Reinforcement Learning for Optimisation of a Cascade River System

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Master of Engineering with endorsement in Software Engineering

at

The University of Waikato

by

Nicolaas John Waddington

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2025

Abstract

Getting the most out of renewable power stations has financial benefits and the environmental benefits of displacing electricity generation from fossil fuels. This thesis considers how to reduce losses in the Waikato Hydro Scheme that are due to inefficient operation while ensuring no operating constraints are breached. The hydroelectric dams are currently centrally run by a human "hydro controller" who makes decisions on how much generation to assign to each station at any given time. The hydro controller ensures no operating constraints are breached, but it may be possible to improve the running efficiency of the hydro stations. Currently, hydro controllers only have basic tools to assist them when deciding which action to take. The aim of this thesis is to create live recommendations to help them make decisions. To this end, historical data is used to train a model using reinforcement learning that can be used to provide such live recommendations on how much generation to assign to each station. The Waikato River hydro control problem is formulated as a Markov decision process and several variations on the Monte Carlo control algorithm for reinforcement learning are proposed and compared empirically. The Waikato River hydro control problem has a large action space with billions of distinct ways to assign generation to stations. The K-medoids algorithm is used to cluster actions so that reinforcement learning becomes practical. Furthermore, once a model has been learned, a hill-climbing approach is used to improve the quality of the selected actions. Combining these techniques gave better running efficiency than historical human performance in 18% of scenarios without breaching any operating constraints. 34% of scenarios did breach constraints so possible improvements to address this are also discussed.

Acknowledgements

Firstly, I would like to thank Eibe Frank for his support during this thesis. I would not have been able to get this far without him. The number of things he has helped me with are too numerous to list here but I'd especially like to thank him in the following areas: for guiding me in my course of study so I could learn the machine learning skills needed to complete this project, for listening to me for many hours as I explained the nature of the problem while giving me insightful tips on which paths to take, for always being willing to help with the administration side of study and making sure I had everything I needed to work effectively, and for his extensive proofreading of this thesis and always giving timely, clear feedback.

I would like to thank Ben Colcord for proofreading, for his help with resolving Python environment issues, and helping get the "numba" Python library working effectively.

I would like to thank my Dad, Ian Waddington, for his thorough proofreading and general writing advice to make sure concepts are clear and flow together.

I would like to thank Mercury for giving me the time and funding to pursue this project. I would also like to thank the ICT team for providing the development environment and compute time used to develop and train the models for this thesis. I would especially like to thank the hydro control team at Mercury for being willing to explain how everything works to me and answering my many questions.

Finally I would like to thank my wife, Savarna, and son, Alby. Thank you Savarna for always being encouraging and understanding. Your love and support has been what got me through the hardest parts of this project. And thank you Alby for inspiring me through all that you've learnt in the last two years. Any artificial intelligence or machine learning pales in comparison to the real thing I see in you.

Contents

Chapter 1

Introduction

Hydropower is an important part of New Zealand's electricity generation, as exemplified by Mercury NZ, one of New Zealand's major electricity generators and retailers, whose power generation is based in large parts on hydro generation. In particular, it owns and operates Taupō Control Gates and nine generating hydroelectric power stations along the Waikato River in New Zealand, forming the Waikato hydro scheme. Mercury NZ Ltd (2024) states that these stations produce around 10% of New Zealand's electricity. If the running efficiency of this scheme could be improved further, this would yield substantial benefits: A 1% improvement in running efficiency would correspond to approximately 40 Gigawatt hours of renewable energy per year (approximately 18,000 electric vehicles worth[1]). This energy corresponds to about $6 million per year. Hence, even such a seemingly small improvement in efficiency would be significant, giving the environmental and financial motivation for investigating whether machine learning can be used to achieve this.

The Waikato hydro scheme (Section 3.2) is run centrally by a hydro controller, who decides how much generation to assign to each of the hydro stations. The primary objective of the hydro controller is to ensure that enough electricity is generated while not breaching any operating constraints, such as

[1] According to Mercury NZ Ltd (2019), 470GWh per year equates to 210,000 cars giving 2.24MWh per car per year. Applying this to 40GWh of energy gives $\frac{40GWh}{2.24MWh\ per\ car} \approx 18,000\ cars$

a maximum allowable water level at a hydro station. The secondary objective is to run the hydro stations in an efficient manner, making the most of the water that passes each of the stations. These objectives are collectively referred to in this report as the "Waikato River hydro control problem".

Hydro controllers use their experience as well as some simple spreadsheet-based tools to assist in decision making and planning. There is an opportunity here to use more advanced analytical techniques to improve the business outcomes.

Previous attempts have been made to use analytics to optimise efficiency on the Waikato hydro scheme. One such model was called "AWMA" (internal Mercury project), which was developed in the late 1990s. AWMA was based on linear programming and attempted to make an optimised plan of station generation, flows, and levels for a couple of days. It optimised for running efficiency with the operating constraints built into the model. This approach produced a plan that followed operating constraints and had a good efficiency for the period that it modelled. It had two main shortcomings. The first was that it did not consider the value of having storage in lakes at the end of the plan. The consequence of this was that it often left all lakes empty at the end of the plan, making the following day infeasible or very inefficient. The other shortcoming was that it was too slow to be run in real-time and keep up with the changes on the river.

Another approach was called HAT (Hydro Advisory Tool), created in 2008. It performed a mathematical solve to optimise the hydro unit setpoints for a given station setpoint in real time. This optimised each station individually but did not consider the river as a whole. It provided some efficiency improvements for a short time but fell out of use because it failed to address the co-optimisation of real-time efficiency with longer-term needs of having the right amount of water in the right place over several days.

These two attempts highlight the importance of a model that is able to consider the river as a whole and to create a good approximation for the

Waikato River hydro control problem. Extra care needs to be taken with the end of a plan because the Waikato River hydro control problem is continuous; it does not stop at the end of a plan. It is also important for the model to run quickly to be able to keep up with the changes on the river.

The proposed solution is to use reinforcement learning to give real-time recommendations to hydro controllers. Reinforcement learning can be trained offline in a simulation and then give fast recommendations based on the trained model. Live data feeds can be connected to the trained model so it can stay up to date with what is actually happening on the river. The aim is to obtain similarly efficient plans as previous linear programming approaches without breaking constraints while being fast enough to be run in real-time.

To apply reinforcement learning, the Waikato River hydro control problem is formulated as a Markov decision process (MDP. See Section 2.1.1). The problem has a large continuous state and action space, increasing exponentially with the number of stations, so it is important to use mechanisms that keep the computation manageable. Function approximation using artificial neural networks is used to handle the continuous state space. The action space is discretised using the K-medoids algorithm (2.2). Extensions on the discrete action space using a hill-climbing algorithm are also explored. The Monte Carlo control algorithm (Sections 2.1.4 and 2.1.5), a form of Q learning, which is a well-known method for reinforcement learning, is evaluated. Two different sets of input features for the Monte Carlo control are compared, one with and one without a time dimension.

The thesis is structured as follows. First, it covers background on reinforcement learning and the Waikato hydro scheme. The MDP-based problem formulation is discussed next. Subsequently, the simulation environment that has been developed to enable reinforcement learning is described. Finally, results are presented for each of the Monte Carlo control approaches.

Chapter 2

Background

This chapter introduces reinforcement learning and the Markov decision processes (MDP) that it aims to solve. It also discusses K-medoids clustering and a hill-climbing algorithm that will be used in conjunction with reinforcement learning to enhance performance.

2.1 Reinforcement Learning

2.1.1 Markov Decision Process

In an MDP (Markov Decision Process), there are two separate components, the agent and the environment. The agent's role is to observe the state S of the environment and to take an action A. The environment handles the process of taking this state and action, and producing a reward for that action and the next state. Usually, the aim is to find an agent that maximises a reward R over time, with positive rewards representing good outcomes in the environment and negative rewards representing bad outcomes. A key assumption is that the current state of the environment contains all information about the system. Previous states are not used to determine future states and rewards. The next state is then observed again by the agent and another action is taken. This is repeated until a terminal state is reached. This process generates a trajectory with a sequence of state-action pairs and observed rewards repeated until a

termination time step T:

$$S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2 ... S_{T-1}, A_{T-1}, R_{T-1}, S_T \tag{2.1}$$

The transition model of an MDP is a function representing the probability distribution of state transitions. $\mathcal{S}$ represents the set of all possible states. $\mathcal{A}(s)$ represents the set of feasible actions that can be taken by the agent in state s. For a given state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$, the transition function to a new state $s' \in \mathcal{S}$ while receiving reward $r \in \mathbb{R}$ is given as:

$$p(s', r|s, a) = Pr\{S_t = s', R_t = r \mid S_{t-1} = s\} \tag{2.2}$$

As with any probability distribution, the sum over all elements is 1:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1 \quad \forall \quad s \in \mathcal{S} \quad \text{and} \quad a \in \mathcal{A}(s) \tag{2.3}$$

The agent aims to maximise reward over time by maximising a discounted return G_t where

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \tag{2.4}$$

and $0 \leq \gamma \leq 1$ (Sutton and Barto (2018), p. 48-55). Discounting is done because in many tasks, receiving a positive reward sooner is better than receiving it later. Heavy discounting emphasises rewards closer to the present and gives less consideration to future rewards. A discounting rate γ that is close to 1 represents not much discounting and might be used in an MDP where actions taken in the present affect rewards many steps into the future. Conversely, a discounting rate closer to zero represents heavy discounting and might be used in an MDP where actions taken only affect the present.

The agent learns how to perform actions by establishing a policy π, which gives a probability distribution of actions from a given state. $\pi(a \mid s)$ will be used as a shorthand for the probability of an action a being taken from a state s. This can also be used to model deterministic policies. For a deterministic policy, the probability of the chosen action will be one and the probability of all other actions will be zero.

2.1.2 MDP Example

As an example to illustrate the components of an MDP, consider the game of noughts and crosses. This game consists of a three by three grid where two players, one with 'nought' tokens and one with 'cross' tokens, take turns to place their tokens in the grid, each attempting to be the first player to get three of their tokens in a row. For the purposes of this example, the crosses player always starts.

In this example, the agent is the crosses player and the environment consists of the rules of the game and the opposing noughts player. The agent is trying to win the game and the maximisation of the reward function is used to represent this goal. The state s of the environment is the current grid and tokens that have been placed. The action a is the crosses player placing a cross on some position in the grid. State transitions $s, a \rightarrow s', r$ occur every time an action is taken, in this case when a cross is played. The state transition in this example has three steps:

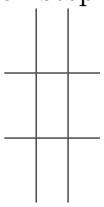
1. The environment checks if the crosses player has played three crosses in a row; if so, the crosses player wins and the new state s' is the terminal state S_T, indicating the end of the game. The reward R_T is +1 from the crosses player winning. In the case of a draw, where the grid is full, but neither player has achieved three in a row, the new state s' is the terminal state S_T. The reward R_T is zero for the draw.

2. If the game is continuing, the opposing player places a nought decided by some (unknown) probability distribution. The environment checks if the noughts player has played three noughts in a row; if so the noughts player wins and the new state s' is the terminal state S_T, indicating the end of the game. The reward R_T is -1 from the noughts player winning.

3. If neither of steps 1 and 2 have ended the game, the transition goes to the new (non-terminal) state s', reflecting both players' moves. The reward r is zero because neither player has won yet.

For simplicity, the discounting factor $\gamma = 1$, meaning there is no discounting. The policy is the strategy that the agent, the crosses player, follows throughout the game. It is how the player decides where to place their cross, based on the current tokens in the grid. Here is a walk-through of an episode showing what the state, action and reward are for each step:

The initial state S_0 is the empty grid:



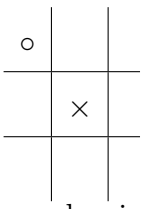
The agent has a policy $\pi(a|S_0)$:

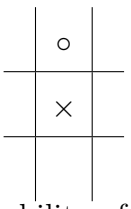
- $\pi(Place \times in\ top\ left \mid S_0) = 0.4$
- $\pi(Place \times in\ centre \mid S_0) = 0.6$

with all other actions having a probability of zero in the policy. Based on this policy, let us assume the agent randomly chooses $A_0 = Place \times in\ centre$. Once the action is taken, the state transition $S_0, A_0 \rightarrow S_1, R_0$ takes place. Now, let us assume the environment, which includes the (unknown) policy of the opponent, gives a 70% chance of a nought being placed in the top left and a 30% chance of a nought being placed in the top-centre position. Neither player can achieve three-in-row so the reward $R_0 = 0$. This gives state transition probabilities:

$$p(S_1 = [see\ grid\ 1a], R_0 = 0\ |S_0, A_0) = 0.7$$

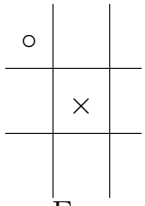
$$p(S_1 = [see\ grid\ 1b], R_0 = 0\ |S_0, A_0) = 0.3$$

Grid 1a 

Grid 1b 

Assume all other state-reward pairs have a probability of zero. The state S_1 and reward R_0 are chosen randomly by the environment according to the state transition probability above.

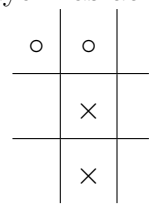
In this case, a possible response by the environment would be $R_0 = 0, S_1 =$



For a second time, the agent takes an action. The policy π takes the state as an input so the policy from S_1 can be completely different to the policy from S_0. Let us assume the policy $\pi(a|S_1)$ takes new values of:

- $\pi(Place \times in\ bottom\ middle \mid S_1) = 0.1$
- $\pi(Place \times in\ bottom\ left \mid S_1) = 0.9$

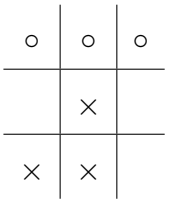
All other actions have probability zero. The agent randomly chooses $A_1 = Place \times in\ bottom\ middle$ using the probabilities from $\pi(a|S_1)$. Another state transition $S_1, A_1 \rightarrow S_2, R_1$ occurs. This time the environment gives a probability of 100% for a nought to be placed in the top middle space in the grid. Neither player has achieved three-in-a-row so the reward $R_1 = 0$ and the new state S_2 is:



Once again, the agent takes an action. The policy $\pi(a|S_2)$ takes values:

- $\pi(Place \times in\ top\ right \mid S_1) = 0.8$
- $\pi(Place \times in\ bottom\ left \mid S_1) = 0.2$

All other actions have probability zero. The agent randomly chooses $A_2 = Place \times in\ bottom\ left$. The state transition $S_2, A_2 \rightarrow S_3, R_2$ occurs. This time, assuming the policy of the opponent is reasonable, the environment has 100% probability of placing a nought in the top right, achieving three noughts in a row along the first row of the grid:



This makes the state S_3 the terminal state, ending the episode. The final reward $R_2 = -1$ because the noughts player won. This gives a trajectory for the episode of:

$$S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2, S_T$$

The return for the episode from each time step can be calculated from the rewards in the trajectory:

$$\begin{aligned} G_0 &= R_0 + \gamma R_1 + \gamma^2 R_2 && = 0 + 1 \times 0 + 1^2 \times -1 && = -1 \\ G_1 &= R_1 + \gamma R_2 && = 0 + 1 \times -1 && = -1 \\ G_2 &= R_2 && && = -1 \end{aligned}$$

Looks like this agent needs a better policy!

2.1.3 Solving an MDP

An MDP is considered "solved" if an optimal policy is found or, equivalently, if the value function under the optimal policy $q_*(s, a)$ is found. An optimal policy is one that always takes the action that gets the highest return or cumulative reward G (see Equation 2.4). One approach to solving MDPs is with dynamic programming. Dynamic programming for MDPs aims to solve a Bellman equation

$$q_*(s, a) = \sum_{s', r} p(s', r|s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right] \tag{2.5}$$

(Sutton and Barto (2018), p. 73). Where $q_*(s, a)$ is the state-action value function defined by the expected discounted future reward from the state-action pair.

To solve the MDP, a system of Bellman equations is formed; one for each state-action pair. Then there is an equation and an unknown variable $q_*(s, a)$ for each state-action pair, and one can solve the system of equations directly. This approach requires complete knowledge of the state transition function p. Also, given that the number of equations to solve is proportional to the

number of state-action pairs, the solution can become difficult to compute as the state and action space become very large.

Another approach that can be taken to solve an MDP is through Monte Carlo control. This consists of repeated iterations of policy evaluation and improvement. Policy evaluation involves estimating the future expected return for a given a policy π. This estimate, from a state s and taking an action a, is denoted $\hat{q}_\pi(s, a)$ where $\hat{q}_\pi$ is the estimated future return function from following a policy π.

Policy improvement involves evaluating $\hat{q}_\pi(s, a)$ for each possible action a in state s. Instead of selecting the action that follows policy π, the one that maximises the value of $\hat{q}_\pi$ is selected, and this becomes the new policy for that state $\pi'(s)$ (Sutton and Barto (2018), p. 97):

$$\pi'(s) = \arg\max_a \hat{q}_\pi(s, a) \tag{2.6}$$

This is referred to as "greedy" action selection because it selects whatever action seems best at the time without regard for future possibilities. A similar alternative is an ϵ-greedy policy where a random action is chosen with probability ϵ and the greedy action is chosen with probability $1 - \epsilon$. Each random action has a uniform probability of being chosen. The ϵ-greedy policy is written as:

$$\pi(a|s, w) = \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(s)| & if\ a = \arg\max_a \hat{q}_\pi(s, a, w) \\ \epsilon/|\mathcal{A}(s)| & if\ a \neq \arg\max_a \hat{q}_\pi(s, a, w) \end{cases} \tag{2.7}$$

Policy evaluation and improvement can be brought together by the "exploring starts" algorithm (Sutton and Barto (2018), p. 99). In exploring starts, a random state-action pair S_0, A_0 is chosen and an episode following π is generated:

$$S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2 ... S_{T-1}, A_{T-1}, R_{T-1}, S_T, A_T, R_T \tag{2.8}$$

For each time step t in the episode, the discounted future reward for the episode is calculated as $G = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$. This G value is used for the

policy evaluation step. The $\hat{q}_\pi$ estimate for each state action pair is given by the mean of all G values obtained by taking the action in the corresponding state. Every time the $\hat{q}_\pi$ value estimate is updated, a step of policy improvement is performed, updating π as above. By alternating steps of evaluating the action-value of a policy and improving on the policy through greedy selection of actions, the policy is gradually improved. With a policy π, taking a single step of policy improvement, the new policy π' is guaranteed to have an equal or better expected return for all states. This is shown in the policy improvement theorem (Sutton and Barto (2018), p. 97-98):

$$\begin{aligned} &\forall s \in \mathcal{S} \\ \hat{q}_\pi(s, \pi'(s)) &= \hat{q}_\pi(s, \arg\max_a q_\pi(s, a)) \\ &= \max_a \hat{q}_\pi(s, a)) \\ &\geq \hat{q}_\pi(s, \pi(s)) \end{aligned}$$

The benefit of using this approach of alternating policy evaluation and improvement is that a policy can be iteratively improved through sample episodes. This avoids the need for complete knowledge of the state transition function p. One of the drawbacks is that each state-action pair must be visited multiple times to get an accurate estimate of $\hat{q}_\pi$.

2.1.4 Monte Carlo Control with Linear Function Approximation

For MDP's with larger state or action spaces, it may be impractical to visit every state-action pair multiple times. Function approximation provides a way to generalise experiences across states and actions. One simple way to do this is through linear function approximation. The estimated future return function $\hat{q}_\pi(s, a)$ can be represented by a dot product of weights and features derived from the state and action.

$$\hat{q}_\pi(s, a, \mathbf{w}) \doteq \mathbf{w}^T \mathbf{x}(s, a) \doteq \sum_{i=1}^{d} w_i x_i(s, a), \tag{2.9}$$

where

- s is the state to evaluate the value at
- a is the action taken from state s to evaluate
- $\mathbf{w}$ is a weight vector
- $\hat{q}_\pi$ is the approximation of the state-action value function under a policy π
- $\mathbf{x}$ is the weight vector calculated as a function of the state and action.

To fit the $\hat{q}_\pi$ function, a training loop is repeated. In each loop, an episode is generated following π. π is defined as an epsilon greedy policy following Equation 2.7. Upon completion of the episode, the G value is calculated for each time step, same as for basic Monte Carlo control. The objective function used to compute the weight vector minimises the square distance between the q value approximation and the true discounted future return:

$$min(\hat{q}_\pi(S_t, A_t, \mathbf{w}) - G_t)^2 \tag{2.10}$$

Applying stochastic gradient descent to this objective function gives the following update rule to apply:

$$\mathbf{w}' \doteq \mathbf{w} + \alpha[G_t - \hat{q}_\pi(S_t, A_t, \mathbf{w})]\mathbf{x}(S_t), \tag{2.11}$$

where

- S_t is the state from the episode at time t
- A_t is the action taken in the episode at time t under policy π
- $\mathbf{w}$ is the current weight vector
- $\mathbf{w}'$ is the new updated weight vector

- G_t is the discounted future reward from time t observed in the episode.
- α is a learning rate constant parameter

This Monte Carlo method for the linear case is guaranteed to converge to or near the global optimum, assuming a linear model is sufficient to model the true q function (Sutton and Barto (2018), p. 205).

Linear function approximation is limited in the functions it can represent. This can be partially overcome by feature engineering where additional inputs are concatenated to the feature vector $\mathbf{x}$. Rather than simply taking the raw state and action values, non-linear combinations of different raw values using domain knowledge can be added in to improve the function representation.

2.1.5 Monte Carlo Control with Neural Network Approximation

If a better representation of the $\hat{q}_\pi(s, a)$ function is required, non-linear methods such as neural networks can be used. In this case, the $\hat{q}_\pi(s, a)$ function is represented by an n hidden layer neural network taking the feature vector $\mathbf{x}$ as input and targeting the discounted future value of G in the output. In the case where $n = 0$, this is equivalent to the linear function approximation case. For $n > 0$ a richer function approximation is obtained but there is only a guarantee of convergence to a local optimum (Sutton and Barto (2018), p. 202).

To train the neural network, a training loop is repeated. Similarly to the linear case, in each iteration of the loop, an episode is generated following π and the G value is calculated for each time step. Stochastic gradient descent can also be used in the neural network case. The update rule for this is taken as the derivative of the objective function with respect to the weight vector, multiplied by the learning rate $-\alpha$.

2.2 K-medoids Clustering

Environments with large or continuous action spaces can pose problems for some reinforcement learning algorithms. One possible way to reduce the number of actions in the environment considered in this thesis is clustering groups of similar actions to a single action. The K-medoids algorithm is a clustering algorithm similar to K-means. The aim of both algorithms is to split a dataset into K clusters of similar objects. For n objects in a dataset with p variables to fit into K clusters the K-medoids method can be used. A medoid is a representative central object in a cluster of objects. The key difference to K-means is that K-medoids clusters around medoids where K-means clusters around centroids. K-medoids is less sensitive to outliers and ensures the clusters are represented by an actual point in the dataset (Park and Jun (2009)). Having an actual data point as the cluster representative is important because this ensures that the action for that cluster has been used before, so is a valid, feasible action.

2.2.1 Fast K-medoids algorithm

A fast implementation of the algorithm, based on Euclidean distance, is described in Park and Jun (2009):

1. Pairwise Euclidean distances d_{ij} are calculated for every possible pair of data points

2. A total distance metric v_j is calculated for every object j as:

$$v_j = \sum_{i=1}^{n} \frac{d_{ij}}{\sum_{l=1}^{n} d_{il}} \tag{2.12}$$

3. Initial medoids are taken as the objects with the k smallest distance metrics v_j.

4. All other objects are assigned to the cluster represented by their nearest medoid.

5. New medoids are selected by finding the object in each cluster with the minimum total distance to all other objects in the cluster.

6. This process of assigning objects to clusters and updating medoids is repeated until the total distances for each cluster stop changing.

2.3 Hill-climbing Algorithm

A key part of Monte Carlo control is evaluating the action that gives the highest state-action value: $\arg\max_a \hat{q}_\pi(s, a, w)$ (See Equation 2.7). For a small number of actions, it is possible to take the simple approach of evaluating $\hat{q}$ for all possible actions and selecting the action with the highest $\hat{q}$ value. For a larger number of actions, this approach becomes computationally intractable. A method to search the action space without enumerating every single action is necessary.

Lim et al. (2006) presents a hill-climbing algorithm for the matrix bandwidth reduction problem. This problem aims to minimize the bandwidth in a graph. It defines a neighbourhood rule $N'(v)$ where v is an edge in the graph. The neighbourhood rule $N'(v)$ gives a set of edges within a distance, according to some distance metric, of the edge v. The hill-climbing approach works as follows:

1. For all critical nodes (a critical node in this case is a node directly impacting the objective function), determine all edge swaps (u, v) between neighbouring edges $u \in N'(v)$ and v.

2. For each of the valid edge swaps (u, v), check if the objective function has improved. If it has improved, keep the swap. Otherwise, swap back.

3. Repeat this process until there are no more improvements to the objective function.

This approach gives a locally optimal solution to the matrix bandwidth reduction problem.

The hill-climbing algorithm can be adapted for Monte Carlo control. It starts by taking an initial action a. This could be initialised as any arbitrary action. In this case the action that was taken in the previous time step is used as the initial action a. The hill-climbing algorithm considers all neighbours $a' \in \mathcal{C}(a)$ of the action a as defined by some neighbourhood rule $\mathcal{C}(a)$. The neighbourhood rule defines a set of actions that is a subset of the full action space $\mathcal{C}(a) \subset \mathcal{A}$ where $\mathcal{C}(a)$ contains all actions within some distance of action a as defined by some distance metric. The algorithm is adapted as follows:

1. Take an action candidate a as an initial starting point.

2. Take all neighbouring actions a' according to the neighbourhood rule $a' \in \mathcal{C}(a)$.

3. For each of the valid neighbouring actions C', check if the state-action value function $\hat{q}_\pi(s, a', w)$ has improved. If it has improved, the swap is kept and a' becomes the new action candidate $a = a'$.

4. Repeat this process until there are no more improvements to the objective function.

This process gives a locally optimal action a with respect to the state-action value function $\arg\max_a \hat{q}_\pi(s, a, w)$.

Chapter 3

Waikato Hydro Scheme

This chapter begins by discussing the physical characteristics of the Waikato Hydro Scheme. Subsequently, hydro units are explained and the mathematical relationship between flows, levels and generation is introduced. Finally, the current practice of hydro controllers is stated along with the goals and constraints associated with operating the Waikato Hydro Scheme.

3.1 Waikato River

The Waikato River flows from Lake Taupō, in the central North Island, northwards to Port Waikato. The mean inflow through Taupō Control Gates is about 160.8 cubic metres per second (cumecs) and the mean outflow at the Karāpiro Dam is about 247.2 cumecs (Roper (2001), p. 49). The difference between these flows is made up from tributary flows. For context, 247.2 cumecs is enough water flowing to fill an Olympic-size swimming pool in about 10 seconds.

3.2 Waikato Hydro Scheme

The Waikato Hydro Scheme consists of Taupō Control Gates and nine generating hydro stations along the Waikato River in New Zealand. Water flows to the stations from Lake Taupō, controlled by Mercury from Taupō Control

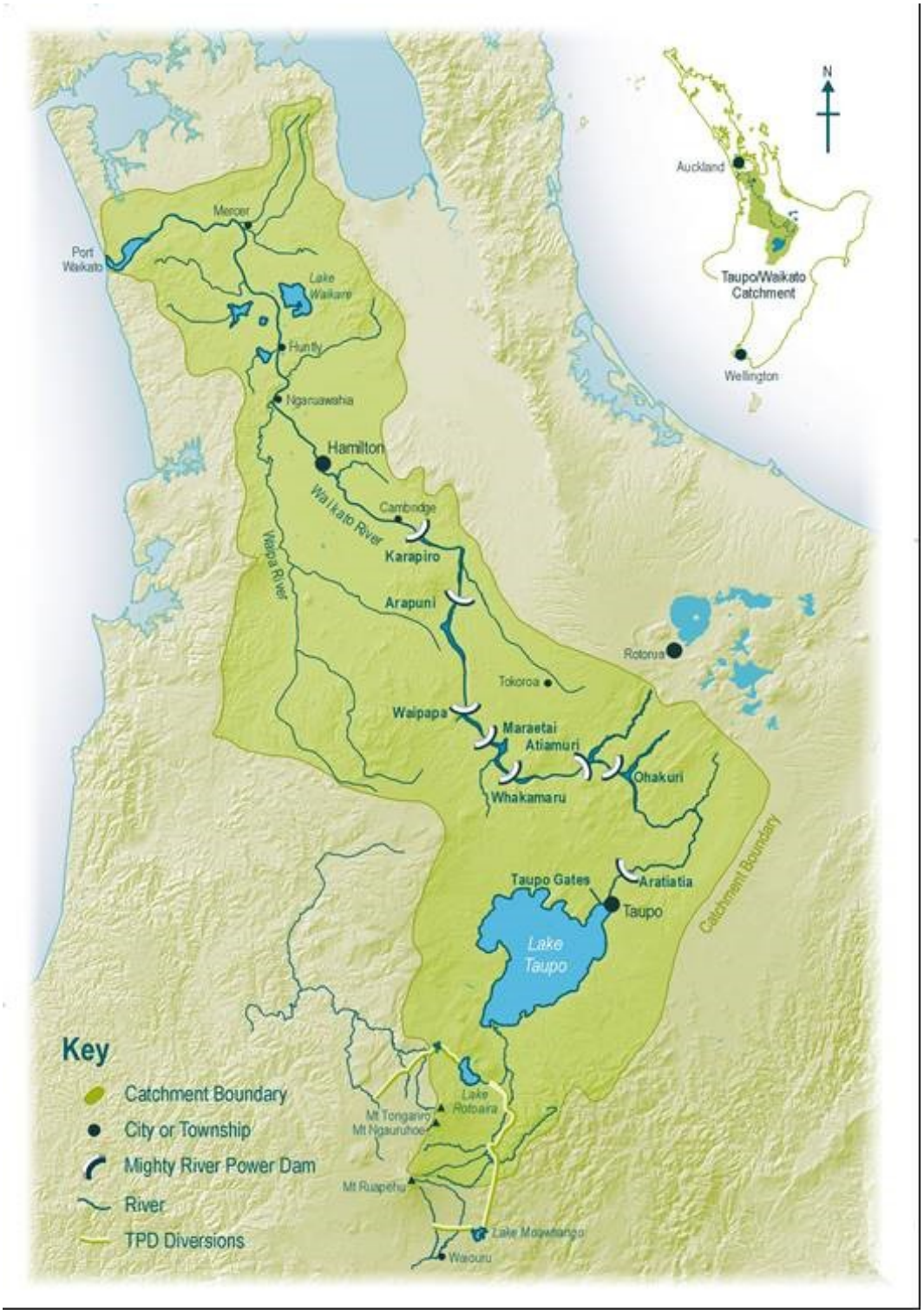


Figure 3.1: Map of Waikato Hydro Scheme

Gates, and several small tributary rivers along the Waikato River. Taupō control gates control the flow of water from Lake Taupō into the Waikato river, but they do not generate any electricity. The stations are listed in river order below:

- Taupō Control Gates (No generation)
- Aratiatia
- Ōhakuri
- Ātiamuri
- Whakamaru
- Maraetai 1 and Maraetai 2
- Waipāpa
- Arapuni
- Karāpiro

Maraetai 1 and Maraetai 2 are two stations side-by-side. While they are physically separate structures, the problem is simplified by considering them a single combined station. For the rest of this thesis, they are considered a single station, reducing the total number of stations from nine to eight along with Taupō Control Gates.

Hydro dams produce electricity by converting gravitational energy into electrical energy. The gravitational energy comes from the difference in water levels upstream and downstream of the dam (known as the headwater level and tailwater level respectively). This difference between the headwater level and tailwater level is known as the "station head". Due to changes in electricity supply and demand, hydro stations are regularly required to increase or decrease their power output. These instructions to increase or decrease come from Transpower, New Zealand's grid owner and system operator, as a

dispatch instruction of a generation amount D. This dispatch instruction is telling the hydro controllers that D MW must be produced now and hydro controllers adjust hydro unit setpoints to ensure that the correct amount of energy is generated. These dispatch instructions can come in as frequently as every five minutes and hydro controllers are expected to acknowledge and make setpoint changes within a maximum of four minutes.

3.3 Hydro Units

Each hydro dam has several generating units and a spillway where water can pass the station without being used for generation. The flow of water through generating units is referred to as the penstock flow and water through a spillway is referred to as the spill flow. Across the river there are 39 generating units split between the nine stations. In each generating unit, there are wicket gates to allow water to flow through to a turbine, spinning it. A shaft connects the turbine to a rotor. As the rotor spins inside a stator, electrical energy is produced. Mercury uses two types of hydro turbines, Francis and Kaplan turbines. The Francis turbine "is the most widely used turbine design today, particularly for medium head and large flow rate situations" (Lewis et al. (2014)) while "the Kaplan turbine became the most widely used type of device for the use of low heads and variable flow rates" (Polák (2021)). For the purpose of this thesis, the only difference that matters is that the hydro units have different efficiency profiles, depending on the station head and flow. The amount of power produced, for either type of turbine, is governed by the equation below:

$$\eta_{u,t} = \frac{P_{u,t}}{\rho g Q_{u,t} H_{s,t}}, \tag{3.1}$$

where

- $\eta_{u,t}$ is the overall efficiency of the unit u at time t. This includes hydraulic, mechanical and volumetric efficiencies (from the gravitational energy of the water to the physical spinning of the turbine). It does not

include electrical losses from the generator, transformer or grid connection. It is dimensionless with a value in the range [0, 1].

- $P_{u,t}$ is the power available at the turbine in Watts at time t. Power is usually converted to megawatts for convenience
- ρ is the density of water, which is assumed as a constant $997kgm^{-3}$.
- g is the acceleration due to gravity. It is assumed as a constant $9.81ms^{-2}$.
- $Q_{u,t}$ is the flow rate of water into the turbine at time t in cumecs (m^3s^{-1}).
- $H_{s,t}$ is the gross head in metres for the station at time t. It is calculated by taking the headwater level in masl (metres above sea level) less the tailwater level in masl.

(Jain et al. (2010))

This can be aggregated up to a station level where:

$$P_{s,t} = \sum_{u} P_{u,t} \; where \; u \in station \; s \tag{3.2}$$

$$Q_{s,t} = \sum_{u} Q_{u,t} \; where \; u \in station \; s \tag{3.3}$$

$$\eta_{s,t} = \frac{P_{s,t}}{\rho g Q_{s,t} H_{s,t}} \tag{3.4}$$

- $\eta_{s,t}$ is the overall efficiency of the station s
- $P_{s,t}$ is the total power at the station s
- $Q_{s,t}$ is the penstock flow through the station s in cumecs (m^3s^{-1})

The efficiency of a hydro unit can vary significantly as a function of Q and H, which are the main variables that can be controlled. For example, Figure 3.2 shows how the efficiency varies with flow at a fixed station head. For this Maraetai unit, efficiency peaks at 60 cumecs with very high or very low flows having poor efficiency.

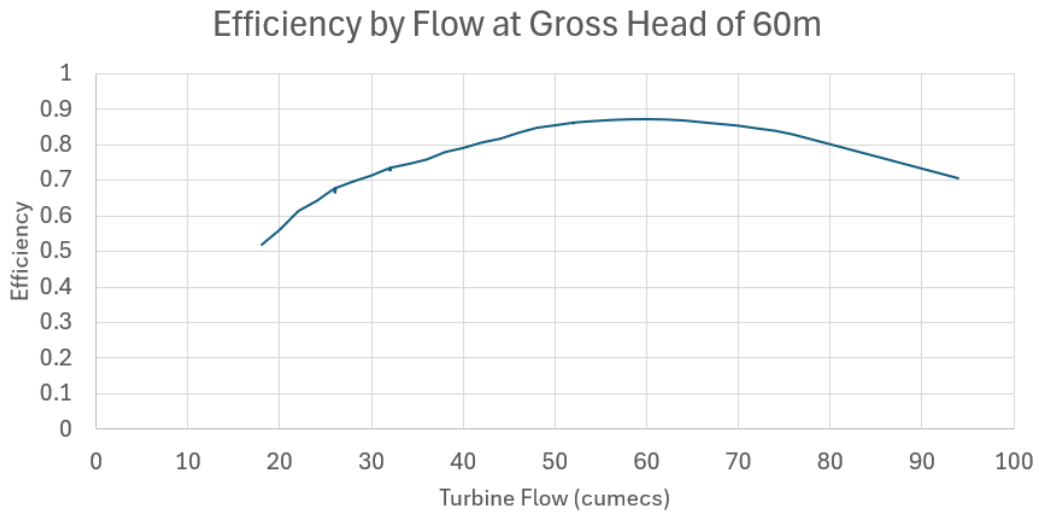


Figure 3.2: Efficiency curve from a unit at the Maraetai station showing how efficiency changes with flow at a fixed gross head of 60 metres

3.4 Operational Constraints

Hydro stations have a number of restrictions on how they can operate. Some examples of these restrictions are listed below:

- The total power produced across the Waikato hydro scheme must align with the Transpower dispatch generation. This can be expressed for a given time t as:

$$\sum_{s} P_{s,t} = D_t \tag{3.5}$$

  Also, occasionally Transpower will issue a dispatch instruction to an individual station within the Waikato hydro scheme due to transmission constraints, but this rare case is ignored for simplicity.

- During maintenance, one or more hydro units can be unavailable, or have their power output restricted. These maintenance outages can be as short as half an hour or can last several months. This is reflected in a maximum generation availability for each station. This can be expressed as $P_{s,t} \leq MaxGen_{s,t}$. One example of hydro units being unavailable is when there is a station shut at the Whakamaru station, the fourth station in the river sequence, for a day. In this case, all units are unavailable

and the $MaxGen$ would be set to zero for the station Whakamaru at all times that fall on the day of the station shut. Another example would be if two out of the four units at the Whakamaru station were out for maintenance for a week. Each unit at Whakamaru is capable of 31MW of generation, so if only two units were available, then $MaxGen$ would be set to $31 \times 2 = 62$MW for Whakamaru for all times that fall on the week of the outage.

- Headwater levels must be held between a minimum and maximum value. These values are determined by factors like engineering safety limits and resource consents. Occasionally additional headwater level restrictions are added for a short time in support of community events such as rowing or duck shooting.

- Minimum flows are in place at some stations to ensure the river continues to flow. This is part of the resource consent for the Waikato hydro scheme.

- Hydro units with Francis turbines have rough running ranges. These are ranges of unit power output that cannot be used for extended periods of time because they are damaging to the turbine.

- Hydro units have minimum and maximum generation levels by unit.

- At the Aratiatia power station, there is a "Tourist Spill" that takes place four times a day in summer and three times a day in winter. A Tourist Spill is where the station spillway opens at 65 cumecs for 12.5 minutes as entertainment for tourists. It is part of the resource consent for the Waikato hydro scheme.

3.5 Hydro Control

Hydro controllers work to provide continual operation for the Waikato hydro scheme in 12 hour shifts. They make adjustments to the generation setpoints,

which control how much water flows through turbines and how much electricity is generated. They receive dispatch instructions from Transpower. These are instructions to generate a certain amount of electricity at a particular station or group of stations, i.e., the dispatch instruction is to increase or decrease the total generation of the Waikato hydro scheme. Hydro controllers acknowledge the dispatch and adjust the setpoints of the generating units along the river to reflect the change. They ensure all operating constraints are met and maintain communication with stakeholders to ensure the constraints are always up to date.

Hydro controllers primarily aim to reduce the risk of breaching any restrictions with a secondary aim to maximise the yield from water flowing through the stations. Maximising this yield is done by minimising spill, running units at flows that have a high efficiency on the efficiency curve, and maximising gross head where possible. Spill is sometimes necessary because of hydro unit outages, high inflows into the river, or low demand for electricity.

Chapter 4

Literature Review

This chapter discusses the current literature relevant to the Waikato River hydro control problem considered in this thesis. The literature search yielding the publications discussed in this chapter focussed on research that modelled and optimized cascade river systems. Research on real-time optimization and minimising waste of water were of particular interest. This chapter begins by considering two published reviews of existing literature; then two papers with direct relevance are discussed. The rest of the chapter discusses some papers that are partially relevant but do not fully align with the objectives of the hydro control scenario considered in this thesis.

4.1 Reviews on Real-Time Hydro Electricity Optimisation

Giuliani et al. (2021) discuss different approaches to optimising hydro reservoirs. One approach is stochastic dynamic programming. This involves formulating the problem as a Markov decision process and solving by calculating a cost for each river state. Theoretically, this can give an optimal policy for all possible scenarios. Unfortunately as Giuliani et al. state, the time to solve grows exponentially with the number of reservoirs. This effect is known as the "curse of dimensionality". For a longer chain of Hydro stations like the

Waikato, there would likely be challenges with solve time. Giuliani et al. conclude: "Our analysis identifies Approximation in Value Space methods as the approach that better copes with the curse of dimensionality". Reinforcement learning and model predictive control are two of the main approaches they state as methods of approximation in value space. A paper applying both methods to the optimisation of hydro electricity in a cascade river scheme is discussed in Section 4.2.

Bernardes et al. (2022) review applications of machine learning to hydropower operations. They look at papers considering different time horizons: real-time, short-term, and long-term. They also look at different types of hydropower systems: single reservoir, reservoirs in cascade, and multiple reservoirs. The Waikato Hydro Scheme is a cascade river system, so research considering reservoirs in cascade operation optimization, for real-time or short-term decision making, is the most relevant, but some papers on reinforcement learning applied to longer time horizons or other types of river may also have some applicable insights. Interestingly, Bernardes et al. do not discuss any papers looking at the operational optimisation of reservoirs in cascade in a real-time setting. In the time since their review, two papers that consider the real-time setting for reservoirs in cascade have appeared. These two papers are discussed in the next section.

4.2 Machine Learning for Real-Time Optimisation of Hydro Electricity

Castro-Freibott et al. (2025) apply reinforcement learning to a cascade hydro system. They model the full power curve of the hydro stations, making it relevant to the efficiency optimisation needed for the Waikato River hydro control problem. Their work looks at three policy gradient algorithms: (a) a continuous soft actor critic method, (b) a proximal policy optimisation (PPO) with discrete actions, and (c) an adjustments method that iteratively improves the

action. These three RL models are compared with (d) mixed-integer programming and (e) a simple baseline to optimize the financial output of the system at an intra-day level. The baseline approach is described as: "This approach involves keeping the gates of all dams fully open at all times, meaning that each reservoir's outflow is set to its maximum value in every period. Even though this procedure does not make use of the reservoirs' capacity, it represents the previous operational strategy of the hydropower station that motivated this study". They show that reinforcement learning performs better than the baseline for a system of 2 hydro stations and one of 6 hydro stations. However, for 6 hydro stations, reinforcement learning takes longer to train and the best model is only 0.1% better than the baseline while also yielding slightly more constraint violations. The mixed integer linear program approach was by far the best of all the models for the two-reservoir scenario but struggled to scale to the six-reservoir scenario, performing 6.7% worse than the baseline. Their best performing reinforcement learning algorithm is the PPO approach. The small improvement on the simple baseline demonstrates the challenge of adapting the reinforcement learning approach to a complex cascade river system like the Waikato Hydro Scheme with 8 stations. Their work also indicates that discretising actions instead of using a continuous value actions can be beneficial. Note that the work of Castro-Freibott et al. was published very recently; thus, there was no opportunity to consider adaption of their techniques to the setting considered in this thesis.

Ye et al. (2023) apply model predictive control for real-time control in the Shaping hydropower station in the Dadu River basin. Model predictive control is an optimization method that uses a mathematical model of the system dynamics and constraints to minimize a cost function. Ye et al. explore a number of methods to model the river dynamics, including a physics-based water balance model and machine learning models. They evaluate several methods of applying model predictive control including an exhaustive search, particle swarm optimization, and differential evolution. To get their approach to work,

they reduced the action space by fixing the station MW at the historical generation value for each time step. This is a sensible simplification for their situation and gave them good results but is an unacceptable simplification for the Waikato River hydro control problem, where the decision of how much generation to assign to each station is a core part of the problem. However, it highlights the importance of reducing the action space where possible.

4.3 Other Reinforcement Learning Approaches for Hydro Electricity

Now, we briefly review work that applies reinforcement learning to problems similar to the Waikato River hydro control problem considered in this thesis such as long term optimisation or single or dual reservoir systems. The papers we review primarily use either Q learning approaches or policy gradient methods.

Lee and Labadie (2007) successfully model the Guem river basin in South Korea with Q learning. Their work models station generation, spill, and the relationship between flows and station head, all of which are also important in the Waikato Hydro Scheme. It optimises for a weighted sum of multiple objectives. This is particularly relevant because the Waikato River hydro control problem requires optimising efficiency and feasibility, which can be modelled as distinct objectives. The Guem river basin only has two stations, so applying the same techniques to the Waikato with 8 stations may be problematic due to the higher dimensionality of the state space. The paper compares Q learning with stochastic dynamic programming and finds that Q learning yields better results in the case study considered. This work demonstrates that reinforcement learning can be successfully applied to the problem of optimising the generation of hydro electricity. However, it does not consider a larger cascade river system with more stations.

Xu et al. (2021) applies deep Q learning to the Huanren Reservoir in China.

Their work looks at longer term optimisation of the reservoir, breaking up a year into 10-day segments. They discretise the states and actions to manage the continuous state and action space. An example of this is modelling a continuous water level in 1 metre segments. Their approach successfully outperforms stochastic dynamic programming and a decision tree approach, indicating the promise of Q learning applied to hydropower generation. They also explore effects of the granularity of the discretised states and actions. They demonstrate that a more granular state discretisation of 0.25m segments gives a better policy, but the model takes longer to train. This trade-off will be also be important to explore in the Waikato River hydro control problem considered in this thesis. A constant efficiency is assumed in the study so the method is not directly applicable to the Waikato River hydro control problem, which requires optimising for efficiency.

Abdalla (2007) apply reinforcement learning to the reservoirs on the Peace and Columbia Rivers in British Columbia. They focus on the long term optimisation of revenue from cascade river schemes. A Q learning approach is used, comparing function approximation of state values with a lookup table of state values. Abdalla trained the Q learning model by randomly sampling the state space and running an episode for a pre-determined planning horizon. This approach of training a reinforcement learning model by taking a state and running for a pre-determined planning horizon is also applicable to the scenario considered in this thesis.

Shabani (2009) implements reinforcement learning to optimize the hydro scheme subject to flood control constraints in the Columbia River Basin. This work applies Q learning. It optimises the revenue from selling and purchasing energy to and from the US and Alberta as well as valuing the amount of water in storage. Shabani emphasises the importance of modelling uncertainty to get a more robust solution. In particular, there is a focus on uncertainty in electricity demand, water inflows, and market prices. Similar uncertainties are present in the Waikato River hydro control problem considered in this thesis

so care is needed to ensure these uncertainties are modelled.

Riemer-Sørensen and Rosenlund (2020) use deep reinforcement learning to determine the long term scheduling of hydro electricity. Their work uses actor-critic reinforcement learning (a policy gradient method). The scenario considered is slightly different from the Waikato River hydro control problem considered in this thesis, as it has a longer term focus and brings in pricing to optimise for revenue. It ignores generation efficiency. The efficiency optimisation in the short term is a key focus of the work considered in this thesis. While there are major differences in the problem formulation, the deep reinforcement learning approach using the actor-critic approach may be a valid alternative to the Q learning method used in this thesis.

Luo et al. (2025), in work published just before submission of this thesis, apply reinforcement learning to hydro generation in the Dadu River basin. This is a cascade river system with three stations and their optimisation is done in real-time so their research is particularly relevant for the Waikato River hydro control problem considered in this thesis. In particular, they look at scheduling every 15 minutes for the day ahead. A policy gradient method called soft actor critic (SAC) is used. They combine SAC with evolutionary hindsight experience replay (EVHER), a method for experience replay to help with learning. Part of the objective function they use is to penalize time spent close to lake limits. This is used to produce a policy that keeps lake levels at an acceptable target level, not too high or low. This approach may be useful in the hydro control scenario considered in this thesis, in particular given that the approach developed previously, "AWMA" (Section 1), produced solutions that emptied out lakes. Moreover, keeping lake levels away from the extremes is also desirable in the Waikato River hydro control problem.

In similarly recent work, Phankamolsil et al. (2025) apply reinforcement learning to a multi-reservoir hydro system in the Chao Phraya River Basin. They use deep deterministic policy gradients (DDPG) to maintain target levels at reservoirs in the long term. Although this hydro scheme does not gener-

ate electricity, the control problem shares many common elements with the optimisation of hydro dams, such as the need to keep water levels between certain limits. Phankamolsil et al. emphasise the importance of the formulation of the reward function in reinforcement learning: "The design of reward function is one of the critical aspects of DRL applications for optimal operation of reservoir systems". Their work indicates that policy gradient methods such as DDPG could be useful for the Waikato River hydro control problem and that the design of the reward function is important. The recency of the work, and the work of Luo et al., also demonstrates the continued relevance of reinforcement learning in the optimization of cascade river systems.

Wu et al. (2024) apply reinforcement learning to a dual-hydropower reservoir system in the Colorado River Valley. Their work explores using a policy gradient method with a transformer model to encode the state space into a more compact encoding with the aim to speed up reinforcement learning. They primarily focus on a longer term hydro optimisation without consideration for the intra-day detail. This makes their approach not directly applicable to the Waikato River hydro control problem considered in this thesis. On the other hand, the approach of creating an embedding for the state to speed up reinforcement learning is interesting and may be worth exploring if convergence for the reinforcement learning is too slow.

4.4 Other Machine Learning Methods for Hydro Electricity

Chen et al. (2020) present Gaussian process regression (GPR) as a methodology for mapping outflows of upstream reservoirs to inflows of downstream reservoirs, focusing on the application to the Three Gorges-Gezhouba cascade reservoirs (TGGCR). The primary area of interest is the short term modelling of flows, which is relevant in a cascade river system like the Waikato Hydro Scheme. Current practice for the Waikato Hydro Scheme is to use a lag model.

A lag model assumes that water from an upstream station takes a constant "travel time" to reach the downstream station. The true travel time may not be constant and may vary with river conditions such as weather and flow rate. The GPR method, a form of Bayesian regression, provides a more flexible approach. In particular, it is found to characterise the hourly flow for the TGGCR better than the lag model. Thus, it may improve estimation of flow accuracy for the Waikato Hydro Scheme as well.

4.5 Literature Summary

This thesis presents a reinforcement learning approach for the real-time optimisation of the Waikato Hydro Scheme. This method lessens the "curse of dimensionality" discussed above by Giuliani et al. (2021). The review by Bernardes et al. (2022) highlights the gap in current research in the real time cascade river optimisation space. Castro-Freibott et al. (2025) provide recent success in the application of reinforcement learning for optimising a cascade river system. Ye et al. (2023) present a model predictive control approach that works for their use case but would require unacceptable simplifications to adjust to the Waikato River hydro control problem considered in this thesis. Lee and Labadie (2007), Xu et al. (2021), Abdalla (2007) and Shabani (2009) apply Q learning successfully to hydro electricity but in different contexts to the hydro control setting we consider. They discuss various ways of formulating rewards and managing large action spaces, providing justification for the problem formulation in the next section. Riemer-Sørensen and Rosenlund (2020), Luo et al. (2025), Phankamolsil et al. (2025) and Wu et al. (2024) discuss applying policy gradient methods to various hydro schemes. They discuss experience replay and reward formulations that provide a basis for the practical application of reinforcement learning. Policy gradient methods are not used in this thesis but are discussed in the part of this thesis considering future work (Section 9.1). Chen et al. (2020) use Gaussian process regression

to improve the accuracy of flow forecasting in cascade reservoirs. This could also form part of future work (Section 9.1).

Chapter 5

Problem Formulation

This chapter describes how the Waikato River hydro control problem can be formulated as an MDP. Lee and Labadie (2007) gives a general framework for modelling a river system as an MDP and applying reinforcement learning. Some changes to the formulation are required because the Waikato Hydro Scheme has many more stations than the Guem River, particularly for actions. The actions are formulated in Section 5.2 with an approach to reduce the size of the action space.

5.1 States

In the environment of the MDP, the states represent the physical state of the Waikato river as well as current forecasts. The components of the physical state include headwater levels, flows, station generation, and requested total river generation. The forecast component of the state includes forecast generation, inflows, and station generation limits. The physical state component and forecast component of the state are discussed separately below.

5.1.1 River Physical State

When water is released at one station, it takes time for it to reach the next station in the chain. This travel time can be as short as twenty minutes for

water flowing from Ōhakuri to Ātiamuri. It can also be as long as twelve hours for water flowing from Aratiatia to Ōhakuri. To fully represent the flow state of the river, one needs to consider not just the current station flows, but the water in motion between stations. Thus, as part of the river state, the current flows and flow history are kept up to the maximum travel time of twelve hours. These flows include both flows through hydro stations, as penstock flow or spill flow, and tributary flows into the river. Current headwater levels are also included in the river state. As stated in Section 3.2, the Maraetai 1 and Maraetai 2 power stations are consolidated to a single Maraetai station, giving a total of eight generating hydro stations.

5.1.2 Forecast Component

The first forecast is the generation forecast. This is the estimated total generation requirement for the Waikato hydro scheme over the next 60 hours. The next is the inflow forecast. There is one forecast per station in the Waikato hydro scheme going out seven days. Finally there is the station generation limit forecast. This is based on the latest plan determining which hydro units are going to be available and which are going to be out for maintenance. These unit states are converted into a total station megawatt availability forecast. Full history of forecast generation, forecast availability, and forecast tributaries was not available for the dataset that was used so the actual historical values for the corresponding times were used as a substitute. This effectively assumes that the forecasts are totally accurate. The forecasts for the next four hours rarely change significantly close to the time. Further out than four hours gets a little bit less reliable and over a day out can be significantly different. Given that up to 4 hours is reliable, this is probably a reasonable assumption for the real-time use case looked at in this thesis. Using this assumption in a model used for longer term planning may need further investigation.

5.2 Actions

The actions represent the change in generation and spill from one time step to the next. Time steps are defined as when the Transpower dispatch generation D changes or 15 minutes pass, whichever comes first.

The actions are formulated as discrete actions encoding different ways the generation and spill could change. For example, if a dispatch instruction to generate 20MW of additional electricity is received, a possible action could be to increase generation at one station by 20MW and leave spill unchanged. Another possible action could be to split the 20MW between three different stations and to increase spill at another station by 50 cumecs (see Section 3.5 for reasons why spill might be required). To consider all possible discrete actions for assigning 20MW of generation to 8 stations, there would be $\binom{20+8-1}{8-1} = 888030$ possible actions assuming integer MW. This is before considering spill combinations as well. For reinforcement learning, this is too large an action space to use a typical discrete action approach. To reduce the number of actions, K-medoids is used to cluster similar actions together as described in Section 2.2 and Section 5.2.1 below.

After some initial experimentation, getting the model to manage both Taupō Control Gates and the generation and spill of every station on the river produced poor results. This was due to the small storage and travel time between Taupō Control Gates and Aratiatia. Hence, the formulation used in this thesis assumes that Taupō Gates is managed separately to achieve an Aratiatia headwater level of 337.5 metres above sea level. This restricts the action space to just the eight station generation and eight spill amounts.

5.2.1 Relative Generation Proportion Model

One way of clustering the actions is looking at a relative change in generation and spill from one time point to the next. For generation, this is represented by a vector of 8 values representing the proportion of the total MW change to

assign to each station. For example, if the dispatched MW change increased the total river MW by 10MW, the action represented by $[1, 0, 0, 0, 0, 0, 0, 0]$ would indicate assigning all of that 10MW change to the first station. In another example, if the dispatched MW change increased the total river MW by 30MW, the action represented by $[0, 0.5, 0.5, 0, 0, 0, 0, 0]$ would indicate assigning half of the change in generation (15MW) to the second station and the other half to the third station. K-medoids clustering is used on historical generation change data to form a set of representative generation proportions.

The spill component of the action is represented by a vector of 9 values (the extra value comes from Taupō control gates, which have controlled water flow but no generation). These values represent the change in spill at each station or at Taupō Control Gates, with Taupō Control Gates being indexed first in the vector. For example, the vector represented by $[40, 0, 0, 0, 0, 0, 0, 0, -50]$ would indicate increasing the flow at Taupō Control Gates by 40 cumecs and decreasing the flow at the last station by 50 cumecs. K-medoids clustering is used on historical spill data to form a set of representative spill changes.

For both generation and spill, changes are limited to not be below the station's minimum generation or spill or above the station's maximum generation or spill. The generation and spill representative sets are cross joined to form a combined set of actions.

One shortcoming of using relative changes to generation is that it considers all station MW setpoints equally without regard for the efficiency or feasibility of the setpoint. For example, if the Maraetai station generates 30MW and the Transpower dispatch requires reducing the total river MW by 25MW, the generation action of $[0, 0, 0, 0, 1, 0, 0, 0]$ would reduce the Maraetai generation to 5MW. 5MW has a very low efficiency, so it would be preferable to only consider this option if absolutely necessary. In a similar example, if the Transpower dispatch were to reduce the total river MW by 28MW, the same action would reduce the Maraetai generation to 2MW, which is not a feasible setpoint for any of the units there.

In practice, it was found that this resulted in most action candidates being infeasible or very inefficient giving very poor results. Hence, this method was not used in the results discussed in this thesis and was replaced by the unit order model discussed in the next section.

5.2.2 Unit Order Model

As an attempt to improve the efficiency and feasibility of action options, a unit order model was developed. This method uses two stages. The first stage consists of rankings that determine the order in which units can be turned off and on. The second stage converts the unit rankings to a station MW setpoint using efficient MW ranges.

For the first stage, each of the 39 hydro units is given a ranking from 1 to 39 to indicate the order in which units are to be turned on, so the unit with rank 1 will be the first unit to be turned on and the last unit to be turned off. Conversely, the unit with rank 39 will be the last unit to be turned on and the first unit to be turned off. In order to determine a set of different rankings, K-Medoids clustering is used on historical daily unit run hours. Using daily run hours ensures realistic unit configurations (for example, running all ten units at the Maraetai station and zero units at the downstream Waipāpa station would be unrealistic due to the extreme flow difference) while still getting a variety of different hydrology and unit availability configurations.

The second stage is based on the efficient ranges for each unit. These ranges are defined by the efficiency of generating at the relevant MW levels being over a certain threshold. To generate a river-wide total of D MW, units are turned on in rank order until D MW can be generated with all units in the efficient range. While doing this, minimum and maximum station generation constraints are followed. If it is not possible to generate D MW with all units in an efficient range, some units may generate above or below the efficient range.

This method means that different actions will usually be at an efficient

MW setpoint and will almost always be feasible. It makes the actions more focused on which units should be generating, which is a simpler problem than determining which units should be generating and how much should they be generating.

Figure 5.1 shows an example of how the unit order model takes a unit ranking and converts it into a set of generation values for each unit (which can then be summed at each station to get the station generation). This process is done with the following steps:

1. For each unit in the unit ranking, get the efficient minimum and maximum for each unit. These are given in the example Figure 5.1 as the "Efficient Minimum for Unit" and "Efficient Maximum for Unit" columns.

2. These minimum and maximum values are then summed cumulatively in the "Cumulative Minimum" and "Cumulative Maximum" columns.

3. The first row where $Cumulative\ Minimum \leq D_t \leq Cumulative\ Maximum$ is found. All rows after this row are discarded and the corresponding units considered as off at time t. This is shown in grey in Figure 5.1

4. All remaining rows and their corresponding units are considered as on for time t.

5. Let U_m be the "marginal unit". For this example the marginal unit is the row with the 34th unit highlighted in yellow. All units in rows above the "marginal unit" are assigned with the efficient maximum MW value. All units in rows below the "marginal unit" are assigned with the efficient minimum MW value. The marginal unit is chosen and MW assigned such that all units combined sum to D_t, which in this example is 500MW. In this example, the marginal MW is chosen as 19MW, which is between the minimum of 17MW and the maximum of 20MW for the 34th unit.

Unit Ranking	Efficient Minimum for Unit	Efficient Maximum for Unit	Cumulative Minimum	Cumulative Maximum	Unit MW assigned
0	22	25	22	25	25
36	20	28	42	53	28
37	20	28	62	81	28
1	22	25	84	106	25
25	9	16	93	122	16
28	23	24	116	146	24
29	23	24	139	170	24
30	23	24	162	194	24
31	23	24	185	218	24
38	20	28	205	246	28
32	17	20	222	266	20
15	28	33	250	299	33
26	9	16	259	315	16
33	17	20	276	335	20
34	17	20	293	355	19
11	28	31	321	386	28
16	28	33	349	419	28
12	28	31	377	450	28
7	16	20	393	470	16
3	23	28	416	498	23
4	23	28	439	526	23
8	16	20	455	546	0
27	9	16	464	562	0
17	28	33	492	595	0
13	28	31	520	626	0
18	28	33	548	659	0
9	16	20	564	679	0
2	22	25	576	704	0
5	23	28	599	732	0
19	28	33	627	765	0
14	28	31	655	796	0
35	17	20	672	816	0
6	23	28	695	844	0
10	16	20	711	864	0
20	28	33	739	897	0
21	28	33	767	930	0
22	28	33	795	963	0
23	28	33	823	996	0
24	28	33	851	1029	0

Figure 5.1: Example showing unit order model process for $D_t = 500MW$

There are some cases where there is no row that satisfies the condition in step 3. In this case, the same process is attempted again but instead of using efficient minimum and efficient maximum in step 1, efficient maximum and unit maximum are used. This represents running units at higher MW than efficient to achieve a feasible setpoint. If the step 3 condition is still not satisfied, the process is attempted one last time using the unit minimum and efficient minimum. This represents running units at a lower setpoint than efficient to achieve a feasible setpoint. In the rare case that this still fails to satisfy the step 3 condition, the unit order is simply removed as an action candidate and the agent will select a different unit order that is feasible according to its policy.

Spill is managed by assuming spill can only take place if a station is generating as much as it can. If a station is generating at its maximum, then the spill is controlled by the spill component of the action, similarly to the relative generation proportion method above. Otherwise, if a station is not generating at its maximum, that station's spill is set to zero.

5.3 Rewards

To achieve the business objective of improving efficiency without violating any of the operating constraints, a reward function is used. The reward consists of two parts, an infeasibility penalty and an inefficiency penalty, formulated in terms of the cost of lost generation. The reward is discounted over time by multiplying it by a discount factor at each time step as in Equation 2.4. This discounting encourages the agent to delay large penalties where possible. The goal of this reward formulation is to encourage long feasible episodes, while minimising lost generation.

5.3.1 Infeasibility Penalty

The infeasibility penalty is applied whenever a constraint is breached. This is applied to:

- Headwater level minimum and maximum constraints.
- Minimum flow constraints at Taupō Control Gates, Aratiatia, and Karāpiro. A minimum penstock and spill flow of zero is applied where there is no other limit.
- Station generation minimum and maximum constraints. These are dynamic depending on hydro unit availability at the station.

The infeasibility penalty is a large negative constant, and when it is incurred, the episode ends. Having a large negative reward at the end of the episode and applying discounting over time should encourage longer episodes where no operating constraint is breached.

5.3.2 Cost of Lost Generation

Lost generation is driven by three factors: station spill, inefficient setpoints and low headwater level. The cost of lost generation is added at every time step. Lost MW are calculated as follows:

$$MW_{lost} = k \times Q - P, \tag{5.1}$$

where k is a vector containing a constant for each station. The k constant represents a megawatts per cumec benchmark that is used to compare running efficiency. The lost megawatts are converted to lost megawatt hours by multiplying by the number of hours until the next time step.

5.4 State Transitions

The state transition function $p(s', r|s, a)$ is not easy to estimate given the continuous state space, large action space, and uncertainty in forecasts. Because of this, model free methods are used such as Monte Carlo control with neural network approximation as described in Section 2.1.5. A state transition occurs whenever a new Transpower dispatch is received or fifteen minutes have passed, whichever is sooner.

Episodes begin at a random time in a six-year dataset with the corresponding starting state. From the initial state, an action is chosen. With the state-action pair, a state transition is calculated, and a reward and a new state are generated. Another action is selected and this process is repeated until the episode ends. A terminal step is reached when either a constraint is breached or the end time T is reached and the episode ends.

Chapter 6

Simulation Description

A simulation is used for the environment of the MDP. The simulation handles initialisation and state transitions. Initialisation determines the starting state for an episode S_0. State transitions take a given state-action pair S_t, A_t and perform a sample state transition to get the next state S_{t+1} and the reward R_t. From Section 5.1, the state contains the following river state information:

- Station generation $P_{s,t}$ as the amount of generation at station s at time t
- Station spill $Spill_{s,t}$ as the amount of spill through station s at time t
- Station flow $Q_{s,t}$ as the total flow through station s at time t including penstock and spill flow
- Station headwater level $HWL_{s,t}$ as the headwater level of the station s at time t

and the following forecast information:

- Transpower generation dispatch D_t as the total generation required on the river at time t
- Tributary flows $Tributaries_{s,t}$ towards the station s at time t
- Total available station generation $MaxGen_{s,t}$, which is the maximum station s can generate at time t

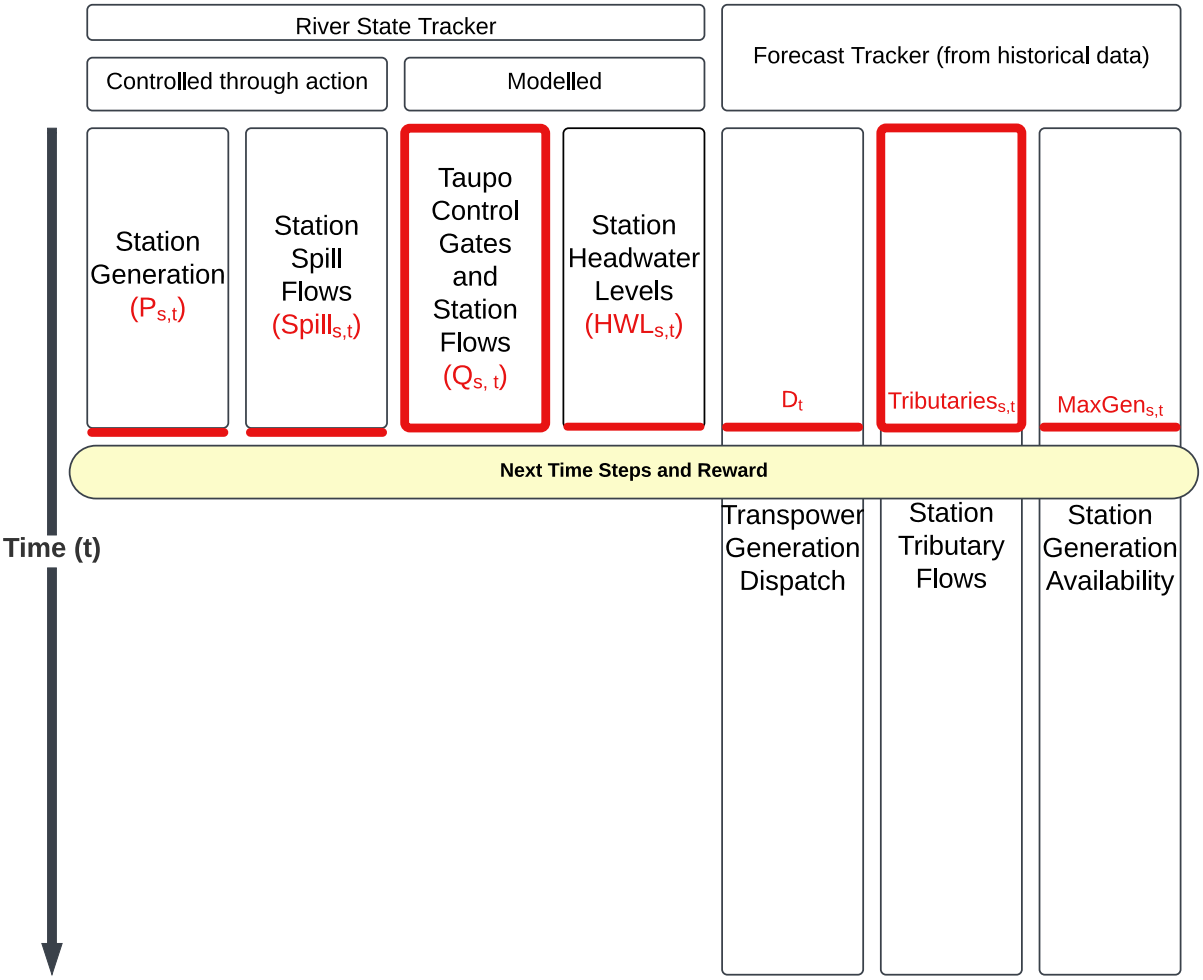


Figure 6.1: Data used in the simulation. Red highlighting indicates values used in the 'State'. A red line indicates just the latest time step is used and a box indicates that the full history with many time steps is used.

The simulator takes these state values (highlighted in Figure 6.1 in red) and uses them to generate a time step (spanning 5-15 minutes) of river state data. The station generation and spill for the next time step are taken from the agent's action. Station flows and headwater levels are modelled as discussed below. Historical data from the core dataset (described in the next section) is used to provide the forecasts and to initialise the river state. Using different time points in historical data ensures the simulator gives a variety of different scenarios with different weather, electricity demand and station availability conditions.

6.1 Core Data Set

A core data set of six years from 2017-2022 was used as the core data set for initialising simulations and providing forecast data. 1% of data was removed due to missing records for some stations and times. This left approximately 3.1 million records, each representing a minute. The attributes in this data set are:

- Timestamp of the record
- Transpower generation dispatch in megawatts (used in forecast data)
- Taupō Control Gates flow in cumecs (used in river state)
- Total station generation for the eight stations in megawatts (used in river state)
- Spill flow at each of the eight stations in cumecs (used in river state)
- Headwater level at each of the eight stations in metres above sea level (used in river state)
- Total station flow at each of the eight stations in cumecs. This includes both spill flow and penstock flow (used in river state)
- Tributary flows upstream of each of the eight stations in cumecs (used in forecast data)
- Total available generation at each of the eight stations in megawatts (used in forecast data)

This core dataset contains all data required to represent the states and actions of the MDP.

6.2 Other Simulation Data

The simulation uses some other data outside of the core data set from Section 6.1. This extra data is either used in the state transition calculations or is a

constraint used in the simulation.

- Travel times are used in the headwater level change calculation in Section 6.6 to model how long it takes water to travel from one station to the next.

- Headwater level limits are used to determine the terminal state in a simulation. If the headwater level at a station falls out of these minimum and maximum values, the episode is ended.

- Minimum flows are the given minimum flows by station as described in Section 5.3. These are used to constrain actions to only use flows equal or greater than this limit.

- The megawatt per cumec benchmark is used in the reward calculation (k_s in Equation 5.1). The station generation and flow are compared to this benchmark to determine how efficiently the station was run.

- The tailwater level model parameters are used in the tailwater level model (Represented by $C_{x,s}$ in Section 6.5.2). These parameters were determined through curve fitting on historical data.

The values for these additional data sources for the simulation are listed in the appendix.

6.3 Initialisation

The simulator chooses a random starting time in the core data set. The random starting time is chosen such that there is enough history and forecast data to accommodate all simulation and forecasting requirements. It generates two objects at initialisation, the river state tracker, and the forecast tracker as shown in Figure 6.1. The river state tracker maintains the river state information required to represent the state of the environment. It begins the simulation with only actual historical data and simulated river state data is

added to this over the course of the simulation. The forecast tracker maintains the forecast data and is always taken from the historical actual data over the course of the simulation.

The river state tracker is initialised with twelve hours of data (720 records) leading up to the start time. Using twelve hours ensures that enough flow history is retained to represent the water in motion between stations. Some minor data changes are done just to the starting state S_0 to ensure that the scenario does not result in an immediate infeasibility. These minor changes are:

- Reducing any station generation to below the total station availability.
- Clipping headwater level at each station to be at least 20 centimeters away from the minimum and maximum headwater levels for that station.
- Setting the current Transpower dispatch to be equal to the sum of the station generation.
- Increasing the spill flows at each of the stations if they are below the minimum flows.

The purpose of setting the current Transpower dispatch to be equal to the sum of the station generation is that there can be minor differences of less than 5MW between the sum of all generation and the Transpower dispatch. This adjustment removes this small discrepancy.

The forecast tracker is initialised with enough forecast records to last the duration of one episode of reinforcement learning. The maximum episode duration is set to 10,000 minutes (approximately one week). The features used in later models depend on forecast data so an extra 3,600 minutes of forecast data is used to ensure that there is always enough data to compute any of the features needed. This totals to 13,600, which is the number of records used in the forecast tracker. As discussed above, the attributes used are the Transpower dispatch generation forecast, the tributary forecast and the total

available station generation available. These are taken from historical actual values in the core data set. As stated earlier, these are actual values that are used as a substitute for the forecast values because the full forecast history was not available for this dataset.

6.4 State Transition Implementation

The state transition of the simulator provides a sample from the probability distribution $p(s', r|s, a)$ stated in Equation 2.2. The next state S_{t+1} is simulated using models taking input from S_t, A_t. The breakdown of how each component of the state is simulated, and which models are used, is listed below.

- Total station generation is taken from the generation component of the action A_t.
- The flow through Taupō control gates and spill flow at other stations are taken from the spill component of the action A_t.
- Total station flow is calculated using the efficiency curve model described in Section 6.5.1 using station generation from A_t.
- Headwater level is calculated using the headwater level model described in Section 6.6. This is calculated based on flows and tributaries from S_t. It also uses the current station flows calculated using A_t and the efficiency curve model in Section 6.5.1.
- The Transpower generation dispatch is taken from the corresponding forecast component of the current state S_t.
- Tributary flows are taken from the corresponding forecast component of the current state S_t.
- Total available generation is taken from the corresponding forecast component of the current state S_t.

The reward is determined as per Section 5.3. To calculate lost generation, station flows and generation are used as described above. To determine the infeasibility penalty, headwater levels are used as described above and compared to station headwater level limits.

The terminal state occurs and the episode ends when either a constraint is breached or 10,000 minutes have passed. States, actions, and rewards are stored in a replay buffer for later use (described later in Section 7.1.3).

6.5 Flow Model

Station flows are a part of the river state and a model is needed to simulate future flows. To calculate flows for the next state, station generation needs to be converted to a station flow. This is done with the efficiency curve method below.

6.5.1 Efficiency Curve Model

Rearranging Equation 3.4 gives:

$$Q_{s,t} = \frac{P_{s,t}}{\eta_{s,t}\rho g H_{s,t}} \tag{6.1}$$

To simplify the computation, the efficiency η is approximated as a function of power $P_{s,t}$. Where there are multiple values of η for the same $P_{s,t}$, the higher efficiency is chosen (For example, if 30MW can be generated at Maraetai by one unit or two units, the more efficient . Station head $H_{s,t}$ is also simplified as a function of station generation and headwater level. Bringing these approximations into the overall equation gives:

$$Q_{s,t} = \frac{P_{s,t}}{\eta_{s,t}(P_{s,t})\rho g H_{s,t}(P_{s,t}, HWL_{s,t}, HWL_{s+1,t})} \tag{6.2}$$

Figure 6.2 shows a comparison of the efficiency approximation with the true efficiency values at three different station heads for a unit at the Maraetai station. The close alignment between the approximation and the true values shows that this is a reasonable simplification.

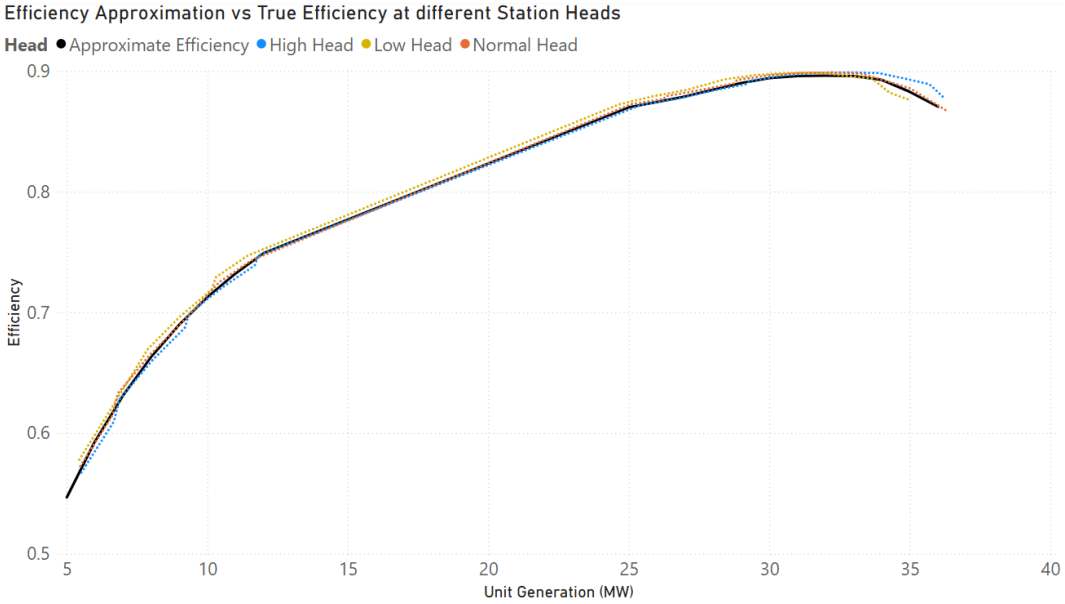


Figure 6.2: A comparison of the approximate efficiencies (solid line) against the true efficiencies at three different station heads (dotted lines). It is noteworthy that between 13MW and 24MW is missing in the chart (there is a straight line from 12MW to 25MW). This is due to 13MW to 24MW being the rough running range of this Maraetai unit.

The station generation $P_{s,t}$ is part of the action taken by the agent. Station efficiency $\eta_{s,t}$ is taken from a lookup table indexed on station generation. Station head $H_{s,t}$ is calculated as per Section 6.5.2 below. These approximations remove flow from the computation of efficiency and head to allow it to be a simple one-step calculation. Otherwise, a more time-intensive iterative solve would be necessary to calculate the flow.

6.5.2 Station Head Model

The station head is calculated by taking the headwater level of a station, less the tailwater level of the station:

$$H_{s,t}(P_{s,t}, HWL_{s,t}, HWL_{s+1,t}) = HWL_{s,t} - TWL_{s,t}(Q_{s,t}, HWL_{s+1,t}) \quad (6.3)$$

The headwater level is given from the river state but the tailwater level depends on the station flow and therefore the station generation. From a tailwater level model (internal Mercury model), tailwater level can be calculated

as:

$$TWL_{s,t}(Q_{s,t}, HWL_{s+1,t})$$
$$= \begin{cases} TWL_{Flow,s,t}(Q_{s,t}) + \max(TWL_{DSWL,s,t}(Q_{s,t}, HWL_{s+1,t}), 0) & if\ C_{type,s} = 1 \\ TWL_{Flow,s,t}(Q_{s,t}) & if C_{type,s} = 0 \end{cases} \tag{6.4}$$

where:

$$TWL_{Flow,s,t}(Q_{s,t}) = C_{a,s} + C_{b,s}Q_{s,t}^{C_{c,s}} \tag{6.5}$$

$$TWL_{DSWL,s,t}(Q_{s,t}, HWL_{s+1,t})$$
$$= C_{d,s} + C_{e,s}HWL_{s+1,t} + (C_{f,s} + C_{g,s}HWL_{s+1,t}) \ln(Q_{s,t} + C_{h,s}) \tag{6.6}$$

In these equations $C_{x,s}$ is a constant with $x \in a, b, c, d, e, f, g, h, type$ as an identifier and s as the station it applies to.

There is a slight complication in that this tailwater level model is a function of the flow $Q_{s,t}$, the object of the calculation. To resolve this, a simplified generation to flow conversion is used for the tailwater level calculation.

6.5.3 Linear Flow Model

The linear flow method assumes constant station head and efficiency. It is less accurate than the efficiency curve model, but it gives a good approximation for the purposes of calculating the tailwater level. The simple method for converting station generation into station flow is to use a linear model where flow is directly proportional to generation. In this model, flow is calculated as: For each station s

$$Q_{s,t} = P_{s,t} \times k_s \ \forall\ s \in \mathcal{S}, \tag{6.7}$$

where

- $\mathcal{S}$ is the set of all stations
- k_s is a constant multiplier determined by best fit on historical data

From Equation 6.7, the tailwater level can be converted from a function of flow. So the station head model can be transformed to only need station generation and headwater levels as follows:

$$H_{s,t}(P_{s,t}, HWL_{s,t}, HWL_{s+1,t}) = HWL_{s,t} - TWL_{s,t}(Q_{s,t}, HWL_{s+1,t}) \quad (6.8)$$

$$= HWL_{s,t} - TWL_{s,t}(P_{s,t} \times k_s, HWL_{s+1,t}) \quad (6.9)$$

6.6 Headwater Level Model

The change in headwater level is based on a water balance, so the volume of water in the system is conserved. When the volume of water going into a reservoir equals the volume of water going out of the reservoir, there is no change in headwater level. If the flow coming into the station exceeds the flow going out, the headwater level at the station is expected to rise. Conversely, if the flow coming into the station is less than the flow going out, the headwater level at the station is expected to drop. The flow coming into the station comes from two sources, tributary flows, and station flows from the upstream station:

$$WaterIn_{s,t} = Q_{s-1,t} + Tributaries_{s,t} \quad (6.10)$$

The flow going out of a station is the total station flow including penstock and spill flows:

$$WaterOut_{s,t} = Q_{s,t} \quad (6.11)$$

The change in headwater level at a station is governed by the equation below:

$$\Delta HWL_{s,t} = Days \times \frac{(WaterIn_{s,t-travel\ times_s} - WaterOut_{s,t})}{FillFactor_s}, \quad (6.12)$$

where

- $\Delta HWL_{s,t}$ is the change in headwater level at station s and time t. The change is measured in metres.

- $Days$ is the time duration that the headwater level change is measured over
- $WaterIn$ and $WaterOut$ are calculated as above.
- $FillFactor$ is the volume of water in one metre of headwater level measured in cubic metre days per metre. A high fill factor will mean a greater flow difference is required for the same headwater level change and vice versa.
- $travel\ time_s$ is the amount of time taken for water to flow from the upstream station to station s measured in hours.

6.7 Simulation Worked Example

In the following, an example is considered with a walk through the calculation of one state transition $S, A \rightarrow S', R$. Values for stations will be listed in river order (see Figure 3.1):

- Taupō Control Gates (spill and flow only)
- Aratiatia
- Ōhakuri
- Ātiamuri
- Whakamaru
- Maraetai
- Waipāpa
- Arapuni
- Karāpiro

The initial state is assumed to take the values stated in the following.

The values used for the river state of the example starting state S are:

- Station Generation $P_{s,t}$ of (15, 0, 0, 0, 0, 0, 0, 40) in megawatts
- Station Spill $Spill_{s,t}$ of (55, 0, 0, 0, 0, 0, 0, 0, 0) in cumecs
- Station Flow $Q_{s,t}$ of (50, 50.4, 0, 0, 0, 0, 0, 0, 149.3) for all t in the last 6 hours and (195, 200, 0, 0, 0, 0, 0, 0, 250) for all t from 12 hours ago to 6 hours ago in cumecs
- Station headwater level $HWL_{s,t}$ of (337.5, 287.0, 252.0, 226.0, 188.5, 127.0, 110.5, 52.0) in metres above sea level

The values used for the forecasts of the example starting state S are:

- Transpower dispatch generation D_t of 85 megawatts for the next 15 minutes and 100 megawatts for all t after that
- Tributary flows $Tributaries_{s,t}$ of (5, 5, 5, 5, 5, 5, 5, 5) cumecs for all t in the last 12 hours, (6, 6, 6, 6, 6, 6, 6, 6) cumecs for all t in the next 15 minutes and (7, 7, 7, 7, 7, 7, 7, 7) cumecs for all t after that.
- Station generation availability $MaxGen_{s,t}$ of (78, 112, 74, 116, 352, 54, 192, 96) for all t in megawatts.

Figure 6.3 shows a visualisation of the initial state S and how it is represented in the river state and forecast trackers.

From the Transpower dispatch of 85MW and the current total generation of $15+40 = 55MW$, there is a 30MW difference. In this example, it is assumed that the agent takes the following action:

- Station Generation $P_{s,t}$ is set to (17, 0, 0, 0, 28, 0, 0, 40) megawatts
- Station Spill $Spill_{s,t}$ is set to (60, 0, 0, 0, 50, 0, 0, 0, 0) cumecs

Of the 30MW of extra generation, 28MW has been assigned to Maraetai and 2MW has been added to Aratiatia. This is below the generation availability for each of the stations so this action is feasible. Taupō control gates has

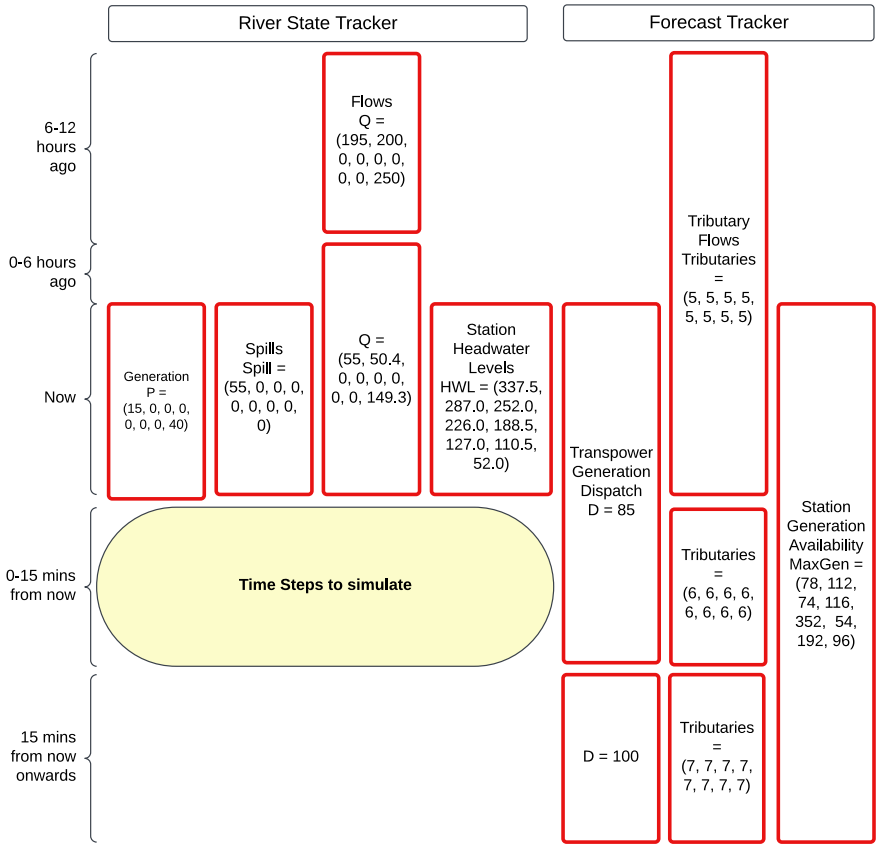


Figure 6.3: Initial state example state representation in river state and forecast trackers

also been increased from 55 cumecs to 60 cumecs of flow. Whakamaru station has begun spilling at 50 cumecs.

The next state S', takes place at the next Transpower dispatch change, 15 minutes after S. To compute S', new values for the river state and forecasts are needed.

6.7.1 Worked Example - Forecast State Transition

For the forecasts, the current time t is shifted forward 15 minutes. The new values for S' become:

- Transpower dispatch becomes 100MW for all future t.
- Tributary flows become (5, 5, 5, 5, 5, 5, 5, 5) cumecs for all t between 12 hours ago to 15 minutes ago. They become (6, 6, 6, 6, 6, 6, 6, 6) cumecs for all t in the past 15 minutes and (7, 7, 7, 7, 7, 7, 7, 7) cumecs for all t in the future.
- Station generation availability $MaxGen_{s,t}$ of (78, 112, 74, 116, 352, 54, 192, 96) megawatts for all t.

6.7.2 Worked Example - River State Transition

For the river state, the station generation and spill can be taken directly from the action A. For station flow, Equation 6.2 can be used. Below, Aratiatia is used as an example. At 17 megawatts (or 17×10^6 watts), Aratiatia has an efficiency $\eta_{Aratiatia}(17) = 0.869$.

First, the tailwater level needs to be calculated. Aratiatia has type:

$$C_{type,Aratiatia} = 0 \tag{6.13}$$

and constants of:

$$C_{a,Aratiatia} = 300.31$$

$$C_{b,Aratiatia} = 0.1179$$

$$C_{c,Aratiatia} = 0.634$$

The linear flow model (equation 6.7) can be used to estimate a station flow for the purposes of calculating the station tailwater level.

$$\begin{aligned} Q_{s,t} &= P_{s,t} \times k_s \\ &= 17 \times 3.52 \\ &= 59.8 \; cumecs \end{aligned}$$

From equations 6.4 and 6.5:

$$\begin{aligned} TWL_{Aratiatia}(Q_{Aratiatia}) &= C_{a,Aratiatia} + C_{b,Aratiatia} Q_{Aratiatia}^{C_{c,Aratiatia}} \\ &= 300.31 + 0.1179 \times 59.8^{0.634} \\ &= 301.89 \; masl, \end{aligned}$$

where masl stands for metres above sea level.

Now with headwater and tailwater levels, the station head can be calculated from Equation 6.8

$$\begin{aligned} & H_{s,t}(P_{Aratiatia}, HWL_{Aratiatia}, HWL_{\bar{O}hakuri}) \\ &= HWL_{Aratiatia} - TWL_{Aratiatia}(P_{Aratiatia} \times k_{Aratiatia}, HWL_{\bar{O}hakuri}) \\ &= 337.5 - 301.89 \\ &= 35.6 \; metres \end{aligned}$$

Bringing the head $H_{s,t}$, the station generation $P_{s,t}$ and efficiency $\eta_{s,t}$ together, the penstock flow for the station can be calculated. From Equation 3.4

the Aratiatia penstock flow is calculated as:

$$
\begin{aligned}
&Q_{Aratiatia} \\
&= \frac{P_{Aratiatia}}{\eta_{Aratiatia}(P_{Aratiatia})\rho g H_{Aratiatia}(P_{Aratiatia}, HWL_{Aratiatia}, HWL_{\bar{O}hakuri})} \\
&= \frac{17 \times 10^6}{0.869 \times 997 \times 9.81 \times 35.6} \\
&= 56 \; cumecs
\end{aligned}
$$

All other stations calculate penstock flow in the same manner. This gives penstock flows of (0, 56, 0, 0, 0, 53, 0, 0, 149) cumecs. Adding in spill flows of (60, 0, 0, 0, 50, 0, 0, 0, 0) cumecs gives a total station flow of (60, 56, 0, 0, 50, 53, 0, 0, 149) cumecs.

Finally, the change in headwater level over the next fifteen minutes is calculated. Again Aratiatia is used as an example. From Equation 6.12:

$$
\begin{aligned}
\Delta HWL_{Aratiatia,t} &= Days \times \frac{(WaterIn_{Aratiatia,t-60m} - WaterOut_{Aratiatia,t})}{FillFactor_{Aratiatia}} \\
&= Days \times \frac{(Q_{Taup\bar{o},t-60m} + Tributaries_{Aratiatia,t-60m} - Q_{Aratiatia,t})}{FillFactor_{Aratiatia}} \\
&= \frac{15}{24 \times 60} \times \frac{55 + 5 - 56}{5.93} \\
&= 0.007 \; metres
\end{aligned} \tag{6.14}
$$

55 cumecs is used as the water in from the upstream Taupō control gates 1 hour ago as that is the travel time between Taupō control gates and Aratiatia. The 56 cumecs is taken from the current Aratiatia flow as calculated in Equation 6.14.

Ōhakuri has a longer travel time at 12 hours. The headwater level change for Ōhakuri would be calculated as:

$$\Delta HWL_{\bar{O}hakuri,t} = Days \times \frac{(WaterIn_{\bar{O}hakuri,t-12h} - WaterOut_{\bar{O}hakuri,t})}{FillFactor_{\bar{O}hakuri}}$$
$$= Days \times \frac{(Q_{Aratiatia,t-12h} + Tributaries_{\bar{O}hakuri,t-12h} - Q_{\bar{O}hakuri,t})}{FillFactor_{\bar{O}hakuri}}$$
$$= \frac{15}{24 \times 60} \times \frac{200 + 5 - 0}{142.09}$$
$$= 0.015\ metres \tag{6.15}$$

The other headwater level changes are calculated in a similar way. These changes can be added to the headwater levels from state S to give headwater levels of (337.507, 287.015, 252.002, 225.994, 188.491, 126.997, 110.500, 51.982) masl for state S'. The complete information for the river state in S' is:

- Station Generation $P_{s,t}$ of (17, 0, 0, 0, 28, 0, 0, 40) megawatts
- Station Spill $Spill_{s,t}$ of (60, 0, 0, 0, 50, 0, 0, 0, 0) cumecs
- Station flow $Q_{s,t}$ of (60, 56, 0, 0, 50, 53, 0, 0, 149) cumecs
- Station headwater level $HWL_{s,t}$ of (337.507, 287.015, 252.002, 225.994, 188.491, 126.997, 110.500, 51.982) masl

With the river state and forecast transition done, everything can be brought together to form the next state S' as shown in Figure 6.4.

6.7.3 Worked Example - Reward Calculation

The reward for this example is calculated as a weighted sum of the lost generation and feasibility scores. The weightings are fixed hyperparameters set in the model. For this example, the weightings will be 0.1 for the lost generation and 10^4 for feasibility.

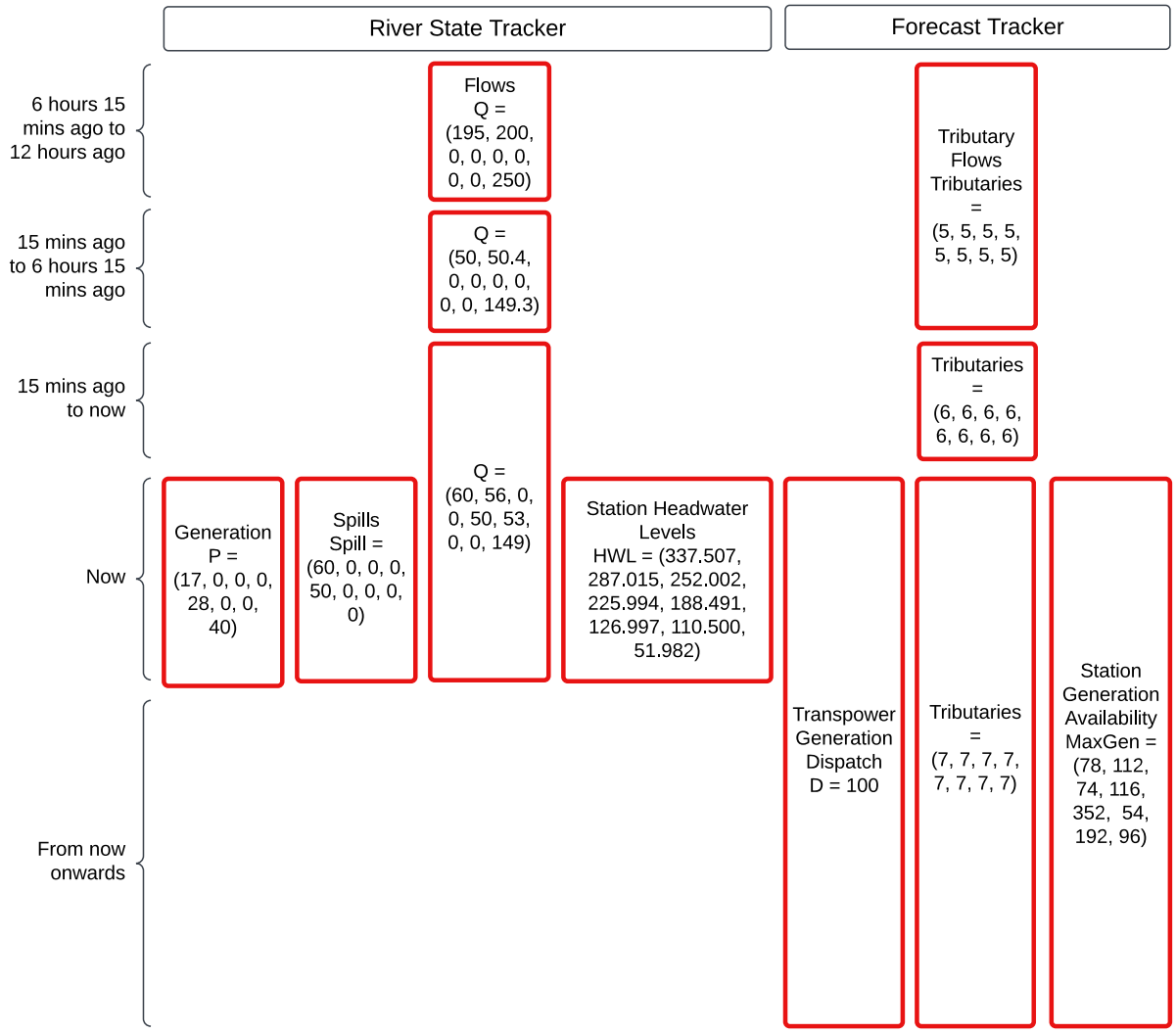


Figure 6.4: New state S' representation in river state and forecast trackers

Lost Generation The lost generation can be calculated with Equation 5.1. Taking Aratiatia as an example, the calculation would be:

$$MW_{lost}^{Aratiatia} = k_{Aratiatia} \times Q_{Aratiatia} - P_{Aratiatia} \tag{6.16}$$

$$= 0.315016 \times 56 - 17 \tag{6.17}$$

$$= 0.64\ megawatts \tag{6.18}$$

All other stations are calculated in the same way and summed together for the total lost generation for that time step. This gives the lost generation score which is then multiplied by 0.1 to give the lost generation component of the reward for this time step.

Feasibility In this case, all the headwater levels are between the minimum and maximum headwater levels (see Appendix 9.2 for minimum and maximum values). This means there is a score of zero for feasibility. If one of the headwater levels had been outside of the acceptable range, the score would have been -1. This would have then been multiplied by the feasibility weighting of 10^4. This would have given -1×10^4 for the feasibility component of the reward.

Chapter 7

Applying Reinforcement Learning

There are many ways of applying reinforcement learning to the Waikato River hydro control problem. In this chapter, three ways of setting up a network architecture and training the model are discussed, and the next chapters will discuss the experiments and results for each of these algorithms to determine which approach is most effective for this problem. As discussed in Section 5.3, the goal of the agent is to achieve long, feasible episodes and minimal lost generation.

7.1 Algorithm 1 - Monte Carlo Control with scalar features

For this algorithm, Monte Carlo control with neural network approximation is used as described in Section 2.1.5. The simulation used is as described in Chapter 6. An ϵ-greedy policy π is used to select actions (see Equation 2.7 for more detail). The action value function uses the model features $\mathcal{F}(s, a)$ described below in Section 7.1.2:

$$\hat{q}_\pi(s, a, w) = \hat{q}_\pi(\mathcal{F}(s, a), \mathbf{w}) \tag{7.1}$$

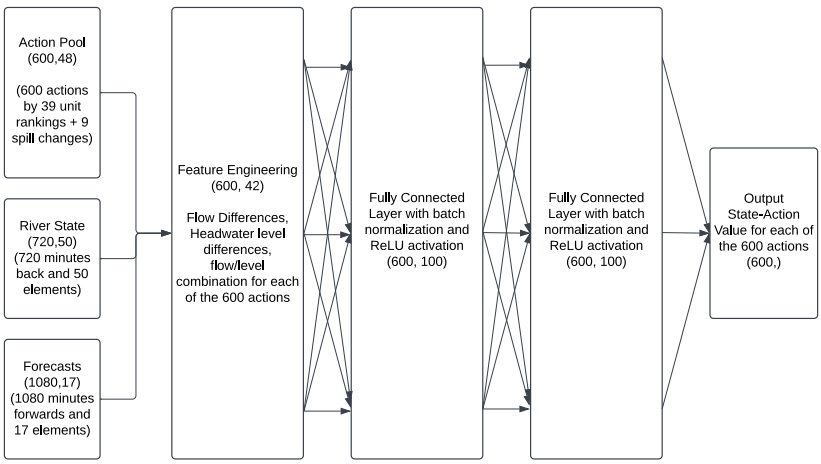


Figure 7.1: Example network architecture for $N_{layers} = 2$ hidden layers $N_{size} = 100$ hidden units per layer with an action pool size of $|\mathcal{A}| = 600$

The model is a standard fully connected neural network with ReLU activation. Batch normalisation is applied before each layer because this allows for higher learning rates to be used, speeding up training (Ioffe and Szegedy, 2015). The network maps the feature set $\mathcal{F}(S_t, A_t)$ to the output target G_t. The network has N_{layers} hidden layers of size N_{size}. An example network architecture is given in Figure 7.1. The action pool gives the possible values that A_t can take and forms part of the input. This is described in the next subsection. The state S_t is initialised from historical data as described in Section 6.3. From there, the simulation computes state transitions for future states based on the selected action as per Section 6.4.

7.1.1 Action Pool

For this algorithm, an action pool consists of $|\mathcal{A}_{unit\ ranking}| \times |\mathcal{A}_{spill\ changes}| = |\mathcal{A}|$ actions. The action pool is the Cartesian product of two sets: a ranking of each of the 39 units and a spill change value for each of the eight stations and Taupō control gates. The unit ranking is a set of $|\mathcal{A}_{unit\ ranking}|$ clusters generated by fitting the clusters to historical unit order data using the K-

medoids algorithm using the unit order model described in Section 5.2.2. The spill change values are a set of size $|\mathcal{A}_{spill\ changes}|$ of changes to station spill, either increasing, decreasing or maintaining the amount of spill as described in Section 5.2.1.

7.1.2 Model Features

The set of features $\mathcal{F}(s, a)$ is a function of the state and action components. The goal of these features is to help the model to predict a state-action value that closely matches the rewards experienced. Given that the biggest driver of the reward is the infeasibility penalty (see Section 5.3.1), it makes sense to focus on features that inform whether an action is likely to cause an infeasible state. The distance of a station's headwater level from the boundary tells us about the likelihood of hitting that boundary. If a station is close to full, it is much more likely to encounter an infeasible state through overfilling that station than if the station was closer to the middle of its range. The flow difference also gives valuable information. If there is a large difference between a station and the upstream station, the station may fill or empty quickly and encounter an infeasible state. Keeping the stations "on flow", where each station's flow is similar to its upstream station, reduces the likelihood of encountering an infeasible state. There are three groups of features within $\mathcal{F}$:

- Features describing what the difference in flow will be after action a is taken, between a station and its upstream station. There are two sets, one for the positive difference (where the station is filling over time) and one for the negative difference (where the station is emptying over time). When a headwater level is close to a boundary, the sign of the flow difference makes a big difference to the outcome. For example, if a station is close to full and the flow difference is indicating that the station is still filling, this will have a high weight as the simulation is approaching an infeasible state, which will lead to termination of the episode and a large negative reward. On the other hand, if a station

is close to full and the flow difference is indicating that the station is emptying, the rate of emptying is less important as it is less likely to encounter a terminal state soon.

These flow differences are defined by:

$$f_{negative\ flow\ difference,s} = \max(Q_{s,t} - Q_{s-1,t} - Tributaries_{s,t}, 0) \quad (7.2)$$

$$f_{positive\ flow\ difference,s} = \max(Q_{s-1,t} + Tributaries_{s,t} - Q_{s,t}, 0) \quad (7.3)$$

- Log differences between the headwater level and the headwater level limits. There are two sets, one for the difference to the upper bound and the other for the lower bound. These are defined by:

$$f_{max\ hwl\ difference,s} = \ln(\max(MaxHWL_s - HWL_{s,t}, 0.01)) \quad (7.4)$$

$$f_{min\ hwl\ difference,s} = \ln(\max(HWL_{s,t} - MinHWL_s, 0.01)) \quad (7.5)$$

- A combination multiplying together the above features for corresponding stations, again with two sets, one to indicate when the headwater level is close to full and has a flow difference indicating that it is filling, and the other indicating when the headwater level is close to empty and the flow difference indicates that it is emptying. These features are useful to give the model an indication of when a model is both close to a headwater level boundary and getting closer. These are defined by:

$$f_{station\ filling,s} = \min(f_{positive\ flow\ difference,s} \times f_{max\ hwl\ difference,s}, 0) \quad (7.6)$$

$$f_{station\ emptying,s} = \min(f_{negative\ flow\ difference,s} \times f_{min\ hwl\ difference,s}, 0) \quad (7.7)$$

- A feature giving the lost generation due to inefficiency is useful to help the model choose actions that are efficient. This is calculated using the same logic as in Section 6.7.3.

$$f_{lost\ MW} = \sum_s MW^s_{lost} \quad (7.8)$$

7.1.3 Replay Buffer

To help with learning, state transition information is processed into a replay buffer $\mathcal{B}$. Sampling from a replay buffer is important because during simulations, one time step is highly correlated to the next step. Uniform random sampling from a sufficiently large replay buffer is a good approximation of the independent, identical distribution condition required for stable training during stochastic gradient descent. In each state transition, an action A_t is taken from state S_t. The feature set corresponding to that action, $\mathcal{F}(S_t, A_t)$, and the discounted future return G_t are recorded. Each record in the replay buffer is a tuple $(f_t \; \forall f_t \in \mathcal{F}(S_t, A_t), G_t)$. Only the features corresponding to the action that has been taken need to be recorded as the other features are unused in the training process. The replay buffer stores up to $|\mathcal{B}|$ tuples and the oldest records are overwritten as new episode data is generated.

7.1.4 Training and Evaluation

For each iteration of the training loop, an episode is generated in the simulator according to the current policy. Then, the episode is processed into the replay buffer. Next, a batch $b_1, b_2, b_3 ... b_{|Batch|} \overset{\text{i.i.d.}}{\sim} \mathcal{B}$ is uniformly sampled from the replay buffer. This batch is then passed to a stochastic gradient descent optimizer with the objective function to minimize being $(\hat{q}_\pi(\mathcal{F}(S_b, A_b), \mathbf{w}) - G_b)^2 \; \forall b \in \{b_1, b_2, b_3 ... b_{|Batch|}\}$. The training loop is repeated for 3000 episodes. The optimizer is configured with a learning rate LR, which is a hyperparameter that takes values discussed in Section 7.1.5.

During training, it is useful to monitor how well the policy performs. Evaluation is done every 150 episodes by running a set of 75 validation episodes, setting $\epsilon = 0$ to disable random exploration. A separate period in the data set is used to extract validation episodes. Another period in the dataset is left unseen during training and validation to form the final test set to evaluate performance of the final model. The average simulation duration and efficiency are recorded. Evaluation is done over the same set of episodes for consistency

across different hyperparameter runs.

7.1.5 Hyperparameter Tuning

During training, some hyperparameters of the learning algorithm were fixed, while others were tuned using validation data. The following fixed parameters were used:

- Reward discount factor $\gamma = 0.995$
- Batch size $|Batch| = 1024$
- Epsilon greedy random action selection rate $\epsilon = 0.1$
- Replay buffer size $|\mathcal{B}| = 50000$ *records*

The tuned hyperparameters were determined using a grid search approach. This approach involves taking the overall set as the Cartesian product of each of the hyperparameter sets. The model was trained and evaluated with each member of the overall set and the resulting mean episode duration and lost MW recorded. To determine which set of hyperparameters had the best performance, the mean episode duration and lost MW were compared.

Two of the hyperparameters that were tuned are the number of hidden layers N_{layers} in the network as well as the size of those layers N_{size}. The network was trained and evaluated on the following options:

- $N_{layers} \in \{0, 1, 2\}$
- $N_{size} \in \{10, 40, 100\}$

The learning rate was also tuned by considering values for $LR \in \{1 \times 10^{-4}, 3 \times 10^{-3}, 1 \times 10^{-2}\}$

Another hyperparameter is the size of the action pool used. The number of distinct unit rankings and the number of spill changes in the action pool can be varied. The network was trained and evaluated on the following options:

- $|\mathcal{A}_{unit\ ranking}| \in \{50, 200\}$

- $|\mathcal{A}_{spill\ changes}| \in \{3, 23\}$

The final hyperparameter set is the efficiency reward weighting EW. This is the multiplier for the lost MW reward. A zero multiplier indicates that the lost MW is not considered in the reward. A positive multiplier indicates the lost MW is considered with a higher weighting meaning more consideration is given to efficiency while training the model. The network was trained and evaluated by considering values for $EW \in \{0, 10^{-4}, 10^{-3}\}$

Overall, the hyperparameter search space contains 324 combinations. Exhaustive search through this space is impractical due to computation limitations, so instead, the hyperparameter search is split into two phases. In the first phase, the model is trained on the hyperparameter sets for the learning rate, the number of layers and the size of those layers. The other hyperparameters were set to:

$$|\mathcal{A}_{unit\ ranking}| = 200$$
$$|\mathcal{A}_{spill\ changes}| = 3$$
$$EW = 0$$

This reduces the size of the hyperparameter space of phase one to 21 combinations [1]. The model with the best validation episode length is then carried forward to phase two. In the second phase, the model is trained with the number of layers, size of layers, and learning rate set to the best values from phase one and the model was re-trained on the 12 different combinations of efficiency weighting and size of action pool. This two-phase approach reduces the number of training runs needed from 324 to 38 [2]

[1] 21 comes from $3+3^2+3^2$ where the first term of 3 is the case of $N_{layers} = 0$, N_{size} does not apply when no hidden layers are used and three different learning rates are evaluated. The second and third terms are where $N_{layers} \in \{1, 2\}$ respectively and three different learning rates and three different hidden layer sizes are evaluated.

[2] 38 comes from 27 + 12 - 1 where 27 is number of combinations in phase one, 12 is the number of combinations in phase two, and we minus one because it is unnecessary to re-test the combination that was used for phase one.

7.2 Algorithm 2 - Monte Carlo Control with Time Series Features

One shortcoming of algorithm 1 is it only considers features based on the present time. This means it is unable to see future changes in electricity demand and unit availability, so is unable to plan for these. For example, if a long period of low demand is forecast, it might be prudent to fill up Karāpiro[3] to maintain the 148 cumec minimum flow that is required from that station. This is because during low demand it may be difficult to flow extra water down the river without spilling water. On the other hand, if a long period of high demand is forecast, it may be better to hold a lower headwater level at Karāpiro to have a buffer if large amounts of water from upstream generation arrive. It is clear that there needs to be some ability for the model to see what demand is coming at what time to be able to plan accordingly. This requires a model that can handle a time dimension to the data.

The second algorithm, designed to tackle this, builds on the approach described in the previous section. The key changes made are to the model features and architecture. More specifically, the modified approach utilises a set of features that have a time dimension. The action pool is configured to have the following sizes for the two subsets concerned:

$$|\mathcal{A}_{unit\ ranking}| = 200$$

$$|\mathcal{A}_{spill\ changes}| = 3$$

The training process is unchanged from Algorithm 1.

7.2.1 Headwater Level Projection

A key concept used in the time series features is the headwater level projection (HLP). Headwater level projection takes a proposed action A and calculates an approximation of what the headwater levels will be a certain number of

[3]Karāpiro is the last station along the river by flow order.

hours into the future. This approximation is computed by:

- Taking the unit order from action A and using the forecast river-wide total generation D_t from the current river state S where t is taken every 30 minutes from the current time.
- Applying the unit order model from Section 5.2.2 to get $\hat{P}_{s,t}$ for each of the stations and times.
- Applying the efficiency curve model from Section 6.5.1 to get $\hat{Q}s, t$ for each of the stations and times.
- Applying the headwater level model from Section 6.6 to get $H\hat{W}L_{s,t}$ for each of the stations and times.

Note that $\hat{P}$, $\hat{Q}$, and $H\hat{W}L$ are computed by assuming that action A is chosen and maintained for all times t.

7.2.2 Model Features

As in Section 7.1.2, the goal of these features is to help the model to predict a state-action value that closely matches the rewards experienced. The headwater level difference features from algorithm 1 are developed further using the HLP method (Section 7.2.1). With HLP, the headwater level is calculated every half-hour for the next 18 hours and a headwater level difference can be computed at each time step. This gives the model the ability to plan ahead and see more accurately when a station is likely to breach a headwater level limit. Combining this with demand gives the model the ability to put a higher weighting on certain station minimum and maximum limits in order to better manage known bottlenecks. Looking at the river as a whole is also important to manage Aratiatia flows, being the first station along the river. This is because if Aratiatia flows are too low, this starves the rest of the river of water, which can take a long time to fix given the 12 hour travel time between Aratiatia and the next station, Ōhakuri. Putting too much flow through Aratiatia

can yield the opposite problem and put too much water down the river leading to unnecessary spill. Using the overall river storage as one of the features gives the model the ability to judge whether an action is likely to make the river too empty or full.

In light of this discussion, the features used in algorithm 2 are log station headwater level differences, headwater level with demand combinations, the log river energy differences and a "lost GWh" feature (the energy lost due to inefficiency). These four sets of features are discussed in turn in what follows.

The station headwater level differences are calculated as:

$$f_{station\ min,s,t} = \ln(\max(HWL_{s,t} - MinHWL_s, 0.01))\ \forall s \in \mathcal{S} \tag{7.9}$$

$$f_{station\ max,s,t} = \ln(\max(MaxHWL_s - HWL_{s,t}, 0.01))\ \forall s \in \mathcal{S} \tag{7.10}$$

where $HWL_{s,t}$ is calculated by HLP as discussed above.

The headwater level with demand combinations are only used for a select few stations where the demand plays a significant role in headwater level planning. Ōhakuri, Waipāpa and Karāpiro all require consideration of the demand forecast during headwater level planning. Below 330MW was selected as low demand as the 20th percentile demand historically. Above 610MW was selected as high demand as the 80th percentile demand historically. More specifically, the features combining headwater level and demand are computed for the previously discussed stations as follows:

$$f_{station\ min\ high\ demand,\bar{O}hakuri,t} = \begin{cases} f_{station\ min,\bar{O}hakuri,t} & if\ D_{mean} > 610 \\ 0, & otherwise \end{cases} \tag{7.11}$$

$$f_{station\ max\ high\ demand,Waip\bar{a}pa,t} = \begin{cases} f_{station\ max,Waip\bar{a}pa,t} & if\ D_{mean} > 610 \\ 0, & otherwise \end{cases} \tag{7.12}$$

$$f_{station\ min\ low\ demand,Karāpiro,t} = \begin{cases} f_{station\ min,Karāpiro,t} & if\ D_{mean} < 330 \\ 0, & otherwise \end{cases} \tag{7.13}$$

$$f_{station\ max\ low\ demand,\bar{O}hakuri,t} = \begin{cases} f_{station\ max,\bar{O}hakuri,t} & if\ D_{mean} < 330 \\ 0, & otherwise \end{cases} \tag{7.14}$$

where: $D_{mean} = \frac{\sum_t (D_t)}{1080}\ \forall t \in [0, 1080]\ minutes$

The river energy differences are calculated as:

$$f_{river\ min,s,t} = \ln(\max(E_t - MinE, 0.01)) \tag{7.15}$$

$$f_{station\ max,s,t} = \ln(\max(MaxE - E_t, 0.01)), \tag{7.16}$$

where $MinE, MaxE$ are constants, with values of 0.68 and 1.1 respectively, and E_t is the scaled total potential energy in the river, calculated using the following steps.

1. Water potential is calculated as:

$$WaterPotential_s = \sum_{i=s}^{KPO} k_s \tag{7.17}$$

   for each station. The sum is performed over the current station and all stations downstream up to the eighth and final station, Karāpiro (KPO for short).

2. Volume in transition $V_{transition,s}$ is calculated as the sum of cubic metres in transit between stations:

$$V_{transition,s} = \sum_{t=0}^{travel\ time_s} Q_{s,t} \times 3600\ seconds\ per\ hour \tag{7.18}$$

3. Volume in reservoir $V_{reservoir,s}$

$$V_{reservoir,s} = A_s \times (HWL_{s,t} - MinHWL_s) \tag{7.19}$$

   where A_s is the area of the reservoir for station s.

4. The energy at time t is calculated as

$$E_t = \sum_s WaterPotential_s \times (V_{transition,s} + V_{reservoir,s}) \tag{7.20}$$

5. The resulting energy E_t is divided by a constant 9.47×10^7 to scale the energy such that zero represents an empty river and one represents a full river assuming $V_{transition,s} = 0 \ \forall s$.

The lost GWh feature is calculated as:

$$f_{lost\ GWh} = \frac{1}{60 \times 1000} \times \sum_{s,t} MW^{s,t}_{lost} \tag{7.21}$$

for all t in the corresponding 30 minute period, where

$$MW^{s,t}_{lost} = k_s \times Q_{s,t} - P_{s,t} \tag{7.22}$$

similar to Section 6.7.3. The divisions by 60 and 1000 are to convert minutes to hours and MWh to GWh respectively.

Considering the above features, the model has features $\mathcal{F}_{v,s,t}$, where s is the station the feature applies to, t is the time dimension of the feature and the type of the feature is represented by:

$$v \in \{station\ min, \tag{7.23}$$
$$station\ max, \tag{7.24}$$
$$river\ min, \tag{7.25}$$
$$river\ max, \tag{7.26}$$
$$station\ min\ low\ demand, \tag{7.27}$$
$$station\ min\ high\ demand, \tag{7.28}$$
$$station\ max\ low\ demand, \tag{7.29}$$
$$station\ max\ high\ demand, \tag{7.30}$$
$$lost\ GWh\} \tag{7.31}$$

The "station min" and "station max" features are for 7 stations each (all stations except the first, Aratiatia). Overall, this means 21 features that are used in the model to predict the state-action value.

7.2.3 Model Architecture

The input to the model consists of the station and river energy differences as discussed in the previous section. To aggregate the time dimension, each headwater level difference is multiplied by an exponential decay function with a learned parameter and then summed across all times. This gives the model the ability to learn whether to focus more on headwater level differences closer to the current time or also consider headwater level differences further out in time. For the modified approach with time series features, the implementation of the neural network model is modified to create and work with the aggregated features rather than the original features. More specifically, the weighted average is computed as:

$$\sum_{t} e^{-k_{v,s}} \times \mathcal{F}_{v,s,t} \tag{7.32}$$

where $k_{v,s}$ is a decay weight learnt by the model.

The output of the model is then simply taken as a weighted sum of all these aggregated features. The overall architecture for this model is shown in Figure 7.2. Note that this model is a very shallow neural network with a low number of trainable weights and with very few layers between the input features and the resulting output.

Due to the low number of weights in the neural network, it is possible to initialise the model with weights corresponding to a "sensible" starting point based on prior business knowledge. Thus, in the final dense layer, all features are initialised with equal importance with weights of 0.1. The decay rates are initialised with values roughly inversely correlated to the corresponding station travel times. This is because for stations with short travel times, only headwater level projections closer to the present are relevant. Therefore, it makes sense to have a high decay rate to limit the effect of times further away. For stations with long travel times, forward planning is often important because it takes much longer before any upstream flow changes will be reflected in the station headwater level. These stations are initialised with a low or even

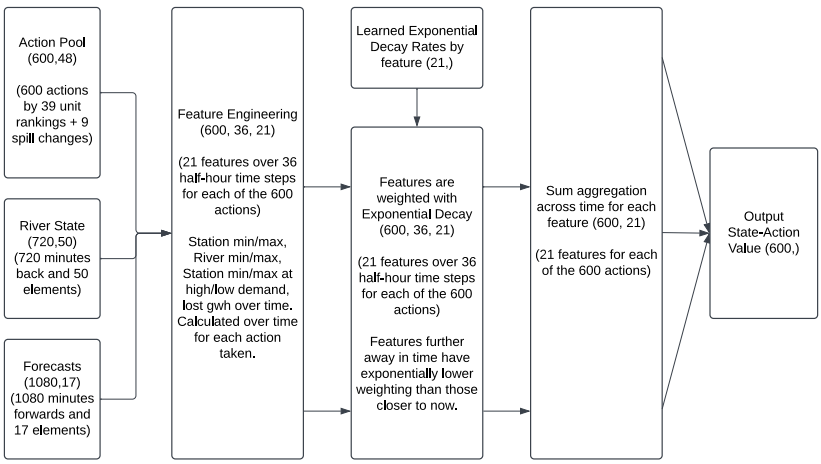


Figure 7.2: Example network architecture for an action pool size of $|\mathcal{A}| = 600$

zero decay rate so that all times are given consideration. The decay rate initial values $k_{v,s}$ can be found in Appendix 9.3.

7.2.4 Hyperparameter Tuning

To find appropriate hyperparameter values for the modified model with time series features, the same grid search approach is used as in Section 7.1.5. The hyperparameters that are tuned are the learning rate and the efficiency reward weighting.

The learning rate was chosen by considering values for $LR \in \{1 \times 10^{-4}, 3 \times 10^{-3}, 1 \times 10^{-2}\}$

The efficiency reward weighting was chosen by considering values for $EW \in \{0, 10^{-4}, 10^{-3}\}$

In the experiments, the model is also trained without the "sensible" weight initialisation to determine the effectiveness of this approach.

7.3 Algorithm 3 - Action Selection using Value Function Hill Climbing

In the previous two approaches, the set of available actions is limited, particularly through the unit order. It contains a variety of actions but is only a small subset of the full action space. To give the agent the ability to select any action in the action space, the hill-climbing algorithm discussed in Section 2.3 can be used to search the action space once a policy has been trained using, for example, the neural network based on time series features discussed above. It is worth noting that the hill-climbing is only applied when a model is deployed or evaluated, not when the neural network weights are trained. The hill-climbing starts from the action used in the previous time step. The action taken in the previous time step was based on a headwater level projection over several hours. Using this same action a few minutes later is likely to be near-optimal so this reduces the number of hill-climbing steps that need to be performed. To perform hill-climbing, a neighbourhood rule is defined. An action a' is considered a neighbour of action a if either of the following conditions are met:

- The unit order from action a' is the same as the unit order from action a except for two units i and j that have their order swapped. All spill values are identical.
- The unit order from action a' is the same as the unit order from action a. The spill value for one station is 30 cumecs higher or lower in action a' compared to action a

The set $\mathcal{C}(a)$ contains the set of all actions that are neighbours of action a. The hill-climbing algorithm works by performing the following steps:

1. Take a trained model giving an approximate state-action value function $\hat{q}_\pi(s, a)$. This could be a trained model from the approach described in either algorithm one or two.

2. The action the agent took in the previous time step A_{t-1} is chosen as the initial action candidate a.

3. Let a' be an action from the set of neighbours $\mathcal{C}(a)$, defined above. The action a' is obtained by taking the action candidate a and either swapping units i and j in the unit order, giving $a'_{i,j}$, or changing the spill amount at a station s by 30 cumecs, giving $a'_{s,+30}$ or $a'_{s,-30}$. The new action candidate is selected as the highest value neighbour. Either:

   - $a = \arg\max_{i,j}(\hat{q}_\pi(S_t, a'_{i,j}, \mathbf{w}))$,
   - $a = \arg\max_s(\hat{q}_\pi(S_t, a'_{s,+30}, \mathbf{w}))$ or
   - $a = \arg\max_s(\hat{q}_\pi(S_t, a'_{s,-30}, \mathbf{w}))$

   , whichever gives the highest $\hat{q}$ value.

4. Repeat step 3 until there is no more improvement in $\hat{q}_\pi$ value. The action candidate a at that point is then chosen as the action for that time step, A_t.

An example of one step of this hill climb algorithm is shown in Figure 7.3. In this example, two possible neighbours of a hypothetical candidate action are shown.

This gives a locally optimal action A_t with respect to the value function $\hat{q}_\pi$. This hill-climbing method can be applied whenever the policy, represented by a neural network model, is used to determine an action during evaluation.

Hyperparameter tuning is not used for this approach; instead, the hyperparameter values established for the trained model are assumed to be suitable and applied to this algorithm as well.

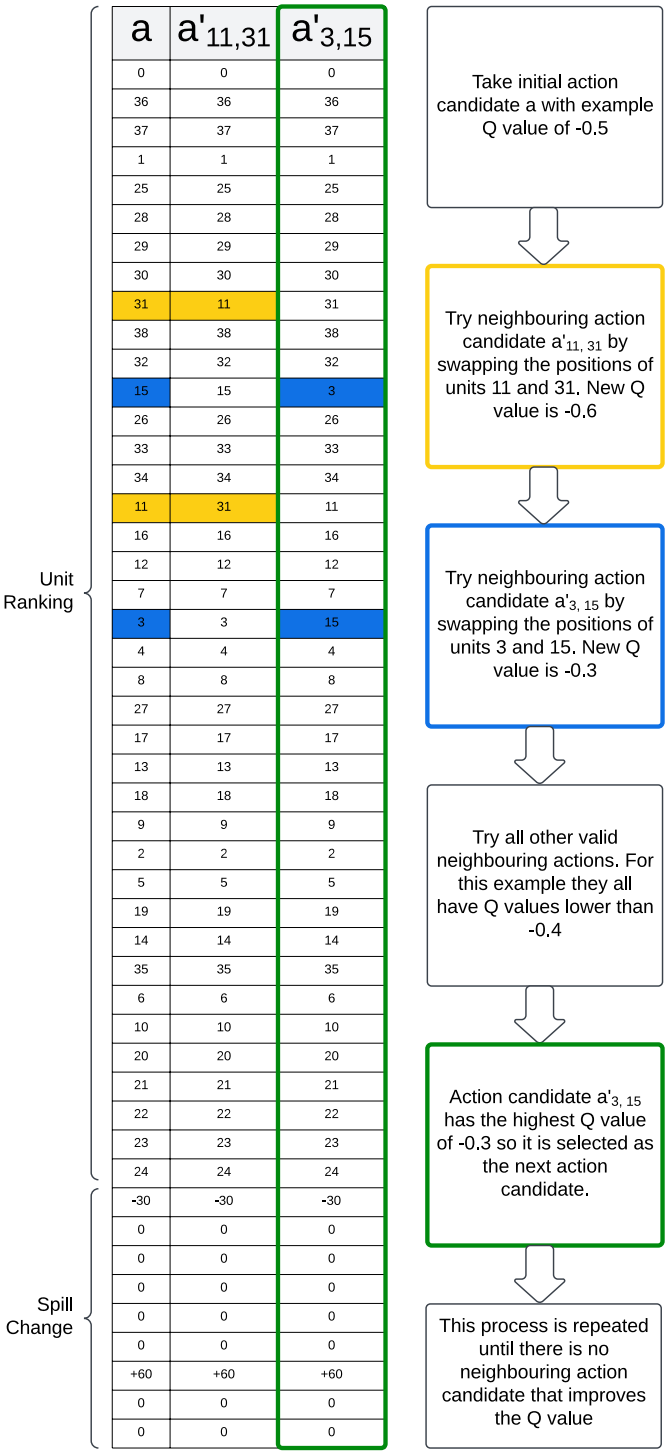


Figure 7.3: Example step of value function hill-climbing algorithm

Chapter 8

Experiments and Results

This chapter discusses the methodology used for the experimental evaluation and results of running Algorithms 1, 2 and 3 from the previous section. To determine which algorithm is most effective, two key metrics that align with the Waikato River hydro control problem are evaluated. The first metric is mean episode duration. This is how long the agent was able to run in the simulator before breaching an operating constraint and reaching a terminal state. The longer the episode duration, the better. The simulation terminates after 10,000 minutes, so the best possible outcome is a mean episode duration of 10,000. The second metric is mean lost generation, measured in MW. This is a measure of efficiency, with a lower average lost MW indicating that the hydro stations are generally running more efficiently. The lost MW are calculated as shown in Equation 5.1. Note that the lost MW are calculated in comparison to a benchmark and can be negative in the case that the hydro stations run more efficiently than the benchmark. The lower the lost MW are, the better.

8.1 Validation and test data

The core data set discussed in Section 6.1 was split into thirds. The first two years were used for training the models, the second two years were used as a validation set in the hyperparameter tuning, and the final two years were used as a test set for a final performance measure of the model. Algorithms 1

and 2 were trained for 3000 episodes for each hyperparameter set evaluation. Every 150 episodes during training, a validation set of 75 episodes was run to monitor performance of the model and both mean episode duration and mean lost generation were recorded. The model was saved if it achieved greater mean episode duration than the previously best validation run.

8.2 Python Environment

The hyperparameter tuning for Algorithm 1 (discussed in Section 7.1.5) was run using a Python notebook on an Amazon web services "r4.2xlarge" cloud instance. This cloud instance was provided by Mercury NZ. Phase 1 of the Algorithm 1 experiments, where the network structure and training parameters were evaluated, took 3 days to run. Phase 2 of the experiments, where the simulation and action space parameters were evaluated, took 4 days to run.

The experiments for Algorithms 2 and 3 were run using Python programs shifted onto multiple on-premises servers to speed up the hyperparameter tuning discussed in Section 7.2.4. These servers were provided by the University of Waikato and had specs as listed in Appendix 9.4. Each of these hyperparameter tuning experiments was run in parallel on distinct servers. The hyperparameter set evaluation for Algorithm 2 took between 11 hours and 122 hours to run in parallel. The evaluation of Algorithm 3 took 58 hours to run.

8.3 Runtime Improvements

Given the long run time of reinforcement learning experiments, it was important to use techniques that allowed faster computation. In Algorithm 2, there are three main time-intensive components in the experiments: the unit order calculation (Section 5.2.2), the headwater level projection (Section 7.2.1), and the state-action value function evaluation.

One technique used in the unit order calculation was vectorisation of calculations. Arora (2024) discusses the use of vectorisation to improve perfor-

mance of Python code. This involves replacing code using "for loops" with code performing the same logic in a vectorised manner. The "NumPy" package in Python was used to vectorise the simulation and unit order model. The computation time of the unit order model was reduced significantly.

Another technique was converting some time-intensive code to "just-in-time" compiled C code. This was done using the "numba" Python package Lam et al. (2015). Lam et al. state: "Numba compiles a subset of the language into efficient machine code that is comparable in performance to a traditional compiled language". This was used to improve the performance of the headwater level projection computation, reducing computation time by a factor of 8 from the NumPy implementation.

After applying all of these techniques, the final runtime breakdown for algorithm 2 was:

- 2% neural network evaluation
- 54% unit order model evaluation
- 9% headwater level projection
- 14% tailwater level calculation
- 7% penstock flow calculation
- 14% other simulation calculations

8.4 Baseline Comparisons

As a baseline, 200 test episodes were run with completely random actions. The mean episode duration with fully random actions was 188 minutes and the mean lost generation was 45.6. This gives a worst case to evaluate training against. Another baseline was computed based on historical actual data. The actual historical average lost generation was 60.6MW including spill and 15.9MW excluding spill. The very high lost generation of 60.6MW suggests

large amounts of spill in some scenarios. Comparing this to the 15.9MW excluding spill shows a large proportion of the lost generation was due to spill. Some of this spill was due to Lake Taupō being too full, which was outside the scope of the simulation developed in this thesis. An example of this might be that due to lots of rain, Lake Taupō is full of water and flow down the river needs to be increased to avoid flooding. This would result in spill at multiple stations down the river, which the model avoids unless this constraint is explicitly modelled. Fixing this discrepancy is discussed in the future work in Section 9.1. To give a fairer baseline from the historical data, the 15% of episodes with the worst lost generation were removed. This cut-off was chosen to remove most of the scenarios where there was multi-station spill as this is usually caused by Lake Taupō being full. This gave a more reasonable lost generation baseline of 23.9MW. No constraint was breached in the historical data, so the mean episode duration was the maximum episode duration of 10,000 minutes (approximately 7 days).

8.5 Algorithm 1 Results

Algorithm 1 (Section 7.1) uses a simple, one-dimensional set of features to determine a value function. As discussed, for hyperparameter tuning, this algorithm was trained using a number of different hyperparameter settings to find the hyperparameter values best suited for the Waikato River hydro control problem. As stated in Section 7.1.5, the hyperparameters that were evaluated in Phase 1 were:

- $N_{layers} \in \{0, 1, 2\}$
- $N_{size} \in \{10, 40, 100\}$
- $LR \in \{1 \times 10^{-4}, 3 \times 10^{-3}, 1 \times 10^{-2}\}$

The remaining hyperparameters were set to:

- $|\mathcal{A}_{unit\ ranking}| = 200$

- $|\mathcal{A}_{spill\ changes}| = 3$
- $EW = 0$

These hyperparameter configurations gave the episode durations and lost generation amounts shown in Table 8.1.

The set of hyper parameter values that gave the longest mean episode duration was $N_{layers} = 2$, $N_{size} = 40$, $LR = 10^{-2}$. This combination had a mean episode duration of 1127 minutes and a mean lost generation of 13.1MW. These parameter values were carried forward to phase 2. In phase 2 the remaining hyperparameter sets were evaluated:

- $|\mathcal{A}_{unit\ ranking}| \in \{50, 200\}$
- $|\mathcal{A}_{spill\ changes}| \in \{3, 23\}$
- $EW \in \{0, 10^{-4}, 10^{-3}\}$

The results of these hyperparameter sets are shown in Table 8.2. No further improvements were obtained using the second phase.

The model with the highest mean episode duration of 1629 (27 hours) and mean lost generation of 13.4MW had the following hyperparameters:

- $N_{layers} = 2$
- $N_{size} = 40$
- $LR = 10^{-2}$
- $|\mathcal{A}_{unit\ ranking}| = 200$
- $|\mathcal{A}_{spill\ changes}| = 3$
- $EW = 0$

These parameters intuitively make sense. For the N_{layers} and N_{size} parameters, this gives the model a large enough network to avoid under-fitting. For

N_{layers}	N_{size}	LR	Duration (mins)	Lost MW
0	N/A	1×10^{-4}	496	14.3
0	N/A	3×10^{-3}	263	34.2
0	N/A	1×10^{-2}	868	7.5
1	10	1×10^{-4}	726	31.9
1	10	3×10^{-3}	562	12.7
1	10	1×10^{-2}	464	10.1
1	40	1×10^{-4}	328	21.6
1	40	3×10^{-3}	653	8.6
1	40	1×10^{-2}	765	10.7
1	100	1×10^{-4}	540	10.4
1	100	3×10^{-3}	309	13.4
1	100	1×10^{-2}	614	18.4
2	10	1×10^{-4}	406	16.6
2	10	3×10^{-3}	614	18.4
2	10	1×10^{-2}	705	15.1
2	40	1×10^{-4}	492	24.6
2	40	3×10^{-3}	872	11.2
2	40	1×10^{-2}	1629	13.4
2	100	1×10^{-4}	758	32.8
2	100	3×10^{-3}	1092	14.2
2	100	1×10^{-2}	1055	21.6

Table 8.1: Results for phase 1 of hyperparameter tuning for Algorithm 1

$\lvert\mathcal{A}_{unit\ ranking}\rvert$	$\lvert\mathcal{A}_{spill\ changes}\rvert$	EW	Duration (mins)	Lost MW
50	3	0	446	12.2
200	3	0	1629	13.4
50	23	0	1038	9.9
200	23	0	244	27.8
50	3	1×10^{-4}	1000	12.4
200	3	1×10^{-4}	558	9.0
50	23	1×10^{-4}	463	18.0
200	23	1×10^{-4}	892	20.4
50	3	1×10^{-3}	591	15.1
200	3	1×10^{-3}	1025	7.5
50	23	1×10^{-3}	329	22.4
200	23	1×10^{-3}	398	13.8

Table 8.2: Results for phase 2 of hyperparameter tuning for Algorithm 1

the learning rate, it seems that higher is better to get faster convergence. Further experimentation would be needed to determine if increasing this further would be beneficial. The $|\mathcal{A}_{unit\ ranking}| = 200$ makes sense because covering more of the action space gives the agent more options to choose from. The best hyperparameter option of $|\mathcal{A}_{spill\ changes}| = 3$ being the best option is surprising because more actions should be better but given that the model only allows spill when stations are generating at maximum, it is possible that spill is not utilised enough for more spill options to be useful. The best hyperparameter option of $EW = 0$ giving the highest mean episode duration makes sense because $EW = 0$ effectively gives the agent the single objective of increasing mean episode duration without needing to consider efficiency.

This model was evaluated on the final two years of unseen "test" data for 200 episodes. This gave a mean episode duration of 1428 and a mean lost generation of 17.2.

LR	EW	Duration (mins)	Lost MW
0.0001	0	5877	15.9
0.003	0	6069	15.0
0.01	0	6504	14.5
0.0001	10^{-4}	5899	17.3
0.003	10^{-4}	6026	16.5
0.01	10^{-4}	6186	16.5
0.0001	10^{-3}	5205	21.2
0.003	10^{-3}	4853	17.9
0.01	10^{-3}	4456	13.5

Table 8.3: Results for hyperparameter tuning for Algorithm 2 with initial weights set (IW)

8.6 Algorithm 2 Results

Algorithm 2 (Section 7.2) used a set of features with a time-series dimension to determine a value function. This algorithm was also tuned to find the hyperparameters best suited for the Waikato River hydro control problem. The hyperparameter sets that were evaluated were:

- $LR \in \{1 \times 10^{-4}, 3 \times 10^{-3}, 1 \times 10^{-2}\}$
- $EW \in \{0, 1 \times 10^{-4}, 1 \times 10^{-3}\}$

The algorithm was also tested with random starting weights and with the starting weight initialisation approach (denoted RW for random weights and IW for initialised weights) discussed in Section 7.2.4. As a comparison, the IW model was also evaluated with no training to see how the model performs with initialised weights alone.

The hyperparameter results for the IW model with the weight initialisation used are shown in Table 8.3.

The hyperparameter results without the weight initialisation used (RW model) are shown in Table 8.4.

LR	EW	Duration (mins)	Lost MW
0.0001	0	1390	12.5
0.003	0	5942	14.1
0.01	0	6536	13.7
0.0001	10^{-4}	1328	19.5
0.003	10^{-4}	5503	13.8
0.01	10^{-4}	6253	13.9
0.0001	10^{-3}	760	26.7
0.003	10^{-3}	3511	14.9
0.01	10^{-3}	3970	13.3

Table 8.4: Results for hyperparameter tuning for Algorithm 2 without initial weights set (RW)

The model with the highest MED (mean episode duration) was the model with hyperparameters of $LR = 0.01$ and $EW = 0$. This was the best setting with and without initial weights set. The RW model performed very slightly better with an MED of 6536 minutes (about 4.5 days) and a mean lost generation of 13.7 MW. The training curves of the models with $LR = 0.01$ and $EW = 0$, for both the RW and the IW model, are shown in Figure 8.1. It is noteworthy that the MED on the validation set is generally longer than that on the training set. This is because in the training set, the epsilon greedy policy is used where there is probability ϵ of a random action being taken. For validation, this random action is not used and the best action according to the $\hat{q}$ function is taken every time, giving a better MED.

The result for the IW model without training was an MED of 6416 and a mean lost generation of 17.7MW when evaluated on the 75 episodes from the validation dataset. Most of the IW hyperparameter training runs actually had worse MED than this, which is interesting. One possible explanation is that the initial weights produce a good policy but a poor value of the objective function $(\hat{q}_\pi - G)^2$. Subsequent training may produce a weaker policy initially

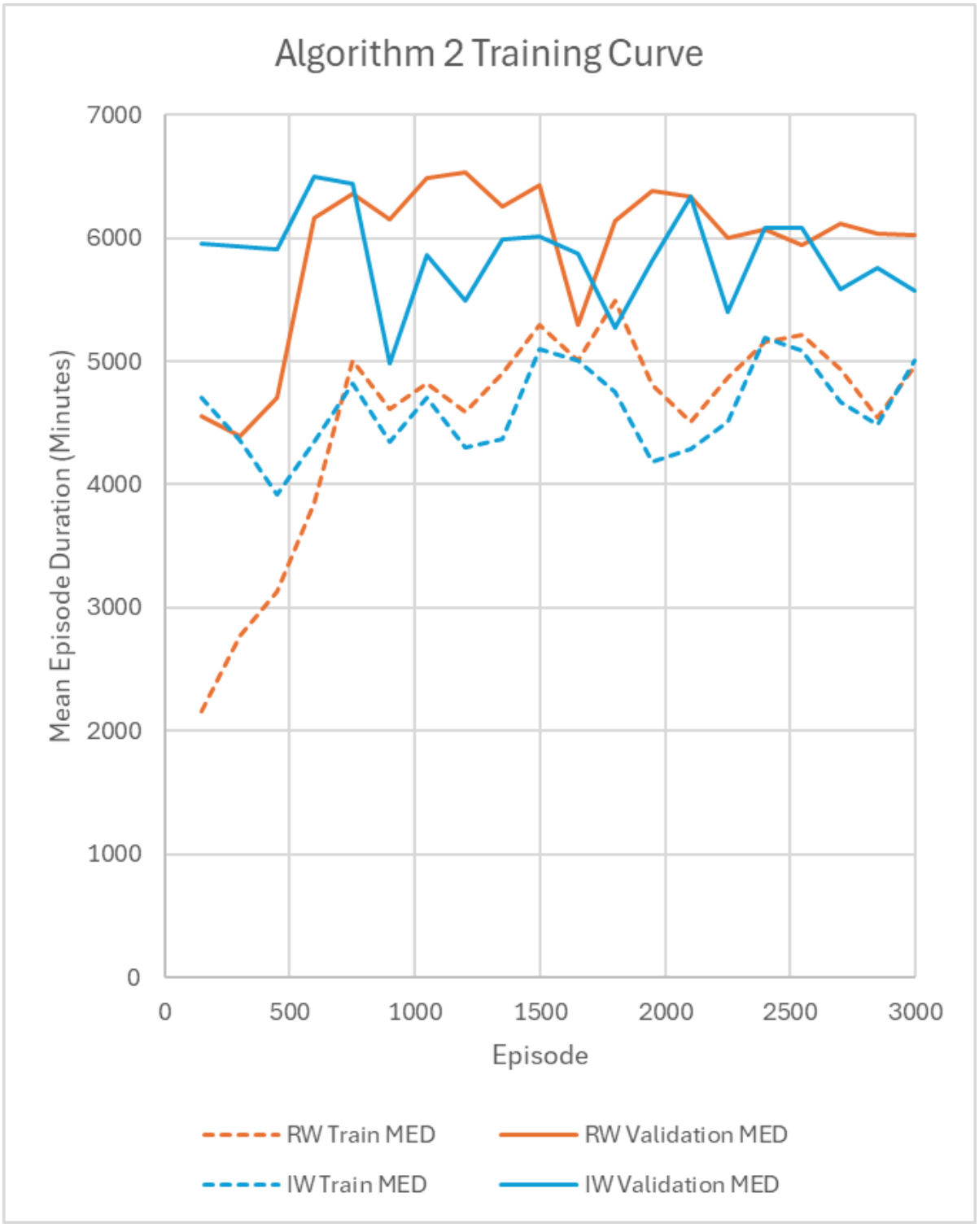


Figure 8.1: Training and Validation curve for both models with hyperparameters of $LR = 0.01$ and $EW = 0$.

to improve the $\hat{q}_\pi$ fit before training is able to improve the policy. It is possible that with longer training, other hyperparameter sets would have improved beyond the MED of 6416 but longer training runs were not feasible due to the constraints of the project.

Interestingly, the lost generation did not correlate strongly with the efficiency weighting in algorithm 1 and 2. One would expect the higher the weighting put on efficiency, the lower the average lost generation, possibly at the expense of a lower mean episode duration. The results for algorithm 2 in Table 8.3 and 8.4 show that often increasing the efficiency weighting makes both metrics worse. A possible explanation for this is that a lot of efficiency optimisation is already taking place in the unit order model (Section 5.2.2). This model makes action candidates that use efficient setpoints where possible. It seems that although the trained model from algorithm 2 may have some impact on efficiency, most of the efficiency optimisation is done by the unit order model.

The final model of algorithm 2, after hyperparameter tuning, was also evaluated on the final two years of unseen "test" data for 200 episodes. This gave a mean episode duration of 4560 and a mean lost generation of 15.3.

8.7 Algorithm 3 Results

Algorithm 3 uses the trained model from Algorithm 2 with the highest MED and applies the hill-climbing approach discussed in Section 7.3. The Algorithm 3 model was evaluated on 200 episodes of test data, just like the final models for Algorithm 1 and Algorithm 2. The mean episode duration was 7451 and the mean lost generation was 15.4MW.

It is worth noting that 66% of the episodes completed with no breaches in constraints. Of the 34% that failed, 87% of the failures were from the Ātiamuri and Waipāpa stations. These are stations with small storage lakes and short travel times.

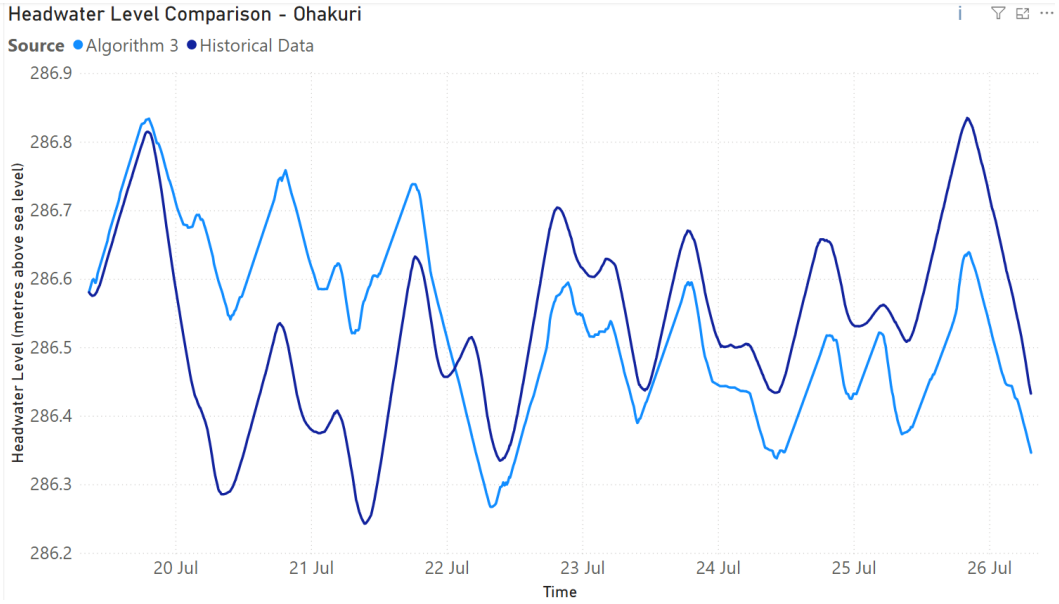


Figure 8.2: This shows the headwater level in Ōhakuri for both the Algorithm 3 results and the historical data. There is some deviation in the first couple of days where Algorithm 3 holds Ōhakuri at a higher headwater level. Otherwise, the headwater level from Algorithm 3 is very similar to the historical data.

8.7.1 Historical Comparison

One episode from the test set was selected for comparison with historical data. This episode reached the maximum duration without any constraint breaches. The episode had an average lost generation of 8.8MW. This was slightly better than the lost generation of 9.7MW in the historical data. Figures 8.2, 8.3, 8.4, 8.5, and 8.6 show graphs comparing the headwater levels from Algorithm 3 with the headwater levels from the historical data for some stations. Overall, the headwater levels from Algorithm 3 behave in a similar manner to the historical data, giving credibility to this approach.

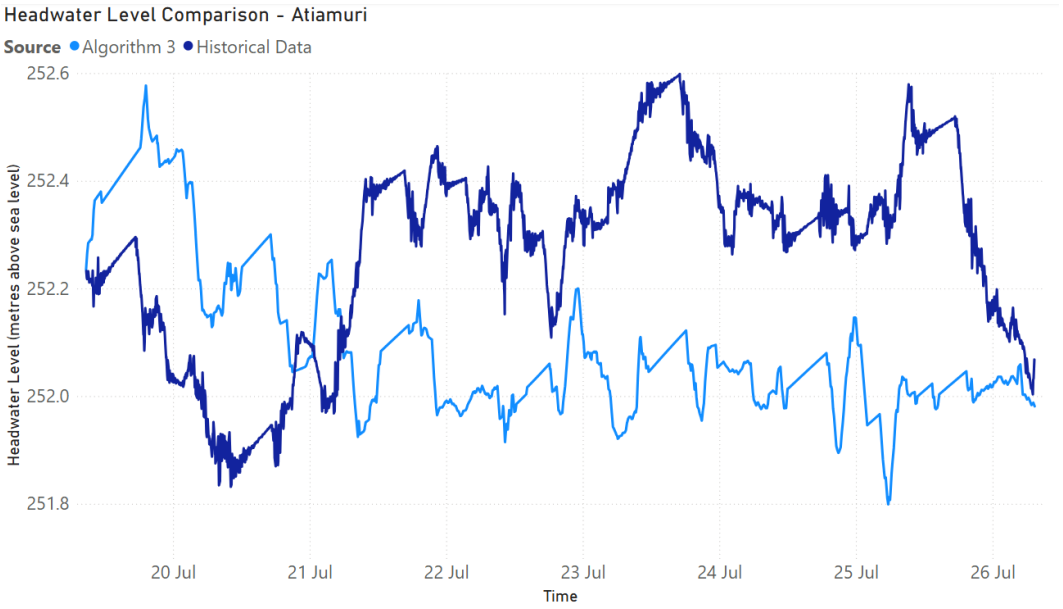


Figure 8.3: This shows the headwater level in Ātiamuri for both the Algorithm 3 results and the historical data. Algorithm 3 seems to hold the headwater level lower at Ātiamuri which may be why there are many constraint breaches there.

Figure 8.4: This shows the headwater level in Whakamaru for both the Algorithm 3 results and the historical data. Algorithm 3 seems to be more conservative with the headwater level. It avoids taking the level lower like in the historical data.

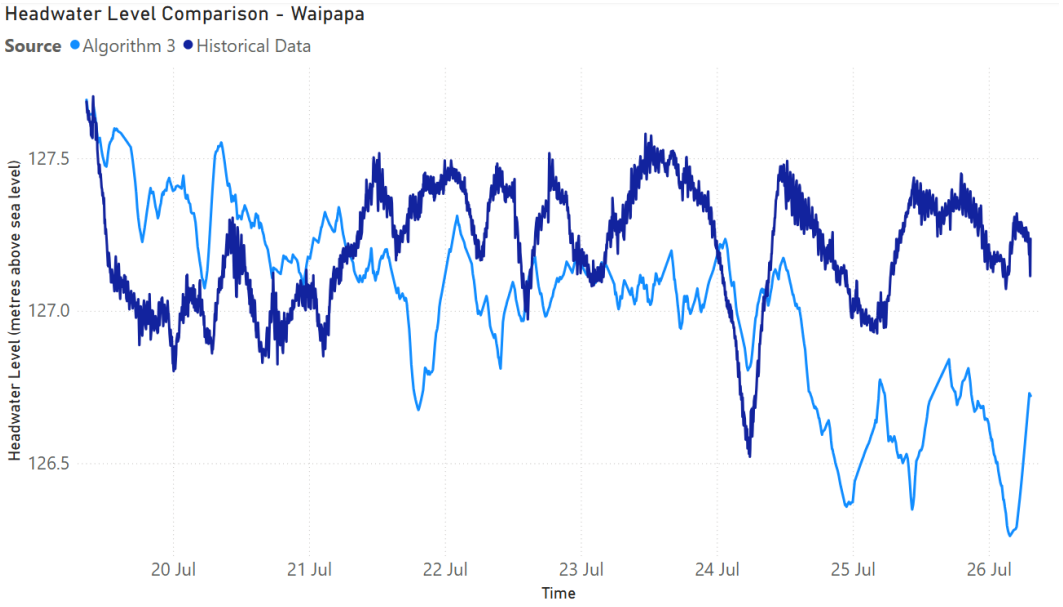


Figure 8.5: This shows the headwater level in Waipāpa for both the Algorithm 3 results and the historical data. The headwater levels are aligned for the first five days but then Algorithm 3 drops the headwater level lower than the historical data. This lower headwater level may be part of the reason for constraint breaches at Waipāpa.

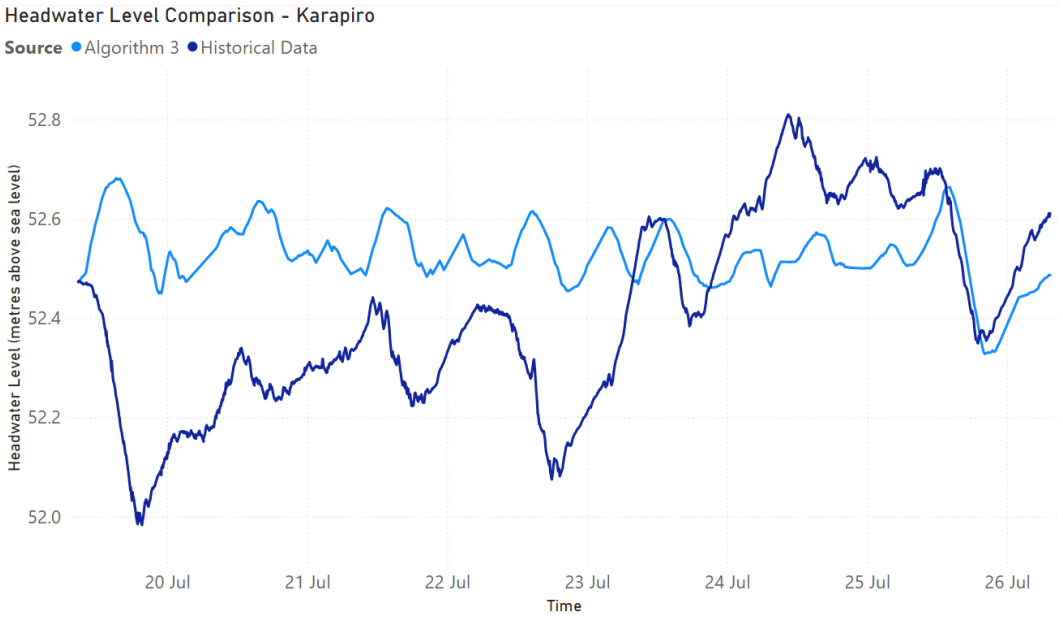


Figure 8.6: This shows the headwater level in Karāpiro for both the Algorithm 3 results and the historical data. Algorithm 3 seems to be more conservative with the headwater level. It avoids taking the level to the upper and lower extremes like in the historical data.

Chapter 9

Conclusion

In this thesis, reinforcement learning was applied to the Waikato River hydro control problem. The aim was to use reinforcement learning to achieve better efficiency outcomes while adhering to operational constraints. A simulation was created that modelled the control problem. Two metrics were measured in the simulation: episode duration and lost generation. Three models were developed to maximise episode duration and minimise lost generation in this simulation.

The final results of evaluating each of the three reinforcement learning approaches on the test set are shown in Table 9.1. Both the random and historical baselines, discussed in Section 8.4, are included for comparison.

Algorithm 1 used Monte Carlo control with a neural network to model the $\hat{q}_\pi$ function. It simplified the large action pool by using the K-medoids

Model	Duration (mins)	Lost MW
Algorithm 1 Model	1428	17.2
Algorithm 2 Model	4560	15.3
Algorithm 3 Model	7451	15.4
Random Action	188	45.6
Historical Performance	10000	15.9 — 23.9 — 60.6

Table 9.1: Comparison of all models with baselines

algorithm to get a smaller number of representative actions. Algorithm 1 applied a simpler set of features without a time dimension. It was trained using the simulation (Section 6) on a variety of hyperparameter combinations in two phases, the first phase for the model structure, the second for simulation related parameters. The best model from the hyperparameter tuning was run on the test set with the results shown in Table 9.1. For mean episode duration and lost MW, it is noticeably better than the random action policy, but still too far from the historical performance to be useful.

Algorithm 2 was designed to improve the $\hat{q}_\pi$ approximation using features with a time component to enable forward planning. These features used a headwater level projection (Section 7.2.1). The network used was very shallow in this approach with not many weights. The hypothesis that starting the model with sensible initial weights would give a better mean episode duration was tested. It turned out that if a good learning rate was chosen, using random weights performed slightly better than using the sensible initial weights. Overall, algorithm 2 had over triple the mean episode duration of algorithm 1, demonstrating the importance of time-based features to the performance of the model. Just like Algorithm 1, Algorithm 2 is still too far from the historical performance to be useful.

Algorithm 3 used the best trained model from algorithm 2. Instead of restricting it to the smaller action set from the K-medoids approach, a hill-climbing method is used to improve the current action. This allows the model to be trained on a smaller action set to allow for faster training, but then be expanded out to a larger action space again to achieve better mean episode duration. The algorithm 3 hill-climbing approach had the best mean episode duration of 7451, showing the importance of expanding the action set while remaining competitive with algorithm 2 in terms of lost MW.

The historical lost generation shown in the table has three different figures: without spill, with spill and outliers filtered, and with spill without outliers filtered, respectively. The baseline with spill and outliers filtered was chosen

as the best representation of the historical lost generation. All three models outperformed this historical baseline of 23.9MW, but only the Algorithm 3 model achieved promising mean duration.

The experimental results obtained from hyperparameter tuning indicate that lost generation of the models did not seem to be affected much by the weighting of the efficiency part of the reward. It was discussed in Section 8.6 that most of the efficiency optimisation happens in the unit order model determining action candidates, rather than in the neural network model.

Algorithm 3 gave high mean episode durations and low lost generation. In the test dataset it got through 18% of the episodes without any constraint breaches and with lower lost generation than that observed in the historical performance. This shows the model can be used to improve outcomes in the Waikato River hydro control problem in some situations. Limitations to how this model can be used are discussed in the next section. After that, future work that could further improve the mean episode duration and applicability to the Waikato River hydro control problem is discussed.

9.1 Limitations and Future Work

Although the algorithm 3 approach improves efficiency, the mean episode duration still falls short with 34% of episodes breaching an operating constraint, which causes an episode to stop before 10000 minutes have expired. The consequences of breaking operating constraints in a real hydropower scheme are significant and any breach is unacceptable. Any use of this model in practice needs to be carefully supervised by a human hydro controller to weigh up the efficiency benefits with the likelihood of breaching a constraint. However, it is worth noting that even if the model were able to achieve 100% of episodes without breaching a constraint in the test data, it would still be necessary to have this human oversight. This is because the historical scenarios are not necessarily representative of future scenarios that could occur. If a scenario

occurred that was not present in the historical data, it is difficult to know if the policy of this model would give a good or bad outcome.

Given that the RL policy will not be able to be followed 100% of the time, it is tempting to try a hybrid human/RL policy. This could be where a human hydro controller follows the RL policy until they see that it is going to breach a constraint, then switching back to manual control. However, this hybrid policy may not actually perform better than the human alone. Further investigation is needed to determine whether an RL policy can be used profitably in a hybrid manner like this.

Another limitation is that the model bases its policy on the current forecast data. If this data is incorrect or uncertain, the policy may not take appropriate actions to manage this uncertainty. Not all of the years from the core dataset had proper forecast data available so the historical actual data was used as a substitute (See Section 6.3). This gives the model an unfair advantage as it has perfect hindsight of the actual generation and inflows. If more data were collected, it would be possible to re-train the reinforcement learning model with proper forecast data that the human hydro controller would have had at the time. This would make the simulation closer to the actual environment, which should make the policy more robust when used in the actual environment.

Another consideration is that any policy generated by reinforcement learning is only as good as the simulation it was trained in. If there are differences between the simulation and reality, the policy could perform well in the simulated environment but behave worse than expected in the real environment. There are several possible differences between the simulation and real life. Three of these are: unit start-stops, Lake Taupō level constraints, and changes to the environment over time.

For unit-start stops, the simulator presented in this thesis assumes that starting or stopping a unit is instant and cost-free. This is not the case in reality. It can take several minutes from when a hydro controller issues a setpoint to when the hydro unit actually reaches that setpoint physically. This

has two main implications, flow accuracy and unit thrash. Flow accuracy is impacted because as a hydro unit gets to the setpoint, it has to pass through all of the other inefficient setpoints on the way to the desired setpoint. This can make the flow output difficult to predict with accuracy. Further investigation is needed to determine whether this discrepancy causes problems with using the reinforcement learning policy. The other implication is unit thrash. Unit thrash is when a policy decides to turn a unit or several units on or off too much. There is a real cost to turning hydro units on and off due to passing through inefficient ranges as discussed earlier, and increasing wear and tear on the unit, increasing maintenance costs. Further investigation is needed to determine if the reinforcement learning policy does excessively thrash the units relative to history. If so, an additional metric may be necessary to capture this extra objective.

For Lake Taupō level constraints, there are situations where Lake Taupō is too full and more water needs to be shifted downstream than can be generated. This situation typically causes multiple stations to spill water. Lake Taupō was left out of scope for this thesis but it seems some sort of constraint is necessary to keep the simulation consistent with reality. A possible change that could be made is to have an increased minimum flow constraint at Aratiatia to represent this extra flow of water. The formulation of spill may also need to be changed to allow spill even at lower generation during these times so that enough water can flow downstream.

There could also be changes to the real environment over time that are not reflected in the simulation. One example of this is hydro station rehabilitation. A station rehabilitation is when the hydro units at a station are at the end of their life and are replaced with new units. This is typically done one unit at a time over several years. These new units often have significantly different efficiency curves than the old units because they benefit from 50+ years of improved technology since the old ones were installed. This causes a discrepancy in the simulation because the efficiency curve for the station will change

multiple times over the course of a station rehabilitation. Some investigation is needed to find ways of handling station rehabilitations effectively in both the training and real-time use of the reinforcement learning model.

Phankamolsil et al. (2025) applied policy gradient methods with success to a real-time cascade hydropower scheme. Switching from the Q learning approach using Monte Carlo control to a policy gradient method may produce better results. Using policy gradient methods would also open up the possibility of using a continuous action space rather than discretising the action space with the K-medoids algorithm. This would allow the model to train on the full action space rather than a restricted set of options. It is unclear whether this would improve the results or not without further investigation.

Another potential improvement is in the headwater level model accuracy. The current headwater level model assumes a fixed lag time for water to travel from one station to the next. A more accurate representation is demonstrated by Chen et al. (2020) with their Gaussian process regression model. This model could be implemented in the simulation to improve the accuracy of the headwater level model.

Bibliography

Abdalla, A. E. (2007). *A reinforcement learning algorithm for operations planning of a hydroelectric power multireservoir system* [Doctoral dissertation, University of British Columbia]. https://doi.org/http://dx.doi.org/10.14288/1.0063269

Arora, I. (2024). Improving performance of data science applications in python. *Indian Journal of Science and Technology*, *17*(24), 2499–2507.

Bernardes, J., Santos, M., Abreu, T., Prado, L., Miranda, D., Julio, R., Viana, P., Fonseca, M., Bortoni, E., & Bastos, G. S. (2022). Hydropower operation optimization using machine learning: A systematic review. *AI*, *3*(1), 78–99. https://doi.org/10.3390/ai3010006

Castro-Freibott, R., García-Sánchez, Á., Espiga-Fernández, F., & González-Santander de la Cruz, G. (2025). Deep reinforcement learning for intraday multireservoir hydropower management. *Mathematics*, *13*(1). https://doi.org/10.3390/math13010151

Chen, X., Zhou, J., Jia, B., Yang, X., & Zhou, C. (2020). Characterizing the hydraulic connection of cascade reservoirs for short-term generation dispatching via gaussian process regression. *IEEE Access*, *8*, 145489–145502. https://doi.org/10.1109/ACCESS.2020.3005941

Giuliani, M., Lamontagne, J. R., Reed, P. M., & Castelletti, A. (2021). A state-of-the-art review of optimal reservoir control for managing conflicting demands in a changing world [e2021WR029927 2021WR029927]. *Water Resources Research*, *57*(12), e2021WR029927. https://doi.org/https://doi.org/10.1029/2021WR029927

Ioffe, S., & Szegedy, C. (2015, July). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd international conference on machine learning* (pp. 448–456, Vol. 37). PMLR. https://proceedings.mlr.press/v37/ioffe15.html

Jain, S., Saini, R., & Kumar, A. (2010). Cfd approach for prediction of efficiency of francis turbine. *The 8th International Conference on Hydraulic Efficiency Measurement*.

Lam, S. K., Pitrou, A., & Seibert, S. (2015). Numba: A llvm-based python jit compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. https://doi.org/10.1145/2833157.2833162

Lee, J.-H., & Labadie, J. W. (2007). Stochastic optimization of multireservoir systems via reinforcement learning. *Water Resources Research*, *43*(11). https://doi.org/https://doi.org/10.1029/2006WR005627

Lewis, B. J., Cimbala, J. M., & Wouden, A. M. (2014). Major historical developments in the design of water wheels and francis hydroturbines. *IOP Conference Series: Earth and Environmental Science*, *22*(1), 012020. https://doi.org/10.1088/1755-1315/22/1/012020

Lim, A., Rodrigues, B., & Xiao, F. (2006). Heuristics for matrix bandwidth reduction. *European Journal of Operational Research*, *174*(1), 69–91. https://doi.org/https://doi.org/10.1016/j.ejor.2005.02.066

Luo, W., Wang, C., Zhang, Y., Zhao, J., Huang, Z., Wang, J., & Zhang, C. (2025). A deep reinforcement learning approach for joint scheduling of cascade reservoir system. *Journal of Hydrology*, *651*, 132515. https://doi.org/https://doi.org/10.1016/j.jhydrol.2024.132515

Mercury NZ Ltd. (2019). *Spades in the ground for new turitea wind farm*. Retrieved February 13, 2025, from https://www.mercury.co.nz/investors/news/20191029-spades-in-the-ground-for-new-turitea-wind-farm

Mercury NZ Ltd. (2024). *Hydro generation*. Retrieved May 28, 2024, from https://www.mercury.co.nz/about-us/renewable-energy/hydro-generation

Park, H., & Jun, C. (2009). A simple and fast algorithm for k-medoids clustering. *Expert Syst. Appl.*, *36*(2), 3336–3341. https://doi.org/10.1016/J.ESWA.2008.01.039

Phankamolsil, Y., Rittima, A., Sawangphol, W., Kraisangka, J., Tabucanon, A. S., Talaluxmana, Y., & Vudhivanich, V. (2025). Deep reinforcement learning for multiple reservoir operation planning in the chao phraya river basin. *Modeling Earth Systems and Environment*, *11*. https://doi.org/10.1007/s40808-024-02265-z

Polák, M. (2021). A brief history of the kaplan turbine invention. *Energies*, *14*(19). https://doi.org/10.3390/en14196211

Riemer-Sørensen, S., & Rosenlund, G. H. (2020). Deep reinforcement learning for long term hydropower production scheduling. *CoRR*, *abs/2012.06312*. https://arxiv.org/abs/2012.06312

Roper, D. (2001). *Taupo waikato resource consents assessment of environmental effects*. Mighty River Power.

Shabani, N. (2009). *Incorporating flood control rule curves of the columbia river hydroelectric system in a multireservoir reinforcement learning optimization model* [Doctoral dissertation, University of British Columbia]. https://doi.org/http://dx.doi.org/10.14288/1.0063141

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning - an introduction*. MIT Press.

Wu, R., Wang, R., Hao, J., Wu, Q., & Wang, P. (2024). Multiobjective multihydropower reservoir operation optimization with transformer-based deep reinforcement learning. *Journal of Hydrology*, *632*, 130904. https://doi.org/https://doi.org/10.1016/j.jhydrol.2024.130904

Xu, W., Meng, F., Guo, W., Li, X., & Fu, G. (2021). Deep reinforcement learning for optimal hydropower reservoir operation. *Journal of Water*

Resources Planning and Management, *147*(8), 04021045. https://doi.org/10.1061/(ASCE)WR.1943-5452.0001409

Ye, S., Wang, C., Wang, Y., Lei, X., Wang, X., & Yang, G. (2023). Real-time model predictive control study of run-of-river hydropower plants with data-driven and physics-based coupled model. *Journal of Hydrology*, *617*, 128942. https://doi.org/https://doi.org/10.1016/j.jhydrol.2022.128942

Appendix

9.2 Simulation Constants

The following constants were used as part of the simulation:

- Travel times in minutes: [60, 720, 20, 60, 60, 20, 150, 90]
- Headwater level minimum: [336.4 , 285.7, 251.0, 225.0, 187.35, 125.0, 110.0, 51.5]
- Headwater level maximum: [337.72, 287.2, 252.9, 226.5, 189.0, 128.1, 111.0, 52.9]
- Minimum flows of 50 cumecs at Aratiatia and 148 cumecs at Karāpiro. All other stations had a zero minimum flow.
- Megawatt per cumec benchmark: [0.315016, 0.289573, 0.206178, 0.343658, 0.545116, 0.142931, 0.481758, 0.270711]

9.3 Initial Network Weights

The decay weights $k_{v,s}$ for Algorithm 2 are initialised as follows:

$$k_{station\ min,s} = [0, 0.2, 0.05, 0.05, 0.1, 0.05, 0.05],$$

$$k_{station\ max,s} = [0, 0.2, 0.05, 0.05, 0.1, 0.05, 0.05],$$

$$k_{river\ min,s} = 0,$$

$$k_{river\ max,s} = 0,$$

$$k_{station\ min\ low\ demand,Karāpiro} = 0.05,$$

$$k_{station\ min\ high\ demand,\bar{O}hakuri} = 0,$$

$$k_{station\ max\ low\ demand,\bar{O}hakuri} = 0,$$

$$k_{station\ max\ high\ demand,Waipāpa} = 0.1,$$

$$k_{lost\ GWh} = 0$$

9.4 Machine Specifications

The machines that algorithm 2 and 3 were run on had the following specifications:

- Intel Core i7-6700k CPU @ 4.00GHz
- 4 CPU cores
- 58GB RAM
- NVidia GeForce GTX 1080 Ti (not used in the experiments in this thesis)

9.5 Code and data links

The Python source code used for training and evaluation of algorithm 2 and 3 is stored in Github here: https://github.com/nickwaddington/hydroRL
The core dataset is required to run the Python code and is available here: https://zenodo.org/records/14921600?token=eyJhbGciOiJIUzUxMiJ9.eyJpZCI

6ImRiMzUwM2YwLThjOTctNGMyMC1hMmI0LTBlODM5MDc3ZWEzYiIs
ImRhdGEiOnt9LCJyYW5kb20iOiJkOTY5OGI5NTg4Y2JlMTEyZGU0ZWE0
MmE3Nzk4NDNhNCJ9.aO5oda50VpBCyeCnpWXp-wQh1dP9Y3Dp1MIOYm5
3rAy80eQmTRFAHJ4DNK7BH7GavsHL3E8b2KATQOLjwfvtUQ