three steps are combined, we have a full rainfall model with the desired properties that are stated at the beginning of this subsection.

3.2.1 Modelling event durations

The first step in simulating rainwater is to model the alternating pattern between dry and rain spells. We define a rain spell as one or more consecutive time periods with non-zero precipitation. Of course, dry spells are defined as one or more consecutive time periods without precipitation.

To be able to model the event durations of dry and rain spells, we first need to find out which durations are common. To do this, we extract precipitation data at a 5 minute level for the years

2018 and 2019. This data is available through the Nationale Regenradar. From this dataset, we can extract the length of dry and rain spells, as well as the amount of rain per rain spell which we need in Section 3.2.2. For now, we are mainly interested in the duration of the dry and rain spells.

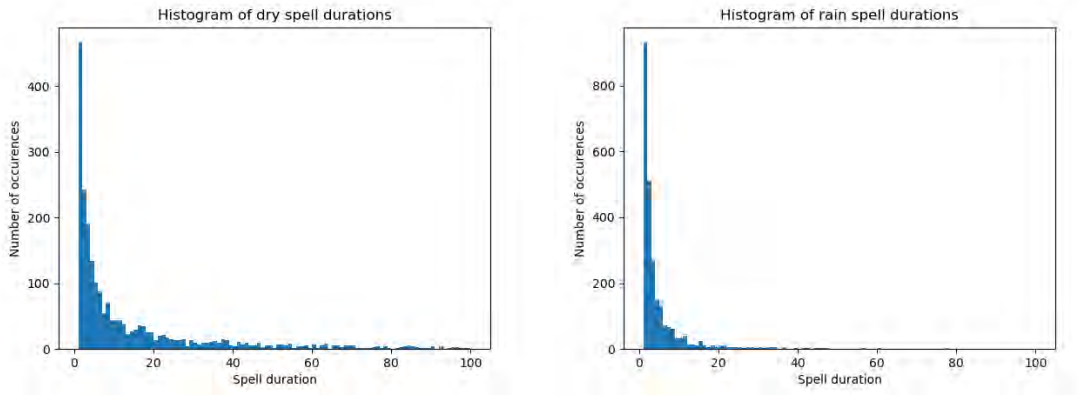

Figure 4: Histograms of dry and rain spell durations

In the figures above we can see histograms of the dry and rain spell durations. These durations are derived from the precipitation dataset at a 5 minute level, using data between the 1st of January 2018 and the 31st of December 2019.

In Figure 4, we can observe histograms for the duration of dry and rainy periods. We have cut off the graph at a spell duration of 100 time periods, which is roughly 8 hours. However, especially for dry spells, spells longer than 100 time periods occur as well with the longest being 4562 periods (nearly 16 days). When we look at the plots we can see that dry spells are usually longer than rain spells. Given that it does not rain most of the time, this makes sense. Dry spells generally last longer than rain spells, with their average durations being equal to 78.13 and 4.98 time periods respectively.

Another observation that can be made is that we see a lot of dry and rain spells with durations close to 0. Because the data that we use has a relatively high frequency, a short break within a rain shower is seen as a new dry spell. It could be the case that we see a high number of short dry spells right after each other, interrupted by long dry spells. If this is the case, the length of the previous dry (and/or rain) spell contains information about the expected length of the next spell and hence, should be taken into account. We investigate this by looking at the first-lag autocorrelation of the durations using two different correlation coefficients, for both the dry and rain spells. These two correlation coefficients are the Pearson correlation coefficient and Spearman's rank correlation coefficient. We find that for both correlation coefficients and weather types, correlations are significant but weak with all four correlation coefficients ranging between 0.05 and 0.15. For this reason, it is decided to treat the event durations as independent of one another.

The next step in modeling event durations is to fit a distribution that we can later use to simulate from. Just like the paper by Acreman (1990), we consider three distributions for this. These

distributions are the exponential, gamma, and log-normal distribution. All distributions have the property that they are non-negative, so they seem like good candidates to simulate durations. The only drawback is that these distributions are continuous, while the durations are discrete. To solve this problem, we can just round up all numbers during our simulation. This works, but it means that on average our simulated event length is 0.5 higher than the original event length[3]. To account for this, we subtract 0.5 from all observed durations. This in turn lowers the average duration from our simulations by 0.5, hence canceling out the bias we introduce by rounding up.

The first distribution we consider is the exponential distribution, which has a single parameter λ. We use the Maximum Likelihood Estimator of this parameter to fit this distribution. Full derivations of the Maximum Likelihood Estimators of all three distributions can be found in Appendix A. The Maximum Likelihood Estimator of λ can be seen in Equation 3.5.

$$\hat{\lambda} = \frac{n}{\sum_{i=1}^{n} x_i} \tag{3.5}$$

Where x_i is the duration of event i and n is the number of events in our dataset. The second distribution that we consider is the gamma distribution. This distribution has two parameters; a shape parameter κ and a scale parameter θ. We can see the Maximum Likelihood Estimator of these parameters in Equation 3.6.

$$\begin{aligned} \hat{\kappa} &= \frac{n \sum_{i=1}^{n} x_i}{n \sum_{i=1}^{n} x_i \ln(x_i) - \sum_{i=1}^{n} \ln(x_i) \sum_{i=1}^{n} x_i} \\ \hat{\theta} &= \frac{1}{n^2} \left(n \sum_{i=1}^{n} x_i \ln(x_i) - \sum_{i=1}^{n} \ln(x_i) \sum_{i=1}^{n} x_i \right) \end{aligned} \tag{3.6}$$

The final distribution to consider is the log-normal distribution. Just like the gamma distribution, this distribution has two parameters. The parameter μ determines the mean of the distribution, whereas σ determines the standard deviation of the distribution. We can see the Maximum Likelihood Estimators of this distribution in Equation 3.7

$$\begin{aligned} \hat{\mu} &= \frac{1}{n} \sum_{k} \ln(x_k) \\ \hat{\sigma^2} &= \frac{1}{n} \sum_{k} (\ln(x_k) - \hat{\mu})^2 \end{aligned} \tag{3.7}$$

We fit these distributions on the extracted dry and rain spells. For every distribution, we can calculate goodness of fit measures like the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC). These measures of fit can be seen in Equation 3.8. In these equations,

[3]The effect of the rounding is uniform between 0 and 1, so on average the simulated event is 0.5 time units longer

k denotes the number of model parameters, n represents the number of observations, and $\mathcal{L}(X|\hat{\boldsymbol{\theta}})$ denotes the log-likelihood of the observation vector X given parameter vector $\hat{\boldsymbol{\theta}}$.

$$\begin{aligned} AIC &= 2k - 2\mathcal{L}(X|\hat{\boldsymbol{\theta}}) \\ BIC &= \ln(n)k - 2\mathcal{L}(X|\hat{\boldsymbol{\theta}}) \end{aligned} \tag{3.8}$$

For the selection of the distribution, we choose the one with the best score on these performance measures. We do this independently for both dry spells and wet spells to have the best possible overall fit. In both cases the log-normal distribution performs best, so this is the distribution that is used to simulate our event lengths.

3.2.2 Modelling event intensities

Now that we have a framework to simulate event durations, we can move on to modeling event intensities. The rain intensity is defined as the average precipitation during a rain shower, measured in millimeters per 5 minutes. Defining rain intensity in this way makes it independent of the duration of the rain event, in contrary to the total rainfall of a rain event.

The strategy required to model event intensities is similar to the strategy to model event durations. We start with inspecting the intensities from the Nationale Regenradar dataset which was also used to extract the event durations. The calculated intensities are shown in Figure 5. From this figure, we can see that most rain events have a low intensity. Furthermore, we see a shape similar to the histograms in Section 3.2.1.

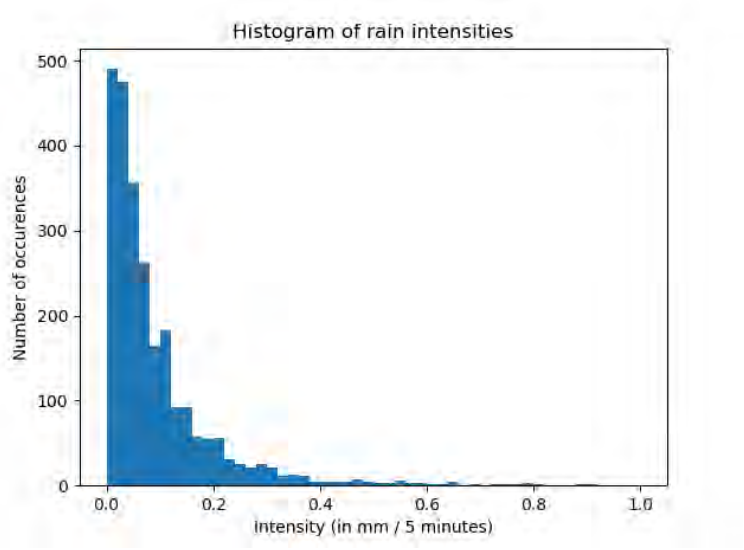

Figure 5: Histogram of rain event intensities

In the figure above we can see a histogram of the observed rain intensity. These intensities are derived from the precipitation dataset at a 5 minute level, using data between the 1st of January 2018 and the 31st of December 2019.

More interesting is the relation between rain spell duration and rain intensity. In Figure 6 we can see the relation between these two variables. What is noteworthy in this plot is that all the extremely intense rain events have a low duration. This can be explained by the following. During events with a high duration, it is much more likely to see both periods of high and periods of low intensity. When we calculate the average intensity throughout the whole rain shower, the periods of high intensity are offset by periods with low intensity, hence creating an event that is overall less extreme. The same reasoning could be applied to events with a low intensity. It is unlikely for high duration events to only have very light rain. Such events are typically broken up by periods of no rain, or periods of higher rain intensity. This idea is supported by the right-hand side of Figure 6, which shows the duration versus intensity relation after the natural logarithm has been applied. The fitted intensity is just the average over all observations. We can clearly see that the variance of the intensities is a lot bigger at low duration events compared to high duration events.

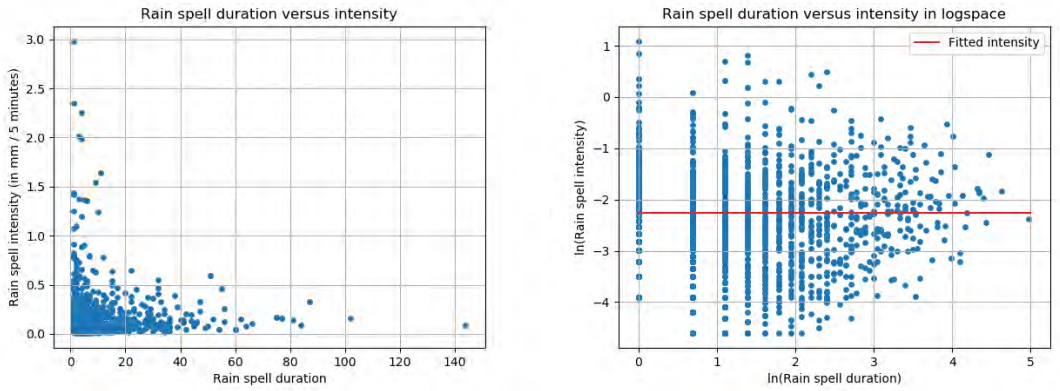

Figure 6: Relation between rain event duration and intensity

In the figures above we can see scatter plots of the observed rain duration versus intensity. On the left-hand side we see the actual values, while on the right-hand side the values have been transformed by applying the natural logarithm. These durations and intensities are derived from the precipitation dataset at a 5 minute level, using data between the 1st of January 2018 and the 31st of December 2019.

The observed relation between the variance of the intensity and the duration of the event is something that could be taken into account during modeling. However, there is no clear benefit in doing so. Ignoring this fact increases the probability of an event with a high duration/intensity. This type of event learns the algorithm to anticipate rain showers since it wants to avoid overflow of the system at all costs. Making them more rare means that we need more training steps to encounter these situations, slowing down learning. Making these events more common (through modeling durations and intensities independent) speeds up learning, so we use this in our research.

Since we model the rain intensity independent of the rain event duration we can just apply the same methodology as in Section 3.2.1. This means that we consider the exponential, gamma, and log-normal distributions and choose the one with the best fit. Since the rain intensity is a continuous variable, we do not have to make the adjustment we previously made to account for rounding. Again, the log-normal distribution performs best, so we use this distribution to simulate rain intensities in the remainder of this thesis.

3.2.3 Modelling event profiles

To model the event profiles, we base our methodology on the approach as taken by Brigandi and Aronica (2019). Under this approach, mass curves are simulated based on a fitted beta distribution for every point of the curve.

The first step of the approach by Brigandi and Aronica (2019) is to extract all rain events and convert them to mass curves of equal length. To be in line with the work by Brigandi and Aronica (2019), we choose to work with 25 steps. This converting step is just a linear transformation of the original rain shower, which is the rainfall amount per time step. If the original event is shorter than 25 time periods, each intensity is spread out over multiple steps. If the original event is longer than 25 time periods multiple intensities are merged into a single step. We can see this transformation into mass curves in Equation 3.9. In this equation, r is the original rain event and x is the steps for which we calculate the fraction of total rain fallen. The length of event r is denoted by $|r|$ and the amount of rainfall in period i is written as r_i, starting from index 0. This formula sums the rainfall until a given fraction of time, assuming constant rainfall within each time period. For example, if we calculate the amount of rain fallen at 50% of time passed in an event of 3 time periods, we would sum the first time period and half of the second time period and then divide this by the total amount of rainfall in the event.

$$f_r(x) = \sum_{0=1}^{|r|-1} \max(0, \min(1, |r| * \frac{x}{25} - i) * r_i \Big/ \sum_{j=0}^{|r|-1} r_j \qquad \forall\, x \in (1,\ 2,\ \ldots,\ 24,\ 25) \tag{3.9}$$

This converting step gives the observed dimensionless mass curves. The mass curves show how the rainfall is increasing as a function of time. In Figure 7 a collection of these mass curves is shown. From this graph, we can see that there is some variation in rain intensities within rain events. It can for example be seen that after 40% of the event duration has passed, the fraction of total rain fallen lies somewhere between 10% and 70%. We further observe that we generally have less than 20% of all rainfall in the first/last 20% of the time. This indicates that most rain events have a relatively slow start and end, with a more intense period in between. The goal is to produce similar behavior with our simulation engine.

The next step is to calculate the fraction between the precipitation registered in a period and the remaining precipitation from that time period. This gives the desired quantities to estimate the parameters of the beta distribution. In Figure 8 we have plotted the average of these proportions for every time point. We can see that in the early stages of a rain event, each time period only represents a small fraction of all the remaining precipitation. As time progresses, this average proportion gets closer to 1 because of the decreasing number of time periods that are left in the event. The final proportion is equal to 1 since there is no precipitation remaining after this time period.

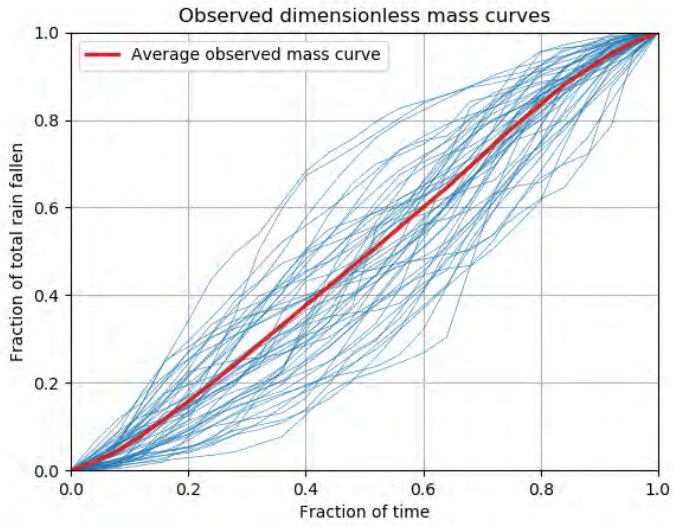

Figure 7: Sample of 50 observed mass curves

In the plot above we can observe 50 observed mass curves along with the average mass curve, as calculated over all observed events. Events are derived from the precipitation dataset at a 5 minute level, using data between the 1st of January 2018 and the 31st of December 2019.

Using the proportions between the precipitation registered in a period and the remaining precipitation from that time period we can estimate the beta distribution parameters. We do this using a method of moments estimator, given that there are no closed-form formulas for the Maximum Likelihood Estimator of the beta distribution (Owen, 2008). These estimators are derived from the analytical equations for the mean and variance of the beta distribution, which are in turn approximated by the sample mean and variance. We can see the parameter estimates in Equation 3.10. A full derivation of these parameter estimates can be found in Appendix B.

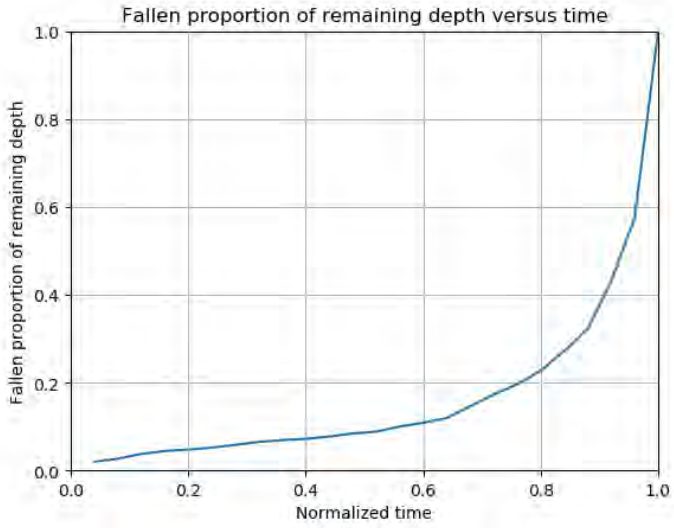

Figure 8: Average proportion of remaining event depth fallen

In the figure above, the average proportion of remaining event depth, registered in a specific time period is shown. These proportions are derived from the precipitation dataset at a 5 minute level, using data between the 1st of January 2018 and the 31st of December 2019.

$$\hat{\alpha} = \bar{x}\left(\frac{\bar{x}(1-\bar{x})}{s^2} - 1\right)$$
$$\hat{\beta} = (1-\bar{x})\left(\frac{\bar{x}(1-\bar{x})}{s^2} - 1\right) \tag{3.10}$$

In the equations above we make use of the sample mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and sample variance $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$. The parameter estimates are calculated for all 25 steps. An overview of the parameter estimates per time period can be seen in Appendix C.

Using these parameter estimates, we can now simulate standardized mass curves. However, in our simulation framework we want to be able to simulate mass curves for all possible event durations and intensities. To do this, we use the following steps during simulation. The first step is to calculate the total rainfall of the event as the product of the duration and intensity. Next, we use the parameter estimates to generate the fraction of remaining precipitation fallen in each of the 25 steps, which are then converted to millimeters using the calculated total rainfall. Finally, we adjust the number of time periods in our simulated event using the same linear transformation as before. This gives a simulated event with the desired length and intensity and whose event profile is based on all observed event profiles.

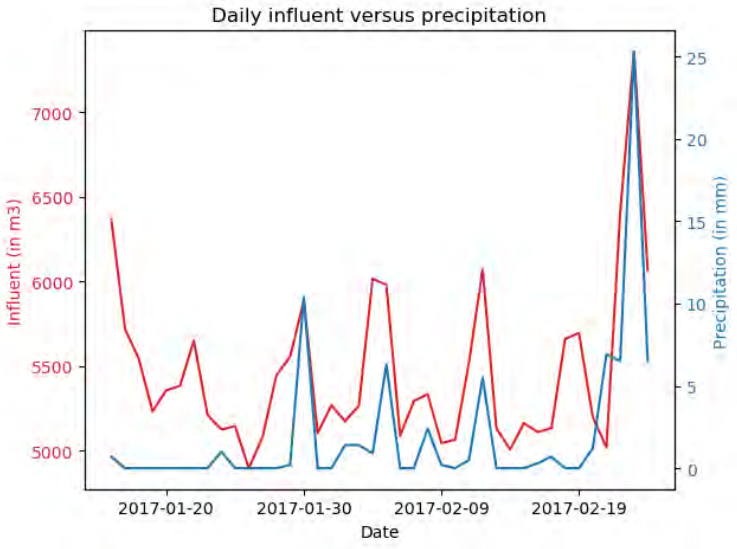

Figure 9: Relation between daily influent and precipitation

In the figure above we can see two time series; the daily influent and the daily precipitation. Daily influent values are calculated by aggregating the 5 minute influent dataset. Precipitation values are taken from the KNMI dataset.

Now that we have our rain event, we still need to convert this into influent values. Based on Figure 9, we can see that there is a strong relation between daily influent and daily precipitation. The reason that we work with daily data instead of 5-minute data is that rainwater does not arrive at the purification facility immediately. This makes it difficult to find the relation between rainfall and influent at a 5-minute level since the precipitation affects multiple future influent values. Working with daily data circumvents this problem, as most of the additional influent caused by

precipitation is registered on the same day. Finding this relation is done by means of a simple linear regression, for which the formula is given in Equation 3.11, where a represents the intercept and b the slope of the regression line.

$$\text{Influent}_t = a + b\ \text{Rain}_t \tag{3.11}$$

3.2.4 Simulation process

In subsection 3.1 and the previous part of this subsection we have described all the elements needed for our simulation framework. All that is left to do to create our single-agent, stochastic environment is to combine these elements into a single algorithm. We can see the pseudocode of this in Algorithm 3. In this algorithm, we can see that we generate dry and rain events in an alternating fashion. We do this until we reached our desired simulation length, which can be reached in somewhere between 1 and *desired length* iterations due to the stochastic duration of our events. Of course, when we generate a rain event, we also need to simulate the intensity and profile.

Algorithm 3: Influent simulation

```
initialization
while simulation length < desired length do
    if next rain = false then
        simulate dry event duration
        next rain = true
    else
        simulate rain event duration
        simulate rain event intensity
        simulate rain profile
        next rain = false
    simulation length += event duration
    store event
```

During our simulation, we make use of some assumptions. Some of these assumptions have already been mentioned, like the fact that we model event durations and intensities independently of each other. Furthermore, an additional assumption is made that the influent from rainwater arrives at the purification facility in the same time period that we register the precipitation. While this assumption does not hold in practice, it helps us to train more efficiently. The reason for this is that under the assumed situation, rain arrives at the purification facility faster. This requires the algorithm to show more anticipating behavior in order to get a high reward, which is something that we want to stimulate.

3.3 Influent forecast approximation

The next step in creating a realistic simulation environment is adding a forecast of the two influent streams that we just described. As mentioned in the introduction, having a forecast of future influent helps to anticipate on future high influent events. To do this, we split the influent forecast into two parts. The first part concerns the forecast of the wastewater, whereas the second part looks at the forecast of the rainfall. This makes sense as these two influent streams are unrelated, so modeling them as one would make the process unnecessarily complex. These two forecasts are joined together to make an overall forecast for total influent.

For this thesis, we work with a forecast window of one hour. There are several reasons to do so. The first one is the amount of water that can be processed in that time window. As we will see in Section 3.5, agents are encouraged to keep water levels below 10% of their maximum capacity. In times without rainfall, agents should be able to keep water levels below this threshold. In case a rain spell is approaching, the one hour time window is enough for the agent to completely empty the storage tank and hence build up capacity for future influent. The second reason is that having a longer forecast window would decrease the precision of our forecast. This would make it harder for agents to select good actions based on the forecasted influent as the forecast contains more noise.

3.3.1 Wastewater forecast approximation

As mentioned in the introduction of this subsection, we first focus on the approximation of wastewater. We know from Section 3.1 that this process is simulated using an Ornstein-Uhlenbeck process with a mean reversion parameter θ and a noise parameter σ^2. We use this knowledge to make a forecast for the wastewater influent.

To find the forecasted influent in the next hour, we go back to the model definition as stated in Equation 3.1. By taking the expectation of this definition and doing some rewriting, we can find the expected value of the next observation in terms of known quantities. This can be seen in the equation below.

$$\begin{aligned}
\mathbb{E}(Y_{t+1}) &= \mathbb{E}(e^{X_{t+1}} D_{t+1}) \\
&= \mathbb{E}(e^{X_t + dX_t} D_{t+1}) \\
&= \mathbb{E}(e^{X_t - \theta X_t + \sigma dW_t} D_{t+1}) \\
&= \mathbb{E}(e^{(1-\theta)X_t + \sigma dW_t} D_{t+1}) \\
&= \mathbb{E}(e^{(1-\theta)X_t} D_{t+1}) + \mathbb{E}(e^{\sigma dW_t} D_{t+1}) \\
&= e^{(1-\theta)X_t} D_{t+1}
\end{aligned} \tag{3.12}$$

It can easily be seen that this formula can be extended to an n-step ahead forecast by adjusting the baseline rate D_{t+1} by D_{t+n} and raising the $(1-\theta)$ to the power n. This gives us the following generic formula for an n-step ahead forecast.

$$\mathbb{E}(Y_{t+n}) = e^{(1-\theta)^n X_t} D_{t+n} \tag{3.13}$$

We can take this generalized formula to make a forecast for the next 12 time periods. These 12 forecasts are aggregated to create a single overall wastewater influent forecast for the next hour.

3.3.2 Rainwater forecast approximation

In the next subsection, we focus on approximating influent from rain events. We saw in the previous subsection that forecasting wastewater was fairly straightforward due to the high dependence on a known daily pattern and the relatively low level of noise in the observations. It is evident that the influent of rainwater has neither of these properties, so we have to look at alternative methods to make a forecast.

When we look at the literature of precipitation forecasting (or nowcasting as it is often called for time windows $<$ 24 hours), we see that all methods rely on geospatial information. This makes sense, as observing whether it rains in areas closeby seems like a good predictor of rainfall in the area of interest. Most of the forecasting models make use of pixel data, on which they then use some type of predictive model to make the precipitation forecast. The predictive model that is used can for example be a Lagrangian dynamic model (Zahraei et al., 2012), a standard neural network (Rivolta et al., 2006) or an LSTM network (Shi et al., 2015, 2017).

In our case, we have to work without pixel data since this type of data is not available to us. To correct for this, we have downloaded precipitation data from 9 different weather stations over a period of 10 years. The station of interest is located in the center, while all the other stations are spread in a circle around the station of interest. This allows us to have information similar to that in the pixel data without actually using that specific data source. We complement the dataset by adding hourly observations of weather variables like humidity, wind speed/direction, atmospheric pressure, and temperature. This gives us a dataset that should be able to produce a good predictive model for our simulation.

However, before we can proceed to create such a predictive model, we still have to impute the weather variables (since these are hourly) and create some features for the model to use. For the weather variables, there are two types of imputation that we use. The first type is linear interpolation, which is used for numeric features where this option makes sense. Examples of these features are temperature and atmospheric pressure. The other type of imputation is simply taking the last known observation. This is used for categorical variables and variables where linear

interpolation does not always make sense. An example of this last type is the wind direction, which is in degrees and can produce weird behavior if the wind direction is around the 0 / 360-degree mark. For the feature engineering, we have mainly focused on the precipitation dataset. For all locations, features are made for the average rainfall in the last n time periods, where several values for n are used ranging between 2 and 24 time periods (equivalent to 10 to 120 minutes). To account for the lack of a forecast in the locations not directly of interest, the same features are made looking forward several time periods. This should give the predictive model enough information to make a decent precipitation forecast.

As our predictive model, we make use of a gradient boosting regressor. This method was introduced by Friedman (2001) as a way to increase the performance of ordinary regression trees. It works by fitting several trees in sequential order, each one fitted on the model residuals created by all the trees before it. The reason why this method is chosen is because it often achieves good performance without the need to optimize the model parameters a lot. Given that it is merely a modeling step of our environment, using such a model is more favorable than developing a complex method with slightly better performance and features. The result of this gradient boosting regressor can be seen in Figure 10.

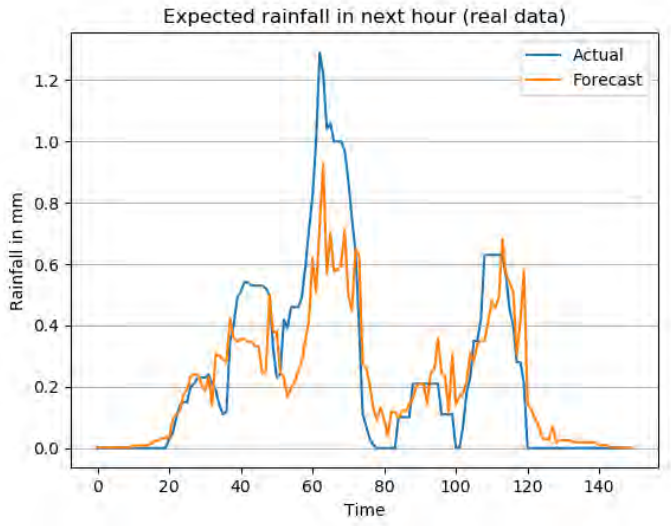

Figure 10: Actual versus forecasted rainwater

In the figure above we can observe the actual and forecasted rainwater influent over a one hour time window, based on the gradient boosting regressor. On the horizontal axis, the time period can be seen whereas the vertical axis shows the rainfall in millimeter over the next hour.

In order to make a realistic simulation environment, we have to produce forecasts with characteristics similar to the one shown in Figure 10. Since we do not have the same dataset available for simulated rainfall events, we have to come up with an alternative method to create such a forecast. From the results in Figure 10 we can deduce several desired characteristics. The first one is that the precision of the model is dependent on the total rainfall in the next hour. From the plot, we can easily see that the error is often much smaller in periods of low expected rainfall compared

to periods of high expected rainfall. The second characteristic is that our prediction shows some kind of autoregressive behavior. This means that when our forecast is too low at a given moment in time, we are likely to see a low forecast in the next time period as well. This makes sense as the windows of two successive time periods overlap for more than 90%.

The first characteristic we deal with is the dependence between model accuracy and rainfall in the next hour. To cope with this feature we split our dataset into several buckets based on the actual amount of rainfall in the next hour. Bucket boundaries are determined in such a way that each bucket is guaranteed to have at least 1000 observations. Grouping observations based on the actual amount of rainfall gives us a good understanding of the typical forecast the gradient boosting regressor would make. Examples of the forecasts made for two buckets are shown in Figure 11. We can see that the shape of the distribution is dependent on the actual amount of rainfall. In the case where we do not observe a lot of rain in the next hour (visible in the left pane), most of our forecasts are centered around 0. In cases where a little bit more rain is expected in the next hour (approximately 0.6 millimeters, shown in the right pane) the distribution looks more like a normal distribution centered around the actual rainfall amount.

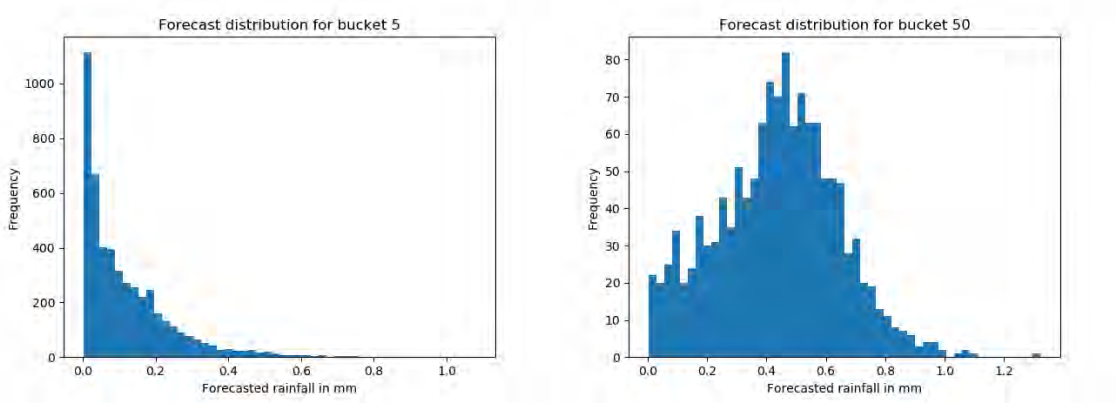

Figure 11: Rainfall forecast distribution for 2 different buckets

In the figures above we observe histograms of the rainfall forecast in the next hour for two different buckets. On the left-hand side, we see a bucket with a low amount of rainfall whereas the bucket on the right has a higher amount of rainfall in the next hour. These forecasts were made by the gradient boosting regressor using data between the 1st of January 2011 and the 31st of December 2019.

For every bucket that is created we estimate the parameters of the log-normal distribution. We use this distribution since it only takes on positive values, works with different shapes of the distribution, and since it works best to model rainfall, as we have seen in Section 3.2. As discussed before, the log-normal distribution has two parameters: μ and σ. The μ parameter determines the mean of the distribution, which helps to make higher forecasts in buckets where more rainfall is registered. The σ parameter determines the variance of the distribution and is able to capture the fact that we see larger errors in situations where more rain is expected in the next hour.

Using the estimated distribution per bucket, we can already make a first version of the rainwater forecast simulator. To do this, we start by taking the simulated rainwater influent using our

approach as described in Section 3.2. Using these simulated values we can calculate the rainfall in the next hour for every time period and match this with the correct bucket. A forecast can be simulated using the parameters of that respective bucket. The problem with this approach is that it does not have the autoregressive characteristic that we have described before.

To incorporate the autoregressive characteristic of the rainwater forecast we introduce one final step. The forecast as described in the previous paragraph is taken as a baseline forecast. To this baseline forecast, simple exponential smoothing is applied to achieve the desired autoregressive characteristic. The formula for this can be seen below where X_t is our smoothed forecast at time t, λ is the smoothing parameter, and Y_t is the baseline forecast at time t. The λ parameter is estimated by minimizing the mean squared error of the forecast and is approximately equal to 0.4 in our case.

$$X_t = \lambda Y_t + (1 - \lambda) X_{t-1} \tag{3.14}$$

To summarize, simulating our precipitation forecast consists of four steps. The first step is to create a rainfall series using the methodology from Section 3.2. After this, we calculate the amount of rainfall in the next hour for every time period. Based on the rainfall in the next hour, we determine the bucket and simulate our forecast using the fitted log-normal distribution. Finally, we apply exponential smoothing to our forecast series to add the autocorrelation property.

3.4 Multi-agent modelling

In the previous subsections, we have introduced environments that stimulate behaviors like buffering (to cope with the daily influent pattern) and anticipation (to deal with future influent peaks). This leaves us with one goal remaining, which is coordination between different agents. In this section, we describe which adjustments are made to the environment in order to create a realistic multi-agent setting.

First of all, all agents have their own set of location parameters. These parameters control the pump and storage tank of each location, as well as the baseline wastewater pattern and the amount of influent each millimeter of rain produces. All these factors are based on input given by waterschap Rijnland and differ for each agent. The advantage of this approach is that we can easily add more agents to our problem. All it requires to do this is a new set of parameters which we estimate based on input from the waterschap.

What the agents share is the scaling of the wastewater pattern, the rainfall simulator, and the forecast of the wastewater and rainwater. When we combine these shared components of our simulator with the agent specific parameters we get an influent series that is unique to every agent.

3.5 Reward function

The final element of our environment is the reward function. The environment that we have created so far will not produce any desired behavior without a good reward function since this element determines which policy is deemed optimal. It is often hard to reason what the optimal behavior is under a given reward function, so the reward function presented here is based on trial and error.

Since we have multiple characteristics that we would like to see from our agents, our reward function consists of a few elements that are joined together. Some of these parts are based on the behavior of single agents, while other parts are based on the joint action of all agents. Of course, the joint action is only of importance when we are dealing with two or more agents.

The first element of the reward function is based on the volume in the storage tank. This element is awarded to each agent individually. The goal is to keep the volume in the storage tank relatively low, so the agent receives a reward of 3 if the volume is above 0 and below 10% of the maximum volume[4]. Of course, it is not desirable to let the pump running if the tank is already empty, so no volume at all is penalized with a negative reward (so a penalty) of -3. For storage tank volumes over 10 % of the maximum capacity, a penalty is given which increases linearly between 0 and -3 depending on the volume present in the tank. The reason that this penalty increases linearly is to motivate the agent to reduce the tank volume even if it exceeds the 10 % threshold.

The second element is given both to each agent individually as well as to the joint action of all agents and depends on the chosen action. We reward the agent(s) if an action is taken other than turning the pump(s) completely off. Furthermore, the reward is largest when the minimum possible action is taken and decreases quadratically to 0 between the minimum and maximum possible action the agent(s) can take. This reward structure is shown in Figure 12.

The reason why such a reward structure is chosen is to stimulate the spreading of influent throughout the day. As mentioned in the introduction, spreading influent over the day as much as possible is desirable from the perspective of the waterschap. This type of reward stimulates actions that are as constant as possible due to the concave shape of the reward function. Together with the flexibility incorporated in the first element of the reward function (the reward is constant between 0 and 10% of the maximum storage capacity), this should lead to some buffering by the agents. Letting the pumps running during the night at a low rate ensures that the agent is getting rewards from the second element of the reward function while staying within the margin provided by the first element. This buffer can be replenished during the day when influent is higher.

[4]Please note that it is not the absolute values of the reward function that are of importance. It is the proportions between the different elements in the reward function that are of relevance (i.e. multiplying all rewards by some constant does not change the optimal behavior of the agent).

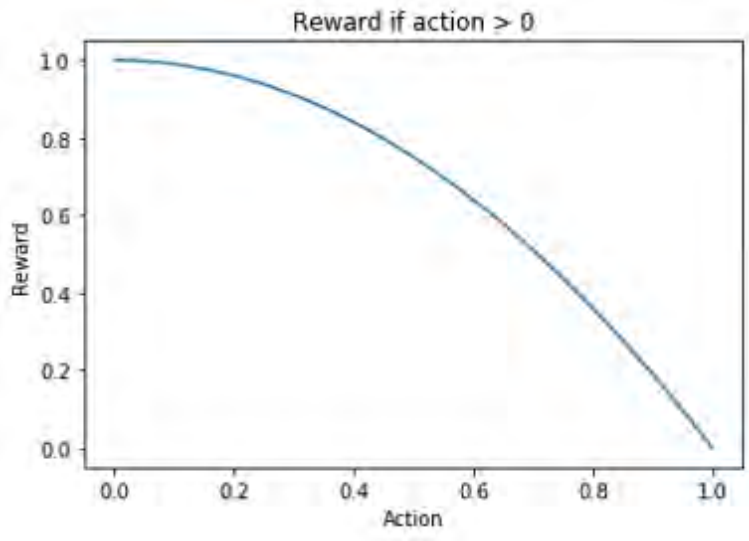

Figure 12: Reward when chosen action > 0

In the figure above we can see the reward the agent(s) receives if it chooses an action other than turning the pump completely off. On the horizontal axis the (joint) action can be seen where 0 corresponds to the minimum possible action and 1 corresponds to the maximum possible action. On the vertical axis the reward is shown.

When we look at the other desired behavioral characteristics we can see that these are covered by this reward structure as well. It can easily be seen that overflow of one of the storage tanks is sub-optimal since this yields the lowest possible reward in the first element of the reward function. Anticipation on rain showers is rewarded under this reward function in two different ways. First of all, agents try to avoid going over 10 % of their storage tank capacity since this prevents them from getting the reward of 3. Secondly, it is beneficial for the agent to spread all influent as evenly as possible. When the agent recognizes a rain shower is approaching the best action is to start processing more water before the rain shower arrives.

4 Agent Design

In the previous section we have discussed the environment, which was the first major methodological aspect. The second major methodological aspect that we discuss are the agents that operate in the environment.

In this section, we discuss several different agents with varying degrees of complexity. Each of the following subsections discusses a different type of agent. We start with an agent that follows the current rule-based policy. This gives us a good indication of the required performance level to be an improvement over the rule-based method. We can see this agent as some kind of baseline performance. After this, we look into value iteration which was discussed in Section 2. Finally, we make use of several deep reinforcement learning methods to see if these give an improvement over the other methods.

The agent that makes use of tabular methods (i.e. the agent that uses value iteration) is not able to cope with large state spaces. This becomes a problem when we introduce the influent forecast or multi-agent scenarios into our environment. For this agent, adaptations are made to enable the agent to work with scenarios like these.

4.1 Rule-based agent

The first agent that is created follows the policy that is currently used at the purification facility. This policy is not based on any kind of optimization, rather, it is based on a simple heuristic to determine the action in the next step. The advantage of this method is that it works with continuous state and action spaces and only needs three parameters to set up. This combination of flexibility and simplicity is probably the reason why it is used by waterschap Rijnland.

The three parameters that need to be set are the minimum initiation level (denoted I_{min}), maximum initiation level (denoted I_{max}), and termination level (denoted T). The initiation levels are the fractions of tank capacity at which the pump is activated at minimum and maximum capacity. The termination level is the fraction of the tank's capacity at which the pump is turned off. The model assumes that $T \leq I_{min} \leq I_{max}$.

The policy of the pump can be formulated as follows. When the water level in the tank is below the termination level T, the pump is off. Water then flows in, increasing the water level. When the minimum initiation level is reached, the pump is activated at minimum capacity. For water levels between the minimum and maximum initiation level, the action is increased linearly with the water level. Furthermore, we never decrease our action compared to the previous time period, unless we reach the termination level when the pump is turned off.

We can also represent this behavior more formally, which is done in Equation 4.1. In this equation, a_t represents the action of the pump at time t, which is scaled between 0 and 1. w_t is the water level at time t and a_{min} is the minimum action the pump can execute.

$$a_t(w_t, a_{t-1}) = \begin{cases} 0, & \text{if } 0 \leq w_t < T \\ a_{t-1}, & \text{if } T \leq w_t < I_{min} \\ \max\left(a_{min} + (1 - a_{min})\frac{w_t - I_{min}}{I_{max} - I_{min}}, a_{t-1}\right) & \text{if } I_{min} \leq w_t < I_{max} \\ 1 & \text{if } I_{max} \leq w_t \end{cases} \tag{4.1}$$

While the policy as described above is nice in terms of flexibility and simplicity, it can lead to some undesired behavior. First of all, the policy can overreact to small peaks in influent if these happen around the minimum initiation level. This small peak pushes up the action that is chosen due to the increased water level, which is then executed until the termination level is reached again. This is not ideal when we want to spread out influent as much as possible.

Another type of undesired behavior is the lack of anticipation this agent shows, for example when a rain shower is expected. It could be that the water level in the tank is right below the minimum initiation level. Under an anticipating policy, we would see that the tank is emptied in order to make room for the influent caused by the rain shower. This agent does not produce such behavior, leading to a higher risk of overflow of the system.

The same lack of anticipation comes into play when we want to correct for the daily influent pattern. In the ideal situation, we would spread out the influent as much as possible by building up a buffer during the day, which is released during the night. The rule-based agent is not able to produce such behavior.

4.2 Value iteration

In order to overcome the undesired properties the rule-based agent has, we make use of planning and reinforcement learning methods. As already outlined in Section 2, the difference between planning and reinforcement learning is that a planning model assumes that the dynamics of the environment are known, whereas a reinforcement learning model learns these dynamics during optimization. For this reason, value iteration falls into the class of planning methods.

In this subsection, we look at how value iteration can be applied to the different environments that have been created. Key in this is to model the dynamics of the environment and the expected reward the agent receives. Furthermore, we need to address the curse of dimensionality in a multi-agent setting. As already argued, the number of state-action combinations grows rapidly when we add more agents. Since we want our solution to be scalable, we have to adjust our method such that it can handle these larger state-spaces.

However, before we address this issue we first look into approximating the transition probabilities and rewards $p(s, r \mid s, a)$. To keep matters simple, we focus on a single agent first. We know that this water level depends on a number of factors. These factors are the water level in the previous period (state information, known), the desired effluent in the current period (action, known), and the influent in the current period (coming from our simulation model, unknown). Luckily, we can approximate the influent distribution using our simulation model from Section 3. To do this, we simulate a large number of observations from this model and convert this into probabilities. The result of this can be seen in Figure 13.

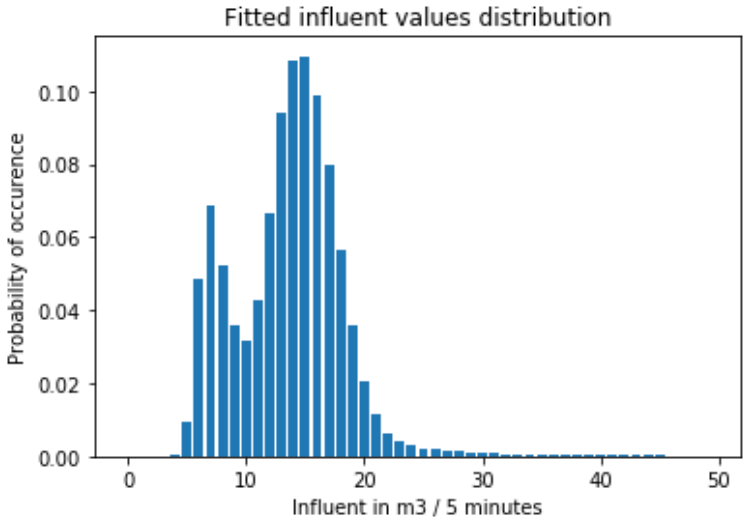

Figure 13: Approximated influent distribution

In the figure above we can observe an approximation of the influent distribution coming from our simulation model. The distribution is based on 2.5 million simulated observations, where every observation is rounded down to the nearest integer value. The daily cycle is not taken into account, so the shown distribution is averaged over the whole day.

Using this approximation, we can just calculate the transition probabilities for every given state, action, and next state. Furthermore, we can calculate the reward for every (s, a, s') combination so we store these as well. With this information, we can execute Algorithm 1 which gives us the optimal state-values and policy. One of the downsides of this approach is that it does not take into account information regarding the time of the day or expected influent in the next periods. Leaving out this information leads to sub-optimal actions when the expected influent is much different from what is shown in Figure 13.

To adapt this for the influent forecast version of the environment we have to make a few changes. We could just take the expected influent and use this as an additional state variable. However, this would increase the state space by such a large amount that we can no longer find the optimal policy within a reasonable time. This means we have to look for a method that still has the benefits of adding the influent forecast to the model, without making the problem computationally infeasible. It is recognized that the influent forecast is most useful in cases where future influent is high, often caused by rainwater.

The solution that is used is to make 10 buckets for the influent forecast and to apply standard value iteration (as we would without the influent forecast) on each bucket. These buckets are made based on the tail of the influent forecast distribution. The influent forecasts are collected in the same way as is done in Figure 13. The first nine buckets are created by taking 0.3 % of the right tail per bucket. So the 10th bucket runs between 99.7 % and 100 %, the bucket before that from 99.4 % and 99.7 % etc. The first bucket takes on all other forecast values and is therefore larger than all the other buckets. The reason why almost all time steps fall in the first bucket is that the agent does not need to change its behavior when there is no or little rainfall. Given that high rainfall events are rare, we only want to change our policy in the tail of the forecast distribution.

Now that we have calculated the boundaries of all the buckets, we are one step away from a working algorithm. We run the value iteration algorithm we have created for environments without a forecast on each bucket to give us a set of optimal policies. One of these policies is carried out, dependent on the influent forecast that is given to the agent.

4.2.1 Multi-agent adaptation

As argued in the introduction on value iteration, adding more agents quickly makes the problem intractable due to the large number of state-action combinations. We already discussed in Section 2 that using function approximation is one way to solve this problem, which is what we use in Section 4.3. For now, we make use of the separable nature of the reward function and state-space to find a good joint action.

We described in Section 3.5 that the total reward function is a combination of several smaller rewards. Some of these smaller rewards only look at individual agents, whereas other rewards look at all agents jointly. In this section, we make use of this reward structure to make a scalable multi-agent model based on value iteration.

The central idea is as follows: for every agent, we can look at his individual contribution to the total reward (i.e. the part of the reward function that only depends on that specific agent). Using this individual part of the reward function and the part of the state space the agent controls, we can do regular value iteration. This is no different than doing value iteration in the single-agent case. This results in a set of state-action values for every agent, which tells us each agent's preference for a given action in each state. At any point in time, we can observe the state and retrieve the accompanying state-action values for every agent.

We could just take the action that maximizes each agent's individual state-action value and use this as an approximation for a good joint action. However, this would neglect the potentially conflicting interests between individual and joint rewards. Hence, we need to find a balance between these two reward sources to guarantee good performance. We do this through a steepest ascent algorithm that is explained in the remainder of this section.

We start with the best action for each agent individually, which we call the greedy action. Since we have calculated the state-action values for every agent it is straightforward to find these actions. With this initial action, we can calculate the reward coming from the joint action. From this point, we look for joint actions that improve the collective reward without losing too much individual reward. To do this, we can calculate the total reward for every combination of actions. However, the action space grows exponentially with the number of agents so this option quickly becomes infeasible. To circumvent this problem we make two adjustments to our considered action space.

The first adjustment we make to our action space is to only consider changes to a single agent at a time. This limits the number of potential actions to consider from $|A|^n$ to $n \cdot |A|$ where $|A|$ is the number of actions each agent can take and n is the number of agents. The second adjustment is to only consider actions with a lower effluent than the greedy action. The reason for this is based on the shape of the collective reward function. In Section 3.5 we explained that the reward decreases quadratically as the total effluent goes up. When we take the greedy actions as our reference point, it can easily be seen that all actions with a higher effluent both decrease the state-action values and collective reward. Hence, these actions can never be better than the greedy actions and are therefore sub-optimal.

For all the actions in our action set we can calculate the change in total reward it would cause. This change in reward is shown in Equation 4.2. In this equation, $\nabla R_k(n, a)$ represents the difference in reward if agent n changes to action a. Furthermore, $A_k(n)$ is the action of agent n in iteration k of our action selection algorithm. $Q_n(s, a)$ is the state-action value of agent n in state s under action a and $R_{col}(A_k)$ is the collective reward when the joint action A_k is selected, which is a vector of all the individual actions $A_k(n)$.

$$\begin{aligned} \nabla R_k(n,a) &= \big[Q_n(s,a) - Q_n(s, A_k(n))\big] + \big[R_{col}(\tilde{A}_k) - R_{col}(A_k)\big] \\ \text{where } &\begin{cases} \tilde{A}_k(i) = a & \text{if } i = n \\ \tilde{A}_k(i) = A_{k-1}(i) & \text{otherwise} \end{cases} \end{aligned} \tag{4.2}$$

As already mentioned earlier in this section, we make use of a steepest ascent method in order to find a good joint action. This means that we normalize the differences in overall return by the change in effluent it causes, which is defined as $|A_k(n) - a|$. The reason for this is that without normalization, the algorithm is incentivized to decrease the total effluent quickly (because this is most beneficial to the collective reward). Without normalization, we would see that after a few iterations, some agents have low effluent values whereas others are still at their greedy action. Due to the concave shape of the collective reward function, there is a point where there is no single agent that can change its action in order to improve the total reward function. If this point is reached before all agents have adjusted their action, we end up with large differences in actions between agents (which is in most cases sub-optimal).

If we use the normalized differences in reward, we take away the incentive to drastically alter actions. Rather, due to the concave nature of the collective reward function, we even stimulate small steps. This causes the algorithm to pick the actions that cause a relatively low change in state-action values and creates the opportunity to gradually lower all actions instead of drastically lowering just a few. However, the algorithm is still able to lower an action with 2 or more steps if this is beneficial for the individual Q-values.

Combining the initialization with greedy actions, reward difference calculations, and normalization results in the following algorithm. In this algorithm, N represents the number of agents. It can be seen that we iteratively calculate the differences in reward and change the action of a single agent until there is no agent that can cause an increase in overall performance.

Algorithm 4: Action-selection algorithm

Observe $s \; \forall \; N$
$A_0(n) = \text{argmax}_a \, Q_n(s, a) \forall n$

while $A_k <> A_{k-1}$ **do**
  $A_{k-1} = A_k$
  for *each* $n \in N$ **do**
    for *each* $a < A_0(n)$ **do**
      $\tilde{A}_k(i) = a$ if $i = n, \qquad \tilde{A}_k(i) = A_{k-1}(i)$ otherwise
      $\nabla R_k(n, a) = \big[Q_n(s, a) - Q_n(s, A_k(n))\big] + \big[R_{col}(\tilde{A}_k) - R_{col}(A_k)\big]$
      $\nabla R_k(n, a) = \nabla R_k(n, a) \; / \; |A_k(n) - a|$

  if $\max_{n,a} \nabla R_k(n, a) > 0$ **then**
    $\bar{n}, \bar{a} = \text{argmax}_{n,a} \, \nabla R_k(n, a)$
    $A_k(\bar{n}) = \bar{a}$

It is important to note that the algorithm as described is not guaranteed to be optimal. There are two main reasons for this. The first reason is that while searching the algorithm can get stuck in local optima. Since we only consider changes in one action at a time, there are some joint actions that will never be evaluated, leaving open the possibility of a sub-optimal action being selected after convergence. The second reason is that we only take into account the direct effect of the collective reward function. Taking the best possible action in the current time step could lead to lower future collective rewards. However, since the collective reward only depends on the action (and not on the state) and we do take into account future rewards at an individual agent level, the negative effect of this is limited.

Finally, we consider how this approach scales with the number of agents. We first have to calculate the state-action values for every agent individually, which scales linearly with the number of agents. When looking at the action-selection mechanism, we can see two places where the number of agents is relevant. First of all the number of actions that are considered scales linearly with the number of agents. Secondly, having more agents is likely to increase the number of iterations of the algorithm.

Assuming that every agent needs on average the same number of updates in the action-selection algorithm, this scales linearly with the number of agents as well. Combined, the duration action-selection mechanism scales quadratically with the number of agents, which is an improvement compared to the exponential scaling we would see if we would perform value iteration on the joint state and action space.

4.3 Deep reinforcement learning

In this final subsection, we discuss how we have set up the deep reinforcement learning agents. In Section 2 we already discussed the different algorithms that are used and gave a general overview of how they work. All these algorithms are implemented in the Stable Baselines python package, which is based on research by OpenAI. This takes away the need to implement the algorithms from scratch, so in this subsection, we mainly focus on the parameter settings we use for each of these algorithms. In the ideal situation, we would try a lot of different parameter settings for each algorithm, however, this is computationally not feasible. Trying 2 or 3 different values for about 5 different parameters gives us anywhere between 32 and 243 different parameter combinations. Combining this with the 4 different environments, multiple runs for each parameter/environment combination, and several different deep reinforcement learning methods it would take months to complete. We therefore only focus on the most important parameters for each algorithm.

For all the algorithms that we discuss one parameter is fixed. This is the γ parameter, better known as the discount rate. We fix this parameter at 0.995 for all algorithms. The reason why this parameter is so high is that we put a lot of emphasis on the long run. It could be argued that we put as much emphasis on the long run as we do on the short run. However, most algorithms cannot cope with a discount factor equal to 1, so a value just below 1 is chosen.

4.3.1 Trust Region Policy Optimization (TRPO)

For the TRPO algorithm, there are 2 parameters that we tune in the search for good model performance. The first of these parameters is the number of time steps per batch. This batch size determines how often we update our network based on the advantage values we have calculated. The default setting for this parameter is to update every 500 time steps, which was used in the original paper by Schulman et al. (2015). For our agent, we try three different values; 5000, 10,000, and 20,000. The reason why these values are much higher than for the default agent is the large role of stochasticity in our environment. We want every update to be a good representation of all the situations an agent can encounter in our environment to make learning as stable as possible. Given that 500 time steps corresponds to a little under two days, we can easily see that such a batch size leads to batches that are far from being a fair representation of the actual environment.

The second parameter that we tune is related to the step size of the policy, denoted by λ. The reason why we choose this parameter is that taking smaller steps helps to average out the stochasticity

of the environment. For this parameter we use the values 0.98 and 0.99, creating a total of 6 parameter settings for this algorithm.

4.3.2 Advantage Actor Critic (A2C)

For the Advantage Actor Critic agent, we again have two parameters that are tuned. The first parameter is, just like with the TRPO algorithm, related to the number of steps before each policy update. However, in this model, the number of steps before each update is a lot smaller with a default value of 5. This is because the stochasticity that is normally averaged out by the batch size is now averaged out by having multiple agents. To stay in line with the TRPO algorithm, our batch sizes are around 10 times the normal size with parameter values of 25, 50, and 100 being tested.

The second parameter we adjust is called the entropy coefficient, which slows down the speed of convergence. This helps to avoid local optima during training, resulting in an overall better agent. The default value for this parameter is equal to 0.1. Given that we are more vulnerable to being stuck in a local optimum due to the high stochasticity in our environment we try a value of 0.1 and 0.2 for this parameter. Overall, we again have 6 total parameter settings that are tested for this algorithm.

4.3.3 Proximal Policy Optimization (PPO)

For the final algorithm we consider there are three parameters to be tuned. Furthermore, there are two different implementations of this algorithm in Stable Baselines, so we consider both of these versions. The parameters that are tuned are the same for both of these implementations.

Like in all the other algorithms, we again start with a parameter for the batch size. Based on some quick trial and error, it is found that the batch sizes we need to consider are 500, 1000, and 2000. This is again about the same order of magnitude bigger than the default value, which is 128.

The second parameter is called the clipping parameter. This parameter influences the step size of the algorithm and should help us to avoid being stuck in local optima. It is somewhat similar to the entropy coefficient parameter in the A2C agent and it takes the same values of 0.1 and 0.2. The final parameter is the same λ parameter we had in the TRPO algorithm. However, in the PPO algorithm, it often takes a lower value, so we test values of 0.95 and 0.98. This gives us a total of 12 parameter combinations to test for this algorithm.

5 Agent Evaluation

The final methodological aspect that is discussed is a framework to evaluate the performance of the agents consistently and reliably. In this section, we first talk about the experimental setup that is created to test agents, after which we discuss the various performance measures that are used along with a motivation of why these measures are relevant.

The experimental setup that is created consists of several steps. The first step is to set up the agent and the environment based on the parameters that we are testing. The next step is to set a seed that is used during training. This serves two purposes; to denote the run that we are doing and to make the outcome of the run reproducible. Denoting the run number is needed since we evaluate each agent/environment/parameter setup a total of 10 times. We do this to limit the impact of chance on the outcomes of our experiments. When this is all done, the next step is to train the agent either until convergence (for the value iteration agent) or until a fixed number of time steps is reached.

After training is completed, we can move on to evaluate our agents during test runs. We use a total of 1000 test runs which all last 1000 time periods (approximately 4 days) to get a good overview of how well our agent performs. We track the performance of the agents based on the seven performance measures that are discussed below. These test runs are the same for all agents to rule out the possibility that one agent performs better merely because it was evaluated based on easier runs. Doing 1000 runs of about 4 days each gives us a diversified set of scenarios for our agent to deal with, ranging from long dry spells to intense rain periods.

Agents are evaluated based on seven different measures that are all related to different aspects of the agent's behavior. Performing well on a single measure does not automatically imply that the agent shows desired behavior. Rather, to assess the performance of an agent all the measures should be evaluated simultaneously.

The first two measures are the mean and standard deviation of the rewards that are obtained during the 1000 evaluation episodes. The mean reward indicates how well the agent is able to align himself with the reward function we have created and could be seen as the most important measure we have. The standard deviation of the reward indicates if the agent can consistently achieve a high reward, or if it is designed to do well in certain situations. Of course, a low standard deviation is desirable since we want the agent to perform well in any given situation.

The third and fourth measures are related to the 10% threshold we have set in the first part of the reward function. It is explained in Section 3.5 that the agent receives a large positive reward if the volume in the storage tank is above 0 and below 10% of the maximum capacity. The measures compute the average number of times these thresholds are exceeded per run. We have split this

into two measures since having a tank volume which is too low has different consequences than having a tank volume which is too high.

The next measure is the standard deviation of the actions (ranging between 0 and 1) chosen by the agent. This measure shows us how well the agent is able to consistently choose the same action over the course of a run. While this is implicitly captured in the reward function (and therefore in the first measure), differences in the achieved reward can also be caused by other factors like exceeding the storage tank threshold. This measure allows us to isolate the agent's ability to choose the same action over and over again.

The final two measures relate to the agent's ability to use the storage tank as its buffer. We have previously discussed that an agent that performs as desired uses the storage tank to build up water during the day and deplete this buffer during the night. To measure this, we calculate the lower and upper 5% quantile of the storage tank after each evaluation run. If the agent does a good job of using the storage tank as its buffer, we will see that the lower quantile has a low value (since the tank is nearly empty towards the end of the night) and the upper quantile has a value close to the 10% threshold.

6 Results

In this section, we move on to discussing the results of our research. We do this in three different subsections that all serve their distinct purposes. We start by discussing the quality of our simulation environment in Section 6.1. We do this by making comparisons with real-world data and showing some examples of the data generated by our simulators. Section 6.2 evaluates the performance of our agents in these environments using the performance measures discussed in Section 5. Finally, we finish this section by performing a sensitivity analysis on some key environment parameters to show the generalizability of our method in Section 6.3.

6.1 Environment Design

In this section, we discuss the outcome of our simulation environment following the same structure as we did in Section 3. This means that we start by discussing the simulation of wastewater, after which we move on to rainwater. Finally, we discuss making a forecast for these two influent streams.

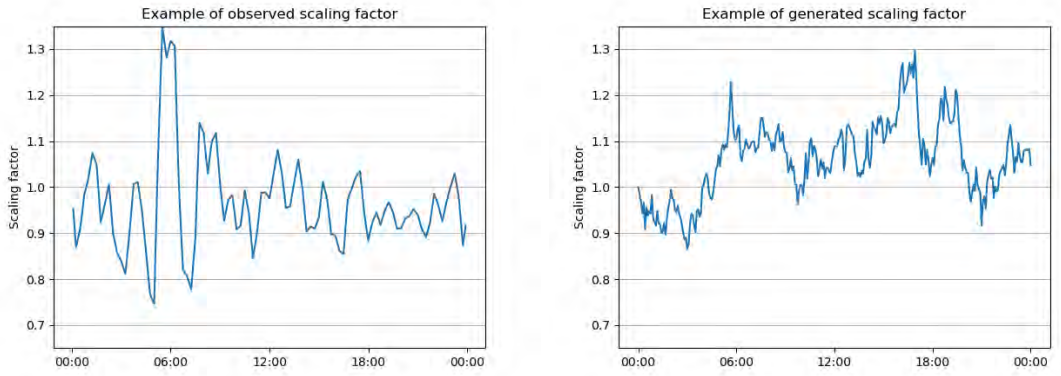

Figure 14: Observed versus simulated scaling factor

In the figures above we can see two different scaling factors. On the left-hand side we can see a sample of the observed scaling factor, extracted from the influent dataset. On the right we can see the scaling factor that is produced by our simulator.

In Figure 14 we can see two different examples of scaling factors. The figure on the left was already shown in Section 3.1 and is the scaling factor as observed in the dataset. We have fitted this process using the Ornstein-Uhlenbeck process and used the parameter estimates to simulate the graph that is shown on the right. We can see that there are some differences between the two graphs. On the left-hand side, the process is often stable for a number of time periods after which we often see a large jump in the scaling factor. This is caused by the way the influent is calculated for the wastewater dataset and this does not always accurately represent reality. In the graph on the right, we do not see this jumping behavior since the process has the same variance in every time step. However, this is a more accurate representation of reality so we are comfortable with using this simulator. Furthermore, both figures show the same characteristics (large dependence

on the previous time step and scaling factors in the same range), albeit in a somewhat different way, so we are confident that our simulator produces a good wastewater influent pattern.

To validate the fitted distributions of the event durations and rain intensity a two-sample Kolmogorov-Smirnov test is performed. This test calculates the probability that two independent samples are drawn from the same continuous distribution. In other words, it tests whether the observed event durations and rain intensities come from the same log-normal distribution that we use in the simulator. When we test this over a simulated period of 2 years (to match the real data) and 10 different runs (since the simulation process is stochastic) we find that in all cases the null hypothesis of different distributions cannot be rejected. In other words, there are no clear indications that the observed data does not follow the log-normal distribution.

For the simulation of mass curves, we cannot apply a distributional test like the Kolmogorov-Smirnov test. To assess the quality of these mass curves we have to rely on visual comparisons. For this reason, we have plotted two different sets of mass curves in Figure 15.

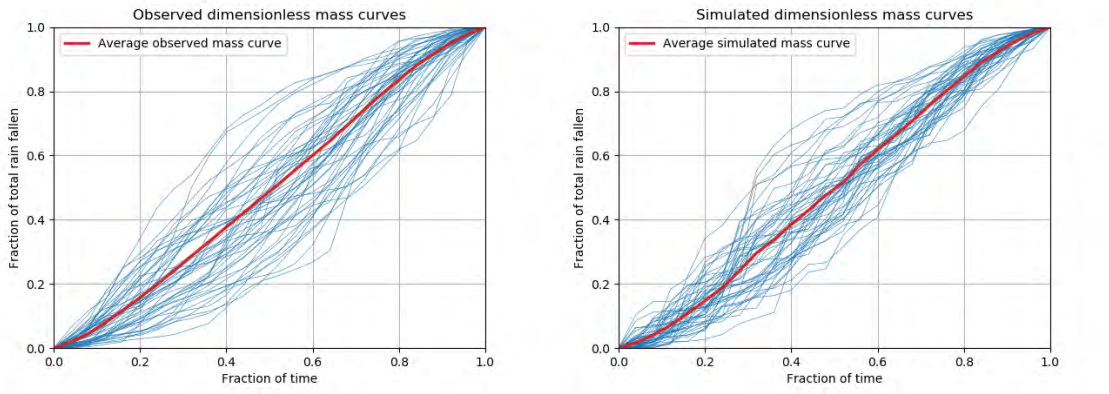

Figure 15: Observed versus simulated mass curves

In the figure above we can observe two different sets of mass curves. On the left-hand side we can see examples of the observed mass curves, extracted from the influent dataset. On the right we can see the mass curves that are produced by our simulator.

On the left-hand side of the figure, we have the mass curves like they are calculated based on the precipitation dataset along with the averaged observed mass curve. We can recall from Section 3.2 (where the same plot is shown) that the average event starts off slowly, has a period of high intensity, and ends with a low-intensity period again. Furthermore, individual events can differ quite a lot regarding when most of the rain falls within the event. When we look at our simulated events (shown on the right) we see a similar behavior for the average event profile. It can be noted that this average simulated mass curve is less smooth than for the observed mass curves, but this is due to the smaller sample size. When we look at the individual mass curves, we can see that these are a bit more volatile in our simulated events compared to the observed events. However, this can be explained by the fact that a lot of observed events are rescaled from relatively short events (e.g. 10 time periods) to the standard 25 steps that we use for our mass curves. Since we

do not know how the rain is distributed during a single time period, it was assumed that rain was constant during this time period, leading to smoother mass curves. If we would convert the mass curves on the right to shorter time windows and then rescale them to the 25 time period scale this would smooth the mass curves, leading to an outcome more similar to the observed mass curves. In practice, this difference is not of importance since all mass curves are converted to the duration that was simulated by the event duration simulator.

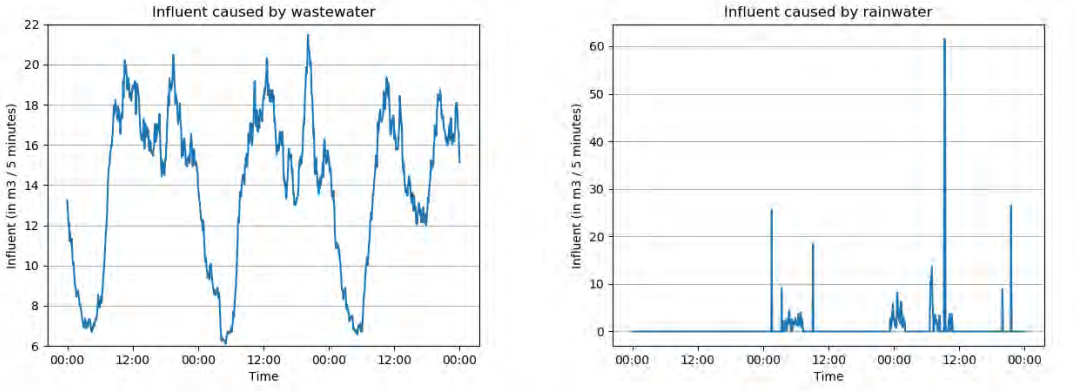

Figure 16: Simulated wastewater and rainwater influent series

In the plots above we can see examples of simulated wastewater and rainwater. These samples are generated over a periods of 3 days using the simulation process as described in Section 3.

The final outcome of our wastewater and rainwater simulators can be seen in Figure 16. On the left-hand side, we can see the influent of the wastewater simulator. In this graph, we can clearly see the daily pattern over a time span of 3 days with added noise coming from the scaling factor. This is most visible during the third day which shows a much larger dip between the morning and afternoon peaks. When we look at the influent caused by rainwater we can see a combination of short but intense rain spells mixed with rain spells with a longer duration, but lower intensity.

If we join the two influent streams we get the total influent that is shown in Figure 17. In this graph, the dotted orange and green lines represent the minimum and maximum capacity of the pumps respectively. We can see that during the nights the total influent dips below the minimum capacity of the pump. This means that the agent will either have to use its buffer or turn the pump off during some time periods. We can also see that during times of intense rainfall the total influent is higher than the maximum capacity of the pump. In scenarios like these the influent forecast can be of help in order to stay below the 10 % threshold we have set.

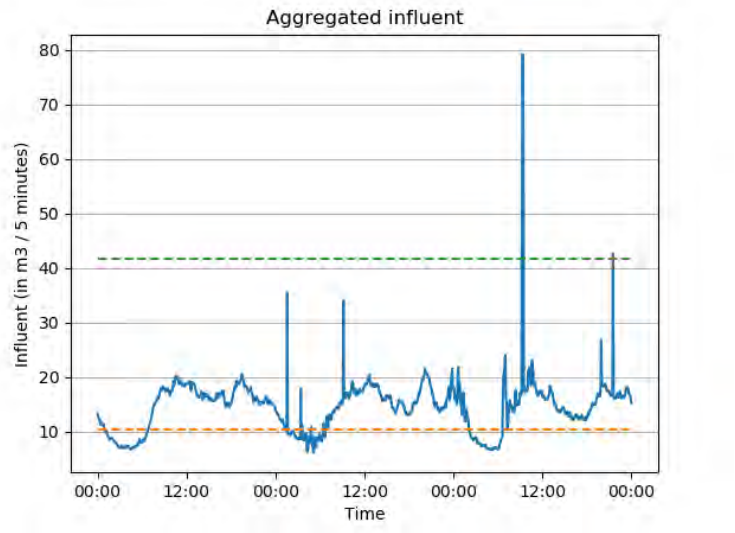

Figure 17: Combined influent series

In the figure above we can observe the aggregated influent of the wastewater and rainwater series that are shown in Figure 16. The orange dotted line corresponds to the minimum available action of the agent, whereas the green dotted line corresponds to the maximum available action of the agent.

The next part of the environment we focus on is the forecast of future influent. Again, we start off with influent coming from wastewater for which we have created a simple forecasting model. The result of this can be seen in Figure 18. In this figure, we can see that our forecast closely follows the actual rainfall in the next hour. The reason for this is that most variance in the wastewater influent comes from the daily pattern, which is known to us. On top of that, the simulated noise is mean reverting, so observing whether we have less or more influent than expected in the current time period tells us a lot about future time periods.

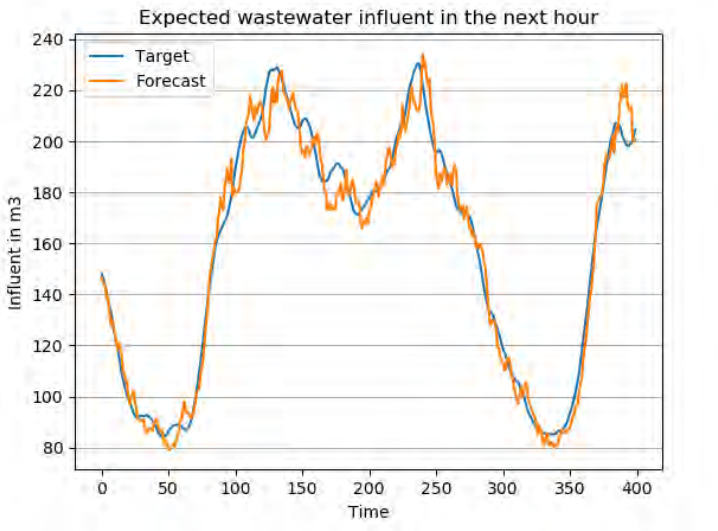

Figure 18: Actual versus forecasted wastewater influent in the next hour

In the figure above we can see influent coming from wastewater in the next hour. The blue series denotes the actual wastewater influent in the next hour, which is generated by our simulator. The orange series is the forecast of this influent using the methodology as described in Section 3.3.

The final part of the environment is to make a forecast for rainwater. In Figure 19 we can see two versions of a simulated forecast. The version on the left shows the forecast that is created without any exponential smoothing. We can see that this forecast produces a lot of outliers and does not show any time dependence. To solve these problems we have introduced the version with exponential smoothing, of which the results can be seen in the graph on the right. This version of the rainfall forecast is much closer to the actual rainfall and does not produce large outliers as we had in the figure on the left. Furthermore, we can see signs of autocorrelation since the forecast usually produces an over/underestimation multiple time periods in a row. Overall the figure on the right is a lot closer to the actual forecasting model we saw in Figure 10 so we use this version of the forecasting model in the remainder of this thesis.

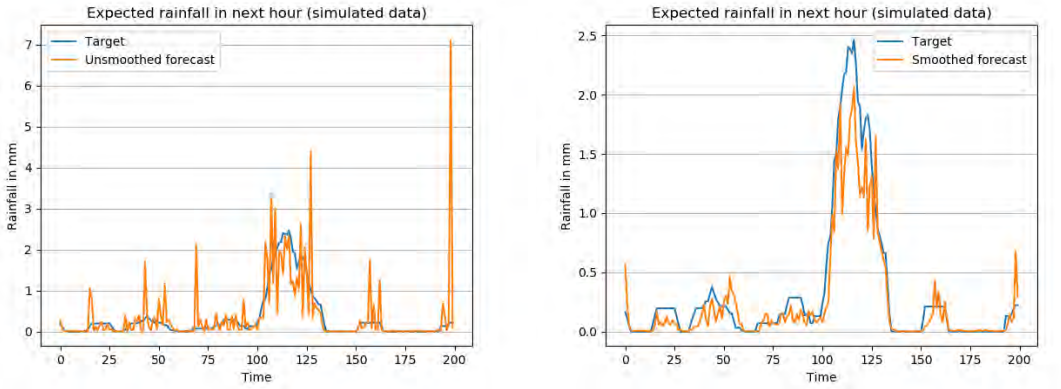

Figure 19: Actual versus forecasted rainwater influent in the next hour

In the figures above we can observe the influent coming from rainwater in the next hour, along with two different forecasts. The blue series denotes the actual amount of rainfall in the next hour. The orange series corresponds to the unsmoothed forecast in the left plot and the smoothed forecast in the right plot.

6.2 Agent Evaluation

The next part of the results discusses how our agents have performed in the four different environments we have created. These are the agents from Section 4 along with a random agent for comparison purposes. For the deep reinforcement learning agents we show the results for those parameter settings that worked best, with results for all the parameter combinations available in Appendix D. After that, we take the best agent for every environment and analyze how the agent is able to obtain its good performance and if it shows all the characteristics we were looking for.

We start with the single agent, no influent forecast environment. Of all the environments that we discuss, this environment is the least complicated. In Table 1 we can see the performance of our seven different agents on the seven different performance metrics that we have defined. When we look at this table, a few things stand out. First of all, we can see that the random agent performs much worse than all the other agents. This is hardly a surprise given that this agent does not take into consideration any state information to make its decision. When we look at the other agents, we can see that most of them are performing at a reasonable level with the exception of the A2C algorithm.

From the five agents that perform at a high level, we see that in terms of average reward the value iteration algorithm performs best, followed by both implementations of the PPO algorithm. These algorithms, as well as the rule-based agent, are also doing a good job of constantly obtaining a high reward, indicated by their low standard deviation of the rewards. We can see that all of these five agents do a good job staying between the lower and upper threshold we have set, making an error anywhere between 0.15 % and 2.5 % of the time. The rule-based agent performs particularly well in this aspect since it starts taking action long before the upper threshold is reached and stops in time to avoid the lower threshold. Where this agent is less effective is choosing the same action over

Table 1: Results single agent, without influent forecast

This table shows the performance of our seven different agents in the single agent environment without an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $> thres$ and $< thres$ columns denote the number of violations of the upper and lower threshold respectively over a 1000 time step run. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Random	-989.70	209.52	3.63	814.30	5.382	**0.000**	0.098
Rule-based	3665.80	**38.78**	**1.66**	**0.00**	8.037	0.224	0.645
VI	**3793.86**	46.89	6.48	**0.00**	4.994	0.023	**0.912**
TRPO	3665.49	128.47	24.64	0.76	6.408	0.314	0.688
A2C	2008.01	346.04	274.38	33.35	4.953	0.385	2.722
PPO1	3756.00	105.45	9.75	**0.00**	5.473	0.034	0.638
PPO2	3740.88	121.93	8.67	0.21	5.738	0.061	0.516

and over again, indicated by the high standard deviation of the actions. We see that value iteration and A2C do well in this aspect, but judging by the number of times the A2C algorithm passes the threshold it seems like this algorithm fails to take action when needed (which also explains the low average reward). Finally, when we look at the quantiles, we can see that value iteration does a good job of making use of the margin between the threshold values. We see that TRPO and PPO tend to stay away from the upper threshold value much more, which likely explains why value iteration outperforms these algorithms. Using this flexibility less also means that these agents need to turn off the pumps in some time periods, explaining their higher action standard deviation. Our suspicion that the A2C algorithm fails to take action when needed is confirmed since the upper quantile indicates that this algorithm often passes the upper threshold.

The next environment we look at is the single agent with influent forecast environment. This environment is exactly the same as we have discussed above, but agents now have new information about the influent forecast. This should enable them to obtain better performance. When we compare Tables 1 and 2 we see that this is indeed the case for most agents (ignoring the random and rule-based since these do not use the additional information). When we look at the performance metrics, we see similar results as we saw above. The random agent is still by far the worst, which was expected due to its simplicity. The A2C agent has improved somewhat, but it is still far worse than all the other agents that use the state information since it often passes the upper threshold (visible in the third and seventh performance metrics).

When we look at the deep reinforcement learning algorithms, we see that the TRPO agent has outperformed the rule-based agent and shows similar performance to the PPO2 algorithm. If we look at the average reward, we can see that the value iteration algorithm is again the best algorithm. It achieves its good performance by having a low standard deviation in its actions and using the margin between the two thresholds better than all other algorithms. All of the five algorithms

Table 2: Results single agent, with influent forecast

This table shows the performance of our seven different agents in the single agent environment with an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $> r\,thres$ and $< thres$ columns denote the number of violations of the upper and lower threshold respectively over a 1000 time step run. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Random	-988.47	209.20	3.63	814.10	5.381	**0.000**	0.098
Rule-based	3665.80	38.78	**1.66**	**0.00**	8.037	0.224	0.645
VI	**3806.69**	**29.31**	1.69	0.12	5.128	0.023	**0.878**
TRPO	3729.27	67.76	7.68	0.78	6.535	0.074	0.694
A2C	3212.48	580.19	166.63	1.46	**4.614**	0.062	1.328
PPO1	3764.26	55.73	3.79	0.05	5.960	0.036	0.510
PPO2	3730.33	103.41	10.31	0.21	5.923	0.069	0.634

that perform well manage to avoid the threshold boundaries. More importantly, they are able to do this a lot better than they were without the influent forecast. For the value iteration, TRPO, and PPO1 algorithms the number of errors dropped by about 70 %. This is a clear indication that adding the influent forecast to the state space helps agents in their decision making.

A somewhat surprising result is that the PPO2 algorithm is not able to improve with the additional information it is provided with. There is no clear reason for this, but given that these results are based on 10 runs, we cannot rule out that this result is just a coincidence.

The next environment that we discuss is the multi-agent environment without the influent forecast. For this environment, most of the metrics are on the exact same scale as they were for the first two environments. This is not the case for the average reward and the standard deviation of the reward. The reason for this is that the maximum achievable reward per time step increases from 4 in the single-agent case to 10 in the multi-agent case. We can therefore expect to see higher rewards and standard deviations for the multi-agent environments. However, the scaling between the single and multi-agent rewards should be around a factor of 2.5 (10 / 4). A threshold violation is counted if at least one of the storage tanks is above its threshold value, the action standard deviation is calculated over the total action the quantiles are calculated as the average of the two storage tanks.

The results of this environment can be seen in Table 3 and are quite similar to what we have seen for the single-agent environments. Like in all cases, the random agent still performs far worse than all other agents. Similarly, the A2C algorithm is still the worst of all the deep reinforcement learning algorithms. What is different this run is that TRPO is now the best deep reinforcement learning algorithm, slightly outperforming both versions of the PPO algorithm. Upon further inspection of the results, we see that this is due to a few bad runs in both the PPO algorithms which lower the average performance of the algorithm. For the PPO1 algorithm, 8 out of 10 runs obtain an

average reward above the TRPO score of 9024. Since the TRPO algorithm does not show any negative outliers, it is highly likely that PPO1 is still the better algorithm for this environment, even though this is not reflected in these results.

Table 3: Results multi agent, without influent forecast

This table shows the performance of our seven different agents in the multi agent environment without an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively over a 1000 time step run. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Random	222.08	381.39	5.90	780.54	8.772	**0.000**	0.158
Rule-based	9022.40	**129.59**	**2.43**	**0.00**	12.223	0.227	0.659
VI	**9452.17**	150.25	12.71	0.04	6.887	0.058	0.953
TRPO	9024.52	1029.03	50.93	0.73	7.454	0.098	1.040
A2C	8150.40	472.52	64.73	17.65	13.215	0.080	0.850
PPO1	8979.62	872.06	67.22	0.23	6.679	0.052	**0.963**
PPO2	8892.67	910.48	67.28	0.08	7.617	0.136	1.022

Another surprising result in this environment is the good performance of the rule-based agent, which is obtained by staying far away from the threshold values. However, it could very well be that this performance stands merely because of the bad performance of the PPO algorithms. When we compare the results of Tables 1 and 3 we see that in both cases the rule-based agent and the TRPO agent perform at similar levels.

Finally, we can see that value iteration outperforms the other algorithms. This is in line with what we saw in the single-agent environments. Again it obtains its good performance by using the full range between the threshold values without making too many errors. Furthermore, when we look at the standard deviation of the actions the algorithm does a good job of creating a constant influent stream to the wastewater purification plant. These results are a clear indication that our multi-agent adaptation works as intended since the algorithm outperforms the deep reinforcement learning agents while still being faster in finding the optimal policy.

The final environment we look at is the multi-agent with the influent forecast. This environment is the most realistic, so the agent that performs best in this environment is likely to do well in real-world scenarios as well. The results for this environment can be seen in Table 4. When we look at the results we see that these are completely in line with what we have seen in all other environments. The random and rule-based agents did not change their behavior compared to the previous environment, so the results for these agents are exactly the same as in Table 3.

When we go to the deep reinforcement learning methods we once again can see that the A2C algorithm is worse than all other deep reinforcement learning agents. Compared to the single-agent environments, the algorithm manages to make fewer errors regarding the lower and upper

Table 4: Results multi agent, with influent forecast

This table shows the performance of our seven different agents in the multi agent environment with an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively over a 1000 time step run. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Random	222.14	382.93	5.90	780.54	8.773	**0.000**	0.158
Rule-based	9022.40	129.59	**2.43**	**0.00**	12.223	0.227	0.659
VI	**9500.74**	**95.47**	5.18	0.07	**7.079**	0.060	**0.952**
TRPO	9202.37	366.82	22.11	0.77	8.734	0.109	0.819
A2C	8447.57	281.19	72.83	7.27	11.876	0.115	0.678
PPO1	9346.92	307.96	13.43	0.21	7.730	0.040	0.646
PPO2	9156.80	442.26	25.20	0.11	8.764	0.098	0.805

thresholds. However, the actions that are selected by the agent change a lot over time, which results in the high action standard deviation. Like we see in most other environments, the PPO1 algorithm performs the best among the deep reinforcement learning methods both in terms of staying within the thresholds and in minimizing the standard deviation of the actions.

However, value iteration is still the best method, a feat we have seen in all four environments. To test if the average reward of the value iteration agent is significantly higher than the average reward for all other agents, a mean comparison test with independent variances is performed. We find that for almost all environments and agents value iteration performs significantly better at a 95% confidence level. The sole exception to this is the PPO1 agent in the multi-agent environment without an influent forecast. While the difference in average reward is quite large between the PPO1 and value iteration agents, the main cause of this was a single bad run for the PPO1 agent. This increases the variance, leading to a mean comparison test with a p-value slightly above 0.05. Overall we can conclude that value iteration is the best agent among those that have been tested.

Since value iteration performs best in all four environments, we use this algorithm to further analyze the behavior of the agent. More specifically, we check if the agent shows the desired characteristics like buffering, coordination between pumps, and anticipation of rain events. Furthermore, we include some plots with actions produced by the rule-based agent as a comparison.

The first environment we consider is the single agent, no influent forecast environment. For this environment, we discuss two scenarios; one without rainfall and one with rainfall. The first scenario can be seen in Figure 20 and consists of 1000 time periods, which is roughly 4 days. We can see in this figure that our scenario does not have any rain due to the lack of peaks in the influent pattern. Furthermore, we can clearly see the daily pattern characterized by the double peak structure.

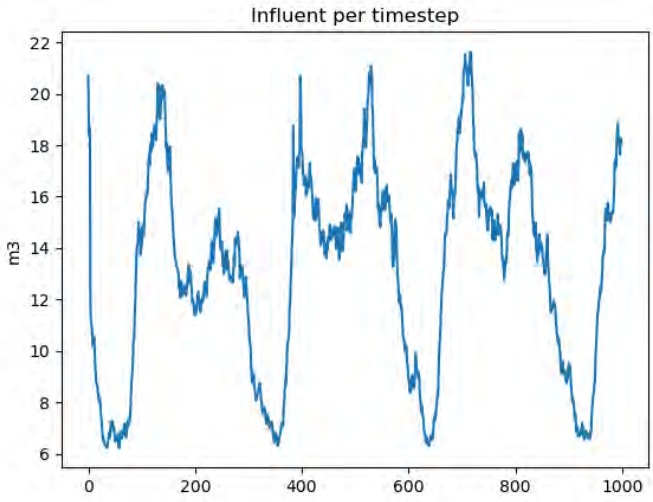

Figure 20: Simulated influent in the single agent, no forecast environment (without rainfall)

In the figure above we can see an example of a simulated influent pattern in the single agent without influent forecast environment. On the horizontal axis the time periods are shown, where 1000 time periods corresponds to approximately 3.5 days. On the vertical axis the influent is shown in m^3 per 5 minutes.

To see how the rule-based agent and value iteration agent copes with this scenario we need to go to Figure 21. On the left-hand side, we can see the volume and actions of the rule-based agent. This agent has a type of on/off behavior which can be seen in both the volume and action plots and is explained in Section 4.1. The value iteration agent shows a different type of behavior. First of all, the volume does not fluctuate as much as we see for the rule-based agent. Furthermore, the volume that is registered approaches the upper threshold value closer (which is 350 in this case) and goes closer to 0 as well. This is caused by choosing the minimum possible action when possible, with the pump turned off in all the other time periods. Using the minimum possible action during the day fills the storage tank, which is then depleted during the evening/night. We can see this as a clear indication that the agent possesses the buffering characteristic we were looking for.

We can easily relate these plots to the results we saw in Table 1. In terms of reward, both algorithms are able to collect the reward for staying between the threshold values in all time periods. However, since value iteration only uses the minimum possible action to pump out water, it receives the action-related part of the reward function more often (which is only given when an action > 0 is chosen). This also explains how the algorithm is able to have a lower standard deviation of its actions. The fact that value iteration uses buffering and gets closer to the threshold value explains the results we saw for the volume quantiles.

The next scenario comes from the same single-agent, no influent forecast environment, but this time it includes rainfall. We can see the influent pattern of this scenario in Figure 22. In this scenario, there are several rain spells with the most noteworthy around the 500 time period mark. However, given that the maximum capacity of the pump is 50 cubic meters of water per time period, this should not pose a problem for both algorithms.

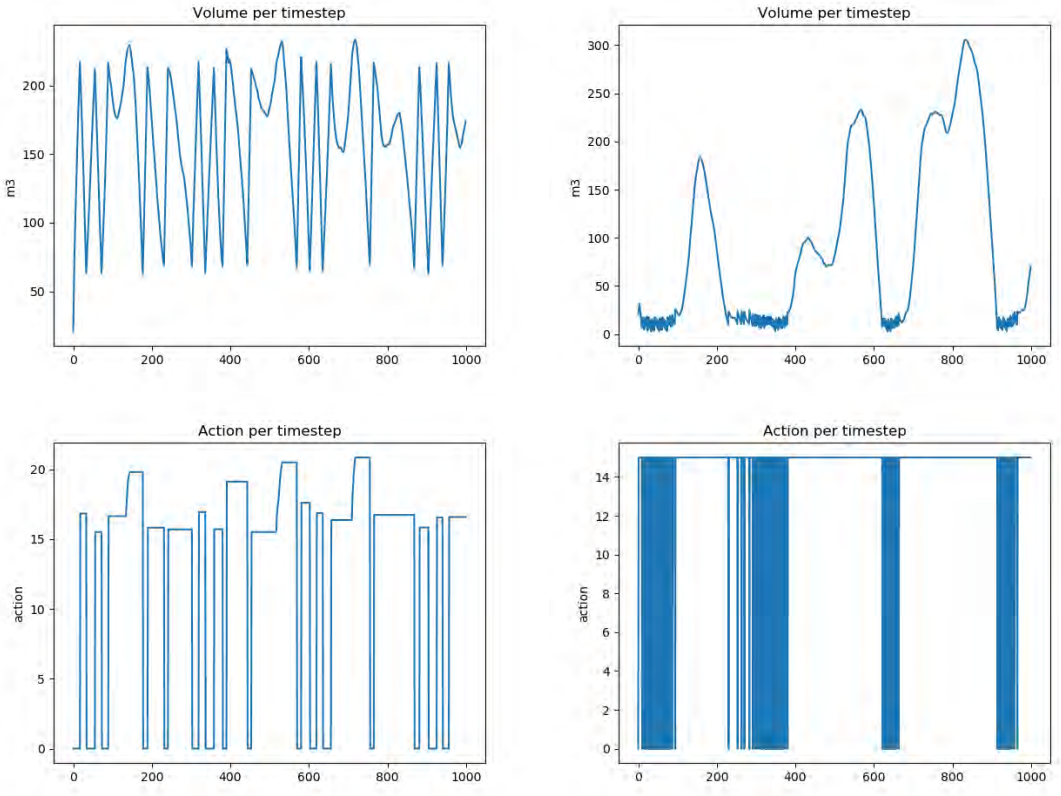

Figure 21: Actions and volumes for the rule-based and value iteration agents (without rainfall)

In the four plots above can see the volume and actions of the rule-based and value iteration agents based on the scenario shown in Figure 20. In the top left corner we can see the volume for the rule-based agent, with the actions of this agent shown in the left bottom plot. On the right-hand side we can see the volume and actions for the value iteration agent. In all plots the time step is shown on the horizontal axis, with the volume / action shown on the vertical axis.

When we look at the behavior of the rule-based algorithm, we see that the rain spells cause longer periods of non-zero actions by the agent. Some of these periods have a high, but brief peak. This can be explained by the way the rule-based agent chooses its action. When the action reaches a certain level, the agent does not lower the action until it stops the pump altogether. This can be seen as a kind of overreaction by the agent since it still selects high actions after the peak in influent has passed.

Value iteration takes a different approach to cope with rain spells. We can see that it uses the extra influent to fill up the storage tank as long as it does not approach the upper threshold value. This additional influent can be used during the night to keep the pump at minimum capacity for a longer period of time. This is the more favorable approach since it results in an influent stream to the wastewater treatment facility that is more evenly spread throughout the day. Furthermore, it is another indication that the agent is able to use the storage tank as a buffer. Only when there is rainfall and the storage tank is near the upper threshold value the action selected by the agent goes up.

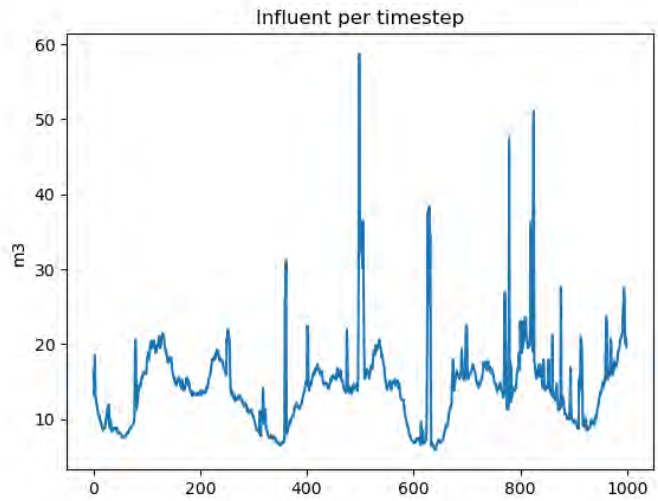

Figure 22: Simulated influent in the single agent, no forecast environment (with rainfall)

In the figure above we can see an example of a simulated influent pattern in the single agent without influent forecast environment. On the horizontal axis the time periods are shown, where 1000 time periods corresponds to approximately 3.5 days. On the vertical axis the influent is shown in m^3 per 5 minutes.

We can confirm this strategy by looking at the policy the value iteration agent follows, which is shown in Figure 23. From this figure, it becomes clear that the value iteration agent selects the minimum possible action (at 0.2 in this plot) as long as it stays below the 10% threshold. When the threshold is violated, the optimal action goes up to lower the tank volume as quickly as possible.

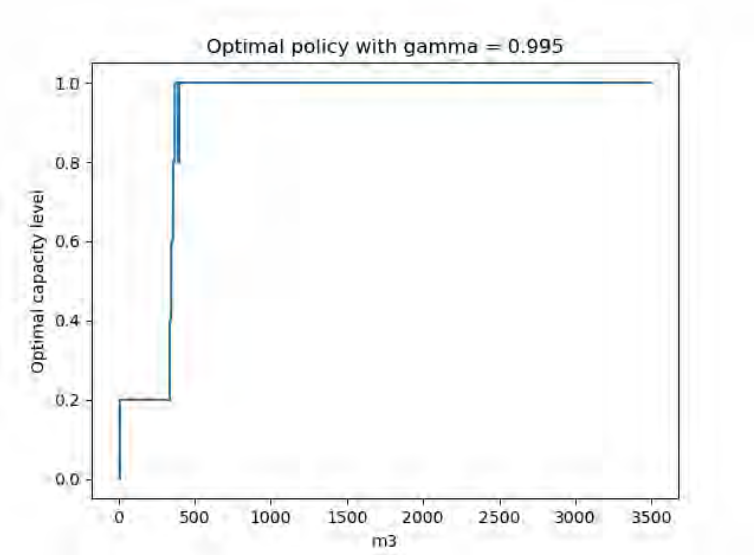

Figure 23: Policy of the value iteration agent

In the figure above we can see the policy the value iteration agent uses in the single agent, no influent forecast environment. On the horizontal axis we can see the volume in the storage tank in m^3. On the vertical axis we can see the optimal action, scaled between 0 (no action) and 1 (maximum possible action).

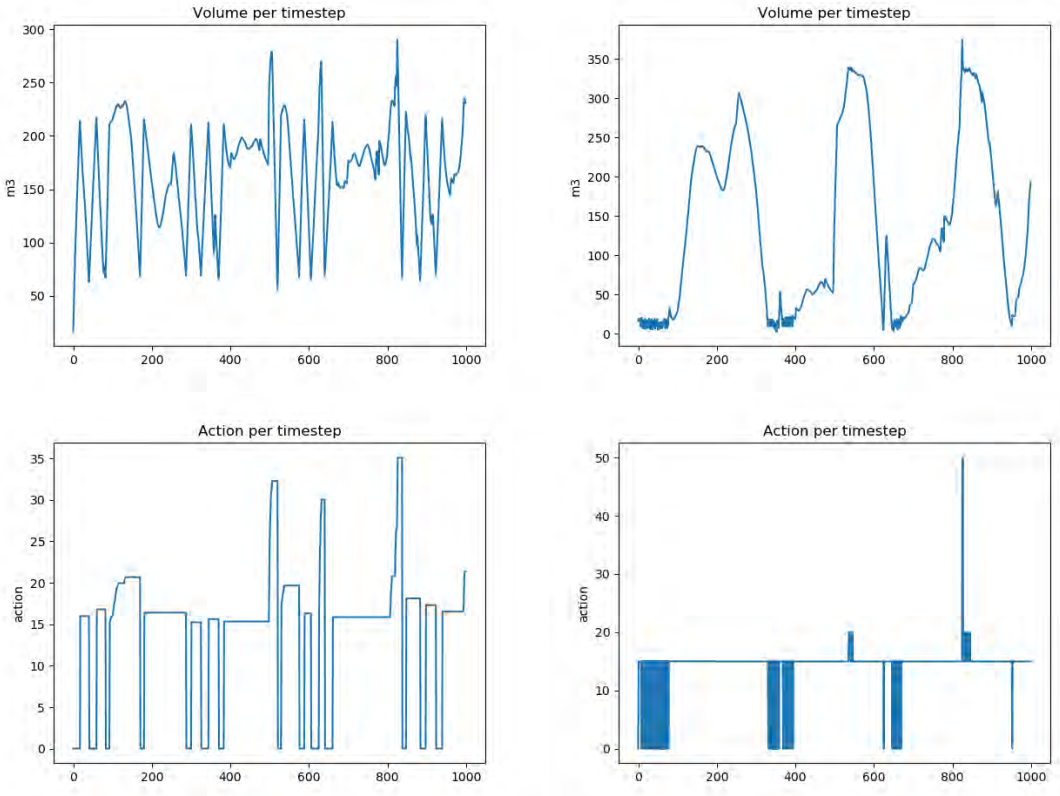

Figure 24: Actions and volumes for the rule-based and value iteration agents (with rainfall)

In the four plots above can see the volume and actions of the rule-based and value iteration agents based on the scenario shown in Figure 22. In the top left corner we can see the volume for the rule-based agent, with the actions of this agent shown in the left bottom plot. On the right-hand side we can see the volume and actions for the value iteration agent. In all plots the time step is shown on the horizontal axis, with the volume / action shown on the vertical axis.

The next environment we look at is the single agent with influent forecast environment. Since the rule-based agent does not use any of this additional information (and therefore does not change its behavior), we only focus on the value iteration agent here. This environment allows us to check whether the agent shows anticipating behavior when a large amount of influent is approaching. To show this, we have selected a scenario with several rain spells, all with different characteristics. We can see this scenario in Figure 25. In this figure we can see several rain spells. The first is in the early stages of the scenario and is rather small. After that a brief, intense rain spell occurs, followed by a longer intense rain event. The final noteworthy rain event occurs around time period 600.

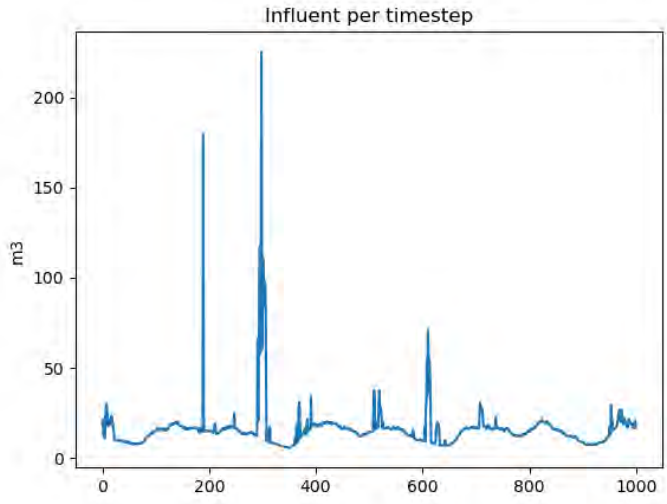

Figure 25: Simulated influent in the single agent, with forecast environment

In the figure above we can see an example of a simulated influent pattern in the single agent with influent forecast environment. On the horizontal axis the time periods are shown, where 1000 time periods corresponds to approximately 3.5 days. On the vertical axis the influent is shown in m^3 per 5 minutes.

To see how the value iteration algorithm copes with this scenario we can look at Figure 26. We can see that the first rain occurs when the storage tank is nearly empty. Given its low intensity the agent processes this rain event by using the storage tank and letting the pump run at minimum capacity. The second rain event occurs when the storage tank is around the 200 mark. We can see a brief drop in the volume followed by a rapid increase. This drop indicated that the agent anticipated the additional influent by increasing its action before the rain event arrived. We can see the same more clearly in the longer intense rain spell. Before the rain event arrives there is a drop in volume from around 300 to 150. This is followed by a spike to 800, but during this time the pump is running at the maximum capacity of 50. This means that during this time the agent cannot avoid going over the upper threshold of 350. Finally, we can see how the agent deals with the rain event around time period 600. While this event is less intense than the two previously discussed events, it still has influent values above the maximum action the agent can take. This means that some anticipation is required to avoid going over the upper threshold. When we look at the volume plot we see that this is indeed the strategy that the agent follows, indicated by the brief dip and resurgence in volume around the 600 time period. These last three events are a clear indication that our agent is able to anticipate based on the influent forecast it is provided with.

The next scenario we look at comes from the multi-agent with no influent forecast environment. We are mainly interested if the agents can coordinate their individual actions to make the total action as constant as possible. This is visible in times of low influent, as agents need to turn off the pump at certain times. For this environment we use the same scenario as in Figure 20 but rescaled for all agents. This scenario has no influent from rainwater, so we are able to see what happens during the night where there is low influent. We see a scenario with rainwater later in this subsection when we combine the multi-agent and influent forecast environments.

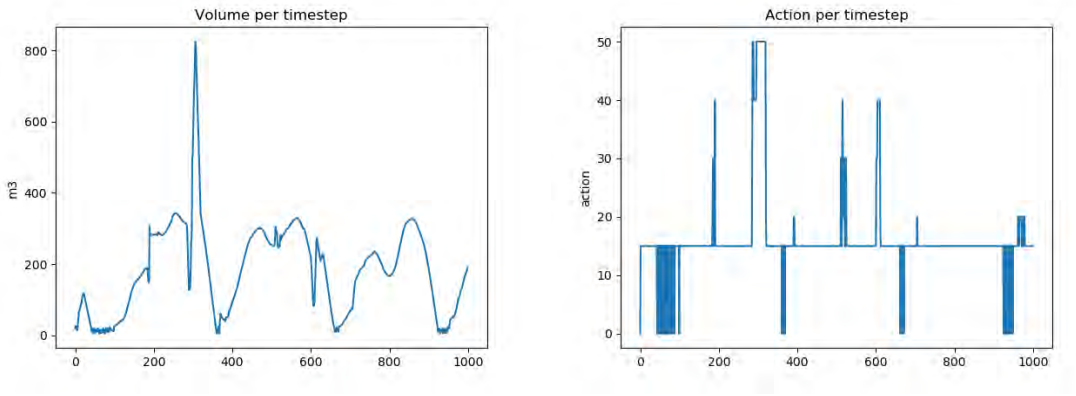

Figure 26: Actions and volumes for the value iteration agent, single agent with influent forecast environment

In the plots above can see the volume and actions of the value iteration agent based on the scenario shown in Figure 25. On the left-hand side we can see the volume , whereas the right-hand side shows the actions taken by the agent. In all plots the time step is shown on the horizontal axis, with the volume / action shown on the vertical axis.

In Figure 27 we can see the actions taken by the individual agents on the left, as well as the combined actions on the right. We can see that the first agent only uses the minimum possible action along with no action, whereas the second agent uses the same combination in the beginning, but switches between the lowest two actions later on. We can see that in the second part of our scenario our agents are out of sync (i.e. only one of the agents is changing actions during a certain time period). However, at the beginning of our sample we can see that both agents have to choose between doing the minimum possible action and no action. If we were to see coordination between agents agent 1 would do the minimum possible action when agent 2 does no action and vice versa. When we look at the combined action of the two agents, this is indeed what we observe. Only in the first time period when both storage tanks are at 0, we see that both agents choose to do no action. After that the agents alternate between three options, only agent 1 does the minimum action (resulting in a total action of 15), only agent 2 does the minimum action (resulting in a total action of around 10), or they both do the minimum possible action (resulting in a total action of around 25). The fact that the total action is always above 0 is a clear indication that the agents coordinate their actions, meaning that our multi-agent adaptation of value iteration works as intended.

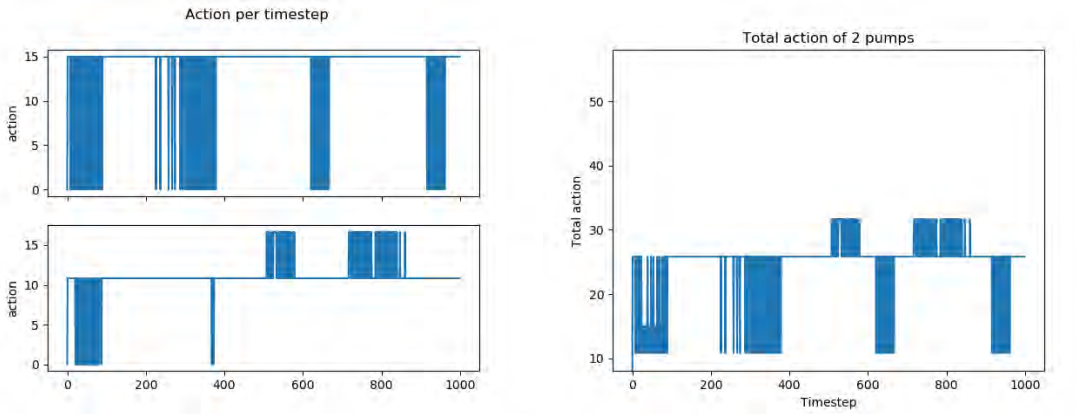

Figure 27: Individual and total action for the value iteration agent, multi agent without influent forecast environment

In the two plots above we can see the actions per agent, as well as the joint action of the two agents. These actions are selected in response to the influent pattern shown in Figure 20. In all the plots the time step is shown on the horizontal axis, with the action shown in the vertical axis.

Finally, we move on to a multi-agent with influent forecast environment. So far we have seen that our agent is able to show all the desired characteristics when tested in isolation, which were buffering, anticipation, and coordination. This final environment allows us to test if this still holds when we apply our agent to the most realistic environment.

In Figure 28 we can see four different types of graphs. In the top left corner we see the influent for our two agents. These influent patterns are very similar and differ only in the scaling of the daily pattern and the amount of influent each millimeter of rain generates. Given that these parameters are close to each other in our case, the two influent patterns look very similar. In this pattern, we see several rain events, the two most important of those in the middle of our scenario.

The first characteristic we are looking for is buffering. We can see clear evidence of this characteristic in the right upper corner, which displays the volumes in both storage tanks. We can see for both agents that they let the volume increase during the day to right under the threshold (350 and 250 for agents 1 and 2, respectively). During the night this buffer is used to choose the minimum possible action for a longer time than otherwise would have been possible.

The second characteristic we are looking for is anticipation. To do this we once again look at the volume plot in the top right corner. For the first large rain event we observe that agent 2 lowers the volume in the storage tank before the event arrives. Agent 1 seems to do this but only to a smaller degree, which can be explained by the margin between the volume and upper threshold at that time. The influent coming from the rain event is processed with the maximum action available to the agents as shown in the bottom left corner. When the second rain event arrives we see that both agents anticipate this event by lowering their volumes, albeit to a lesser degree this time. This is explained by the lower intensity of the second rain event. Overall, we still see clear signs of anticipation in our agents.

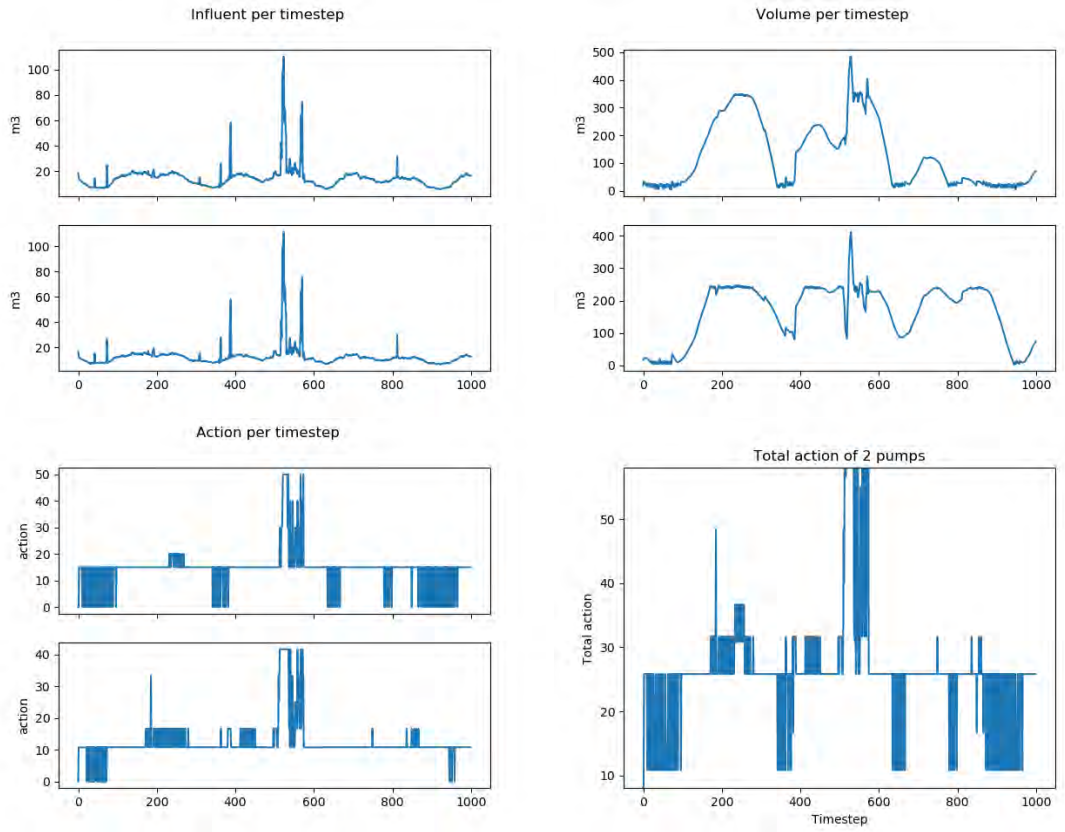

Figure 28: Behaviour of the value iteration agent in the multi agent with influent forecast environment.

In the four figures above we can see various aspects of the value iteration agents behaviour in the multi agent with influent forecast environment. In the top left we see the influent the agents are faced with. In the top right the volume of both agents is shown over time. In the bottom two plots we can see the actions of the individual agents (left) and their collective actions (right).

The final characteristic we look for is coordination. To do this, we take the same approach as we did in the multi-agent, no influent forecast environment. This time we see two parts in our sample where both agents have to turn off the pumps in some time periods, which is at the beginning and end of our scenario. When we look at the combined action of our two agents we see that other than the very first time period, at least one of the two agents selects a non-zero action in every time period. Given that this occurs quite often for each agent individually this is a clear indication that coordination is still present between our agents in this final environment. To conclude, we have clear evidence that our value iteration agent displays all the desired characteristics, both when tested in isolation and combined into a single realistic environment.

6.3 Sensitivity analysis

The final part of this results section focuses on performing a sensitivity analysis. A sensitivity analysis is defined as *"a local measure of the effect of a given input on a given output"* (Saltelli et al., 2004, p. 42). So far we have seen that the value iteration agents work very well in the environments that were created. To improve the generalizability of these results we adjust the parameters that

determine the environment and observe if our findings from the previous subsection still hold. We do this by adjusting these environment parameters one at a time in both an upward as downward direction. We have chosen to adjust these parameters by 25% since this gives a shock large enough to change the environment in a meaningful way without creating unrealistic scenarios.

There are six different environment parameters we adjust in this sensitivity analysis. Three of these parameters are related to the pumps the agents can control. These parameters are the maximum volume of the storage tank, the minimum action the agent can take, and the maximum action the agent can take. The volume of the storage tank also determines the upper threshold value, since this is defined as 10% of the maximum capacity. The minimum and maximum possible actions determine the range of actions each agent can take. Furthermore, since the value iteration algorithm has a finite number of actions which are spaced linearly between 0 and the maximum action this impacts the actions of the value iteration agent as well. Another parameter is the amount of influent each millimeter of rain causes, which determines the balance between wastewater and rainwater. The final two parameters are κ and σ, which play a role in simulating the scaling factor of the wastewater. The mean reversion rate is determined by κ, whereas σ determines how much variation this scaling factor has.

Along with the six parameters that were just discussed there is one other factor we include in this sensitivity analysis. This is the number of agents in our multi-agent environments. The results of Section 6.2 were based on a setting that had 2 agents. Since it is not unusual for a facility to have more than 2 agents we test our algorithms in a setting with 5 agents. In this setting agents 1, 3, and 5 share the same environment parameters and agents 2 and 4 share environment parameters. This means that these agents get the same influent and use the same threshold values, but their actions differ since these are independent.

To limit the size of this section we focus on three agents in this subsection; rule-based, value iteration, and PPO1. The rule-based agent is selected to allow us to make a comparison with the currently used method, value iteration due to its good performance and PPO1 since it was the best deep reinforcement learning methods. Furthermore, we mainly discuss results for the multi-agent with influent forecast environment. Results for other environments can be found in Appendix E. [5]

We start with the results for the rule-based agent. In Table 5 we can see the results for each of the 13 parameter changes that we have tested. We can see that overall, this agent does a very good job of obtaining results that are close to the baseline results from Section 6.2. In terms of rewards, a few scenarios stand out. The first two that stand out are the minimum action going up and the maximum action going down, which both show below-average rewards. These scenarios limit the range of actions our agent can choose from, making it harder to generate a constant action. Furthermore, for the minimum action going up this results in the agent selecting an action of 0

[5] Results for all other agents are available on request

Table 5: Sensitivity results rule-based, multi agent, with influent forecast

This table shows the performance of the rule-based agent in the sensitivity analysis for the multi agent environment with an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively.

Parameter	Change	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Max. volume	Up	9026.66	126.61	2.11	0.00	12.155	0.230	0.657
	Down	8995.52	132.78	2.91	0.00	12.457	0.222	0.662
Min. action	Up	8713.16	122.37	2.28	0.00	14.144	0.214	0.627
	Down	9157.64	129.22	2.53	0.00	11.977	0.239	0.694
Max. action	Up	9132.67	103.87	1.43	0.00	12.724	0.224	0.637
	Down	8748.42	193.47	5.10	0.00	11.600	0.231	0.714
Influent per mm	Up	8984.39	167.46	4.11	0.00	12.917	0.227	0.678
	Down	9052.05	95.11	1.20	0.00	11.599	0.227	0.646
κ	Up	9029.47	130.99	2.43	0.00	12.123	0.227	0.657
	Down	9015.43	128.81	2.42	0.00	12.342	0.227	0.662
σ	Up	9002.03	131.34	2.44	0.00	12.502	0.226	0.664
	Down	9035.22	126.91	2.43	0.00	12.040	0.227	0.655
Agents	Up	22524.20	313.24	2.27	0.00	31.872	0.226	0.656

more often, which is not rewarded. The other scenario that stands out is the scenario with five agents. However, this high reward is caused by the increased number of agents and disappears when we scale the reward back to two agents ($22,500/5 * 2 \approx 9000$).

Next, we look at the value iteration agent. When looking at this agent in isolation we see the same patterns as we saw for the rule-based agent. It is therefore more interesting to compare the value iteration agent with the rule-based agent to see which performs better. When looking at the average reward, we see that the value iteration agent performs better in all the scenarios it is provided with. It can obtain these higher rewards in the same way as we saw in Section 6.2, which is by using the range between the threshold values more effective and choosing actions with a lower standard deviation. This does come at the cost of surpassing both threshold values more often, but the loss in reward because of this is marginal compared to the benefits this strategy has.

Finally, we look at the results of the best deep reinforcement learning algorithm we have, based on the results from Section 6.2, which is PPO1. The results for this algorithm are shown in Table 7. We again see that the model performs less well in the minimum action up and maximum action down scenarios. Furthermore, we see a low average reward in the scenario with more noise around the scaling factor. However, this is caused by a single run where the model for some reason was not able to converge to a good policy. Without this run, the average reward would be around 9000, which is in line with all the other scenarios. When we look at the scenario with 5 agents we see that the deep reinforcement learning algorithm performs much worse than we would expect. It seems

Table 6: Sensitivity results value iteration, multi agent, with influent forecast

This table shows the performance of the value iteration agent in the sensitivity analysis for the multi agent environment with an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively.

Parameter	Change	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Max. volume	Up	9517.56	90.42	4.74	0.06	6.637	0.062	0.926
	Down	9462.09	104.51	7.38	0.13	7.696	0.059	0.972
Min. action	Up	9194.77	84.17	2.43	0.23	9.196	0.067	0.573
	Down	9578.65	111.71	7.42	0.15	8.461	0.246	0.983
Max. action	Up	9637.07	66.39	3.20	0.07	7.450	0.059	0.944
	Down	9208.68	169.77	8.35	0.10	6.945	0.062	0.974
Influent per mm	Up	9474.07	135.01	7.46	0.09	7.709	0.065	0.963
	Down	9517.07	62.98	4.03	0.05	6.518	0.056	0.943
κ	Up	9506.51	95.40	5.01	0.05	6.980	0.061	0.948
	Down	9483.95	99.70	6.39	0.13	7.263	0.057	0.956
σ	Up	9485.15	98.91	6.12	0.15	7.299	0.057	0.958
	Down	9509.30	95.25	4.90	0.05	6.918	0.061	0.944
Agents	Up	23744.70	212.06	3.04	0.10	17.985	0.043	0.929

like this is caused by the high standard deviation in the actions that are selected by the agent. An explanation for this bad performance could be that this agent is trained using the optimal parameters in a 2 agent setting. Testing different parameter settings as we did in Section 6.2 could benefit the performance in this scenario for the PPO1 agent.

When we compare the results of the PPO1 agent with the two other agents we see that the rule-based agent and the PPO1 agent often obtain similar rewards. This is in contrast to the results in Section 6.2 where the PPO1 often outperformed the rule-based agent. However, it could very well be that testing different parameters for the PPO1 agent in all scenario's would improve the results this agent obtains. The value iteration agent outperforms both the rule-based agent and the PPO1 agent in all scenarios, strengthening our belief that the value iteration agent is the most effective in finding the optimal policy.

Table 7: Sensitivity results PPO1, multi agent, with influent forecast

This table shows the performance of the PPO1 agent in the sensitivity analysis for the multi agent environment with an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively.

Parameter	**Change**	$\overline{\boldsymbol{Reward}}$	$\boldsymbol{\sigma_{Reward}}$	$>$ **thres**	$<$ **thres**	$\boldsymbol{\sigma_{Actions}}$	$\boldsymbol{Q_{0.05}}$	$\boldsymbol{Q_{0.95}}$
Max. volume	Up	9095.73	304.22	15.23	0.01	11.405	0.080	0.631
	Down	8955.25	196.45	10.55	0.05	13.632	0.138	0.684
Min. action	Up	8416.17	166.54	7.21	0.42	17.403	0.099	0.525
	Down	9156.31	285.95	30.04	0.00	10.766	0.123	0.729
Max. action	Up	9158.19	206.09	16.46	0.01	13.503	0.115	0.707
	Down	8324.36	196.95	7.08	0.42	12.993	0.094	0.480
Influent per mm	Up	8756.84	203.37	9.21	0.04	16.358	0.102	0.569
	Down	8956.35	181.68	8.98	0.07	13.408	0.094	0.573
κ	Up	8922.27	214.77	27.25	0.00	12.609	0.112	0.693
	Down	9036.36	258.09	20.45	0.00	11.709	0.103	0.669
σ	Up	8418.48	298.33	75.79	3.65	12.721	0.121	1.213
	Down	9092.35	275.23	14.96	0.00	11.564	0.103	0.678
Agents	Up	17890.60	424.16	16.55	0.02	59.288	0.183	0.752

7 Conclusion

In today's rapidly digitizing world, waterschappen are faced with the challenge of making their operations more efficient using new technologies. In this thesis, we have looked into the problem of controlling the influent of water into a water purification plant through the use of planning and reinforcement learning methods. These methods are compared to the method that is currently used in four different environments. The environments build on one another adding more complexity every time to create an environment that closely resembles reality.

In the design of the environments, a lot of emphasis is placed on basing our environment parameters on real data. Furthermore, the design of every environment is based on several characteristics that are present in the real dataset. For example, it was noted that the gradient boosting regressor produced errors that were both dependent on time and the amount of rainfall in the next hour. Our simulation environment was designed in such a way that it is able to produce the same type of errors to make our simulator as realistic as possible. From the results in Section 6.2 we can conclude that our simulator can match the same characteristics as the real data / predictive models.

The way the environments and reward function are set up stimulates three different types of behavior. The daily pattern in the wastewater combined with the flexibility the agents are given regarding the storage tank volume stimulates buffering behavior. This buffering behavior means that agents fill up the storage tank throughout the day (when influent is high) and deplete this during the night (when influent is low). The next characteristic is anticipation on periods with high influent (i.e. rainy periods), which can be achieved through the influent forecast agents are provided with. Finally, we stimulate coordination between agents by rewarding their collective actions.

The agents that are used in this thesis have all been set up in such a way that they are close to their best possible performance. The currently used rule-based agent is set up based on information from the waterschap, where it is currently used. The value iteration agent is designed specifically to function well in our environments. Finally, deep reinforcement learning methods are generic methods to solve reinforcement learning problems. To optimize their performance these models are tested using different parameter settings to adapt them to our problem.

When we tested our agents we saw very similar results in all four environments. The rule-based agent did a good job to stay between the thresholds we have set on the volume in the storage tank. However, this agent is less efficient is choosing constant actions due to a lack of buffering behavior. Furthermore, the agent does not anticipate on future rainfall and lacks coordination between the agents. We can conclude that this agent sets a decent baseline performance, especially in single-agent settings. However, it lacks the three behavioral characteristics we were looking for in our agent.

The deep reinforcement learning methods were often able to obtain slightly better performance than the rule-based agent. The downside of this agent is that it obtains these results without a clear strategy. In some instances, the agent improves its performance by approaching the threshold values more often, whereas in other instances it stays far away from these thresholds. The cause of this is the stochastic nature of this class of algorithms, which implies that the agent does not always come to the same strategy. Given that the performance difference between runs can be quite substantial for this class of algorithms using these types of agents is not ideal.

In light of this, we can confidently conclude that our tailor-made value iteration agent was the best in finding the optimal policy. The adaptations made to the value iteration algorithm are one of the key contributions made in this thesis. It constantly outperformed all the other agents, giving significantly higher rewards in nearly all cases. Additionally, our adaptation of the algorithm is near deterministic with the only source of variance being the estimated distributions of influent and forecasted influent. Since these distributions are estimated based on a large number of time steps we see very little variation in the outcome of our algorithm, with average rewards within a 0.2 point margin in our 10 test runs. Furthermore, the policy of this agent is easily visualized using the state-action values which makes it more transparent than deep reinforcement learning.

When we analyzed the behavior of the value iteration agent we concluded that it shows all three behavioral characteristics we were looking for in our agents, both when tested in isolation and combined. We can explain this good performance by the fact that standard value iteration is guaranteed to find the optimal policy. While we lost this guarantee through the discretization of our actions, adding buckets for the influent forecast, and adding the multi-agent adaptation, the loss in performance is relatively small.

The sensitivity analysis we conducted further confirmed the belief that value iteration is the best agent. In the 13 different ways the environment was adjusted, value iteration performed best in all of them. From this we can conclude that value iteration also works in scenarios where the environment contains more/less noise, the agent has a different set of actions, the influent of rain is larger/smaller, and the number of agents is higher. This makes our results promising for other wastewater treatment plants given that these plants are facing a similar problem.

For Ynformed / waterschap Rijnland this thesis is of value in several different ways. First of all, this thesis adds to the knowledge present within Ynformed in the field of planning and reinforcement learning. This is a field in which Ynformed has shown a large interest, but a lack of projects makes it hard to gain expertise. Second of all, the environments that are created can be of added value in future projects in the water sector. Elements like the daily wastewater pattern or rainfall are often of importance and having a simulator for this can be of help. For waterschap Rijnland this thesis serves as a proof of concept that they can manage their wastewater purification plant more efficiently than is currently done.

8 Recommendations for Future Work

In the last section of this thesis, we discuss the shortcomings of our method, how to improve on these shortcomings, and which elements of our method could benefit most from future research. Naturally, some topics benefit from future research that are not discussed in this section. However, based on our current knowledge of our method, these are the most natural extensions of the method

The first extension is related to the simulation of influent for multi-agent environments. Under the current method, all agents receive influent based on the same scaling factor and precipitation. This is converted to an influent series based on the environment parameters for each agent like the scale of the daily wastewater pattern and the amount of influent per millimeter of rain. While it is likely that that the scaling factors and rainfall are related for different agents, there are likely differences present in the scaling factor/rainfall between agents that are currently neglected. The same holds for the forecast that is created for the total rainfall in the next hour.

To improve this element of the simulation process we could adapt a multivariate model to simulate the scaling factor, rainfall, and forecast of rainfall in the next hour. To do this the parameter estimation would have to be adjusted to not only estimate the current model parameters but also some measure of how the two (or more) agents are related. These parameters could be used to simulate from a multivariate model. For rainfall, this process is even more complicated since generating this series consists of three steps. All of these steps must be altered to have a full multivariate model.

Another extension that can be made to the environment is related to the influent rainfall produces. In the current setup, rainfall causes influent in the same time period that it is registered. In reality, there is some lag between the time periods the rainfall is registered and the time period it arrives at the wastewater purification facility. This time difference is related to the time it takes the rainwater to flow from the point where it enters the sewage system to the wastewater purification facility.

The third extension is to expand the parameter analysis step that was performed in Section 6.2. The deep reinforcement learning models take a long time to train and evaluate, hence the amount of parameter combinations that were evaluated in this step is limited. Testing more parameter combinations could lead to better performance of the deep reinforcement learning models. Furthermore, finding good parameter settings could also help with the high variability we see in the strategy of these agents since a good parameter setting will consistently converge to the same policy.

Another extension would be to adjust the reward function to avoid constant action switching. We saw that the value iteration agent alternates between the minimum possible action and no action

during the night. Doing so is not desired from a pump maintenance perspective, as parts wear out quicker when actions alternate a lot. Adding an element to the reward function that penalizes this type of behavior can help to solve this inefficiency of the value iteration agent.

Finally, the value iteration agent can be fine-tuned to improve its performance slightly. This fine-tuning step is related to the bucket boundaries of the influent forecast. Currently, these boundaries are based on the quantiles of the influent forecast distribution, using high quantiles to realize different behavior in times of high expected influent. Setting these bucket boundaries in a different way changes the behavior of our value iteration agent, potentially improving performance.

A Deriving Maximum Likelihood Estimators

During the simulation of rainwater, we have fitted distributions on several aspects of the process. Naturally, the shape of these distributions is determined by one or more parameters. To make the simulation as realistic as possible, we fitted these parameters on the data using a Maximum Likelihood approach. In this appendix, we derive the closed-form expressions for these estimators. We do this for three different distributions; those being the exponential, gamma, and log-normal distribution.

A.1 Exponential Distribution

The first Maximum Likelihood Estimator that we derive is for the exponential distribution. This distribution has a single parameter λ. This parameter determines the average rate at which events occur.

The starting point to derive the Maximum Likelihood Estimator is the density function of the exponential distribution. This density function can be seen in Equation A.1.

$$f(x) = \lambda e^{-\lambda x} \tag{A.1}$$

We can use this equation to derive the likelihood function, which can be seen in Equation A.2.

$$\begin{aligned} L(\lambda) &= \prod_{i=1}^{n} \left(\lambda e^{-\lambda x_i} \right) \\ &= \lambda^n \prod_{i=1}^{n} e^{-\lambda x_i} \end{aligned} \tag{A.2}$$

This likelihood function is transformed into the log-likelihood function. Some simplification steps are performed to make it easier to take the derivative of this function.

$$\begin{aligned} \mathcal{L}(\lambda) &= \ln \left(\lambda^n \prod_{i=1}^{n} e^{-\lambda x_i} \right) \\ &= n \ln(\lambda) + \sum_{i=1}^{n} \ln(e^{-\lambda x_i}) \\ &= n \ln(\lambda) - \lambda \sum_{i=1}^{n} x_i \end{aligned} \tag{A.3}$$

If we take the derivative of the log-likelihood function with respect to λ and set this to 0, we can find the Maximum Likelihood Estimator $\hat{\theta}$.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \lambda} =& \frac{n}{\lambda} - \sum_{i=1}^{n} x_i \\ \Rightarrow& \frac{n}{\hat{\lambda}} - \sum_{i=1}^{n} x_i = 0 \\ \Rightarrow& \frac{n}{\hat{\lambda}} = \sum_{i=1}^{n} x_i \\ \Rightarrow& \hat{\lambda} = \frac{n}{\sum_{i=1}^{n} x_i}\end{aligned} \tag{A.4}$$

A.2 Gamma Distribution

The second Maximum Likelihood Estimator that we derive is for the gamma distribution. This distribution can be characterized in multiple ways. For this derivation, we use the version with a shape parameter κ and a scale parameter θ. The density of this characterization can be seen in Equation A.5.

$$f(x) = \frac{1}{\Gamma(\kappa)\theta^{\kappa}} x^{\kappa-1} e^{-x/\theta} \tag{A.5}$$

In the density above, $\Gamma(\kappa)$ represents the Gamma function. Again, we create a likelihood function based on the density as stated above.

$$L(\kappa, \theta) = \prod_{i=1}^{n} \left(\frac{1}{\Gamma(\kappa)\theta^{\kappa}} x_i^{\kappa-1} e^{-x_i/\theta} \right) \tag{A.6}$$

We derive the log-likelihood function by taking the natural logarithm of this likelihood function. We simplify the log-likelihood function by recognizing that the log of a product is equal to the sum of the individual logarithms.

$$\begin{aligned}\mathcal{L}(\kappa, \theta) &= \ln\left(\prod_{i=1}^{n} \frac{1}{\Gamma(\kappa)\theta^{\kappa}} x_i^{\kappa-1} e^{-x_i/\theta} \right) \\ &= \sum_{i=1}^{n} \left((\kappa - 1)\ln(x_i) - \frac{x_i}{\theta} - \ln(\Gamma(\kappa)) - \kappa \ln(\theta) \right) \\ &= (\kappa - 1)\sum_{i=1}^{n} \ln(x_i) - \sum_{i=1}^{n} \frac{x_i}{\theta} - n\ln(\Gamma(\kappa)) - n\kappa\ln(\theta)\end{aligned} \tag{A.7}$$

The problem with this log-likelihood function is that when we take the derivative with respect to κ, it is impossible to find a closed-form solution for $\hat{\kappa}$. To circumvent this, we use the approximation from Zhi-Sheng and Chen (2017). Their method derives these estimators based on the generalized gamma distribution using log-moment estimators. This method is not fully equivalent to Maximum

Likelihood, but the authors show that the found approximations have similar performance and efficiency. The parameter estimates are given by the following formulas.

$$\begin{aligned}\hat{\kappa} &= \frac{n\sum_{i=1}^{n} x_i}{n\sum_{i=1}^{n} x_i \ln(x_i) - \sum_{i=1}^{n} \ln(x_i) \sum_{i=1}^{n} x_i} \\ \hat{\theta} &= \frac{1}{n^2}\left(n\sum_{i=1}^{n} x_i \ln(x_i) - \sum_{i=1}^{n} \ln(x_i) \sum_{i=1}^{n} x_i\right)\end{aligned} \tag{A.8}$$

A.3 Log-normal Distribution

Finally, we derive the Maximum Likelihood Estimator for the log-normal distribution. This distribution has two parameters, those being μ and σ. These parameters denote the mean and variance of the distribution respectively. Again, we start off by writing down the density of the distribution as shown in Equation A.9.

$$f(x) = \frac{1}{x\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(\ln(x)-\mu)^2}{2\sigma^2}\right] \tag{A.9}$$

This density is converted into a likelihood function. We simplify this equation slightly to make it easier to take the logarithm in the next step.

$$\begin{aligned}L(\mu,\sigma^2) &= \prod_{i=1}^{n}\left(\frac{1}{x_i\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(\ln(x_i)-\mu)^2}{2\sigma^2}\right]\right) \\ &= (2\pi\sigma^2)^{-n/2}\prod_{i=1}^{n}\left(\frac{1}{x_i} \exp\left[-\frac{(\ln(x_i)-\mu)^2}{2\sigma^2}\right]\right)\end{aligned} \tag{A.10}$$

We obtain the log-likelihood by taking the logarithm of Equation A.10. The result of this can be seen below.

$$\begin{aligned}\mathcal{L}(\mu,\sigma^2) &= \ln\left((2\pi\sigma^2)^{-n/2}\prod_{i=1}^{n}\left(\frac{1}{x_i} \exp\left[-\frac{(\ln(x_i)-\mu)^2}{2\sigma^2}\right]\right)\right) \\ &= -\frac{n}{2}\ln(2\pi\sigma^2) - \sum_{i=1}^{n}\ln(x_i) - \sum_{i=1}^{n}\frac{(\ln(x_i)-\mu)^2}{2\sigma^2}\end{aligned} \tag{A.11}$$

Using the log-likelihood function, we can now take the derivative with respect to the parameters and set these derivatives to 0. When we solve for μ and σ, we get the Maximum Likelihood Estimators. The result of this can be seen in Equations A.12 and A.13

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mu} &= \sum_{i=1}^{n} \frac{2(\ln(x_i) - \mu)}{2\sigma^2} \\
&\Rightarrow \sum_{i=1}^{n} \frac{(\ln(x_i) - \hat{\mu})}{\sigma^2} = 0 \\
&\Rightarrow \sum_{i=1}^{n} \ln(x_i) = \sum_{i=1}^{n} \hat{\mu} \\
&\Rightarrow \sum_{i=1}^{n} \ln(x_i) = n\hat{\mu} \\
&\Rightarrow \hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} \ln(x_i)
\end{aligned} \tag{A.12}$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \sigma^2} &= -\frac{n}{2}\frac{1}{\sigma^2} + \sum_{i=1}^{n} \frac{(\ln(x_i) - \hat{\mu})^2}{2(\sigma^2)^2} \\
&\Rightarrow \sum_{i=1}^{n} \frac{(\ln(x_i) - \hat{\mu})^2}{2(\hat{\sigma^2})^2} - \frac{n}{2\hat{\sigma^2}} = 0 \\
&\Rightarrow \sum_{i=1}^{n} \frac{(\ln(x_i) - \hat{\mu})^2}{\hat{\sigma^2}} = n \\
&\Rightarrow \hat{\sigma^2} = \frac{1}{n} \sum_{i=1}^{n} \big(\ln(x_i) - \hat{\mu}\big)^2
\end{aligned} \tag{A.13}$$

Which correspond to the Maximum Likelihood Estimators that were used in Section 3.

B Deriving Method of Moments Estimators

To model the event profiles in Section 3.2 we made use of the beta distribution. The parameters of this distribution were calculated on different points of the mass curve using a method of moments estimator. In this appendix, we derive this estimator and show that the equations used (as stated in Equation 3.10) are the correct ones.

B.1 Beta Distribution

Deriving the method of moments estimator for a given distribution involves taking the analytical expressions for the first few moments and rewriting them into parameter estimates. Since the beta distribution has two parameters, we need expressions for the mean and variance. As stated by Owen (2008), these expressions are as follows.

$$\begin{aligned} \mathbb{E}(x) &= \frac{\alpha}{\alpha+\beta} \\ \mathrm{Var}(X) &= \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \end{aligned} \tag{B.1}$$

We approximate the mean and variance by the sample mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and sample variance $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$. Using this approximation we can write one of the parameters (β) in terms of the other parameter α and sample mean.

$$\begin{aligned} \bar{x} &= \frac{\alpha}{\alpha+\beta} \\ &\Rightarrow \alpha+\beta = \frac{\alpha}{\bar{x}} \\ &\Rightarrow \beta = \frac{\alpha}{\bar{x}} - \alpha \end{aligned} \tag{B.2}$$

Using this relation between the distribution parameters we can derive an expression for α in terms of the sample mean and variance.

$$
\begin{aligned}
s^2 &= \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \\
&\Rightarrow (\alpha+\beta)^2(\alpha+\beta+1)s^2 = \alpha\beta \\
&\Rightarrow (\alpha+\frac{\alpha}{\bar{x}}-\alpha)^2(\alpha+\frac{\alpha}{\bar{x}}-\alpha+1)s^2 = \alpha\left(\frac{\alpha}{\bar{x}}-\alpha\right) \\
&\Rightarrow \left(\frac{\alpha}{\bar{x}}\right)^2\left(\frac{\alpha}{\bar{x}}+1\right)s^2 = \frac{\alpha^2}{\bar{x}}-\alpha^2 \\
&\Rightarrow \alpha^2\left(\frac{1}{\bar{x}^2}\right)\left(\frac{\alpha}{\bar{x}}+1\right)s^2 = \alpha^2\left(\frac{1}{\bar{x}}-1\right) \\
&\Rightarrow \frac{\alpha}{\bar{x}}+1 = \left(\frac{1}{\bar{x}}-1\right)\left(\frac{\bar{x}^2}{s^2}\right) \\
&\Rightarrow \frac{\alpha}{\bar{x}}+1 = \frac{\bar{x}-\bar{x}^2}{s^2} \\
&\Rightarrow \frac{\alpha}{\bar{x}} = \frac{\bar{x}(1-\bar{x})}{s^2}-1 \\
&\Rightarrow \alpha = \bar{x}\left(\frac{\bar{x}(1-\bar{x})}{s^2}-1\right)
\end{aligned}
\tag{B.3}
$$

Substituting this into the found relation between α and β gives us an analytical expression for β as well.

$$
\begin{aligned}
\beta &= \frac{\alpha}{\bar{x}}-\alpha \\
&\Rightarrow \beta = \frac{\alpha}{\bar{x}}-\frac{\alpha\bar{x}}{\bar{x}} \\
&\Rightarrow \beta = \alpha\left(\frac{1-\bar{x}}{\bar{x}}\right) \\
&\Rightarrow \beta = \left(\frac{1-\bar{x}}{\bar{x}}\right)\bar{x}\left(\frac{\bar{x}(1-\bar{x})}{s^2}-1\right) \\
&\Rightarrow \beta = (1-\bar{x})\left(\frac{\bar{x}(1-\bar{x})}{s^2}-1\right)
\end{aligned}
\tag{B.4}
$$

Which combined gives us the same parameter estimates as stated under Equation 3.10.

$$
\begin{aligned}
\hat{\alpha} &= \bar{x}\left(\frac{\bar{x}(1-\bar{x})}{s^2}-1\right) \\
\hat{\beta} &= (1-\bar{x})\left(\frac{\bar{x}(1-\bar{x})}{s^2}-1\right)
\end{aligned}
\tag{B.5}
$$

C Estimated parameter values for the event profiles

Table 8: Estimated parameter values for the dimensionless mass curves

In the table below we can observe the estimated parameter for the dimensionless mass curves at 24 linearly spaced points along the curve. These parameter estimates are obtained using the method of moments estimators as derived in Appendix B and are used to simulate event profiles in our influent simulator.

Percentile	Alpha	Beta
0.04	1.816	53.707
0.08	1.718	46.784
0.12	1.604	39.090
0.16	1.510	32.879
0.20	1.525	30.263
0.24	1.645	28.809
0.28	1.646	25.885
0.32	1.362	19.216
0.36	1.990	25.696
0.40	1.914	22.783
0.44	2.057	22.471
0.48	1.507	14.423
0.52	2.107	18.891
0.56	3.151	28.033
0.60	2.630	20.560
0.64	2.712	18.804
0.68	2.426	14.326
0.72	3.948	22.260
0.76	3.278	15.434
0.80	3.973	16.128
0.84	6.095	20.898
0.88	8.182	21.758
0.92	15.192	27.959
0.96	35.710	33.947

D Parameter optimization results for deep reinforcement learning algorithms

D.1 Trust Region Policy Optimization (TRPO)

Table 9: Results TRPO single agent, without influent forecast

This table shows the results of the parameter analysis for the TRPO agent in the single agent environment without an influent forecast. The batch size is denoted by N_{batch} whereas λ denotes the step size parameter. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $> thres$ and $< thres$ columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	λ	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
5000	0.98	3548.37	254.18	67.23	**0.22**	5.499	**0.170**	0.892
5000	0.99	**3665.49**	128.47	**24.64**	0.76	6.408	0.314	0.688
10,000	0.98	3529.16	218.12	74.35	0.31	5.418	0.189	0.920
10,000	0.99	3068.07	**108.17**	114.94	0.25	5.650	0.365	1.637
20,000	0.98	3570.85	226.25	59.40	0.53	5.584	0.247	**0.952**
20,000	0.99	3030.13	183.38	128.7	0.55	**5.355**	0.341	1.717

Table 10: Results TRPO single agent, with influent forecast

This table shows the results of the parameter analysis for the TRPO agent in the single agent environment with an influent forecast. The batch size is denoted by N_{batch} whereas λ denotes the step size parameter. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $> thres$ and $< thres$ columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	λ	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
5000	0.98	3713.02	73.83	9.96	0.72	6.566	**0.059**	0.643
5000	0.99	3055.28	194.28	122.24	**0.65**	**5.779**	0.204	1.731
10,000	0.98	**3729.27**	**67.76**	**7.68**	0.78	6.535	0.074	0.694
10,000	0.99	3554.08	97.11	55.71	0.73	6.699	0.141	**0.754**
20,000	0.98	3722.70	82.87	10.66	0.79	6.379	0.094	0.734
20,000	0.99	3724.99	83.91	8.73	0.78	6.480	0.072	0.676

Table 11: Results TRPO multi agent, without influent forecast

This table shows the results of the parameter analysis for the TRPO agent in the multi agent environment without an influent forecast. The batch size is denoted by N_{batch} whereas λ denotes the step size parameter. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $> thres$ and $< thres$ columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	λ	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
5000	0.98	7886.91	1743.11	187.99	6.47	7.083	0.103	1.764
5000	0.99	8693.79	1325.32	90.54	0.89	7.681	0.118	1.209
10,000	0.98	8591.04	1756.94	104.03	0.83	**7.054**	**0.096**	1.338
10,000	0.99	9014.11	**830.36**	**49.94**	0.84	7.958	0.105	**1.018**
20,000	0.98	**9024.52**	1029.03	50.93	**0.73**	7.454	0.098	1.040
20,000	0.99	8949.55	931.65	55.56	0.77	7.982	0.120	1.030

Table 12: Results TRPO multi agent, with influent forecast

This table shows the results of the parameter analysis for the TRPO agent in the multi agent environment with an influent forecast. The batch size is denoted by N_{batch} whereas λ denotes the step size parameter. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $> thres$ and $< thres$ columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	λ	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
5000	0.98	9074.67	522.90	44.95	0.74	8.395	0.073	0.889
5000	0.99	9157.95	399.78	25.28	0.75	9.173	0.119	0.859
10,000	0.98	9196.81	365.16	25.61	0.75	8.572	0.071	0.817
10,000	0.99	9202.37	366.82	22.11	0.77	8.734	0.109	0.819
20,000	0.98	9191.97	463.62	27.40	0.73	8.256	0.075	0.854
20,000	0.99	9191.63	422.35	22.07	0.73	8.779	0.101	0.777

D.2 Advantage Actor Critic (A2C)

Table 13: Results A2C single agent, without influent forecast

This table shows the results of the parameter analysis for the A2C agent in the single agent environment without an influent forecast. The batch size is denoted by N_{batch} whereas Ent.coef. denotes the entropy coefficient. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $> thres$ and $< thres$ columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	Ent.coef.	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
25	0.01	877.56	400.09	497.05	4.31	2.754	0.725	4.875
25	0.02	**2008.01**	346.04	**274.38**	33.35	4.953	**0.385**	**2.722**
50	0.01	1676.93	140.87	335.97	0.49	5.348	0.589	3.386
50	0.02	-670.89	**59.95**	682.40	**0.09**	**2.651**	1.179	7.151
100	0.01	-662.70	84.72	682.42	0.57	2.808	1.160	7.118
100	0.02	-821.73	160.16	727.62	0.14	2.889	1.122	7.340

Table 14: Results A2C single agent, with influent forecast

This table shows the results of the parameter analysis for the A2C agent in the single agent environment with an influent forecast. The batch size is denoted by N_{batch} whereas Ent.coef. denotes the entropy coefficient. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $> thres$ and $< thres$ columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	Ent.coef.	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
25	0.01	3171.68	707.63	177.29	3.45	4.279	0.041	1.462
25	0.02	3124.32	688.35	180.61	9.01	4.405	**0.039**	1.395
50	0.01	**3212.48**	580.19	**166.63**	1.46	4.614	0.062	**1.328**
50	0.02	2499.38	696.40	291.77	**0.55**	**3.806**	0.225	2.435
100	0.01	1202.27	**61.75**	390.99	6.18	4.114	0.676	4.228
100	0.02	2452.21	95.64	202.30	4.78	5.027	0.372	2.459

Table 15: Results A2C multi agent, without influent forecast

This table shows the results of the parameter analysis for the A2C agent in the multi agent environment without an influent forecast. The batch size is denoted by N_{batch} whereas Ent.coef. denotes the entropy coefficient. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $> thres$ and $< thres$ columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	Ent.coef.	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
25	0.01	-479.52	**274.52**	537.28	278.49	**4.253**	0.952	5.244
25	0.02	-7.99	361.66	743.70	63.79	6.564	1.002	6.224
50	0.01	1002.04	422.44	629.22	50.84	6.661	1.018	5.437
50	0.02	1371.49	345.53	630.52	59.88	8.000	0.874	5.247
100	0.01	5681.09	339.55	208.39	31.78	13.086	0.336	2.216
100	0.02	**8150.40**	472.52	**64.73**	**17.65**	13.215	**0.080**	**0.850**

Table 16: Results A2C multi agent, with influent forecast

This table shows the results of the parameter analysis for the A2C agent in the multi agent environment with an influent forecast. The batch size is denoted by N_{batch} whereas Ent.coef. denotes the entropy coefficient. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $> thres$ and $< thres$ columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	Ent.coef.	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
25	0.01	2598.14	431.35	348.32	240.30	**7.072**	0.589	3.284
25	0.02	2098.07	407.23	447.24	171.21	7.462	0.706	3.948
50	0.01	5795.61	292.43	311.36	30.97	8.731	0.435	2.561
50	0.02	4817.10	388.32	426.73	**2.40**	9.385	0.573	3.634
100	0.01	8310.18	316.53	**50.49**	8.86	13.260	0.119	**0.708**
100	0.02	**8447.57**	**281.19**	72.83	7.27	11.876	**0.115**	0.678

D.3 Proximal Policy Optimization 1 (PPO1)

Table 17: Results PPO1 single agent, without influent forecast

This table shows the results of the parameter analysis for the PPO1 agent in the single agent environment without an influent forecast. The batch size is denoted by N_{batch} whereas λ denotes the step size parameter and Clip.coef. denotes the clipping coefficient. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	λ	Clip.coef.	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
500	0.10	0.95	3671.91	256.92	28.21	0.34	5.728	**0.027**	0.576
500	0.10	0.98	3671.94	330.27	31.07	0.06	5.440	0.044	0.760
500	0.20	0.95	3511.32	388.30	48.25	16.81	5.673	0.030	0.581
500	0.20	0.98	3707.41	189.19	17.46	0.27	6.008	0.029	0.467
1000	0.10	0.95	3631.21	248.00	33.90	6.35	5.452	0.029	0.676
1000	0.10	0.98	3680.83	293.98	26.98	0.30	5.549	0.045	0.690
1000	0.20	0.95	3611.37	313.17	51.51	0.10	5.191	**0.027**	**0.850**
1000	0.20	0.98	3674.75	339.58	28.76	0.01	5.510	0.035	0.702
2000	0.10	0.95	**3756.00**	**105.45**	**9.75**	**0.00**	5.473	0.034	0.638
2000	0.10	0.98	3696.31	236.06	17.39	**0.00**	6.166	0.081	0.661
2000	0.20	0.95	3443.86	455.13	102.56	0.04	**4.816**	0.043	1.094
2000	0.20	0.98	3690.99	332.30	25.92	0.02	5.451	0.061	0.766

Table 18: Results PPO1 single agent, with influent forecast

This table shows the results of the parameter analysis for the PPO1 agent in the single agent environment with an influent forecast. The batch size is denoted by N_{batch} whereas λ denotes the step size parameter and Clip.coef. denotes the clipping coefficient. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	λ	Clip.coef.	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
500	0.10	0.95	3751.77	68.81	4.52	0.43	6.138	0.026	0.415
500	0.10	0.98	3743.38	95.64	5.56	0.44	6.211	0.043	0.426
500	0.20	0.95	3757.39	57.09	3.70	0.40	6.079	0.025	0.410
500	0.20	0.98	3749.15	52.22	**3.10**	0.62	6.392	0.032	0.349
1000	0.10	0.95	3766.81	69.28	5.46	0.04	**5.672**	**0.022**	0.583
1000	0.10	0.98	3764.26	55.73	3.79	0.05	5.960	0.036	0.510
1000	0.20	0.95	**3766.88**	58.37	4.77	0.07	5.766	0.027	0.571
1000	0.20	0.98	3762.50	54.62	3.68	0.04	6.021	0.035	0.491
2000	0.10	0.95	3761.82	54.42	4.51	0.01	5.827	0.034	0.546
2000	0.10	0.98	3628.25	**50.27**	3.67	**0.00**	9.541	0.172	0.530
2000	0.20	0.95	3755.59	58.62	5.75	0.13	5.847	0.024	0.493
2000	0.20	0.98	3747.17	61.75	7.19	**0.00**	6.201	0.090	**0.678**

Table 19: Results PPO1 multi agent, without influent forecast

This table shows the results of the parameter analysis for the PPO1 agent in the multi agent environment without an influent forecast. The batch size is denoted by N_{batch} whereas λ denotes the step size parameter and Clip.coef. denotes the clipping coefficient. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	λ	Clip.coef.	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
500	0.10	0.95	8220.23	1666.81	165.73	1.35	6.233	0.079	1.481
500	0.10	0.98	6400.21	1511.96	240.58	6.98	13.038	0.277	2.598
500	0.20	0.95	8177.01	1599.16	169.90	0.78	6.174	0.080	1.546
500	0.20	0.98	6931.56	1366.57	245.31	3.52	10.914	0.165	2.449
1000	0.10	0.95	8361.82	1483.51	157.74	0.20	**5.725**	**0.074**	1.409
1000	0.10	0.98	7997.77	1077.25	**64.17**	0.05	16.382	0.168	1.032
1000	0.20	0.95	**8979.62**	872.06	67.22	0.23	6.679	0.052	**0.963**
1000	0.20	0.98	7394.86	1299.35	190.60	0.01	11.305	0.188	2.052
2000	0.10	0.95	6118.64	1224.50	285.33	**0.00**	13.330	0.314	2.853
2000	0.10	0.98	-2251.45	**266.49**	769.83	**0.00**	9.093	1.356	8.132
2000	0.20	0.95	8297.67	1009.76	78.62	**0.00**	13.042	0.151	1.244
2000	0.20	0.98	1604.15	758.18	494.06	**0.00**	23.736	0.781	5.478

Table 20: Results PPO1 multi agent, with influent forecast

This table shows the results of the parameter analysis for the PPO1 agent in the multi agent environment with an influent forecast. The batch size is denoted by N_{batch} whereas λ denotes the step size parameter and Clip.coef. denotes the clipping coefficient. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	λ	Clip.coef.	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
500	0.10	0.95	9021.96	629.88	58.86	1.86	**7.080**	0.053	**0.872**
500	0.10	0.98	8135.95	819.40	111.63	5.93	14.838	0.137	1.281
500	0.20	0.95	9113.27	463.42	45.93	0.74	7.549	0.051	0.757
500	0.20	0.98	8859.93	687.45	61.26	0.68	9.803	0.090	1.072
1000	0.10	0.95	9250.20	453.01	28.18	0.18	7.283	**0.040**	0.803
1000	0.10	0.98	8135.38	235.03	11.73	0.07	17.826	0.126	0.611
1000	0.20	0.95	**9346.92**	307.96	13.43	0.21	7.730	**0.040**	0.646
1000	0.20	0.98	8598.87	581.84	78.65	4.58	10.066	0.099	1.282
2000	0.10	0.95	8019.61	169.57	**7.87**	0.20	20.031	0.145	0.603
2000	0.10	0.98	6748.81	180.17	20.28	**0.00**	28.631	0.214	0.781
2000	0.20	0.95	8937.08	227.29	11.20	0.02	13.281	0.109	0.652
2000	0.20	0.98	5832.08	**152.18**	107.81	0.05	24.293	0.220	1.654

D.4 Proximal Policy Optimization 2 (PPO2)

Table 21: Results PPO2 single agent, without influent forecast

This table shows the results of the parameter analysis for the PPO2 agent in the single agent environment without an influent forecast. The batch size is denoted by N_{batch} whereas λ denotes the step size parameter and Clip.coef. denotes the clipping coefficient. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	λ	Clip.coef.	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
500	0.10	0.95	3212.28	720.49	170.71	0.42	**4.288**	0.041	1.432
500	0.10	0.98	3450.79	486.16	99.08	**0.00**	4.864	0.097	1.115
500	0.20	0.95	3355.90	583.07	128.20	0.16	4.565	**0.033**	1.250
500	0.20	0.98	3689.85	252.30	25.22	0.06	5.485	0.052	0.711
1000	0.10	0.95	3304.57	654.52	142.28	0.35	4.513	0.036	1.305
1000	0.10	0.98	3644.88	276.99	38.93	**0.00**	5.404	0.117	0.807
1000	0.20	0.95	3578.94	346.99	61.10	0.30	5.043	0.037	**0.931**
1000	0.20	0.98	3631.28	218.21	41.23	0.11	5.592	0.091	0.692
2000	0.10	0.95	3648.95	247.56	38.64	0.21	5.327	0.042	0.757
2000	0.10	0.98	3730.18	148.18	13.68	0.15	5.581	0.104	0.679
2000	0.20	0.95	**3740.88**	**121.93**	**8.67**	0.21	5.738	0.061	0.516
2000	0.20	0.98	3696.67	202.18	20.45	0.19	5.829	0.138	0.654

Table 22: Results PPO2 single agent, with influent forecast

This table shows the results of the parameter analysis for the PPO2 agent in the single agent environment with an influent forecast. The batch size is denoted by N_{batch} whereas λ denotes the step size parameter and Clip.coef. denotes the clipping coefficient. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	λ	Clip.coef.	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
500	0.10	0.95	3451.64	484.02	99.32	0.86	4.710	0.030	1.139
500	0.10	0.98	3647.38	262.78	42.34	**0.11**	5.116	0.038	**0.952**
500	0.20	0.95	3633.75	191.08	35.08	1.37	5.952	**0.021**	0.505
500	0.20	0.98	3542.05	320.61	66.88	0.84	5.377	0.030	0.863
1000	0.10	0.95	3382.94	492.63	121.29	0.32	**4.596**	0.036	1.222
1000	0.10	0.98	3670.83	168.63	29.08	0.17	5.790	0.080	0.749
1000	0.20	0.95	3405.66	458.24	104.04	3.71	4.968	0.036	1.058
1000	0.20	0.98	3725.56	103.19	**9.30**	0.21	6.121	0.074	0.579
2000	0.10	0.95	3612.10	237.94	48.65	0.47	5.375	0.039	0.803
2000	0.10	0.98	**3730.33**	103.41	10.31	0.21	5.923	0.069	0.634
2000	0.20	0.95	3626.94	224.00	42.57	0.29	5.549	0.063	0.817
2000	0.20	0.98	3705.19	**96.17**	10.82	0.28	6.601	0.131	0.640

Table 23: Results PPO2 multi agent, without influent forecast

This table shows the results of the parameter analysis for the PPO2 agent in the multi agent environment without an influent forecast. The batch size is denoted by N_{batch} whereas λ denotes the step size parameter and Clip.coef. denotes the clipping coefficient. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	λ	Clip.coef.	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
500	0.10	0.95	7837.73	1426.63	219.40	0.88	**5.950**	0.083	1.731
500	0.10	0.98	8212.68	1146.84	117.08	**0.03**	8.947	0.157	1.259
500	0.20	0.95	8103.04	1207.39	179.18	0.42	6.552	0.061	1.527
500	0.20	0.98	7732.16	1374.99	217.79	0.11	7.435	0.167	1.747
1000	0.10	0.95	8664.64	1130.25	111.31	0.04	6.539	**0.060**	1.170
1000	0.10	0.98	**8892.67**	**910.48**	**67.28**	0.08	7.617	0.136	**1.022**
1000	0.20	0.95	8028.77	1405.16	186.53	0.66	6.711	0.104	1.535
1000	0.20	0.98	8617.69	937.71	102.80	0.44	8.166	0.098	1.091
2000	0.10	0.95	8580.90	1273.20	117.31	0.15	6.814	0.062	1.250
2000	0.10	0.98	8439.05	935.81	125.94	0.11	7.499	0.119	1.334
2000	0.20	0.95	8689.67	1113.24	103.95	0.11	6.830	0.082	1.174
2000	0.20	0.98	8860.15	913.09	69.82	0.18	7.875	0.131	1.032

Table 24: Results PPO2 multi agent, with influent forecast

This table shows the results of the parameter analysis for the PPO2 agent in the multi agent environment with an influent forecast. The batch size is denoted by N_{batch} whereas λ denotes the step size parameter and Clip.coef. denotes the clipping coefficient. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively. The best score for each performance measure is shown in bold.

N_{batch}	λ	Clip.coef.	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
500	0.10	0.95	8855.91	803.82	81.21	1.12	6.983	**0.047**	1.056
500	0.10	0.98	8781.88	421.92	59.96	**0.01**	10.531	0.165	**0.959**
500	0.20	0.95	8486.93	981.45	134.04	0.54	6.662	0.057	1.235
500	0.20	0.98	8949.71	429.43	54.86	0.11	8.705	0.119	0.905
1000	0.10	0.95	8505.51	933.91	137.69	0.25	**6.392**	0.069	1.269
1000	0.10	0.98	9037.83	448.65	39.31	0.11	8.844	0.114	0.921
1000	0.20	0.95	8693.23	677.89	103.62	0.14	7.393	0.086	1.140
1000	0.20	0.98	9076.05	**346.76**	36.04	0.22	8.830	0.111	0.773
2000	0.10	0.95	9151.65	549.49	36.21	0.10	7.608	0.054	0.918
2000	0.10	0.98	**9156.80**	442.26	**25.20**	0.11	8.764	0.098	0.805
2000	0.20	0.95	9119.41	424.83	38.97	0.17	8.056	0.067	0.797
2000	0.20	0.98	9071.37	404.49	36.02	0.42	8.918	0.128	0.874

E Parameter optimization results for deep reinforcement learning algorithms

E.1 Rule-based

Table 25: Sensitivity results rule-based, single agent, without influent forecast

This table shows the performance of the rule-based agent in the sensitivity analysis for the single agent environment without an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively.

Parameter	Change	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Max. volume	Up	3673.40	37.78	1.43	0.00	7.891	0.226	0.644
	Down	3651.52	40.70	2.00	0.00	8.306	0.219	0.648
Min. action	Up	3547.45	35.51	1.55	0.00	9.981	0.212	0.615
	Down	3721.76	39.29	1.74	0.00	7.491	0.235	0.687
Max. action	Up	3689.45	30.28	0.90	0.00	8.416	0.222	0.627
	Down	3597.89	60.26	3.43	0.00	7.642	0.227	0.688
Influent per mm	Up	3657.01	51.42	2.70	0.00	8.341	0.224	0.658
	Down	3673.01	26.33	0.72	0.00	7.771	0.224	0.637
κ	Up	3667.66	38.51	1.65	0.00	7.993	0.224	0.643
	Down	3662.86	39.43	1.66	0.00	8.108	0.224	0.648
σ	Up	3660.25	38.97	1.66	0.00	8.181	0.224	0.650
	Down	3669.54	37.74	1.63	0.00	7.945	0.224	0.641

Table 26: Sensitivity results rule-based, single agent, with influent forecast

This table shows the performance of the rule-based agent in the sensitivity analysis for the single agent environment with an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively.

Parameter	Change	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Max. volume	Up	3673.40	37.78	1.43	0.00	7.891	0.226	0.644
	Down	3651.52	40.70	2.00	0.00	8.306	0.219	0.648
Min. action	Up	3547.45	35.51	1.55	0.00	9.981	0.212	0.615
	Down	3721.76	39.29	1.74	0.00	7.491	0.235	0.687
Max. action	Up	3689.45	30.28	0.90	0.00	8.416	0.222	0.627
	Down	3597.89	60.26	3.43	0.00	7.642	0.227	0.688
Influent per mm	Up	3657.01	51.42	2.70	0.00	8.341	0.224	0.658
	Down	3673.01	26.33	0.72	0.00	7.771	0.224	0.637
κ	Up	3667.66	38.51	1.65	0.00	7.993	0.224	0.643
	Down	3662.86	39.43	1.66	0.00	8.108	0.224	0.648
σ	Up	3660.25	38.97	1.66	0.00	8.181	0.224	0.650
	Down	3669.54	37.74	1.63	0.00	7.945	0.224	0.641

Table 27: Sensitivity results rule-based, multi agent, without influent forecast

This table shows the performance of the rule-based agent in the sensitivity analysis for the multi agent environment without an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively.

Parameter	Change	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Max. volume	Up	9026.66	126.61	2.11	0.00	12.155	0.230	0.657
	Down	8995.52	132.78	2.91	0.00	12.457	0.222	0.662
Min. action	Up	8713.16	122.37	2.28	0.00	14.144	0.214	0.627
	Down	9157.64	129.22	2.53	0.00	11.977	0.239	0.694
Max. action	Up	9132.67	103.87	1.43	0.00	12.724	0.224	0.637
	Down	8748.42	193.47	5.10	0.00	11.600	0.231	0.714
Influent per mm	Up	8984.39	167.46	4.11	0.00	12.917	0.227	0.678
	Down	9052.05	95.11	1.20	0.00	11.599	0.227	0.646
κ	Up	9029.47	130.99	2.43	0.00	12.123	0.227	0.657
	Down	9015.43	128.81	2.42	0.00	12.342	0.227	0.662
σ	Up	9002.03	131.34	2.44	0.00	12.502	0.226	0.664
	Down	9035.22	126.91	2.43	0.00	12.040	0.227	0.655
Agents	Up	22524.20	313.24	2.27	0.00	31.872	0.226	0.656

E.2 Value iteration

Table 28: Sensitivity results value iteration, single agent, without influent forecast

This table shows the performance of the value iteration agent in the sensitivity analysis for the single agent environment without an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively.

Parameter	Change	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Max. volume	Up	3806.09	44.00	5.09	0.00	4.711	0.020	0.855
	Down	3771.15	54.31	9.91	0.00	5.365	0.029	0.951
Min. action	Up	3625.12	39.53	3.15	0.00	8.315	0.022	0.358
	Down	3880.72	48.13	5.51	0.14	4.710	0.059	0.847
Max. action	Up	3829.22	38.41	5.20	0.00	5.123	0.023	0.905
	Down	3720.13	66.42	8.26	0.00	4.928	0.023	0.924
Influent per mm	Up	3787.51	59.80	8.47	0.00	5.162	0.024	0.921
	Down	3797.09	36.23	4.94	0.00	4.872	0.023	0.897
κ	Up	3796.60	46.32	6.25	0.00	4.933	0.023	0.903
	Down	3788.91	47.80	6.89	0.00	5.104	0.023	0.919
σ	Up	3790.51	46.98	6.56	0.00	5.107	0.023	0.923
	Down	3796.50	46.97	6.42	0.00	4.908	0.023	0.893

Table 29: Sensitivity results value iteration, single agent, with influent forecast

This table shows the performance of the value iteration agent in the sensitivity analysis for the single agent environment with an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively.

Parameter	Change	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Max. volume	Up	3815.15	28.37	1.19	0.12	4.853	0.019	0.807
	Down	3792.40	32.33	2.86	0.13	5.499	0.029	0.936
Min. action	Up	3630.65	29.27	0.78	0.11	8.369	0.022	0.309
	Down	3895.29	28.61	1.36	0.14	4.983	0.072	0.889
Max. action	Up	3840.77	24.19	1.17	0.11	5.198	0.023	0.883
	Down	3735.27	46.34	2.75	0.14	5.052	0.023	0.890
Influent per mm	Up	3804.60	39.59	2.55	0.15	5.324	0.024	0.901
	Down	3807.81	21.57	1.04	0.06	4.961	0.023	0.866
κ	Up	3808.99	28.46	1.63	0.11	5.061	0.023	0.868
	Down	3802.78	30.75	2.08	0.11	5.228	0.023	0.902
σ	Up	3803.03	30.27	2.14	0.11	5.232	0.023	0.905
	Down	3809.48	28.31	1.67	0.10	5.033	0.023	0.858

Table 30: Sensitivity results value iteration, multi agent, without influent forecast

This table shows the performance of the value iteration agent in the sensitivity analysis for the multi agent environment without an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively.

Parameter	Change	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Max. volume	Up	9473.39	143.69	11.65	0.04	6.454	0.062	0.927
	Down	9403.68	165.39	16.33	0.08	7.486	0.056	0.973
Min. action	Up	9163.13	121.12	7.45	0.21	9.138	0.063	0.603
	Down	9512.17	176.76	18.03	0.14	7.912	0.253	0.984
Max. action	Up	9595.22	115.30	9.58	0.06	7.287	0.056	0.941
	Down	9147.21	224.96	18.12	0.04	6.749	0.061	0.985
Influent per mm	Up	9416.96	193.44	16.31	0.07	7.439	0.061	0.968
	Down	9480.57	110.72	9.69	0.04	6.389	0.054	0.942
κ	Up	9458.73	150.58	12.42	0.03	6.784	0.058	0.949
	Down	9438.55	151.37	13.42	0.11	7.065	0.055	0.958
σ	Up	9440.11	150.18	13.13	0.13	7.097	0.056	0.96
	Down	9458.64	151.95	12.76	0.02	6.730	0.059	0.944
Agents	Up	23644.60	332.42	9.24	0.04	17.758	0.041	0.929

E.3 Proximal Policy Optimization 1 (PPO1)

Table 31: Sensitivity results PPO1, single agent, without influent forecast

This table shows the performance of the PPO1 agent in the sensitivity analysis for the single agent environment without an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively.

Parameter	Change	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Max. volume	Up	3781.20	53.94	3.47	0.00	5.551	0.027	0.507
	Down	3650.38	161.64	38.52	0.00	5.617	0.041	0.837
Min. action	Up	3606.06	103.48	7.71	0.12	8.327	0.037	0.371
	Down	3783.71	153.53	20.33	0.00	4.409	0.031	0.757
Max. action	Up	3705.19	258.72	31.98	0.50	5.410	0.034	0.756
	Down	3513.54	289.06	61.88	0.02	5.144	0.032	0.876
Influent per mm	Up	3724.75	166.64	18.03	0.02	5.741	0.032	0.696
	Down	3724.39	119.02	20.54	0.00	5.276	0.036	0.706
κ	Up	3615.39	286.35	52.51	0.04	5.119	0.041	0.865
	Down	3759.94	78.39	7.24	0.00	5.792	0.035	0.599
σ	Up	3704.09	177.13	24.30	0.00	5.427	0.036	0.769
	Down	3643.94	243.07	44.74	0.02	5.112	0.038	0.813

Table 32: Sensitivity results PPO1, single agent, with influent forecast

This table shows the performance of the PPO1 agent in the sensitivity analysis for the single agent environment with an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively.

Parameter	Change	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Max. volume	Up	3764.05	55.45	4.73	0.00	5.720	0.025	0.509
	Down	3753.76	57.69	5.62	0.05	5.939	0.038	0.576
Min. action	Up	3621.14	43.64	2.66	0.04	8.455	0.031	0.313
	Down	3831.85	46.27	3.12	0.00	5.036	0.026	0.393
Max. action	Up	3798.32	43.43	3.45	0.02	5.939	0.029	0.503
	Down	3695.17	71.82	6.00	0.00	5.592	0.026	0.617
Influent per mm	Up	3753.08	64.73	5.53	0.08	6.129	0.030	0.519
	Down	3768.59	38.55	2.83	0.12	5.696	0.021	0.496
κ	Up	3768.65	52.56	4.13	0.00	5.783	0.033	0.549
	Down	3758.10	52.02	4.00	0.02	6.023	0.026	0.500
σ	Up	3759.81	53.21	4.19	0.03	5.943	0.027	0.518
	Down	3766.40	55.19	4.53	0.10	5.756	0.029	0.510
Agents	Up							

Table 33: Sensitivity results PPO1, multi agent, without influent forecast

This table shows the performance of the PPO1 agent in the sensitivity analysis for the multi agent environment without an influent forecast. The performance on seven different performance measures is shown, which are defined in Section 5. The average reward over all runs is denoted by $\overline{Reward}$, with σ_{Reward} denoting the standard deviation of these rewards. The $>$ *thres* and $<$ *thres* columns denote the number of violations of the upper and lower threshold respectively. The standard deviation of the actions of the agent is denoted by $\sigma_{Actions}$. Finally, the lower and upper 5% quantile are represented by $Q_{0.05}$ and $Q_{0.95}$ respectively.

Parameter	Change	$\overline{Reward}$	σ_{Reward}	$>$ thres	$<$ thres	$\sigma_{Actions}$	$Q_{0.05}$	$Q_{0.95}$
Max. volume	Up	7646.66	1376.97	205.76	0.24	8.60	0.152	1.828
	Down	8576.78	708.27	31.72	0.02	16.36	0.210	0.966
Min. action	Up	8125.72	918.79	63.91	0.02	14.51	0.154	1.099
	Down	8259.81	862.26	98.61	0.00	12.03	0.171	1.479
Max. action	Up	6894.52	1203.66	287.08	0.72	10.29	0.277	2.672
	Down	8246.36	924.26	45.20	0.00	10.57	0.109	0.797
Influent per mm	Up	6597.93	1361.58	263.74	1.31	12.85	0.229	2.654
	Down	6091.66	1219.43	323.48	2.23	11.82	0.318	3.180
κ	Up	8096.26	966.14	116.05	0.00	13.46	0.159	1.337
	Down	8207.51	740.16	109.79	0.00	13.87	0.170	1.198
σ	Up	8927.90	770.54	38.31	0.00	10.54	0.173	0.890
	Down	8160.18	1315.12	120.28	0.04	11.18	0.168	1.429
Agents	Up	8902.47	2190.96	421.49	0.38	43.825	0.394	3.728

References

Acreman, M. C. (1990). A simple stochastic model of hourly rainfall for Farnborough, England. *Hydrological Sciences*, 35(2):119–148.

Adam, S., Busoniu, L., and Babuska, R. (2011). Experience Replay for Real-Time Reinforcement Learning Control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 42(2):201–212.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):835–846.

Bellman, R. E. (1954). The Theory of Dynamic Programming. *Bulletin of the American Mathematical Society*, 60(6):503–516.

Bellman, R. E. (1957). A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6(5):679–684.

Bellman, R. E. and Dreyfus, S. E. (1962). *Applied Dynamic Programming*. Princeton University Press, 1st edition.

Blazkova, S. and Beven, K. (1997). Flood frequency prediction for data limited catchments in the Czech Republic using a stochastic rainfall model and TOPMODEL. *Journal of Hydrology*, 195(1):256–278.

Brigandi, G. and Aronica, G. T. (2019). Generation of Sub-Hourly Rainfall Events through a Point Stochastic Rainfall Model. *Geosciences*, 9(5):1–18.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. *International Conference on Machine Learning*, 48:1329–1338.

Favis-Mortlock, D. and Boardman, J. (1995). Nonlinear responses of soil erosion to climate change: a modelling study on the UK South Downs. *Catena*, 25(1):365–387.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232.

Garcia-Guzman, A. and Aranda-Oliver, E. (1993). A Stochastic model of dimensionless hyetograph. *Water Resources Research*, 29(7):2363–2370.

Huff, F. A. (1967). Time Distribution of Rainfall in Heavy Storms. *Water Resources Research*, 3(4):1007–1019.

Kilsby, C. G., Jones, P. D., Burton, A., Ford, A. C., Fowler, H. J., Harpham, C., James, P., Smith, A., and Wilby, R. L. (2007). A daily weather generator for use in climate change studies. *Environmental Modelling & Software*, 22(12):1705–1719.

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic Algorithms. *Advances in Neural Information Processing Systems*, 12:1008–1014.

Koninklijk Nederlands Meteorologisch Instituut (2020). Dagwaarden neerslagstations. `https://www.knmi.nl/nederland-nu/klimatologie/monv/reeksen`. Accessed on 22-01-2020.

Lin, L. J. (1992). Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8(3):293–321.

Mnih, V., Badia, A., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. *International Conference on Machine Learning*, pages 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *NIPS Deep Learning Workshop*, pages 1–9.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, 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*, 518:529–533.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized Sweeping: Reinforcement Learning With Less Data and Less Time. *Machine Learning*, 13(1):103–130.

Organisation for Economic Co-operation and Development (OECD) (2019). Hello, world: Artificial intelligence and its use in the public sector. `https://oecd-opsi.org/wp-content/uploads/2019/11/AI-Report-Online.pdf`. Accessed on 29-06-2020.

Owen, C. E. B. (2008). Parameter Estimation for the Beta Distribution. Master's thesis, Brigham Young University.

Richardson, C. W. (1981). Stochastic Simulation of Daily Precipitation, Temperature and Solar Radiation. *Water Resources Research*, 17(1):182–190.

Rivolta, G., Marzano, F. S., Coppola, E., and Verdecchia, M. (2006). Artificial neural-network technique for precipitation nowcasting from satellite imagery. *Advances in Geosciences*, 7:97–103.

Saltelli, A., Tarantola, S., Campolongo, F., and Ratto, M. (2004). *Sensitivity Analysis in Practice*. John Wiley & Sons Ltd.

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal*, 3(3):535–554.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized Experience Replay. *Proceedings of the International Conference on Learning Representations*, 4.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust Region Policy Optimization. *International Conference on Machine Learning*, 31:1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *Computing Research Repository*, pages 1–12.

Semenov, M. A. and Barrow, E. M. (1997). Use of a Stochastic Weather Generator in the Development of Climate Change Scenarios. *Climatic Change*, 35(4):397–414.

Shi, X., Chen, Z., Wang, H., and Yeung, D. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in Neural Information Processing Systems*, 28:802–810.

Shi, X., Gao, Z., Lausen, L., Wang, H., and Yeung, D. (2017). Deep learning for precipitation nowcasting: A benchmark and a new model. *Advances in Neural Information Processing Systems*, 30:5617–5627.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3:9–44.

Sutton, R. S. (1991). Dyna, an Integrated Architecture for Learning, Planning, and Reacting. *SIGART Bulletin*, 2(4):160–163.

Sutton, R. S. and Barto, A. G. (1981). Toward a Modern Theory of Adaptive Networks: Expectation and Prediction. *Psychological Review*, 88(2):135–170.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. Adaptive computation and machine learning series. MIT Press, Cambridge, MA, 2nd edition.

Tang, C. J. and Chen, S. X. (2009). Parameters Estimation and Bias Correction for Diffusion Processes. *Journal of Econometrics*, 149:65–81.

Uhlenbeck, G. E. and Ornstein, L. S. (1930). On the Theory of Brownian Motion. *Physical Review*, 36:823–841.

van Hasselt, H. (2010). Double Q-learning. *Advances in Neural Information Processing Systems*, 23:2613–2621.

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-learning. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2094–2100.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Williams, J. R. (1990). The erosion-productivity impact calculator (EPIC) model: a case history. *Philosophical Transactions of the Royal Society B*, 329:421–428.

Williams, R. J. (1992). Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3):229–256.

Witten, I. H. (1977). An Adaptive Optimal Controller for Discrete-Time Markov Environments. *Information and Control*, 34(4):286–295.

Zahraei, A., Hsu, K., Sorooshian, S., Gourley, G. G., Lakshmanan, V., Y., H., and Bellerby, T. (2012). Quantitative precipitation nowcasting: A lagrangian pixel-based approach. *Atmospheric Research*, 118:418–434.

Zhi-Sheng, Y. and Chen, N. (2017). Closed-Form Estimators for the Gamma Distribution Derived From Likelihood Equations. *The American Statistician*, 71(2):177–181.