RL4Water: Reinforcement Learning environment for Water Management

Krzysztof Muniak[1]
Supervisor(s): Pradeep Murukannaiah[1]**, Zuzanna Osika**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Krzysztof Muniak
Final project course: CSE3000 Research Project
Thesis committee: Pradeep Murukannaiah, Zuzanna Osika, Luis Miranda da Cruz

RL4Water: Reinforcement Learning environment for Water Management

Krzysztof Muniak[*]
Supervisors: Pradeep Murukannaiah, Zuzanna Osika
EEMCS, Delft University of Technology
Delft, Mekelweg 5, 2628 CD
`k.muniak@student.tudelft.nl`

Abstract

Efficient management of water resources is increasingly critical in the face of growing challenges such as climate change and population growth. This research paper introduces RL4Water, an adaptable framework for simulating water management systems using multi-objective reinforcement learning (MORL). Adhering to the Gymnasium API standard, RL4Water ensures seamless integration with existing MORL algorithms. The framework includes diverse facility classes to accurately model the physical components of water networks. Its generalizability is enhanced by allowing users to modify both the physical properties of these components and the key features of the MORL simulations. RL4Water's capabilities are demonstrated through two case studies: simulations of the Nile River and the Susquehanna River, validating its accuracy and flexibility in managing both large, distributed water systems and centralized systems with complex reservoirs. By bridging the gap between water management and reinforcement learning, RL4Water offers a unified platform for developing and researching water management simulations.

1 Introduction

The management of water resources in the face of climate change presents an urgent challenge that demands innovative solutions [3, 10]. It requires us to model complex problems, such as water flow control of a river, to make informed decisions when planning new facilities or controlling existing ones.

Traditional methods for addressing water management issues often rely on custom algorithms tailored to specific problems, which often share many similarities with reinforcement learning (RL) techniques. Reinforcement learning is an area of machine learning where an agent learns to make decisions by interacting with a given environment. For its actions, the agent receives feedback in the form of a reward, allowing it to improve decision-making policy over time. This makes reinforcement learning suitable for sequential decision-making problems, such as control of water management systems, that change over time and need continuous adaptation.

Complex water management systems often include different objectives that need to be balanced. These goals might include minimizing water deficit in a region, maximizing hydropower generation or ensuring steady water flow through the system. Modelling the problem as a multi-objective allows splitting the decision-making process of which objectives should be prioritized, from the simulation calculating the sets of feasible solutions. As a result, people making decisions based on the simulation results get more insightful knowledge and do not rely on the frameworks used for the simulations to weigh different objectives that are individual to each problem.

[*]An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Existing simulations in the water management field often do not explicitly utilize reinforcement learning methods, nor do they leverage specific algorithms developed within the field of computer science. This presents an opportunity to bridge the gap between these two domains and explore the potential benefits of applying reinforcement learning to address water management challenges. Existing literature provides detailed descriptions of various water system components and can serve as a foundation for developing an RL-based water management framework [15, 8].

In our research, we want to address this gap by answering the following research question: **How can the water systems simulation be generalised for multi-objective reinforcement learning?** To do that we will develop a new framework for integration of various reinforcement learning algorithms into complex water management systems. We want to create a solution that will allow future researchers to easily customise and adapt it to their specific water management problems, removing the need to develop custom algorithms. The implementation will test the Nile River and Susquehanna River simulations [15, 8], by connecting them with a multi-objective reinforcement learning (MORL) [5] algorithm to demonstrate the applicability of the developed framework in real-world scenarios.

The rest of the paper has the following structure. Section 2 provides the background information on the reinforcement learning. The methodology chosen to conduct the research is described in Section 3. The overview of the RL4Water framework and its generalization properties is outlined in Section 4. Section 5 provides the application of the framework to respectively, the Nile River and Susquehanna River case studies. The responsible research part of the paper is described in Section 6. Section 7 discusses the created framework. The conclusions and future works are provided in Section 8.

2 Background

2.1 Water management environments

The Nile River and Susquehanna River environments, detailed in Section 5, are designed to optimize reservoir control policies within complex water management systems. They incorporate multiple facilities, such as power plants, demand districts, and catchments, to simulate the components of real-world water systems. These systems can span large regions, often crossing national boundaries and involving multiple stakeholders, each with its objectives and priorities that need to be addressed.

2.2 EMODPS

To facilitate the need to optimize multiple objectives within the simulations, the Evolutionary multi-objective direct policy search (EMODPS) algorithm was developed by the water management community [4]. It combines the direct policy search (DPS) with multi-objective evolutionary algorithms (MOEA) to come up with Pareto-approximate operating policies [16]. The DPS maps the state and observations from the given environment into action, while the MOEA search through the solution space using evolutionary algorithms.

2.3 Multi-objective reinforcement learning

The MORL simulations consist of two main components, shown in Figure 1: agent and environment. The environment defines the world in which the agent is placed. At each step of the simulation, it provides the agent with the state that includes all information the agent can observe about the environment. The agent, based on the observed state, decides on an action to take. The action changes the state of the environment and provides the agent with a set of rewards.

In traditional reinforcement learning, the agent tries to optimize a single objective. However, in MORL, the agent needs to consider multiple goals at the same time, often requiring trade-offs between them. This aspect is critical in complex decision-making scenarios, such as water management, where different factors such as water supply, flood risk or power production need to be balanced.

To take all of these objectives into account, a Multi-Objective Natural Evolution Strategies (MONES) [5] was integrated with the RL4Water framework. MONES is designed to produce a Pareto-front of alternative policies. The Pareto-front represent a set of non-dominated solutions, allowing the user to choose the most suitable policy based on their specific priorities. It separates the simulation logic of finding policies from the analysis of the trade-offs between objectives.

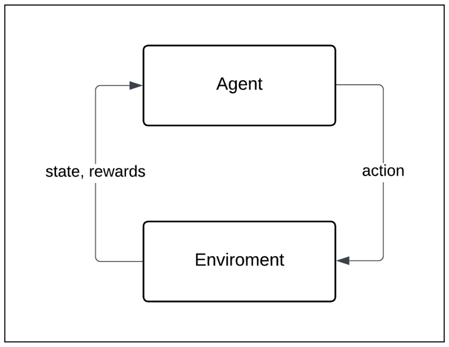

Figure 1: Multi-objective reinforcement learning interaction of the agent with the environment.

2.4 Gymnasium API

To allow easy integration of the environment with existing reinforcement learning algorithms the choice to use the Gymnasium API standard [1] was made. The Gymnasium library was developed by OpenAI and allows for easy separation of the environment from the agent and training algorithms. The library imposes a unified structure of the environment to guarantee compatibility across different code bases. Each environment class must implement the following methods:

- *step*: Updates the environment with action and returns the next observation, reward, whether the action caused environment termination or truncation and additional information from the environment.
- *reset*: Resets the environment to its initial state.
- *render*: Renders the visual environment, allowing users to better understand the simulation process.
- *close*: Closes the environment, it is needed for handling connections to external software or libraries.

Additionally, the environments have a few attributes that help to define the space of their implementation:

- *action_space*: Space representing all valid actions that can be taken in the environment.
- *observation_space*: Space representing all valid observations of the environment state.
- *reward_space*: Space representing all possible rewards received from the environment.

3 Methodology

The initial stage of the research involved a detailed study of the Nile River simulation [15] to understand the underlying principles of water management systems and identify their core component. This step was necessary to create a bridge to integrate the reinforcement learning techniques into the water management field. Following the study, an analysis of the Gymnasium API was done. The goal was to understand the structure the framework needs to have to be compatible with existing MORL algorithms.

The next step focused on adopting the Nile River simulation to be compatible with the Gymnasium library and connecting it with MONES. At that stage, the goal was not to create a framework that could be applied to a variety of water management systems, but to show the feasibility of the approach.

To ensure the correctness of the translated simulation and allow for an easy generalization of the software at later stages, a verification process was created. It involved comparing the behaviour of the simulation before and after adaptation to assert identical results. The verification process allowed for preventing bugs in the development at further steps.

With the base implementation the most important process was started, generalizing the framework to apply to a wide range of water management problems. The objective was to make the software easily customizable, by allowing users to configure component's parameters and adjust their behaviours without modifying the core codebase.

Afterwards, to validate the capabilities developed at the previous stage, a simulation of the Susquehanna River was re-implemented using the RL4Water framework. This step demonstrated the developed solution can be adapted to different water management systems, proving its generalization capabilities.

Finally, the RL4Water framework was compared with another reinforcement learning software developed for controlling water management systems [7]. The differences in approaches, as well as the results produced by both of these solutions, were discussed.

4 Water management reinforcement learning framework

The implementation of the RL4Water framework was done in Python programming language, because of its ease of use and vast amount of packages for data science and machine learning. Moreover, both Nile River and Susquehanna River simulations were developed in Python, making it a natural choice as the programming language for the framework.

4.1 Overview of the RL4Water framework

The RL4Water framework provides a **WaterManagementSystem** class that supports the Gymnasium API [1]. It is responsible for interacting with the agent by providing information about the state of the environment and rewards for action taken by the agent. To model-specific components of the water systems, it utilizes other classes divided into three groups: flows, facilities and controlled facilities. The general structure of the framework, showing the relations between all of its components, is presented in Figure 2.

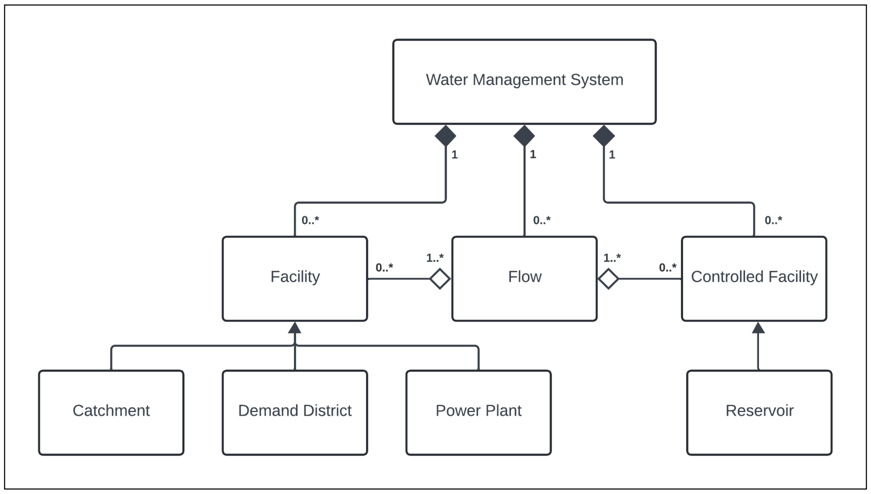

Figure 2: Structure of the RL4Water framework.

The **Flow** class represents the water flow in the system. It allows users to define connections between different facilities. It can connect multiple sources of inflow to multiple outflow destinations, allowing for the creation of complex water flow networks. The class can be easily customized with a few parameters to better match the needs of specific simulations:

- *max_capacity*: The maximum capacity of the flow. Can be used to model flooding scenarios by providing negative rewards or terminating the simulation. Additionally, it can be set to infinity to disable the feature.
- *evaporation_rate*: The evaporation rate of the flow that caused the water loss.
- *delay*: The number of time steps it takes the water to get from the source to the destination, it can be used to model long connections.

- *default_outflow*: The default amount of water flowing through the flow when the *delay* parameter was used. The *default_outflow* is used for the first number of steps equal to the value set for the *delay* parameter.

Both **Facility** and **ControlledFacility** classes represent components of the simulation that can influence the water system and provide a reward based on their state. In contrast to the **Facility** class that represents a static part of the system that cannot be controlled and only responds to the state of the environment, the **ControlledFacility** class can receive an action from the agent to actively interact with the state of the environment. The current RL4Water framework provides an implementation of three different facilities: catchment, demand district, power plant and the implementation of one controlled facility: reservoir.

The **Catchment** class is the simplest facility that allows the modelling of additional inflows to the system. It can be initialized with *all_water_accumulated* parameter that takes a list of inflows the catchment provides for each step of the simulation.

The **DemandDistrict** class models a facility that consumes water from the system to meet its needs. The default implementation of this facility provides a reward based on the proportion of the demand met. The demand of the facility per each step of the simulation can be specified by *all_demand* initialization parameter.

The **PowerPlant** class represents a facility with the ability to produce power from the water flow. To do that it needs to be connected to a reservoir that provides it with stored water available for power production. The class needs multiple parameters to define its physical attributes necessary for calculating power production capabilities:

- *efficiency*: Efficiency coefficient of the power plant.
- *head_start_level*: The water level required for the power plant to generate water.
- *max_capacity*: Maximum power capacity of the power plant.
- *operating_hours*: The number of hours the power plant operates for.
- *min_turbine_flow*: Minimum flow through the turbines to produce hydro energy.
- *max_turbine_flow*: Maximum flow through the turbines for hydro energy production.
- *water_usage*: Ratio of water flowing through the turbines that is consumed.

The **Reservoir** class allows for controlling the water flow through the system. It can accumulate the water in its storage and release it depending on the action chosen by an agent. It allows for complex storage shapes, e.g. cones, to be modelled using functions that describe the relation of the amount of water stored to its level and surface area. The functions can be passed to the system as following sets of data points, that will be further interpolated by the **Reservoir** class:

- *storage_to_minmax_release_relation*: Relation of the stored water amount in m^3 to the minimum and maximum amount of water that can or needs to be released from the reservoir.
- *storage_to_level_relation*: Relation of the stored water amount in m^3 to the water height level in meters, used for calculating the power production.
- *storage_to_surface_relation*: Relation of the stored water amount in m^3 to the surface area in m², used for calculating the amount of evaporated water.

Moreover, the **Reservoir** class can be initialized with a few parameters that define additional attributes of the object:

- *max_capacity*: Maximum amount of water that can be stored in the reservoir.
- *stored_water*: Initial amount of water stored in the reservoir.
- *evap_rates*: Evaporation rates of the reservoir's storage per simulation step, that can be used to represent environmental effects.
- *integration_timestep_size*: Time units per each reservoir's state should be re-evaluated.

Finally, the reservoir can control water release in multiple directions. This is achieved by providing the class with one action for each of the connected destination facilities. The reservoir then calculates the total possible outflow and determines the proportion of target outflows (*split_release_ratio*), based on the ratio of the received actions. This parameter is afterwards used by the **Flow** class to redistribute water to downstream facilities.

4.2 Generalization features of the RL4Water framework

The RL4Water framework can be easily adapted to model different water management scenarios by selecting suitable attributes for the facility classes. As a result, the framework becomes resilient to changes in the physical properties of the model components, reducing the amount of work needed to adjust the simulation. Apart from the option to input the class attributes, the framework allows for easy control of the simulation parameters, such as actions, observations, objectives and time frame of the simulation run.

4.2.1 Custom action space

As every **ControlledFacility** class can be controlled by the agent, it needs to accept actions that define its behaviour. The type and the space of the allowed actions are determined by the *action_space* attribute, which needs to be set for each controlled facility during its initialization. For the **Reservoir** class implemented in the framework, the action space represents all possible values of the amount of water the reservoir can release. While it's not possible to change the action type to a different than the amount of released water, the space that defines the action can be easily modified to be a discrete or continuous space, using respectively **Discrete** and **Box** classes available from the Gymnasium library.

To ensure seamless integration between the MONES algorithm and the **WaterManagementSystem** class, a **ReshapeArrayAction** wrapper was added. MONES outputs a single array of actions for the environment, without subdividing them for each controlled facility. The implemented wrapper converts this action array into a dictionary, using the facility name as the key and the corresponding actions as values. It automatically extracts all necessary information, supporting multidimensional action spaces and simplifying the process for the user.

4.2.2 Custom observations

Each of the **ControlledFacility** classes requires information about the type and space of observations it can provide. That information is defined by the *observation_space* attribute, which is passed during the class initialization. The **Reservoir** model, by default, expects a single, one-dimensional, continuous space with the lower and upper bounds equal respectively to the minimum and maximum amount of water that can be observed in the storage. The observation value that is provided to the agent at each step of the simulation is defined by the *determine_observation* method, which out of the box, returns the amount of water stored in the reservoir's storage. To modify the observation outputted by the environment, one only needs to adjust the *determine_observation* method to return the required variable or set of variables and make sure that the controlled facility class gets initialized with the space correctly defining the new observation.

4.2.3 Custom objectives

Different water management scenarios want to maximize different objectives, thus it is essential to make them interchangeable. In the RL4Water framework, the objectives are implemented using two components: rewards and objective functions.

The rewards are calculated by the *determine_reward* method that needs to be implemented for each **Facility** and **ControlledFacility** class. The method defines which class attributes are relevant to compute the reward, e.g. water demand and inflow for the demand district or water level in storage for the reservoir. Then the method can utilize the *objective_function* attribute, passed during the class initialization, allowing it to compute the final reward from the chosen components. There are nine different objective functions implemented in the **Objective** class that allow for various objective calculations.

The objective of the simulation can be easily modified by overriding the *determine_reward* method to use a different set of attributes or by swapping the *objective_function* during class initialization.

Moreover, the framework can combine objectives from different classes together. It can be done by setting the same *objective_name* attribute for all the classes that should share the same objective. As a result, simulation components that contribute to the same goal can provide a singular reward, e.g. multiple demand districts that provide water for the same area.

4.2.4 Custom time step unit

Often in reinforcement learning the agent interacts with the environment only for a limited amount of steps. To set that limit, for simulation done using the RL4Water framework, the **TimeLimit** wrapper class from the Gymnasium library can be used. It takes as arguments the environment and the maximum number of steps we want the simulation to run for.

Additionally, different water management systems might want to run for different periods, using different step sizes, e.g. the Nile River simulation was run over 20 year time period with each step being one month [15], while the Susquehanna River simulation was run over the frame of one year with a step size of four hours [17]. To facilitate that the RL4Water framework allows to set the size of each step with the *timestep_size* attribute of **WaterManagementSystem** class. The time step size can represent any amount of time from multiple years to a second. Moreover, to allow for accurate computations, that take into account leap years and the specific number of days each month has, a *start_date* argument needs to be provided to specify when the simulation starts.

Finally, to allow reservoirs to take more granularly controlled actions, the RL4Water framework implements an integration step. The integration step divides the main step of the simulation into multiple smaller phases, each with the size set by *integration_timestep_size* attribute of the **Reservoir** class. Furthermore, the actions are applied at each integration stage, instead of the simulations step, ensuring that all physical constraints of the reservoir are respected, as it would not always be possible when taking release actions once over the large period of the whole simulation step, e.g. one month for the Nile River simulation [15].

5 Case studies

The capabilities of the RL4Water framework are shown through two case studies: the re-implementation of the Nile River simulation [15] and the Susquehanna River simulation [8], further referred to as original simulations. Each case study introduces the setting of the environment and describes its structure based on the used framework components. To validate the accuracy of the RL4Water framework, its behaviour was compared to that of the original Nile River simulation.

5.1 Case study of Nile River simulation

The case study of Nile River simulation focuses mainly on simulating the Blue Nile part of the basin. The next two largest tributaries: the White Nile and Atbara River are only modelled as water inflows and do not include any facilities. The area covered by the case study spans over territory of three African countries: Egypt, Sudan and Ethiopia. Each of these actors in the region has different facilities and objectives that are modelled in the case study.

Following the original simulation, the assumption was made, that the agent has complete information on the state of the river basin [15]. Moreover, the agent single-handedly decides on the release actions for all reservoirs. As a result, the scenario modelled in the simulation provides a set of solutions trying to compromise between objectives of different actors in the region.

The simulation horizon was set to twenty years, from 2022 until 2042, with each step of the simulation lasting one month. The size of the integration step was set to half an hour, to ensure that the physical constraints of the reservoirs are not violated [15].

The topological overview of all components of the Nile River simulation model is shown in Figure 3. It demonstrates all facilities used in the case study, as well as the flow between them.

Reservoirs There are four reservoirs modelled in this case study: High Aswan Dam (HAD), Sennar, Roseries and Grand Ethiopian Renaissance Dam (GERD). The HAD as the most downstream reservoir is responsible for controlling the water supply of Egypt and provides significant economic gains for the country [9]. Both Sennar and Roseries dams are located in Sudan and play a significant part in meeting the irrigation demands of the region [13]. Finally, the GERD as the most upstream reservoir is located in Ethiopia. With its primary goal to increase access to electricity across the

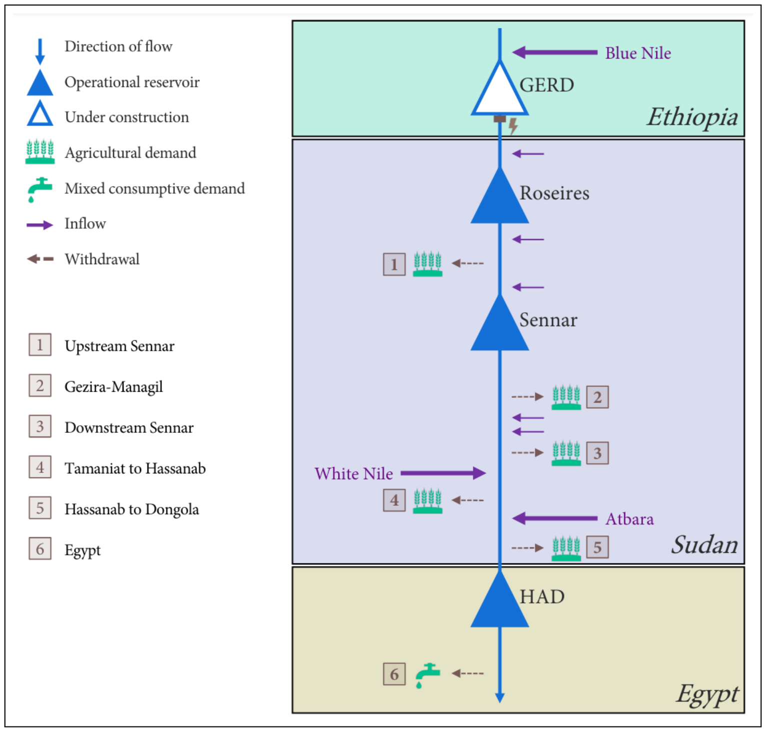

Figure 3: The topology of the Nile River case study detailing all components of the simulation. Adapted with permission from: [15]

country, it will be the largest hydroelectric power plant on the continent [11, 13]. The initialization setup of the GERD reservoir is shown in Code Listing 1.

```
GERD_reservoir = Reservoir(
    name="GERD",
    observation_space=Box(low=0, high=80_000_000_000),
    action_space=Box(low=0, high=10000),
    integration_timestep_size=relativedelta(minutes=30),
    objective_function=Objective.no_objective,
    stored_water=15_000_000_000,
    evap_rates=np.loadtxt(evap_rates_file),
    storage_to_minmax_rel=np.loadtxt(storage_to_minmax_rel_file),
    storage_to_level_rel=np.loadtxt(storage_to_level_rel_file),
    storage_to_surface_rel=np.loadtxt(storage_to_surface_rel_file),
)
```

Code Listing 1: Code for initialization of the Reservoir class.

Power Plants Even though all four reservoirs have the capacity to generate hydroelectric power, our simulation will focus only on the power plant connected to the GERD. Additionally, the original simulation did not include power production for other countries as an objective, making the other three power plants irrelevant to the modelled water management system [15]. The GERD power plant got loaded using the code from the Code Listing 2.

```
GERD_power_plant = PowerPlant(
    name="GERD_power_plant",
    objective_function=Objective.scalar_identity(1 / 1000000000),
    objective_name="ethiopia_power",
    efficiency=0.93,
    min_turbine_flow=0,
    max_turbine_flow=4320,
    head_start_level=507,
    max_capacity=6000,
    reservoir=GERD_reservoir,
)
```

Code Listing 2: Code for initialization of the PowerPlant class.

Demand Districts The case study simulation models six different irrigation districts. Five of them are part of Sudan, and one is located in the territory of Egypt, shown in the Code Listing 3. The Sudanese districts are Upstream Sennar, Gezira-Managil, Downstream Sennar, Tamaniat to Hassanab, and Hassanab to Dongola, while the single Egyptian district aggregates the irrigation needs of the entire country. The districts were selected based on their demand aggregation zones [12].

```
Egypt_irr_system = DemandDistrict(
    name="Egypt_irr",
    all_demand=np.loadtxt(egypt_irrigation_demand_file),
    objcetive_function=Objective.deficit_minimised,
    objective_name="egypt_deficit_minimised",
)
```

Code Listing 3: Code for initialization of the DemandDistrict class.

Catchments The simulation represents the Blue Nile as the primary inflow to the Nile River by modelling it using the **Inflow** class. The other two main tributaries of the Nile River are implemented as catchments, with the Atbara inflow setup demonstrated in the Code Listing 4. The case study includes an additional five smaller catchments, all located within Sudan, that contribute to the inflow of the water management system.

```
Atbara_catchment = Catchment(
    name="atbara_catchment",
    all_water_accumulated=np.loadtxt(atbara_inflow_file),
)
```

Code Listing 4: Code for initialization of the Catchment class.

Objectives Four different objectives were implemented to meet the specific needs of each country located in the eastern Nile River basin. Egypt's primary focus is maintaining the reliability of the water supply by minimizing the deficit in water demand of their irrigation district. Moreover, to keep the energy supply, they want to keep the water level of HAD above the minimum level of 159 meters needed to produce the hydroelectric power [14]. Similarly to Egypt, Sudan also strives to minimize the aggregated water deficit across all its irrigation districts. Finally, Ethiopia prioritizes maximizing hydropower generation from the GERD, as it was the main reason for its construction.

5.2 Case study of Susquehanna River simulation

The Susquehanna River is the longest river on the East Coast of the United States, spanning over three states: New York, Pennsylvania and Maryland. The case study models the lower Susquehanna River basin, particularly the largest nonfederal dam Conowingo constructed in 1928. Since the reservoir is responsible for controlling a large share of river flow in the basin, it impacts multiple shareholders with different objectives.

In 1968 the Muddy Run Pumped Storage Hydroelectric Facility was connected to the Conowingo reservoir. The original simulation modelled the additional facility as controlled by a predefined set of rules, making it not suitable for control by a reinforcement learning agent. Additionally, it did not directly contribute to any of the modelled objectives, resulting in a decision not to include the Muddy facility in the modelled environment.

The environment was simulated for one year, with one step having a duration of four hours. Additionally, the included reservoir had an integration step with a size of one hour allowing it to calculate storage and evaporation changes more granularly.

The overview of the Susquehanna River simulation components, which include one reservoir and four demand districts, is illustrated in Figure 4.

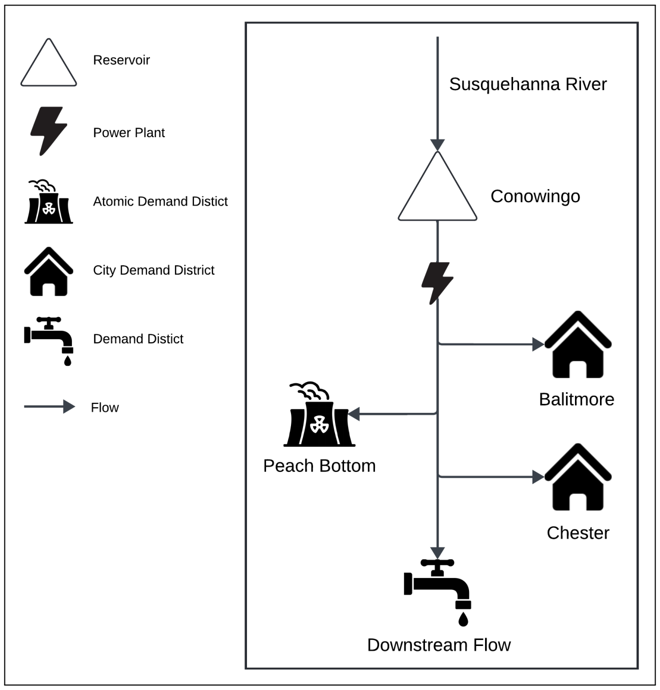

Figure 4: The topology of the Susquehanna River case study detailing all components of the simulation.

Reservoir The Conowingo reservoir is responsible for controlling the downstream water flow, supplying Chester and Baltimore cities and providing water for cooling of the Peach Bottom atomic power plant. Moreover, the reservoir's body of water is used for recreation. In contrast to the reservoirs described in the Nile River case study, which always had a single destination, the Conowingo dam distributes the water flow in multiple directions.

Power Plant The simulation includes a single power plant connected only to the downstream outflow of the Conowingo reservoir. The maximum capacity of the power plant was defined as the combined capacity of its thirteen turbines, and the minimum corresponds to that of the smallest turbine.

Demand Districts The lower Susquehanna River basin contains three demand districts modelled in the simulation: Chester, Baltimore and the downstream region. Both cities have water demands, changing over the time frame of the simulation, that need to be met to sustain their population. On the other hand, the downstream region of the basin needs to meet minimum flow requirements defined by the Federal Energy Regulatory Commission (FERC) to conserve fishing resources [2].

Objectives The Susquehanna River environment modelled four objectives: hydropower revenue, water supply, environment and recreation [8]. The hydropower revenue is computed as a product of the power production (MWh) at the Conowingo hydropower plant and the power prices (US$/MWh). Baltimore, Chester and the Peach Bottom atomic power plant share the same objective of water supply calculated as a water supply ratio to water demand. The environmental objective (J^{SI}) is described as the daily average shortage index in relation to the minimum flow required by FERC, defined using the Equation 1. The quadratic formulation discourages large water deficits while allowing for minor shortages. The final objective maximizes the recreational value of the Conowingo reservoir by promoting the state when the water level in the storage is above the minimum level of 32.5 m (106.5 ft) during the weekend.

$$J^{SI} = -\left(\frac{\max(Demand - Supply, 0)}{Demand}\right)^2 \tag{1}$$

5.3 Verification

To verify the correctness of the RL4Water framework, the behaviour of the Nile River case study was compared with the behaviour of the original simulation. The Nile River simulation was chosen as it included more different components than the Susquehanna River Simulation. Moreover, its original simulation had a clearer codebase with a more detailed explanation of the behaviour in the paper, allowing for a more detailed comparison.

The validation process was conducted by running the original simulation for the set period of 240 steps of one month each, corresponding to the 20 years of the desired simulation horizon. At each step, the action used to control each of the reservoirs, alongside the state of the environment, was logged into a file. The state of the simulation was defined by the amount of water stored and released from each of the four reservoirs (HAD, Sennar, Roseries, GERD) and the power produced by the GERD.

Afterwards, the case study simulation was run using the same simulation horizon and actions described above. At each step, the state of the water management system was asserted to match the variables previously logged in the file, with a relative difference of less than $1\mathrm{e}{-10}$. The only modification made to the RL4Water framework during the validation process was skipping the leap years by incrementing the *current_date* attribute of the **WaterManagementSystem** by one year each time the leap year was encountered. This alteration was made to match the number of days per month used in the original simulation that did not support leap years.

The Nile River environment, developed using the RL4Water framework, takes 28.47 seconds to run a 20-year simulation horizon. This timing exclusively measures the environment's step method using pre-determined actions, excluding the action selection process by the agent. The benchmarking experiment was conducted on a laptop with a Windows 10 operating system and an AMD Ryzen 7 4800H processor.

6 Responsible Research

6.1 Reproducibility

In this research, we adhere to Findable, Accessible, Interoperable, Reusable (FAIR) data principles. To ensure the reproducibility and allow for collaborative development of the RL4Water framework, we have made the entire codebase publicly available on GitHub[2]. The repository can be accessed by anyone, allowing for transparency and increasing community involvement. Users can comment on the software and provide suggestions for further improvements. The chosen developer platform supports version control, allowing everyone to view the exact changes made to the framework to validate the research process. All data used for creating the simulations and running the environments is included in the published code, ensuring an integrated solution that can be analyzed and processed by others. The software was developed using clear naming and comments, making it straightforward to reuse or modify. Moreover detailed descriptions of the framework's properties are provided in Section 4, ensuring that users can fully understand and utilize the capabilities of the framework.

[2] https://github.com/krmuniak/rl4water

6.2 Framework usage

The presented paper describes and provides simulations of two water management systems: the Nile River and the Susquehanna River. The data used for creating these environments was directly adapted from previous studies [15, 8] without any modifications. It is important to note that the presented simulation should not be used as an indicator for making decisions about reservoir control in these regions. The case studies were intended to demonstrate the feasibility of the framework to model such scenarios, but they do not validate the accuracy of the results. While the RL4Water codebase provides a platform for modelling and experimentation, selection of the input data to correctly and validation of the results, are necessary for practical decision-making in real-world water management systems.

7 Discussion

This section discusses the similarities and differences between the RL4Water framework developed using a top-down approach and another MORL framework based on a bottom-up methodology [7]. Moreover, it analyses the possibilities of extending the existing codebase with new functionalities and considers the limitations of the developed framework.

7.1 Similar work

The RL4Water framework was developed by first adapting the Nile River simulation for MORL and then generalizing it to meet the needs of similar water management problems. This top-down approach focused on creating modular classes that simplify the setup process of the environments. The pre-defined methods and classes make it easier to implement water management scenarios based on reservoir control.

In contrast, the bottom-up framework was developed by extracting core principles from different water management problems. This approach focused on creating a generalized system based on water flows between abstract nodes. While this method offers greater flexibility and customization, it requires users to define functions that specify the interactions of these nodes with the environment. Thus, the bottom-up approach requires more effort during setup and configuration, making it less straightforward.

Both frameworks aim to address similar problems but from different perspectives. The RL4Water framework provides an out-of-the-box solution with pre-built components, while the bottom-up framework offers more customization options at the expense of increased complexity.

7.2 Extendability of the framework

The RL4Water framework was designed to be easily extended with new facilities. This was achieved through a unified integration of **Facility** and **ControlledFacility** with the **WaterManagementSystem** and **Flow** classes, specified in the step method. As a result, adding a new facility does not require any changes to the core integration. Each facility only needs to implement specific functionalities that define its properties and behaviour. For the Facility class, the necessary methods are:

- *determine_reward*: Provides the reward of the facility based on its current state.
- *determine_consumption*: Calculates the amount of water the facility uses, which allows for the automatic calculation of the outflow.

For the Controlled Facility class, the required methods are:

- *determine_reward*: Provides the reward of the facility based on its current state.
- *determine_outflow*: Calculates the water outflow from the facility based on an action provided to the method.
- *determine_observation*: Provides the observations of the facility, allowing the agent to make more informed decisions.
- *is_terminated*: Defines whether the simulation should be terminated based on the state of the facility.

By implementing these methods, new facilities can be seamlessly integrated into the RL4Water framework, improving its versatility and applicability in various water management scenarios.

7.3 Limitations

The RL4Water framework has a few limitations. One is the lack of unit tests, as the current implementation only includes an end-to-end test of the entire Nile River simulation. Adding more tests would help to minimize the risk of unnoticed bugs and simplify further development. This would allow for more detailed verification of the codebase and more precise identification of potential error sources.

Another limitation is the lack of static validation for the simulation setup. The framework depends on users to understand its structure and configure simulations correctly. Users can set up environments with invalid parameters without receiving any warnings, leading to incorrect simulation results or unexpected crashes. For instance, users might create facilities that are not connected to the system or set incompatible parameters, such as a power plant with a *min_turbine_flow* greater than its *max_turbine_flow*. Introducing an assertion step at the beginning of the environment setup would mitigate these issues, reducing the dependency on the user's understanding of the framework and making it more robust.

8 Conclusions

The developed framework bridges the gap between water management and reinforcement learning fields by demonstrating a practical application of MORL to real-world reservoir control problems. It creates a unified system for building water management simulations, enabling easier research in the field.

The RL4Water software offers a general solution supporting simulations of water management problems with multiple objectives. It handles complex water flow network systems that consist of facilities with different specifications and purposes, including reservoirs, power plants, catchments and numerous demand districts. By using Gymnasium API, the framework enables seamless integration with reinforcement learning algorithms, such as MONES.

One of the most essential features of the constructed platform is its generalization capabilities. It allows for easy setup of component parameters and modification reinforcement learning properties, including action and observation spaces, objectives, and simulation horizons. This flexibility was demonstrated through two case studies of Nile River and Susquehanna River simulations.

The Nile River case study showcased the codebase's ability to manage large distributed water systems with multiple facilities. In comparison, the Susquehanna River simulation demonstrated the capability to model a centralized system with a single complex reservoir controlling multiple surrounding facilities. The correctness of the software was verified by comparing its behaviour with the adapted Nile River simulation.

8.1 Future works

One of the possible further improvements to the developed RL4Water framework includes automatic detection of the flow network topology. Currently, environments need to be initialized with an ordered list of flows and facilities from the most upstream to the most downstream elements. It makes the users responsible for ensuring a correct setup and requires them to correctly organize the network flow. This feature would reduce the complexity of the simulation configuration process, making it even easier.

Additionally, validation of the framework against other water management simulations is desired to further confirm the generalization capabilities of the codebase. Such a process could include the development of new facilities and covering a larger set of water management problems to utilize reinforcement learning. Including a verification step for that simulation would boost the confidence in the framework's reliability and effectiveness.

Another important step in verifying the feasibility of applying MORL for reservoir control problems is bench-marking the MONES algorithm with the EMODPS, which is commonly used in the water management field. The comparison of these algorithms was done using the Nile River simulation [6], however, extending it to other water management problems would allow to verify the results and help to identify new ways to further increase the performance of both approaches.

Finally, translating of mentioned water management algorithms to be compatible with the Gymnasium structure would help in making the RL4Water framework a unified solution in the field. It would enable the use of well-established algorithms, together with the ease of creating new environments that the developed codebase provides.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[2] Federal Energy Regulatory Commission. Order approving settlement agreement for the conowingo hydroelectric project. Tech. Rep. Proj. 405-009, Docket EL80-38-000, 1989.

[3] Caroline Figuères, Johan Rockström, and Cecilia Tortajada. *Rethinking Water Management: innovative approaches to contemporary issues*. Routledge, 2003.

[4] Matteo Giuliani, Andrea Castelletti, Francesca Pianosi, Emanuele Mason, and Patrick Reed. Curses, tradeoffs, and scalable management: Advancing evolutionary multiobjective direct policy search to improve water reservoir operations. *Journal of Water Resources Planning and Management*, 2016.

[5] R. Hayes, C. F. Radaulescu and Bargiacchi. A practical guide to multi-objective reinforcement learning and planning. 2022.

[6] Kontak Jakub. Measuring the performance of multi-objective reinforcement learning algorithms - nile river case study. 2024.

[7] Faber Jorian. Bottom-up formulation of water management systems as a reinforcement learning problem. 2024.

[8] Witvliet Mark. Multi-sector water allocation: The impact of nonlinear approximation network hyperparameters for multi-objective reservoir control. 2022.

[9] K.M. Strzepek, G.W. Yohe, R.S.J. Tol, and M. Rosegrant. *The value of the high aswan dam to the Egyptian economy*. Number 111 in FNU. Centre for Marine and Atmospheric Science, 2006.

[10] G Várallyay. The impact of climate change on soils and on their water management. 2010.

[11] Ngambouk Vitalis, Valery Ngo, Raoul Fani Djomo Choumbou, Sianga Mutola, Judith Seember, Grace Mbong, and Enjeckayang Forkim. The grand ethiopian renaissance dam, egyptian national security, and human and food security in the nile river basin. *Cogent Social Sciences*, 7:1875598, 01 2021.

[12] Kevin Wheeler, Mohammed Basheer, Zelalem Mekonnen, Sami Eltoum, Azeb Mersha, Gamal Abdo, Edith Zagona, Jim Hall, and Simon Dadson. Cooperative filling approaches for the grand ethiopian renaissance dam. *Water International*, 41:1–24, 05 2016.

[13] Kevin Wheeler, Jim Hall, Edith Zagona, Dale Whittington, and Marc Jeuland. Understanding and managing new risks on the nile with the grand ethiopian renaissance dam. *Nature Communications*, 11, 10 2020.

[14] Kevin G. Wheeler, Jim W. Hall, Gamal M. Abdo, Simon J. Dadson, Joseph R. Kasprzyk, Rebecca Smith, and Edith A. Zagona. Exploring cooperative transboundary river management strategies for the eastern nile basin. *Water Resources Research*, 54(11):9224–9254, 2018.

[15] Sari Yasin. Exploring trade-offs in reservoir operations through many objective optimisation: Case of nile river basin. 2022.

[16] Jazmin Zatarain, Patrick Reed, Jonathan Herman, Matteo Giuliani, and Andrea Castelletti. A diagnostic assessment of evolutionary algorithms for multi-objective surface water reservoir control. *Advances in Water Resources*, 92, 2016.

[17] J. Zatarain Salazar. Multiobjpolicy-rbf analysis kit: Exploring radial basis functions for multiobjective policy optimization, 2023.