SAND2023-10478

SANDIA REPORT
SAND2023-10478
Printed November 2022

Sandia National Laboratories

Machine Learning Solutions for a Stable Grid Recovery

Stephen J. Verzi
Ross T. Guttromson
Asael H. Sorensen

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

Sandia National Laboratories
U.S. DEPARTMENT OF ENERGY NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods/

ABSTRACT

Grid operating security studies are typically employed to establish operating boundaries, ensuring secure and stable operation for a range of operation under NERC guidelines. However, if these boundaries are severely violated, existing system security margins will be largely unknown, as would be a secure incremental dispatch path to higher security margins while continuing to serve load. As an alternative to the use of complex optimizations over dynamic conditions, this work employs the use of machine learning to identify a sequence of secure state transitions which place the grid in a higher degree of operating security with greater static and dynamic stability margins. Several reinforcement learning solution methods were developed using deep learning neural networks, including Deep Q-learning, Mu-Zero, and the continuous algorithms Proximal Reinforcement Learning, and Advantage Actor Critic Learning. The work is demonstrated on a power grid with three control dimensions but can be scaled in size and dimensionality, which is the subject of ongoing research.

ACKNOWLEDGEMENTS

The authors would like to recognize and thank Christian Birk Jones, Filepe Wilches-Bernal, Michael Livesay, and Joseph Lubars for their technical contributions during the early portions of this work. Additionally, the authors would like to acknowledge and thank Ali Ghassemian of the Department of Energy's Office of Electricity Advanced Grid Modeling program for his generous project support and sponsorship

CONTENTS

LIST OF FIGURES

LIST OF TABLES

This page left blank

ACRONYMS AND TERMS

Acronym/Term	Definition
DNN	Deep Neural Network
DQN	Deep Q Learning
DRL	Deep Reinforcement Learning
FACTS	Flexible Alternating Current Transmission Systems
FACTSMDP	Flexible Alternating Current Transmission Systems Markov Decision Process
IC	Initial Conditions (in optimization)
MDP	Markov Decision Process
ML	Machine Learning
NERC	North American Electric Reliability Corporation
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning

1. INTRODUCTION

1.1. Problem Statement

During near blackout conditions, grid operators may be able to restore the system to a safe condition using a Machine Learning (ML) real-time decision support tool. Doing so would require the ability to transition (or dispatch) the grid into a 'good' state, defined as a unique dispatch where the grid has a high margin of operational security and is serving a near maximum amount of load. Under severe emergency conditions, it may not be possible to determine this operating point using optimization, nor might there be sufficient time to find it by study. Additionally, when operating outside of planned operating criteria, an improved dispatch solution requires following a path from the current state to the final state to ensure avoidance of instabilities or other detrimental operating conditions. Therefore, this paper presents an alternative approach to improve grid security through a series of incremental dispatches using Reinforcement Learning (RL).

1.2. Approach

An RL agent is referred to as the navigable player, but more strictly represents the decision process to change the overall grid dispatch to an improved state of security while serving near-maximum load. The RL solution method offers the potential of a speedy solution, with a high probability of achieving a solution. The incremental dispatch solutions are not intended to be proven optimal, although they have been demonstrated to be feasible and 'good' during off-line testing. Furthermore, the RL agent selects incremental changes in the dispatch which represent a secure transition path from the current state to the final state, all the while improving upon load served when possible. The security of this state transition is otherwise not guaranteed since the operation is outside of planned criteria.

This work formalizes these transitions as a Markov Decision Process (MDP) on which a learning agent, such as an RL agent, or a non-learning agent acts to improve its state. To assess the security of the grid during changes in dispatch, we employed specific security metrics to ensure safe transitions during off-nominal operation. This constitutes a major portion of the reward for our MDP and is a key difference between this and prior work [1]–[3]. The RL agent learns to navigate the MDP state space by increasing the stability margins in as few actions as necessary. Each 'action', however could represent the redispatch of all generators in the system, and a discrete process of accomplishing this typically implies a safe transition from the initial to the final state. However, the actual dispatch process is not discrete, as its execution requires a continuous change of each generator's output. These transitions are accomplished at the same time but are not precisely synchronized, and less so during a severe contingency. Therefore, the greater the distance between to the next dispatch point, the less that is known about the state of the grid during that transition path.

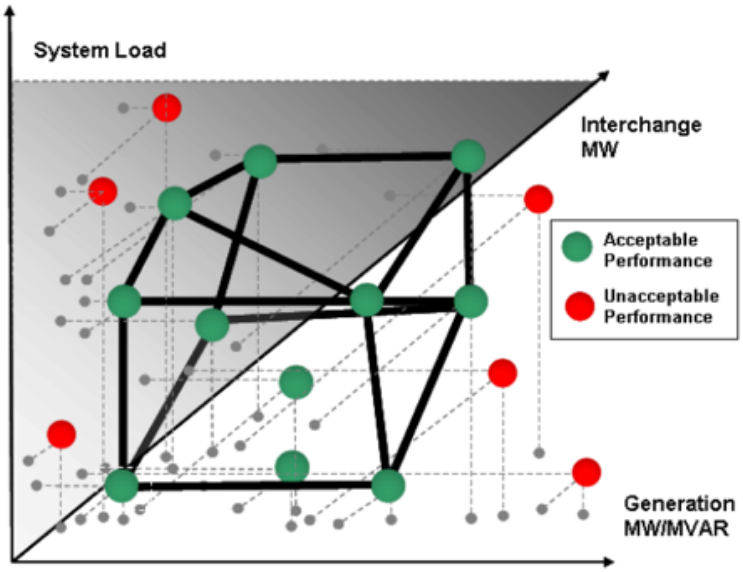

Figure 1-1 Notional scatter plot of planning scenarios indicating planning boundaries. (Image taken from the NERC Reliability Concepts Documents, page 40)

Figure 1-1 is a highly simplified diagram which explains the concept of planning boundaries. In this example, there are three system state variables shown: system load, generation output, and interchange power. Planners determine the operating criteria using a combination of models and operating experience, developing a box (colored in green in Figure 1-1) that is generally safe for operation. Grid operators don't have a need to move the system state outside the box, so that area is not checked by planners to determine if it is safe for operation, or rather, what parts are safe for operation. Thus, the area outside of the box can be thought of as a sort of mine field. Not every state outside the box is unstable, but some are. During a severe blackout, the system topology and load conditions may change substantially, so the previously established boundaries may no longer apply. A new system state with high margins of stability should be determined, and a safe (stable) transition to it should be determined. Doing so could be accomplished using traditional planning, which would require significant time, or the process could be accomplished using optimization methods, which may not be solvable due to the very large problem size. The approach outlined in this work uses a machine learning method.

1.3. Optimization Based Approaches

There are several possible ways to solve this problem using optimization techniques, however we generally consider them in two parts: 1) the determination of a global maxima, which is the system dispatch that provides the maximum load served while preserving adequate margins of system security, and 2) the determination of a state transition path from the initial state to that final state, which does not violate minimum security constraints.

A simplified optimization is shown below to present this concept. The MATLAB® code for this is provided in Appendix B and allows the user to change the initial condition to any point on the graph.

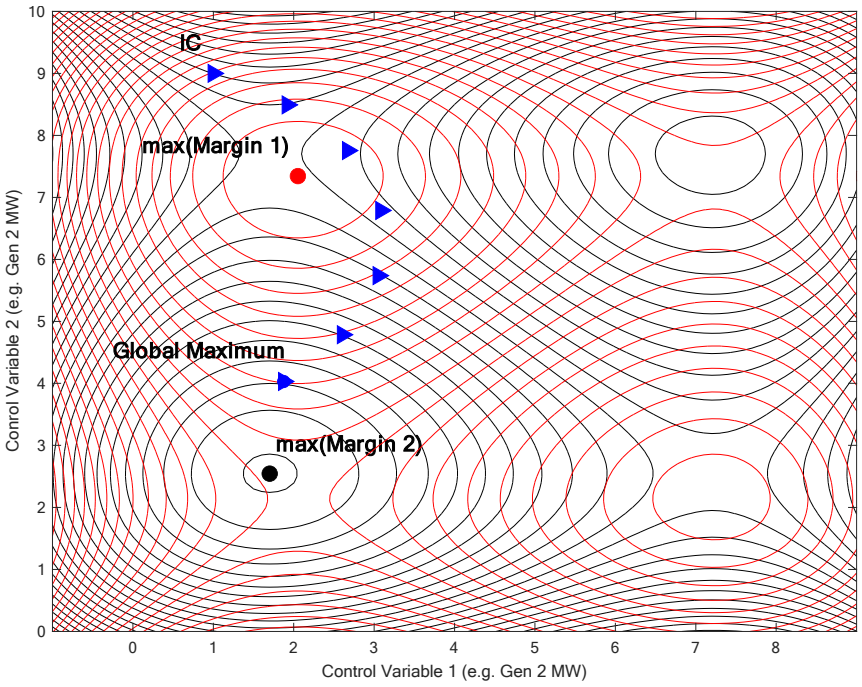

Figure 1-2 Multi-period nonlinear optimization example.

The example n Figure 1-2 shows how closed form optimization techniques can be used to solve the problem presented. The space in the figure is covered with two arbitrary functions that are meant to indicate two different types of grid security margin as a function of two control variables shown in the x and y axes. The addition of these curves was used as the objective function, which was to be navigated, however, the minimum of these two curves (not shown) is a more accurate function that represents the minimum grid security over the two-dimensional control space. The method solves for the global optimal (maximum) and determines a transition path to it which does not violate minimum security constraints, gradient constraints, or point spacing constraints. In this example, the control variables change the dispatch in increments as the system moves from its initial condition to the global optimal. The path is solved as a single solution, not as a sequence of solutions. The algorithm requires a priori knowledge the system security margins, or sufficient information to calculate them. The user may choose any arbitrary initial condition (IC) and obtain an optimal solution from that point.

The ability to carry this solution to a large scale, such as the Western or Eastern Grid models is tenuous. These models are very large, including, for example, 25,000 or more busses, transmission lines, and myriad of loads, transformers, capacitor banks, and Flexible Alternating Current Transmission Systems (FACTS) devices. These are non-linear elements which require complex (both continuous and discrete) algebraic solutions. An optimization model of this size, providing this level of information (as would be provided by system planning engineering studies) has not been developed, and provides the motivation for the work that follows. The authors recognize, however, that a formal mathematical optimization solution would be preferred to a machine learning solution in that optimality bounds may be obtained from the former but not the later. For the results in Figure 1-2, the following set of equations is used to find the optimal path.

$$\min \quad k\nabla S(x,y)_n - (1-k)S(x,y)_n \quad \forall n \tag{1}$$
$$s.t. \quad k \in [0\ 1] \tag{2}$$
$$dx_n^2 + dy_n^2 - d_n^2 = 0 \quad \forall n \tag{3}$$

k *is the objective function weighting factor*
$S(x,y)$ *is security constraint function*
n *are solution point indicies*
d *is the distance betwen points*

1.4. Requirements for Machine Learning Agent Approaches

In order to use agent-based and machine learning models, such as deep reinforcement learning (RL), to help decision-makers during abnormal grid operation, it is important to understand the necessary assumptions and requirements. One of the best ways to represent, describe and understand most of these assumptions and requirements is through the Markov Decision Process (MDP) formalism. A MDP describes the system under study as well as how an active, possibly learning agent, can interact with the system and see how its actions affect the system. Thus, the MDP for grid control will be described next, followed by standard use of incremental prototype generation and ways to quantitatively measure performance of learning.

1.4.1. Markov Decision Process (MDP)

A Markov decision process or MDP is composed of a 4-tuple consisting of a set of states (S), a set of actions (A), a state transition function (p), and a real-valued reward function (r).

$$(S, A, p(S_i,A_i), r(S_j, S_i, A_i)) \tag{4}$$

where $S_i,S_j \in S$, $A_i \in A$, $p(S_i,A_i)$:$S \rightarrow S$, and $r(S_j, S_i, A_i)$:$S,S,A \rightarrow \mathbb{R}$. In a MDP, the set of states is observable by a learning agent, but when the system being research also includes hidden states, the partially observable MDP (POMDP) is more appropriate. For the research described in this report, we exclusively use the MDP formalism, i.e., it is assumed that all state information is visible to each learning or non-learning agent. In the research problem described in this report, the control dimensions define part of the state, and the grid security measures define another part, used in defining the reward function. Actions for the MDP can range from simple to more complicated sets, depending upon domain-specific information. Here, we describe one of the most basic and simple action sets. Given that we have N control dimensions, an agent can either increase or decrease the control output for each dimension or leave them all as is (referred to as the "stay" action), which results in a total of $2N + 1$ actions (i.e., $|A| = 2N + 1$). More sophisticated control action sets will be described when necessary, later in this report. In general, the transition function is embodied either by the physical system, which is being acted upon, or a model of the system, which supports actions as an interface to manipulate the state of the modeled system. The transition function takes a control action as input, along with the current state, and produces the next system state (S_j). In the research described in this report we use a model of the power system, and it is this model that defines the transition function. Note that the exact formulation of the transition function is not available to agents when they choose actions, most especially during learning, as it is the goal of the learning agents to approximate the transition function with respect to their objective (which is to maximize expected reward). The last member of the MDP 4-tuple is a reward function. The reward

function provides direct information from the system (or model of the system) to the agents with respect to the last action choice (A_i) made which resulted in a transition from state S_i to S_j. This information allows a learning agent to improve its performance over time.

1.4.2. Agent Environment

Another important and necessary condition for using RL is defining and implementing an agent environment. This environment provides an interface to the system or model of the system under study, facilitating agent action choice and ultimately agent learning, too. This environment can contain a model of a physical system inside of its interface, if this model runs sufficiently fast. An environment can also be built using a reduced-order model or even be built using data sampled from a system or model itself.

1.4.3. Incremental Prototyping

The best use of machine learning (ML), and especially RL, to tackle hard problems involves using incremental (rapid) prototyping early on. The assumption here is that if we can verify and validate learning agent performance on a simple MDP with reduced (or minimal) dimension, this facilitates building confidence when using these methods in more complex environments, especially where it can be difficult, or even impossible, to validate results in general. Therefore, in this research we start with a simple minimal dimension control model and prototype solution.

1.4.4. Quantifying ML Performance

This research includes quantification of agent performance in terms of both grid security as well as maintaining load served. These measures are important for understanding the advantages as well as disadvantages for using machine learning in controlling grid dispatch during abnormal operating conditions. Measures used in this research are defined in the next chapter (see Section 2.4), and results are presented for both learning and non-learning agents for comparison.

2. FORMALIZING THE GAME

The game has a formal analogy between its play and recovering the grid from a severe unsecure operating condition (see Table 2-1 for details on this analogy). The ML agent itself represents the current state of the grid, and its transition represents state transitions of the grid, which are dispatch changes. As the ML agent seeks an increased reward, it identifies a safe state transition between its current state and each subsequent state as it navigates toward the optimal end state. It's important to recognize that dispatch events are not discrete, and unless the change in outputs of all generators are precisely synchronized, that the state trajectory between the initial and final condition will not be well known. In cases where grid operation is occurring outside planned operating regions, this transition could result in instabilities. Therefore, the transition itself, in addition to the final state, must be managed. After a sequence of several state transitions, the solution will emerge as the final state and its state transitions.

Note that the application of this tool in practice requires a higher degree of control dimensions. In higher dimensional games (those with many independent control variables), state changes could represent other system modifications, such as capacitor bank switching, line switching or even FACTS device operation.

Table 2-1 Comparison of game to transmission stability during state changes

Game Board Attribute	Power Grid Representation
ML Agent: A discretely moveable autonomous agent that seeks to maximize its expected reward as it moves on the game board. The ML agent generates moves according to a policy that has been developed using reinforcement learning.	The location of the ML agent on the game board[1] represents the state of the entire power system. Changes in the ML Agent's location on the game board represent a discrete change in power grid system state and its subsequent security. The optimal position of the ML agent implies an optimal dispatch for a specific grid; that is a system which serves the most load while minimizing operational risk (maximizing security margins).
Game Board Topology: Prior to exercising the ML agent for its intended use, but during the ML agent training process, the values of system security are found using power grid models. These are converted to rewards, corresponding to any possible state represented on the game board. Therefore, each location (state) on the game board has a reward attribute which is used to train the ML agent and are used to play the game after the ML agent is trained. All states observed together represent a reward 'topology'. Thus, the ML agent discretely	The board topology is composed of grid security margin values. Each state has a unique set of security margins representing voltage, damping ratio, transient stability margin, line flow margin, etc. Each of these stability margins are normalized, and the minimum is found for each state, thus the topology is neither guaranteed to be continuous or smooth. This results in a topology that represents system security combined with system control states, which are converted to reward values, for use by the ML agent.

[1] For dimensions greater than three, the game board cannot be visualized so it becomes an abstract state representation.

Game Board Attribute	Power Grid Representation
traverses a topology seeking higher expected reward.	
The ML agent's position on the game board represents a unique state in the grid control dimension space, which is used along with the action used to achieve the current state in computing the reward value. The state of the ML agent is a proxy for the state of the grid dispatch.	The ML agent's position represents the current and unique system state (system dispatch for the current representation) and associated system security.
The ML agent's transition path during the game represents the cumulative reward received, received by the agent, for all actions taken and associated states visited, and the final reward is associated with the last action and final state.	The ML agent's path represents incremental (but discrete) changes in dispatch, each with its associated security margins. Thus, the transition identifies the security of the grid during its transition (series of redispatches) as well as its final state in terms of load served.
The initial state represents the initial state of the grid dispatch, from which the ML agent much chose actions to improve either security, load served or both.	The initial state of the ML agent represents the current, undesirable but marginally stable grid state from which a transition to a more stable state is desired.
The final state represents the final state of game, with the goal of having a high reward value.	The final state represents the last of a series of dispatches aimed toward maximizing the amount of load served while maintaining sufficient security margins.

2.1. Objectives, Penalties, Rewards, Feasibility and Failure

The objective for this research is to maximize load served on the power grid subject to maintaining and/or improving security of the system. Control states are defined by the real power dispatch of grid generators, which can each adjusted up or down, or all can be held at their current dispatch values. The health state of the grid model is defined by the stability/security margins. In the results shown later in this report, four of these health state dimensions (H_k) are used: 1) bus voltage (H_1), 2) line flow (H_2), 3) transient stability (H_3) and 4) small signal stability (H_4). To use these health states in the RL reward objective, they are first normalized such that a value of 1 represents perfect health, 0 means complete failure, and between 0 and 1 indicates non-failure but health can be improved. This logic is represented using a feasibility function (f), see Equation (5) below, where the maximal load will always be found where f is 1.

$$f(H_k) = \begin{cases} 1 & \text{perfect health or feasible} \\ 0 < \& < 1 & \text{The term feasibility was selected as an analogy to an optimization problem. In all cases, the opti} \\ 0 & \end{cases} \quad (5)$$

The overall health of the grid state (S_i, from Section 1.4) is defined as the minimum across the feasibility of each individual health state, see Equation (6) below.

$$f(S_i) = \min_k f(H_k) \quad (6)$$

Note that the feasibility constraint, $f(S_i)$, will always be between 0 and 1, inclusive, where 0 is failure, 1 is feasible; and otherwise it is in penalty (meaning the health/feasibility can be improved before catastrophic failure).

Load served, $l(S_i)$, is defined by the power grid flow model for each dispatch state (S_i). Since load served can be higher in less secure/stable states, those dispatches with perfect health/feasibility (i.e., $f(S_i) = 1$) are rewarded higher than non-feasible ones (i.e., $f(S_i) < 1$). Since staying away from catastrophic failure is paramount to this research, non-failure states (i.e., $f(S_i) > 0$) are rewarded more than failed dispatches (i.e., $f(S_i) = 0$). Thus, the reward objective used by both learning and non-learning agents in this research is defined as follows.

$$r(S_j, S_i, A_i) = \frac{r^*(S_j, S_i, A_i)}{\max_{i,j} r^*(S_j, S_i, A_i)} \quad (7)$$

$$r^*(S_j, S_i, A_i) = \begin{cases} \dfrac{f(S_j) l(S_j)}{\max_j f(S_j) l(S_j)} + 2 & \text{if } f(S_j) = 1 \\ \dfrac{f(S_j) l(S_j)}{\max_j f(S_j) l(S_j)} + 1 & \text{if } 1 > f(S_j) > 0 \\ \dfrac{f(S_j) l(S_j)}{\max_j f(S_j) l(S_j)} & \text{otherwise} \end{cases} \quad (8)$$

Note that the reward objective is also normalized to values between 0 and 1, inclusive.

2.2. The Game Board

Recall that states were defined in Section 1.4 as a combination of control and health (security) dimensions, and actions were defined as choices that can be made to result in a change in state. Each agent, whether learning or non-learning can choose an action from its current state. Each action choice will result in a new state as well as a reward value (according to the MDP defined in Section 1.4.1). Note that an action can result in the system state remaining the same, referred to as "stay". Also, the "stay" action was specifically designed to allow an agent direct control to keep the system

in the same state, even though this might not be the most realistic operationally for the physical grid, but it is consistent with the goal of training the ML agent to learn shorter sequences and remains an active are for future research.

Feasibility was defined in Section 2.1. Each agent knows the current state as well as the reward received for the last action taken. A learning agent will use this information to learn a policy, for choosing an action, that maximizes expected (long-term) reward.

2.3. Rules for Agent State Transition

Actions were briefly defined in Section 1.4.1. An agent can choose any one action at any point in time. An action can only modify one control dimension at a time (i.e., the output of a single generator). Future research will include multi-dimension actions. Thus, with N control dimensions (see Section 1.4.1), there are $2N + 1$ possible actions. These correspond to increasing or decreasing the generator output from any one of the control dimensions ($2N$) or keeping everything the same (i.e., the "stay" action). In the research described in this report, increasing or decreasing generator output happens along unit values. This means that an action can only transition the current system state to a neighboring state, which can be accomplished by discretizing the control dimensions (and adding a control dimension discretization parameter D), which was done for the results presented later.

An entire trajectory of control actions is called an episode, where each element in the episode is an action choice made based upon the system state that was current at the time the action was chosen. The maximum number of episodes is a parameter, M. For example, in the results section we let $M = 50$ due to the control dimension discretization parameter that was used (i.e., since $D = 25$). Each agent acts according to its policy, but the objective is to maximize reward without causing a catastrophic system failure. Note that if an agent is already in the optimal state, the best action choice would be to "stay" in the current state.

2.4. Measuring ML Agent Performance

To understand the potential performance benefits from reinforcement learning (RL), several different measures were developed and computed. These include the percentage of penalized optimal final load served, the percentage of episodes that result in a final state in the feasible region without encountering failure and the average and maximum reward achieved by each agent. Values from these metrics are computed for each starting state used during testing. Two non-learning agents were designed and implement for comparison to the RL agent.

2.4.1. ML Agent Performance Metrics

The first metric is called the percentage of optimal penalized final load served. This metric is computed by recording the final state load served for each agent trajectory (S_j^*), and then these values are penalized for when the final state is not in the feasible region.

$$\boldsymbol{l_{pf}\left(S_j^*\right)} = \begin{cases} \boldsymbol{l\left(S_j^*\right)} & \text{if } \boldsymbol{f\left(S_j^*\right) = 1} \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{9}$$

Every state from the transmission grid model under study has an associated load served ($l\left(S_j^*\right)$), determined using output from the power flow model. The maximum of these is used to normalize

the penalized final load served between 0 and 1 for averaging (across K test starting states) to get the percentage.

$$\%\ \text{final load} = \mathbf{100} * \frac{\underset{\mathbf{1} < j < K}{\text{avg}}\ \boldsymbol{l_{pf}}\left(\boldsymbol{S_j^*}\right)}{\underset{\mathbf{1} < j < K}{\max}\ \boldsymbol{l_{pf}}\left(\boldsymbol{S_j^*}\right)} \tag{10}$$

This metric compares performance of agent trajectories chosen by looking at how close the final load served is to the optimal possible final load served, averaged across all test starting states. Note that ideally each agent will choose to navigate as close as possible to the optimal and stop (using the "stay" action) when at the optimal.

The next metric counts how many times the final state of each agent trajectory lies within the feasible region. This metric is computed using all K test starting states as follows.

$$\boldsymbol{f_p}\left(\boldsymbol{S_j^*}\right) = \begin{cases} \mathbf{1} & \text{if } \boldsymbol{f}\left(\boldsymbol{S_j^*}\right) = \mathbf{1} \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{11}$$

$$\%\ \text{feasible} = \mathbf{100} * \frac{\mathbf{1}}{\boldsymbol{K}} \sum_{\boldsymbol{j}=\mathbf{1}}^{\boldsymbol{K}} \boldsymbol{f_p}\left(\boldsymbol{S_j^*}\right) \tag{12}$$

This measure computes how secure and stable the agent episode is at the end of its control navigation trajectory.

The third metric looks at both the average and maximum final reward values. For each final state achieved by a navigating agent across all test starting states, these metrics are computed as follows.

$$\text{average final reward} = \frac{\mathbf{1}}{\boldsymbol{K}} \sum_{\boldsymbol{j}=\mathbf{1}}^{\boldsymbol{K}} \boldsymbol{r}\left(\boldsymbol{S_j^*}, \boldsymbol{S_i^*}, \boldsymbol{A_i^*}\right) \tag{13}$$

$$\text{maximum final reward} = \underset{\mathbf{1} < k < K}{\max}\ \boldsymbol{r}\left(\boldsymbol{S_j^*}, \boldsymbol{S_i^*}, \boldsymbol{A_i^*}\right) \tag{14}$$

The maximum possible achievable reward only occurs when the starting state is adjacent to the optimal state, and the agent immediately navigates to the optimal state (i.e., the agent choses the action which results in the system transitioning to the optimal state) followed by "stay" action choices until the episode terminates. This episode would result in a final reward value of M, since the reward objective function is normalized to range between 0 and 1, inclusive.

As mentioned in Section 1.4, the control dimensions used in this research are generator output values, which can be increased, decreased or kept the same using the appropriate action choice. A two-dimensional slice of the reward objective defined in Section 2.1 is shown in Figure 2-1.

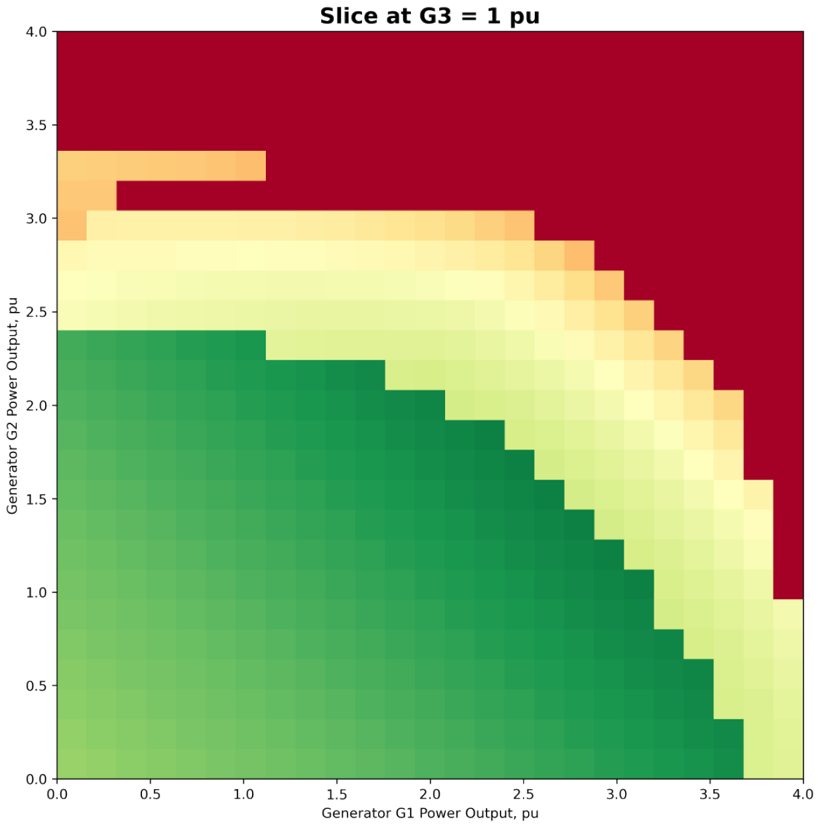

Figure 2-1. Two-dimensional slice of reward objective for 3 generator transmission system where the 3rd generator is held constant at 1pu.

This reward objective is computed using power flow model outputs (for load) as well as security/stability (system health) values computed after the power flow model solution is obtained. Note that when the power flow model did not resolve to a solution, the health state metrics were set to 0 (the failure condition), which is a conservative decision. The graphic in Figure 2-2 indicates each power-flow solution with a dot or an x, which respectively represent power flow convergence and non-convergence, for the same two-dimension slice shown previously (see Figure 2-1). The system load served output of the power flow model is overlaid with the security/stability measures in Figure 2-2.

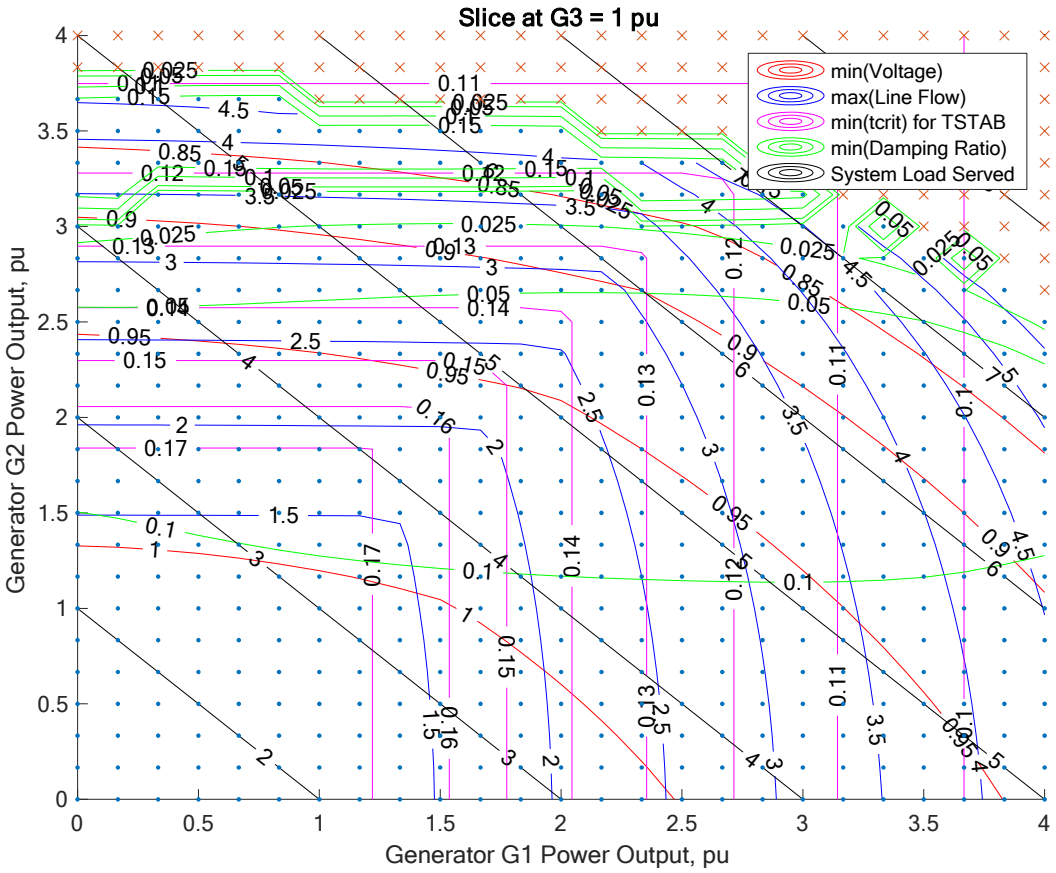

Figure 2-2. Two-dimensional slice of 3D state space, showing load served overlaid with bus voltage, line flow, transient stability and small signal stability for 3 generator transmission system where the 3rd generator is held constant at 1pu.

It is easier to visualize the three-dimensional feasibility space by first looking at bus voltage, line flow, transient stability and small signal stability separately. These heath state metrics are show in Figure 2-3, where green represents feasible ($f(S_j) = 1$), red indicates failure ($f(S_j) = 0$) and yellow is penalty ($0 < f(S_j) < 1$).

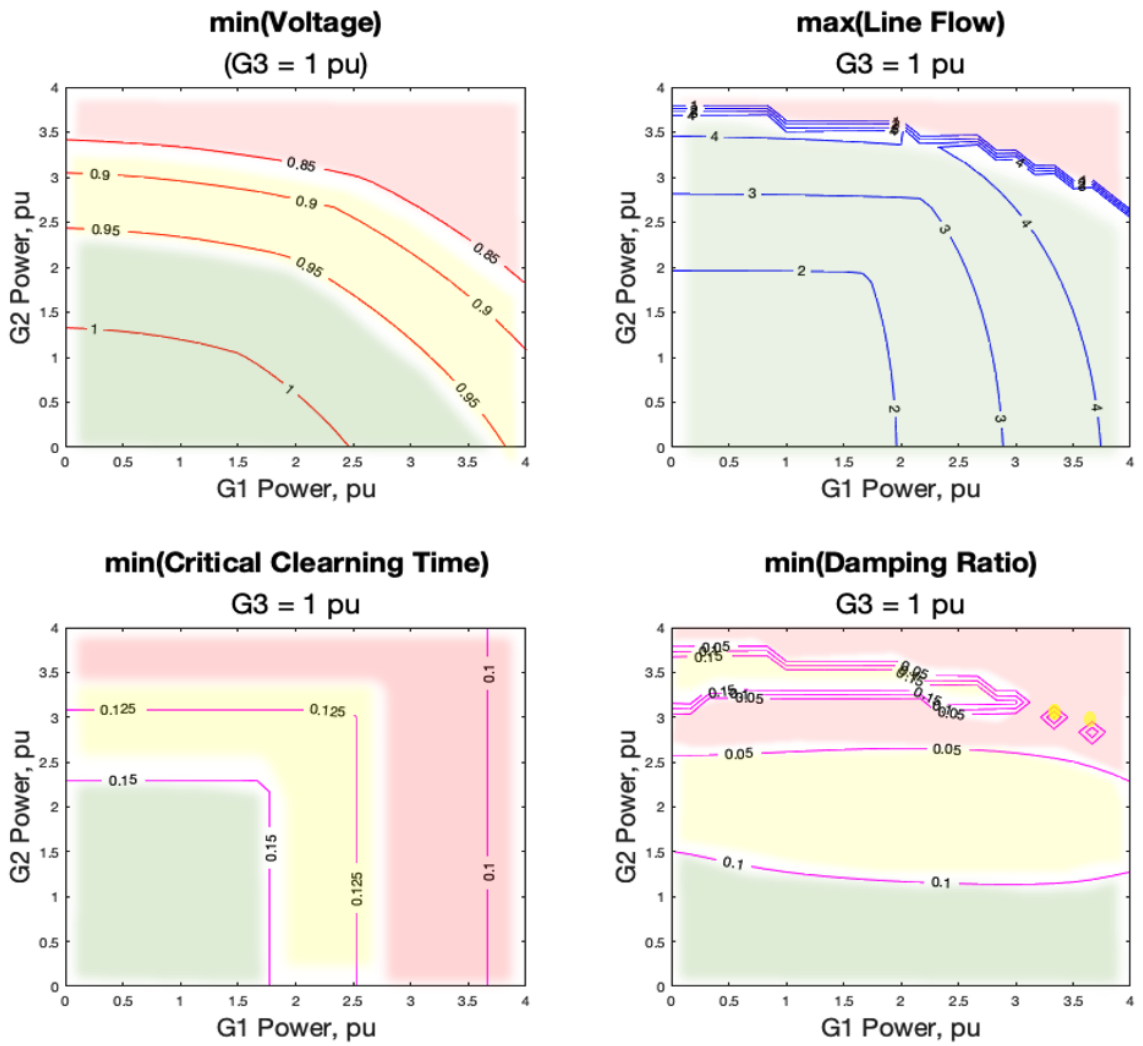

Figure 2-3. Two-dimensional slice of a 3D state space, showing system security metrics of bus voltage, line flow, transient stability, and small signal stability for 3 generator transmission system. The third generator is held constant at 1pu, and each sub-plot is color-coded where green is feasible, red is failure and yellow is penalty.

A three-dimensional visualization of the feasibility space is shown in Figure 2-4.

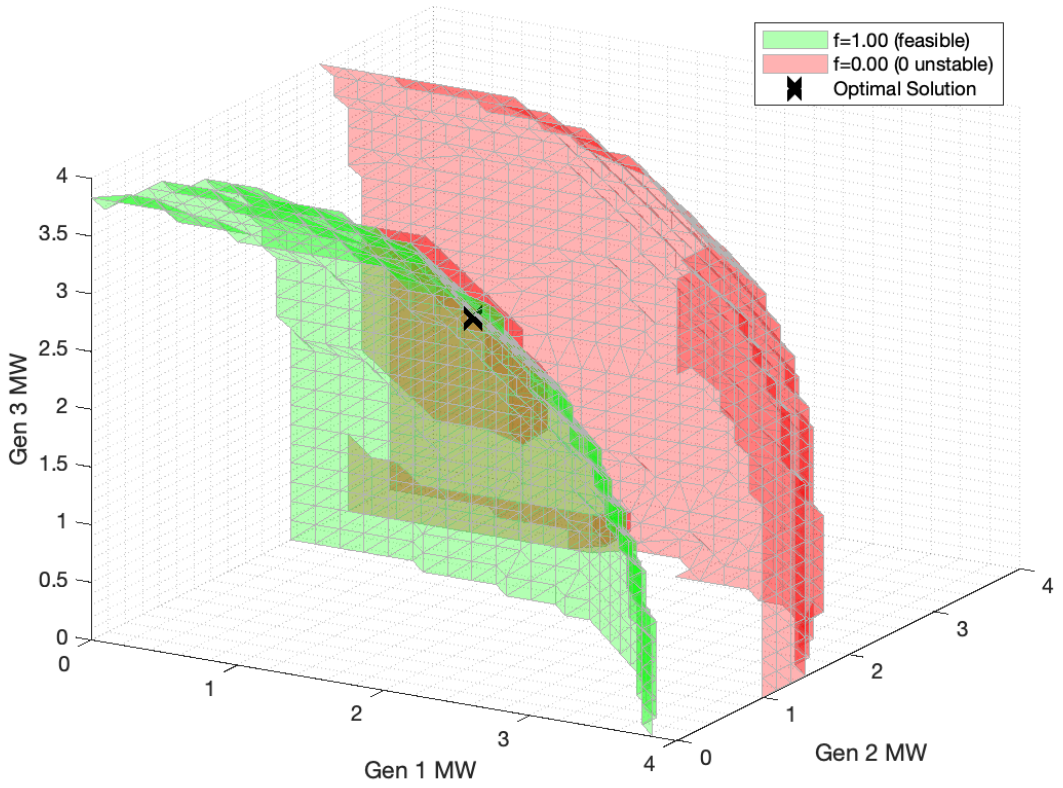

Figure 2-4. Three-dimensional image of the state space for three generator transmission system; color-coded iso-surfaces mark failure and feasibility boundaries. Red and higher indicate the infeasible/failure region and green and lower indicate feasible region. The region between green and red indicates the penalty region.

In Figure 2-4, the black X marks the optimal operating point in this three-generator transmission system and is determined by a combination of maximizing load served while simultaneously maximizing the margins to the security constraints. The point is the largest amount of load served while in the feasible region, as defined previously in Equation (6) and also in Table 3-1 below.

Metrics used to compare performance of the ML agent versus non-learning agents are defined in Equations (9-14), defined previously in Section 2.4.1, and listed again for convenience below.

- % of penalized optimal load served,
- % of scenarios that result in final state in the feasible region, and
- maximum reward achieved.

In future research, the plan is to use transfer learning to test generalizability across learning in one grid topology while testing across another.

3. POWER GRID PROBLEM FORMULATION

The purpose of the ML agent is to safely select transitions to power system states while maximizing the load served and preserving the minimum required security margins. The state changes of the ML agent are a proxy for the safe, incremental redispatch of the grid. A grid model is therefore used to provide two essential features, 1) the ability to change the grid state via changing the dispatch of its generators (control variables) and 2) the ability to quantitatively determine the security margins for each state. Within this model, these include attributes of a grid's stability and security are transient stability margin, small signal stability margin, line flow margin, and bus voltage margin.

3.1. Dimensionality and Reduction

To train and deploy the RL agent, the dynamic power grid model was developed as shown in Figure 3-1. The system is an asymmetric, three-area, three generator system which exhibits sufficient complexity to demonstrate the need for the RL agent to navigate states while attempting to improve its security margins and increasing the amount of load served. Three control dimensions define the state space; these are the dispatch outputs of generators 1, 2 and 3. Total system load was automatically scaled to ensure it was equal to total system generation plus losses for each dispatch state. Additionally, for each dispatch state, values for operational security metrics were determined using Power System Toolbox (PST) [4], [5] analysis running on MATLAB® and mapped to each discrete state. The state space covers all combinations of outputs for Generators 1, 2, and 3. For each state, security metrics were calculated using PST. The metrics include minimum and maximum bus voltages, minimum critical clearing time, maximum line flow, minimum damping ratio and system load served. The objective of the controlling agent (either learning or non-learning) is to maximize the system load served while maintaining a sufficient margin of operational security. The RL agent's incremental changes in system dispatch holds an analogy to a grid multi-period non-linear optimization which seeks a global maximization of system load using a series of incremental dispatches while maintaining operation within a set of constraints. This problem was not solved; however, it is highly relevant for large state transitions occurring outside of planned operating criteria. Under these conditions, there is no guarantee that a large state transition will occur through stable states only. Therefore, a curved or nonlinear trajectory is often required from the initial state to the final state, to ensure stability throughout the transition.

3.2. Grid Model and Topology

The grid topology utilized for this work is shown in Figure 3-1. The operational security metrics (or constraints) that the RL agent must navigate are described below, while the RL agent attempts to increase system load. A three-dimensional discrete state space was developed of size 25^3 (i.e., each generator's dispatch was discretized into 25 equally spaced values). The RL Agent's objective is to incrementally adjust the system dispatch in a manner that increases system load while simultaneously improving system security.

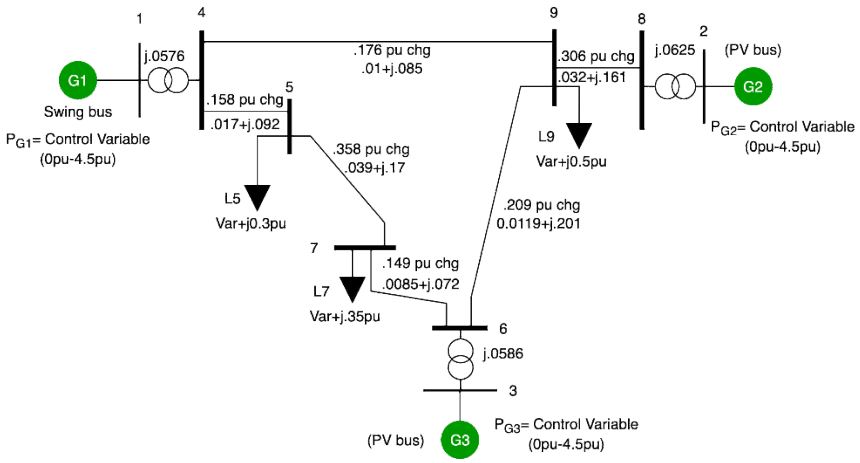

Figure 3-1 System topology for RL Agent navigation and training

System security metrics were calculated using a power flow and dynamic grid model, as they would during typical supervised machine learning (ML) training. The metrics were made available to the RL agent for limited feedback during its navigation process. Figure 2-2 shows a 2-D slice of this 3-D state space, holding Generator 3 power output constant at 1.0 pu. A limited set of operational security metrics were calculated as follows [6], [7]. Although dynamic security was assessed, time domain simulation was not necessary for reasons explained in Section 3.2.1.

For the Grid shown in Figure 3-1, a two-dimensional view of these metrics is presented in Figure 2-2 to provide an intuitive visualization. Figure 2-2 is decomposed into four panels, shown in Figure 2-3. Within each panel of Figure 2-3, the green region represents adequate system security, the yellow region represents stable but poor grid security, and the red region represents unacceptable system security (instability). These boundaries for these regions are presented in Table 3-1.

3.2.1. Dynamic Stability in a non- time domain simulation

The development of security margin values using a time-based simulation would require significant computational resources and time, for even small grids. Although time-based simulations provide a more accurate answer, they typically also provide more information than is necessary. For an ML agent approach, which can require significant reinforcement-based exploration (during learning), training was based on rewards, which in turn were based on the previously calculated grid security margins (see Section 2.1). Thus, to develop an extensive set of security margin data, time-domain simulations were avoided. This approach was applied to two different security metrics: damping ratio and transient stability.

For the calculation of the damping ratio security margin, the system dynamic models were linearized around the operating point of each discrete state. Then a linear dynamic state transition matrix was developed, and eigenvalues were found. These were converted into a damping factor. At each state, this process was repeated, and no time-domain simulation was conducted.

Transient stability security margins were also calculated without using time-domain simulation. At each generator, the system was reduced to its Thevenin equivalent, and an Equal-Area criterion analysis was performed. The transient stability margin metric was selected as the critical clearing

time, which is the minimum time required to clear a 3-phase fault before the generator pulls out of synchronization with the grid.

3.3. Security Metrics

Security metrics for the power grid were calculated at each discrete state. These metrics included bus voltage, line flow, transient stability, small signal stability, voltage stability, and system reserve margin. It was determined that for this simple system, the voltage metrics and voltage stability metrics, when normalized, were essentially duplicative, therefore the voltage stability metrics were omitted from the calculation of feasibility and reward. Likewise, the system reserve margin value did not provide unique information for the navigation of the ML agent, and thus it was removed.

- Bus Voltage: The voltage contours (shown in Figure 2-2 and Figure 2-3) represent the minimum of all nine bus voltages as determined by load flow for each state. There were no conditions that resulted in overvoltage conditions for the system.
- Line flow: The line flow contours (shown in Figure 2-2 and Figure 2-3) are represented as the maximum line flow for all lines at each state, as determined by a load flow analysis.
- Transient Stability: Critical clearing time was calculated for each generator, for each state. For each calculation, a Thevenin reduction was performed, equal area criterion applied using the appropriate dispatch, and critical clearing time (tcrit) was calculated.
- Small Signal Stability: For each state, the dynamic generator, governor, exciter and power system stabilizer models were linearized. The system A-matrix was formed, eigenvalues determined, and damping ratios calculated. The lowest damping ratio was reported for each state. Generally, the small signal modes were well damped, therefore the damping ratio constraints were set higher than is typically seen in normal grid operation to force the RL agent to react to these constraints.

Table 3-1 RL Agent Conditions Defining Feasibility, Penalty and Failure

Security Metric	RL Agent Penalty Condition	Region Defined	Explanation
Bus Voltage	0.95 pu ≤ min(Vbus) -and- max(Vbus) ≤ 1.05 pu	Feasible	All bus voltages are within normal specifications
Line Flow	Line_Flow ≤ 4 pu	Feasible	all line flows are within their normal continuous ratings
Transient Stability	min(tcrit) ≥ 125 ms	Feasible	All generator critical clearing times are within normal specifications
Small Signal Stability	damping_ratio ≥ 7%	Feasible	The least damped system mode is well damped
Bus Voltage	1.05 pu < max(Vbus) ≤ 1.15 pu -and- 0.85 pu ≤ min(Vbus) < 0.95 pu	Penalty	Some bus voltages are out of spec but will not cause tripping
Line Flow	4.0 pu < max(line flow) ≤ 4.5 pu	Penalty	Some line flows are overloaded, but not at risk of tripping on zone 3 protection
Transient Stability	100 ms ≤ min(tcrit) < 125 ms	Penalty	Some generators have a critical clearing time that is high but not egregiously so [2]
Small Signal Stability	1.5 % ≤ damping_ratio < 7 %	Penalty	The least damped system's mode Damping Ratio is low but not unstable[1]
Bus Voltage	max(Vbus) > 1.15 pu -or- min(Vbus) < 0.85 pu	Failure	One or more bus voltages is grossly out of specification and is assumed to cause system failure
Line Flow	4.5 pu < max(line flow)	Failure	One or more line flows are grossly out of specifcation and is assumed to cause system failure
Transient Stability	min(tcrit) < 100 ms	Failure	One or more generator's critical clearing time is grossly out of specification and is assumed to cause system failre

[2] Both tcrit and damping factor establish less restrictive limits than those typically found in industry practice. This ensures that the ML agent will be required to navigate to avoid these limits.

Security Metric	RL Agent Penalty Condition	Region Defined	explanation
Small Signal Stability	damping_ratio < 1.5 %	Failure	The least damped system's mode Damping Ratio is exceptionally low or unstable[1]

4. MACHINE LEARNING APPROACHES DEVELOPED

In this research, several different machine learning approaches are used to tackle the problem of controlling a transmission system power model from an unknown state to one that is more stable, without dropping load served. These methods include simple deep q-learning as well as both model free (similar to Alpha Zero) and model-based (similar to Mu Zero). For model-free learning, the learner is attempting to learn a policy from which actions are chosen, and the policy learned maximizes expected reward. In model-based learning, the learner attempts to learn an internal model to predict the next state from the current state, in addition to the policy used in model-free learning. The learning agent's internal model of state transition is used to maximize expected reward and also allows the agent to attempt hypothetical exploration from the current state forward. Model-free and model-based learning are described more thoroughly in Section 4.3.3.

4.1. Simple Deep Q-learning (DQN) Model

The simplest model implemented is a variant of q-learning [8] that uses a deep neural network, also called a Deep Q-Network (DQN) [9]. The specific software used for this research was developed previously [10], which was extended for state navigation in the transmission grid environment in our current research [11]. The score the DQN reinforcement learning (RL) agent recieves is the immediate reward for taking an action (A_i) from the current state (S_i) that results in a new state (S_j, see Equation 7 in Section 2.1). Note that if an infeasible (or failure) state (defined in Equation 6 in Section 2.1) is reached due to an action choice, then navigation is stopped and no further reward received. Navigation is not stopped when the maximal reward is received, as learning agents must learn to recognize when to stop and also to minimize the number of actions taken during navigation. Thus, the "stay" action (see Section 1.4.1 and also Action Index 0 in Table 4-1) is included in which the dispatch does not change.

Table 4-1 Possible Agent Actions for Given Current State

Action Index	Action Description
0	Keep dispatch the same
1	Decrease State 1 (generation at G1)
2	Increase State 1 (generation at G1)
3	Decrease State 2 (generation at G2)
4	Increase State 2 (generation at G2)
5	Decrease State 3 (generation at G3)
6	Increase State 3 (generation at G3)

These actions and the reward, described in Equations (5-8) in Section 2.1, serves as the objective for any agent during navigation. The DQN agent will learn a policy to choose the action that maximizes expected reward for any given current state.

Training for the DQN agent is very simple. Given the current state, it chooses an action, and sometimes the action choice is driven by randomness (exploration) otherwise it is driven by the policy that the DQN agent has learned up to this point in training (exploitation). During non-training operation, the DQN agent will always choose the action according to its learned policy, which is an approximation for maximizing expected reward using the reward it has experienced thus far.

According to DQN learning, expected reward for action A_i given the current state S_i is updated using:

$$\boldsymbol{Q(S_i,A_i) = Q(S_i,A_i) + \alpha\left[r(S_{i+1},S_i,A_i) + \gamma \max_{a} Q(S_{i+1},a) - Q(S_i,A_i)\right]} \qquad \textbf{(15)}$$

where α is the learning rate, γ is the discount rate, $r(S_{i+1},S_i,A_i)$ is the reward received at step $i+1$ (see Equation 7 in Section 2.1), and a ranges over the actions that can be taken at step $i+1$ when the environment is at state S_{i+1}. The ε-greedy approach is used to promote exploration, defined as:

$$\boldsymbol{\varepsilon_i = \varepsilon_{\min} + (\varepsilon_{\max} - \varepsilon_{\min}) e^{-\frac{i}{\delta}}} \qquad \textbf{(16)}$$

which means that there will always be at least $\varepsilon_{\min}$ stochastistity (or randomness) in action choice for the DQN agent during learning. Note that network structure and hyperparameters for this effort have not been optimized, but these will be addressed in future work. The DQN agent model parameters are listed in Table 4-2.

Table 4-2 DQN Agent Model Parameters

Parameter	Value
γ (Discount Rate)	0.8
α (Learning Rate)	0.00025
$\varepsilon_{\min}$	0.1
$\varepsilon_{\max}$	0.9
δ (ε Decay Rate)	1000
Replay Memory	50,000
Target Update Delay	50

4.1.1. DQN Training

The DQN Agent was trained using 15,000 (or more) navigation episodes, where each episode consists of a maximum of 50 steps (or actions). The use of 50 steps ensures the Agent has the possibility of navigating to the optimzal point, while limiting computational run-time. A rolling replay memory of 50,000 episodes is maintained, from which 1,024 samples are randomly chosen for training. Policy target update is delayed by 50 episodes, meaning that the DQN agent only updates its target network after every 50 episodes are completed. Both of these parameters are also listed in Table 4-2.

4.1.2. Benchmark Comparison for Sufficiency of DQN Training

To assess the training performance of the DQN agent, final reward achieved during training is output for comparison to benchmark agents. To measure how well the trained DQN agent performs, it is compared to both Random and one-state-look-ahead Greedy agents.

The Random agent is a highly simplistic agent, and it serves as a lower bound for comparison to the DQN agent. Using a uniform random variable, the Random agent chooses an action which causes a state transistion (see Table 4-1) from its present (or current) state. The Random agent is constrained to make only a single discrete state transition per step.

The Greedy agent is implemented using the same reward space as used in training the RL agent. During action choice, it sees more information than the RL or Random agents, including all neighboring states and rewards reachable from the current state, and thus it serves as a reasonable upper bound for comparison. The Greedy agent combines information about its potential rewards to form a Decision Policy (i.e., always choose the action, see Table 4-1, that results in the maximum reward). Note that it is unlikely that the Greedy agent will find the optimal state, unless grid environment conditions are simple enough. But, it is expected that the RL agent should approach, and possibly even surpass the Greedy agent's performance given enough training and sufficient grid environment complexity.

A random starting location was selected for each episode during training for the DQN agent. Note that the Greedy and Random agents do not require training. The same reward space and reward values were use used for Greedy and Random benchmarking agents, however. Tests were conducted using all possible starting states for results presented in this paper. All starting locations were chosen from within the 3-D penalty area identified in Table 3-1. During training, each starting location was chosen randomly using uniform random chosen to prevent the RL agent from over-training on any subset of the entire set of starting states. It also prevents the game from ending before an agent's first action choice (or step), as it would if the initial condition were placed in the failure region.

4.2. Variations considered

In the research described in this report, three reward objectives were considered. The most successful object, see Equation 7 in Section 2.1, is used for the results reported in the main portion of the manuscript (see Section 5), and the other reward objectives considered are described in the appendix (see Appendix A).

In training the RL agent, both 15,000 and 150,000 episodes were used, but performance did not significantly improve when using 150,000 training episodes (see Appendix A.6). Part of the reason for these results is that starting states very near the failure region often result in immediate failure, due to random exploration, which provides very little useful information for the RL agent. More training was beneficial when combined with prioritized replay (see Section 4.2.2 and Appendix A.7).

Even though complete hyperparameter optimization was not conducted, several values were tested for both discount rate (γ) and minimum random exploration ($\varepsilon_{\min}$) used in epsilon-greedy learning. For the discount rate, values 0.5, 0.8 and 0.9 were tested, but the most stable results occurred using 0.8. Minimum exploration values of 0.2, 0.1, and 0.01 were tested, but best results came using 0.1. In future research, we would like to attempt to optimize the DQN agent model parameters more thoroughly. We would also like to combine parameterization with prioritized replay.

4.2.1. Curriculum Learning

Curriculum learning is the process of teaching agents a simple task and gradually increasing the difficulty [12]. We hypothesized it might be useful as the grid size increases and with that in mind, we experimented with using it on our problem. Initially the game action space was two-dimensional and we had agents trained for that context. When we converted over to a three-dimensional action space we used agents trained on the two-dimensional to seed the more complex problem. Given the scale of the problem, there was little difference in training from scratch and using seeded weights, thus we didn't investigate much in this context but plan on exploring it further for cases where the action space is larger.

4.2.2. Prioritized Replay

Prioritized replay involves selecting starting points for successive training episodes. In standard game play inside the environment, the learning agent always starts at a randomly chosen state, but when prioritized replay is use, the agent's starting states are biased towards those for which it needs further learning. We have observed that points that are directly adjacent to the failure region are extremely challenging to learn, since they have a higher probability of immediately resulting in failure with little or no useful information for the RL agent to learn from. We did combine prioritized replay with longer training, up to 1,500,000 episodes, and results did improve, especially in terms of the percentage of starting points leading to failure. Note that the Greedy agent cannot achieve a failed state since it will always choose the state in the current state's neighborhood which results in the largest reward. It can (and does) get trapped in local optima that are near the failure region, however.

4.2.3. Dimensionality

In the research conducted for this manuscript, we started initially with a reduced dimension version of the Mini-WECC model [13]. The Mini-WECC model includes 34 generators and 19 loads which were reduced to 9 generators and 8 loads, respectively, see Figure 4-1.

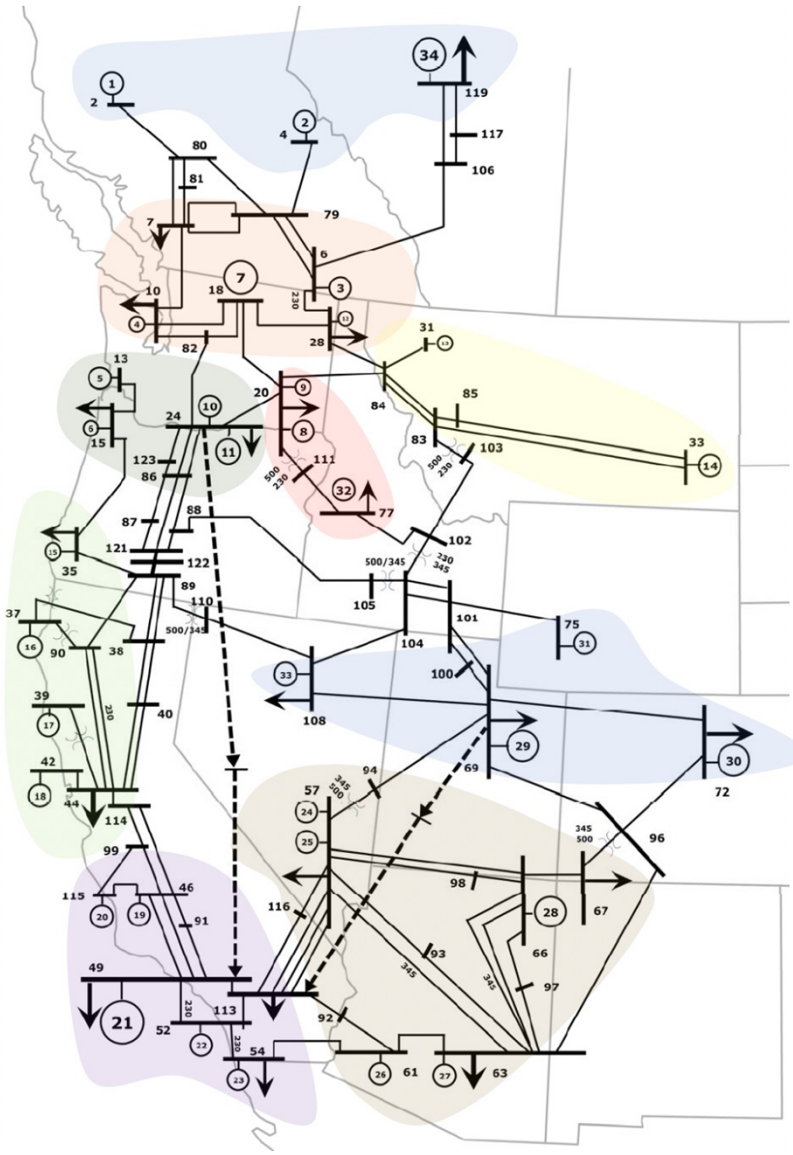

Figure 4-1. Reduced Mini-WECC Model Control State Regions (colored).

The colored areas in Figure 4-1 contain geo-mechanically identified control regions (i.e., spatially, and mechanically connected/related regions). These regions were identified using spectral clustering (see Appendix C for more details). Unfortunately, this dimensionality along with the associated control actions was too complex for an initial prototype to see statistically significant results, especially with respect to greedy agent control.

To address the difficulties in learning with many control dimensions, we drastically reduced our complexity to two dimensions. In two-dimensions, we used two overlapping polynomials each with two variables (the two control variables), see Figure 1-2 in Section 1.3. We were able to get a RL agent to quickly learn to operate in this two-dimensional space (in comparison to a Random agent), even though it is specified using continuous functions of the two control variables. The two-dimension reward space (see Equation (A.2) in Appendix A.2) adjacent to the control space and

three color feasibility regions are shown in Figure 4-2 and Figure 4-3, for discretization values of 11x11 and 101x101 respectively.

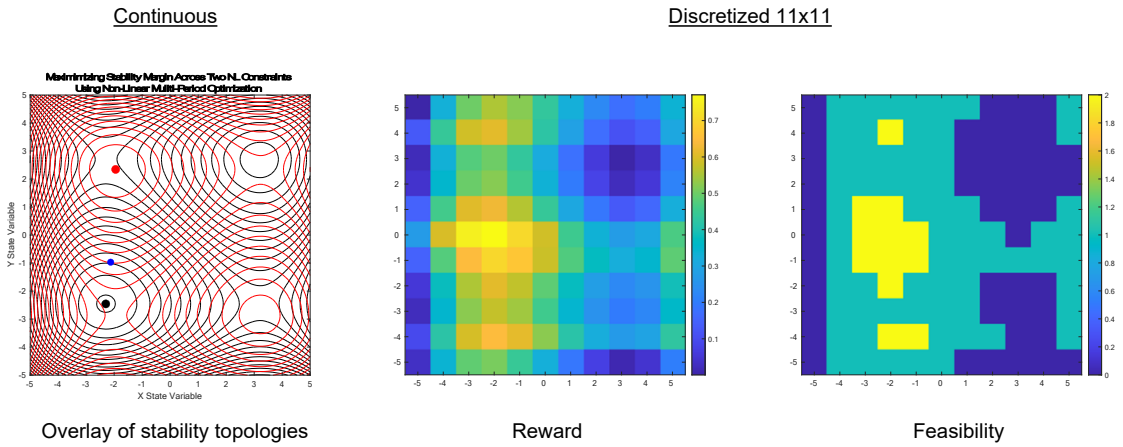

Figure 4-2. Two-dimensional Control Problem using 11x11 Discretization: Stability Overlay (left), Reward Space (center) and Feasibility (yellow)/Failure (blue)/Penalty (teal) Regions (right). For the stability overlay, the red dot is maximum of red overlay. Black dot is max of black overlay. Blue dot is global maximum.

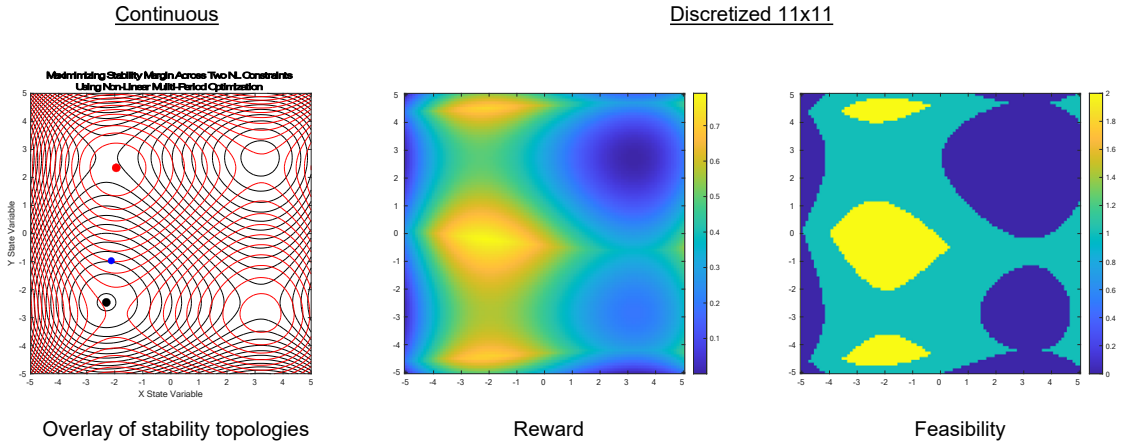

Figure 4-3. Two-dimensional Control Problem using 101x101 Discretization: Stability Overlay (left), Reward Space (center) and Feasibility (yellow)/Failure (blue)/Penalty (teal) Regions (right). For the stability overlay, the red dot is maximum of red overlay. Black dot is max of black overlay. Blue dot is global maximum.

Performance results comparing simple DQN RL agent (trained using 15,000 episodes) with the Random agent are shown in Table 4-3 (both tested using 100 starting points) and Table 4-4 (both tested using 1,000 starting points), respectively.

Table 4-3 Performance Results for Reaching Feasible Region Comparing RL and Random Agents at 11x11 Discretization (tested using 100 random starting points)

Agent	Failure	Penalty	Feasible
Random	72%	22%	6%
RL	11%	18%	71%

Table 4-4 Performance Results for Reaching Feasible Region Comparing RL and Random Agents at 101x101 Discretization (tested using 1,000 random starting points)

Agent	Failure	Penalty	Feasible
Random	11.1%	85.0%	3.9%
RL	3.5%	29.9%	66.6%

The remaining results provided later in this report (see Section 5) are for the three-dimensional grid model described in Section 3.

4.3. Experimental Agents

Considering the end goal of training an agent for an entire power grid and that safety is critical for this application, we experimented with some more complex algorithms. This involved moving from a value method, such as DQN which predicts the value of a given state, to actor-critic methods which incorporate policy to refine decision making. We further incorporated Monte Carlo tree search (MCTS) [14] as a policy refinement method, which simulates future actions and their results in order to choose an action. This was implemented as described a paper describing AlphaZero [15], a model free algorithm, followed by a model-based version based on MuZero [16]. The move to model-based approach was due to the number of computationally expensive grid simulations required in the model free approach. A brief discussion of policy-based methods, an actor-critic algorithm we applied Advantage Actor Critic (A2C), model-free vs. model-based algorithms, and our implementations of the Alpha and MuZero will follow.

4.3.1. Policy Gradient Algorithms

A policy, π, takes as an input a given state, s and outputs an action $\pi(s) \rightarrow a$. It is modeled as a function parameterized by $\theta, \pi_\theta(a|s)$. This differs from a value function in that instead of predicting the value of a given state (the Q value for DQN) it is a method for choosing the best action from a given state. Often these algorithms are sample inefficient because they require training data to be produced from the policy under consideration (called on-policy algorithms). Off-policy algorithms differ from this in that they are trained from previous policies or infer policies from a current policy, which allows them to be more sample efficient by maintaining playback buffers. Both policy- and value-based methods, however, have drawbacks which has led researchers to combine the two in what are known as actor-critic algorithms.

4.3.2. Advantage Actor Critic (A2C)

The simplest actor-critic algorithm we experimented with was Advantage Actor Critic (A2C) [17]. It operates by calculating the advantage, A for a given action, a_t, $A(s_t, a_t) = Q(s_t, a_t) - V(a_t)$. The intuition behind it is that given a policy and value function, if our policy gives us an action that performs better than our value function would predict, we should take that action more often. In the context of safety critical applications, the hope is given the added information it would be better at avoiding failure states.

The version we used for the experimentation was the Stable Baselines 3 implementation [18]. We used Optuna [19] to do hyperparameter tuning, with resulting in $\gamma = 0.9$, a decaying learning rate starting at $1e\text{-}5$, max grad norm of 0.6, and gave lambda of 0.98 performing the best. Our goal was to characterize the performance compared to DQN to see if there was improvement, especially in the area of safety.

4.3.3. Model Free vs Model Based

In RL, whether an algorithm is model free or model based refers to whether the model learns something about the dynamics of the environment it interacts with or not. Imagining an RL algorithm which might simulate future actions and results in order to decide which action to take, a model free version would choose a future action, run an environment simulator to execute the dynamics of an environment to produce a result, and iterate with the simulator in the loop. The model-based version on the other hand would use its own predictions of what the outcome of the environment's dynamics would be, iterating using that. Both, upon deciding on an actual action, would use an environment simulator (or an actual environment) to evaluate the result of a given action, the difference again being that the model-based version uses learned dynamics to produce hypotheticals and does not run the simulator when projecting into the future whereas the model free would run a simulator for each future hypothetical action.

The DQN described in Section 4.1 is an example of a model-free agent. It learns the value of a given state but nothing about the dynamics of a given environment. For it to have any prediction of what the result of an action would be it must use an external simulator.

4.3.4. AlphaZero Adaptation

AlphaZero [15] is an example of a model-free algorithm that we also implemented to test. It employs a MCTS to simulate future actions and their results in order to refine its policy and help choose actions. Again, the hope was by simulating future states it would be better at avoiding critical failures. Our version incorporated a prioritized replay buffer with importance sampling, an actor-critic network with shared weights, MCTS and all elements as described in the paper but applied to the grid action space as opposed to the game of Go.

This algorithm will not be discussed in the results section because it became apparent quickly it would be infeasible to use in a real-world grid control scenario. Each time a decision was to be made, many rollouts were required to predict results of future actions. If there were 50 rollouts, for example, that meant that the algorithm would need to run a grid simulator 50 times in the naïve case (caching and other techniques could be used to reduce this). Given grid simulators are slow, especially for large grids, this would not be successful.

4.3.5. MuZero Adaptation

MuZero solves the limitations of AlphaZero by being model based. Our implementation matched the AlphaZero version in most ways, the differences being based on those described in the MuZero paper [16]. In particular, the deep learning model differed in that instead of taking in a state and outputting policy and a value $s \rightarrow (\pi, v)$, it had an additional encoding step, as well as a dynamics prediction.

Specifically, the new deep learning model (DLM) proceeded as follows. It would accept a state and produce an encoding of that state $s \rightarrow e$. Given an encoding and an action, it predicts the successor encoding (the dynamics function): $(e,a) \rightarrow e'$. It also could take an encoded state and produce a policy, value and reward $e \rightarrow (\pi, v, r)$. These were connected and differentiable end to end, the connected version being $(s,a) \rightarrow (e', \pi, v, r)$, with the benefit that the pieces could be executed independently. Also note the complete dynamics of an environment (the reward and successor encoding) are in separate parts of the network, allowing them to be called as separate functions. This allows rewards to be computed without requiring an action.

Combining the DLM with the previously referenced MCTS, choosing an action would look roughly as follows. The MCTS would be given the current state, s, and immediately encode it as, e. No dynamics would be simulated in this case, so the encoded state would be passed to the portion of the network that generates policy, value and reward (π, v, r). The MCTS would use that information to decide which action, a, to simulate and would do so, getting a successor encoding as well as its policy, value and reward (e', π, v, r). In this way it would explore what future results might look like, ultimately choosing an action, a, based on which looked like the most promising given the projections.

The encoding portion is critical to the success of this algorithm. It is not trained to be invertible, that is it cannot reconstruct the original environment. For this reason, it is not included in the loss function during training. It is meant to learn a reduced representation of the environment that can prediction effective actions, values and rewards.

One product of this effort is a package of this algorithm with an interface that matches that of Stable Baselines 3 [18] that works on any environment that implements the gym API [20], available for future research.

4.3.6. MuZero Adaptation Training

The training setup will not be discussed at length due to the fact that this was an experimental algorithm and required more tuning in order better characterize its performance. That said, a version that was trained and performed reasonably effectively used the following hyperparameters. For the DLM, an encoding size of 70, layer sizes 256, support size of 11 (for better stability in learning rewards and values as discussed in [16]). For the MCTS a Dirichlet alpha of 0.25, a discount factor of 0.98, 15 exploration steps, a c_{init} of 1.25 and a c_{base} of 19652. Generally, it had a learning rate of $1e\text{-}3$ with a decay of 0.8, a replay buffer size 1000 and 3 unroll steps for encodings, would take a training step ever 12 episodes of data generation, and had a batch size of 64. Results shown include only 300 training steps because after that the network showed no improvement with this set of hyperparameters.

4.4. Continuous Machine Learning Algorithms

Continuous state space learning presents very challenging problems, especially since it is possible to have an infinite size space. We addressed this challenge by reducing the state visibility and control action for each agent to specific discrete sub-regions in the continuous space (see Section 3.2). We were able to demonstrate significant learning performance in comparison to a Random agent using two granularities of discretization, where performance results for the more granular sub-region discretization are not as good as those for the less granular case using the same number of training episodes (see Table 4-3 and Table 4-4 in Section 4.2.3), as expected. More research here is planned for future continuing efforts.

5. RESULTS AND CONSIDERATIONS

To demonstrate the performance of reinforcement learning, we provide two sets of results. For the first set of performance results, we look at the load served for the final navigation state. In the second set of results, each agent is ranked based upon how often the final state of each test navigation episode achieves feasibility. Reaching a feasible state quantifies the security that each agent strategy of actions facilitates, and load served is a measure of maintaining grid operation.

The Random and Greedy agents do not learn; thus, they do not need to be trained. The RL agent is trained by randomly choosing a starting state from the penalty region (see Table 3-1), and then it uses reinforcement learning (i.e., exploitation and exploration) to learn a good policy based upon maximizing expected reward received. In the results presented here, the RL agent is trained using 15,000 navigation episodes, where each episode consists of a maximum of 50 actions (i.e., a maximum of 50 state transitions). During testing each agent is presented with a starting state, and it must use a maximum of 50 actions to achieve the goal, i.e., navigating as close as possible to the state with the optimal load served in the feasible region (see Figure 2-4). Every starting state during testing comes from the penalty region, and all possible starting states are used during testing. In the appendix, we provide additional results including: 1) average performance across multiple instances of trained RL agents (see Appendix A.5), 2) performance for RL agents trained for more episodes (see Appendix A.6), and 3) RL agents trained using prioritized initial state choice (see Appendix A.7).

5.1. Penalized Final Load Served

Penalized final load served and the percentage of final load served are defined in Equation (9) and Equation (10), respectively, in Section 2.4.1. Histograms showing penalized final load served are shown in Figure 5-1, and percentages of optimal load served are listed in Table 5-1 for all three agents.

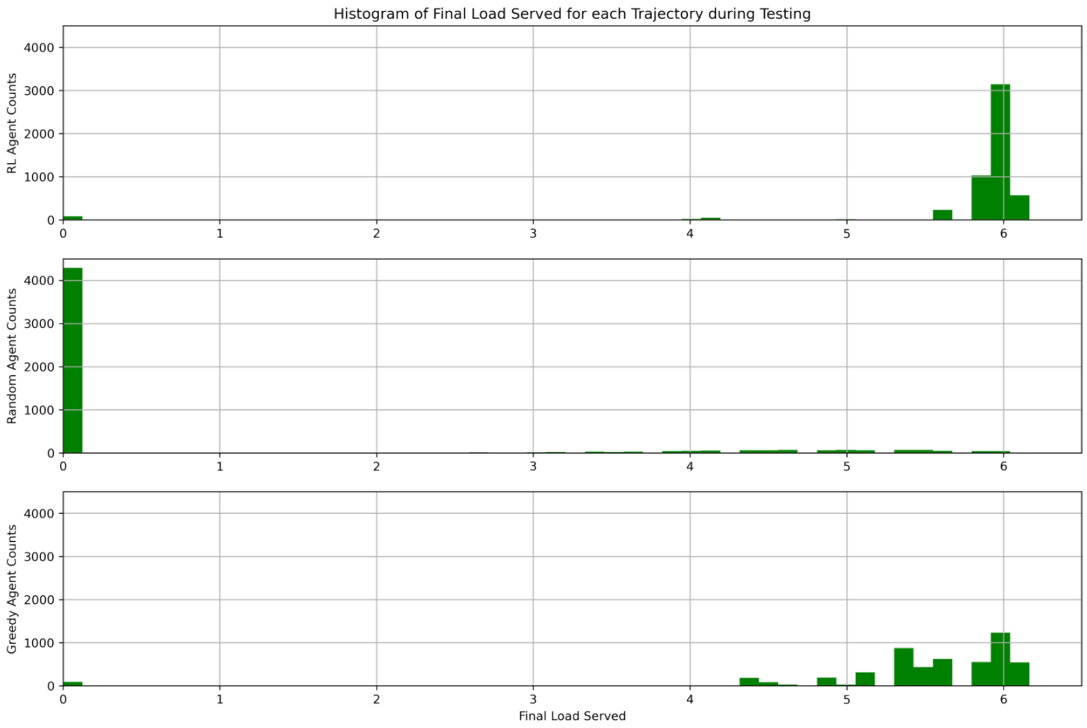

Figure 5-1. Histograms showing penalized final load served during testing for each agent: RL (top), Random (middle) and Greedy (bottom), where the optimal final load is 6.167.

Figure 5-1 demonstrates several key points in this research. First, note that the Random agent (middle histogram) does not get into the feasible region very often. Thus, its final load served is penalized more than not, resulting in the largest bin at 0 (on the left in middle histogram in the figure). Second, the Greedy agent (bottom histogram) never fails, but it also does not always get into the feasible region due to local attractors. Thus, there are a small amount of test episodes for which the Greedy agent receives 0 penalized final load served (on the left in the bottom histogram in the figure). Also, the Greedy agent is further influenced by local attractors (local optima) in the feasible region, which accounts for the distributed binning in its histogram (on the right in the bottom histogram in the figure). The RL agent is also affected by local attractors outside and inside the feasible region, which is visible in its histogram (at the top in the figure). The more concentrated binning in the RL histogram closer to the optimal final load served (in the right in the top histogram in the figure) demonstrates the utility of RL in this problem.

Table 5-1 Percentage of Optimal Final Load Served for Each Agent during Testing

Agent	Percentage of optimal final load served
Random	12.9
Greedy	89.4
RL	94.6

The results in Table 5-1 further demonstrate the utility of reinforcement learning in solving this problem. The optimal final load served is 6.167 (see Figure 2-4). The best possible performance during testing for any agent would be for it end up at the optimal state, but getting near the optimal state, while maintaining feasibility (i.e., ending at a state in the feasible region) is also acceptable. Further, achieving a final state at the edge of the feasible region (the green isoform in Figure 2-4) away from the origin is also preferred. This measure of performance rewards these conditions.

5.2. Getting into the Feasible Region

In the second performance measure, we look at the feasibility (versus penalty or failure) of the final state. Remember that the goal for each agent is to improve security while serving load, from a given starting state. This metric measures the security performance for each agent. Percentage of feasibility, penalty or failure for each starting state during testing for each agent is listed in Table 5-2.

Table 5-2 Percentage of final state feasibility, penalty or failure during testing for each agent

Agent	Failure	Penalty	Feasible
Random	43.8	39.0	17.2
Greedy	0.0	1.7	98.3
RL	1.5	0.1	98.4

The results in Table 5-2 convey several interesting facts. First, the Greedy agent, by design, cannot fail, but it does get trapped in the penalty region some of the time, due to the complexity of the navigation environment (see Figure 5-2). Because all starting states are in the penalty region, the Random agent does not get into the feasible region more than 17.2 percent of the time. And, due to the complexity of the navigation environment, the Random agent fails slightly more than it remains in the penalty region. The RL agent gets out of the penalty region about as often as the Greedy agent, but due to its random exploration, the RL agent still fails 1.5 percent of the time (about as often as the Greedy agent gets stuck in the penalty region). Neither the Greedy nor RL agent is statistically significantly better at getting into the feasible region for the simple grid system under study here.

5.3. Example Navigation Trajectories

Figure 5-2 shows three example starting states, one near the failure region in the upper right, one near the feasible region in the middle and one in a challenging region near the failure region in the lower right.

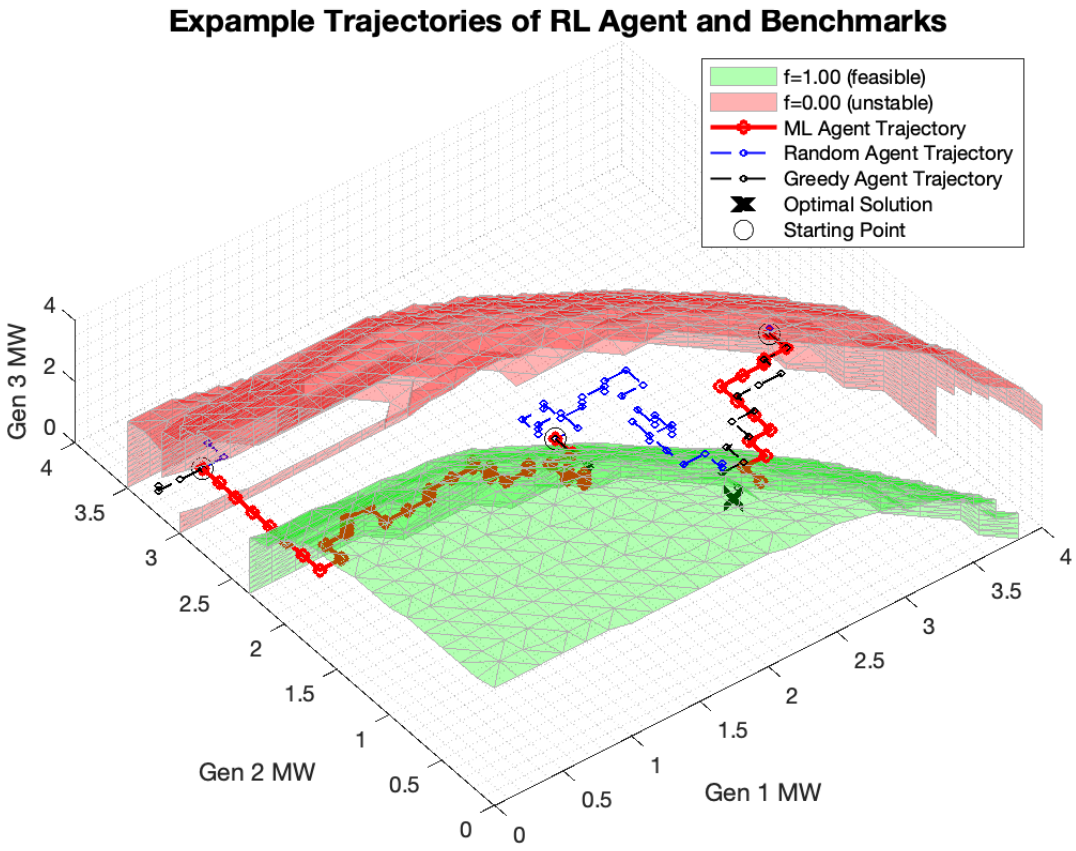

Figure 5-2. Example navigation trajectories for each agent given three different starting states during testing.

For the first starting state (on the right in the figure), the Random agent navigation (in blue) fails almost immediately, but both the Greedy and RL agents navigate into the feasible region, approaching the optimal final load served. With the second starting state (in the middle next to the green feasibility isosurface in the figure), the Random agent can be seen wandering around, where it finally ends up in the feasible region. Both the Greedy and RL agents quickly move into the feasible region and get stuck at local attractors. For the third starting state (on the left in the figure), the Random agent quickly fails, and the Greedy agent moves a small amount then gets stuck at a local attractor. The RL agent is the only agent to navigate into the feasible region from the third starting state, which visualizes the results shown in Table 5-2.

5.4. Experimental Results

The experimental results for all learning agents, including DQN, A2C and MuZero, using Stable Baselines 3 [18] are presented below. Each algorithm was evaluated on all 5,180 starting positions with the percentage of failure, penalty and feasible outcomes presented in Table 5-3.

The experimental algorithms fell short of our hopes but showed some promise. We believe a range of factors contributed to them not performing on par with or outperforming DQN that generally

centered around the larger search space for hyperparameter tuning. More investigation is required to understand why advantage and lookahead didn't succeed in improving the failure rate as hoped (see Table 5-3).

Table 5-3 Experimental Results Using Stable Baselines 3 [18]

Agent	Failure	Penalty	Feasible
DQN (Stable Baselines 3 version)	1.8%	0.2%	98%
A2C	1.8%	25.2%	73%
MuZero	2.2%	23.8%	74%

To further understand the characteristics of these algorithms we also tracked their performance at each training step. In order to better compare A2C with DQN we trained a Stable Baselines 3 [18] version of DQN with matching architecture to A2C. The following discussion was using the best from only a few runs for each algorithm. Given the stochastic nature of training the results are not a rigorous analysis of actual training schedules. For any definitive conclusions a much more thorough analysis should be conducted. That stated, the general magnitude of the training is presented for discussion.

One performance metric we observed for each training step (x axis) was the average total reward achieved for training episodes (y axis). Figure 5-3 shows that the stable baselines version of DQN gains proficiency in roughly 60,000 training steps. This was much faster than the A2C algorithm, which gained proficiency in almost 200,000 training steps (Figure 5-4). Surprisingly MuZero gained that level of proficiency much quicker at roughly 2,000 training steps (Figure 5-5).

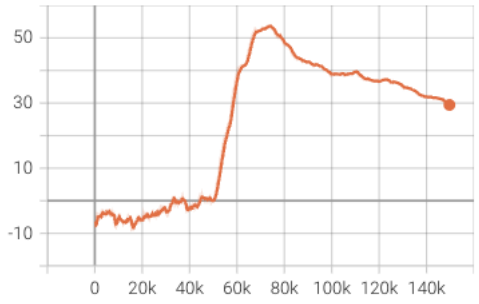

Figure 5-3. DQN episode total reward for each training step.

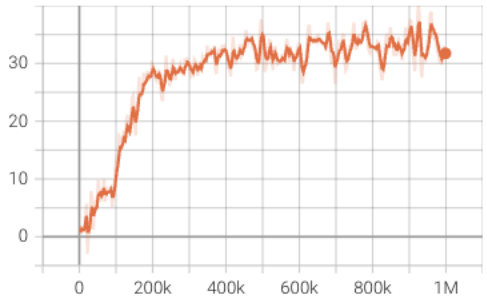

Figure 5-4. A2C episode total reward for each training step.

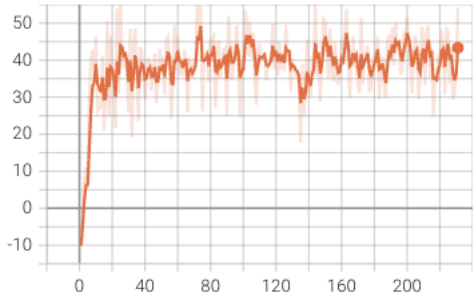

Figure 5-5. MuZero episode total reward for each training step. NOTE each step here is actually 50 training steps (so 40 is 2,000 training steps, 80 is 4,000, etc).

The actual computation used to train each algorithm is not accurately reflected in these graphs because some have many more episodes between training steps to generate training data than others. Also, MuZero, for example, takes much longer to run than the other two because utilizes so many simulations per decision. But it does appear that MuZero is learning in fewer steps than the alternatives.

We also evaluated MuZero running 20 evaluation episodes every 50 training steps and recorded the percent that achieved optimal. It took roughly 7,500 training steps for MuZero to begin to reliably achieve mostly optimal results (see Figure 5-6). Again, this appears to be much faster than the other algorithms discussed.

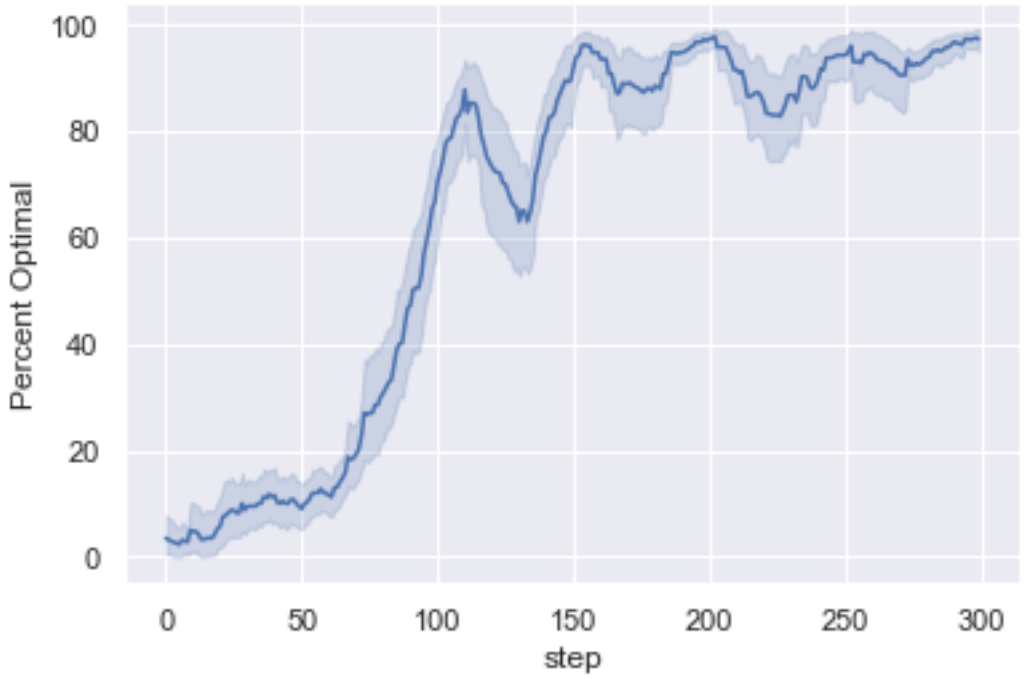

Figure 5-6. MuZero percent optimal results of 20 evaluation runs for each evaluation step. NOTE each step here is actually 50 training steps (so 40 is 2,000 training steps, 80 is 4,000, etc).

While the experimental algorithms didn't achieve the goal of reducing failure states, they showed some promise. We hope to further investigate in the future.

6. FOLLOW ON RESEARCH

Even though we have achieved some small measure of success in using reinforcement learning to stably control a simple grid model from an unsecure, or marginally secure, dispatch state to one which is more secure while continuing to serve load, there are many areas of research that still need to be investigated. In this section, we discuss some of the most important and relevant follow-on research, including use of transfer learning, on-line learning, reduced dimension modeling, scalability of machine learning solution space as well as determining ways to explain learned models, results, policies and action choice strategies for the greater understanding of results obtained, with decision-maker support firmly in mind.

6.1. Transfer learning

In the field of reinforcement learning, transfer learning can have several interpretations. In our research, we are keenly interested in learning general solutions, as opposed to optimizing for a specific environment (i.e., specific grid topology). We know that if we can build a general learner, it can always be further optimized for a specific topology, if that is desired. In our research then, transfer learning is the ability to learn using one grid topology environment, and then deploying inside another one, or learning in one topology, then continue further learning in the new topology, with measurable guarantees of success.

As described in Section 1.4.1, the MDP is very useful in representing the space for learning solutions via reinforcement learning. Thus, if we are given two topologies, T^1 and T^2, with associated MDPs, $\left(S^1, A^1, p^1\left(S_i^1,A_i^1\right), r^1\left(S_j^1,S_i^1,A_i^1\right)\right)$ and $\left(S^2, A^2, p^2\left(S_i^2,A_i^2\right), r^2\left(S_j^2,S_i^2,A_i^2\right)\right)$, it would be impossible to learn using T^1 and deploy (directly) using T^2 if $S^1 \cap S^2 = \emptyset = A^1 \cap A^2$, In this case, we would need a general learning solution that is not dependent upon a particular set of states or actions and in fact can continue to learn as new states and actions are encountered. In this situation, it is more likely that a partially observable MDP (or POMDP) is more appropriate as well as a RL solutions that fits this formalism [21]. If it is possible to determine T^1 and T^2 a priori, then we can learn using T^* where $S^* = S^1 \cup S^2$ and $A^1 = A^1 \cup A^2$, with greater generality, assuming it is possible to define the composite environment, and associated transition function, p^*, as well as appropriate reward function, r^*.

We were able to train an Alpha Zero RL agent using the two-dimensional environment described in Section 4.2.3, and then start with this agent's current learned state to continue learning in the three-dimensional environment described in Section 2. This success gives us confidence that transfer learning is at least potentially viable, but more research is needed.

6.2. On-line Learning

In our research, on-line learning refers to continuous learning in the reinforcement learning (RL) agent. In some areas of RL, this is referred to as (or is very similar and overlapping with) on-policy learning [22]. Unfortunately, pure on-policy RL is not considered viable, since such an agent can drift away from previously learned policies (also called catastrophic forgetting in ML research). We believe that research in the direction of forming algorithms that allow the RL agent to choose when, and to some extent how, to learn might address this issue.

Cleary an RL agent that catastrophically forgets what it previously learned is unacceptable. Thus, an RL agent must have the ability and capacity to maintain some measure of permanent memory (in the form of action choice policy), but it should also have the ability for continuous exploration and learning. Such an agent might need to interact with humans more as well as possibly explain that it might be losing some previous (policy) capability.

6.3. Continued work on dimensional reduction

In this research, we have shown a couple of different levels of grid control dimensionality as well as reduction in this dimensionality. The reduction we employed was, first and foremost, necessary to demonstrate the potential of RL using limited computing resources, but continued research is necessary to find ways to reduce dimensionality to a level that is feasible for learning while not reducing to the point where the solutions learned are only appropriate for such simple topologies and environments.

6.4. Scalability and expansion of solution space

The scalability of our RL solutions and expansion of these to spaces beyond our current three-dimensional topology is important follow-on research. This is intimately related to transfer learning (Section 6.1), on-line learning (Section 6.2) and reduced dimension environments (Section 6.3).

In future research, we envision an RL agent that can chose an appropriate scale (as part of its action set) for the current environment state as well as more freedom to choose when to learn, and more importantly, when not to learn. In our current research, we did explore using prioritized replay (i.e., prioritized choice for starting states during learning) with some limited success, see Appendix A.7, but more research is needed here. One of the reasons that prioritized starting state choice was not as successful (as we would like it to be) is that starting states adjacent to many other failure states, result in quick failure with little learning benefit, many times, before a "better" navigation strategy might be learned. Even when such a "better" strategy is learned, our current simple DQN model employs epsilon-greedy action choice, which includes a non-zero probability of random navigation. If the random navigation leads to failure (over and over), it provides very little useful learning.

6.5. Explainability of learned results

One clear benefit of using reinforcement learning (RL) in our solution is that actions have interpretable and explainable benefit to human decision-makers. If a RL agent recommends a sequence of actions, $A_1, \ldots A_j$, then each action can be interpreted by decision-makers and subject matter experts according to the associated MDP state, $S_1, \ldots S_j$. Unfortunately, long sequences and strategies of action choice can still be difficult to interpret by humans. Further research is needed to facilitate explainable action choice.

Future research will continue to facilitate explainable action choice, by representing more about the grid topology within the policy deep neural network (DNN). In this way, we envision action choice side-by-side with reporting relevant regions with the topology, in terms of control and security dimensions, that are being used to form policy decisions.

7. CONCLUSIONS

In this research, we have demonstrated the utility of RL for controlling a simple transmission grid system. The RL agent was shown to out-perform both Random and locally Greedy agent action control choice. The grid security dimensions were engineered to allow challenging non-failure, non-feasible starting states, and our RL agent was able to navigate out of these regions and into feasibility. That being said, this research has only taken the first steps in grid control during abnormal operating conditions.

In order for our RL agent to be useful in a control room environment, we would need to increase the model dimensionality then obtain access to an accurate but fast operating grid model environment within which to learn. Then RL agents would need to be trained using this environment. In a realistic deployment, it is unlikely that a single RL agent would suffice, thus, we would envision using an ensemble of RL agents, each trained upon a different and tractable portion of the more complex environment. Each agent could be trained upon a different security metric or on a specific geographical sub-net of the whole grid, or one of the many combinations of both of these. Then an aggregate of these results would be used to implement a stepped dispatch solution if and when needed. Currently, our RL solution could add benefit to strategic planning for control room operation in cases of severe abnormal operating conditions and possibly for mitigation of these situations. This tool would be used within a control room as a decision support tool for grid operators. It is not intended to be used during normal operation nor even to be used during contingencies with well matured procedures, but rather when grid security conditions may have degraded to point where such a tool could provide helpful advice to operators when traditional methods are unavailable. Although it is not the intended scope of the current research for RL agents to be made available for normal control room operations, this could be part of future research.

REFERENCES

[1] Y. Zhou *et al.*, "A Data-driven Method for Fast AC Optimal Power Flow Solutions via Deep Reinforcement Learning," *J. Mod. Power Syst. Clean Energy*, vol. 8, no. 6, pp. 1128–1139, Nov. 2020, doi: 10.35833/MPCE.2020.000522.

[2] Y. Zhang, J. Wu, Z. Chen, Y. Huang, and Z. Zheng, "Sequential Node/Link Recovery Strategy of Power Grids Based on Q-Learning Approach," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2019, pp. 1–5. doi: 10.1109/ISCAS.2019.8702107.

[3] M. Mandich, "Power System Stability Assessment with Supervised Machine Learning," *Masters Theses*, Aug. 2021, [Online]. Available: https://trace.tennessee.edu/utk_gradthes/6143

[4] G. R. J. Chow, "Power System Toolbox Version 3." Cherry Tree Scientific Software, 2008.

[5] P. W. Sauer, M. A. Pai, and J. H. Chow, "Power System Toolbox," in *Power System Dynamics and Stability: With Synchrophasor Measurement and Power System Toolbox*, IEEE, 2017, pp. 305–325. doi: 10.1002/9781119355755.ch11.

[6] P. Kundur and O. P. Malik, *Power system stability and control*, Second edition. New York: McGraw Hill Education, 2022.

[7] J. D. Glover, M. S. Sarma, and T. J. Overbye, *Power system analysis and design*, 5. ed., SI edition. Stamford: Cengage Learning, 2012.

[8] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3, pp. 279–292, May 1992, doi: 10.1007/BF00992698.

[9] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning." arXiv, Dec. 19, 2013. doi: 10.48550/arXiv.1312.5602.

[10] T. Bailey, J. Johnson, and D. Levin, "Deep Reinforcement Learning For Online Distribution Power System Cybersecurity Protection," in *2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, Oct. 2021, pp. 227–232. doi: 10.1109/SmartGridComm51999.2021.9631991.

[11] S. Verzi, R. Guttromson, and A. Sorensen, "Using Reinforcement Learning to Increase Grid Security Under Contingency Conditions," in *2022 IEEE Kansas Power and Energy Conference (KPEC)*, Apr. 2022, pp. 1–4. doi: 10.1109/KPEC54747.2022.9814746.

[12] X. Wang, Y. Chen, and W. Zhu, "A Survey on Curriculum Learning." arXiv, Mar. 24, 2021. doi: 10.48550/arXiv.2010.13166.

[13] D. Trudnowski and J. Undrill, "Oscillation Damping Controls, Year 1 report of Bonneville Power Administration contract 37508," ch. The MinniWECC System Model, 2008.

[14] C. B. Browne *et al.*, "A Survey of Monte Carlo Tree Search Methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012, doi: 10.1109/TCIAIG.2012.2186810.

[15] D. Silver *et al.*, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," Dec. 2017, doi: 10.48550/arXiv.1712.01815.

[16] J. Schrittwieser *et al.*, "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, Art. no. 7839, Dec. 2020, doi: 10.1038/s41586-020-03051-4.

[17] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning." arXiv, Jun. 16, 2016. doi: 10.48550/arXiv.1602.01783.

[18] "Stable Baselines3." DLR-RM, Aug. 23, 2022. Accessed: Aug. 23, 2022. [Online]. Available: https://github.com/DLR-RM/stable-baselines3

[19] "Optuna - A hyperparameter optimization framework," *Optuna*. https://optuna.org/ (accessed Aug. 25, 2022).

[20] "openai/gym." OpenAI, Aug. 25, 2022. Accessed: Aug. 25, 2022. [Online]. Available: https://github.com/openai/gym

[21] "Partially observable Markov decision process," *Wikipedia*. Aug. 14, 2022. Accessed: Oct. 20, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Partially_observable_Markov_decision_process&oldid=1104376990
[22] "Reinforcement learning," *Wikipedia*. Oct. 08, 2022. Accessed: Oct. 20, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Reinforcement_learning&oldid=1114883736

APPENDIX A. REWARD OBJECTIVES RESEARCHED

In this research, three reward objectives were considered. Equation (7) in Section 2.1 defines the most successful third reward objective. In the next two sub-sections, the other two are defined and their results are presented in the subsequent two sub-sections. Then we present average results across multiple RL agents, all trained in the same way (i.e., with the same policy DNN configuration and parameterization but different random exploration). Next, results from longer training scenarios for the RL argent are detailed. Following this, results from prioritized initial state choice during training for the RL agent are presented.

A.1. First Reward Objective

The first reward objective is very simply normalized feasibility (Equation (6) in Section 2.1) times load served (see definition for load served in Section 2.1). This reward objective is defined as follows.

$$r(s_j, s_i, A_i) = \frac{f(s_j) l(s_j)}{\max_j f(s_j) l(s_j)} \qquad (A.1)$$

One difficulty that was encountered in using Equation (A.1) both for the Greedy and RL agents is that there are local attractors (local sub-optimal states not in the feasible region but with higher reward than neighbors in the feasible region). Examples of these local optimal points can be seen in the following figure (the white squares in Figure A-1).

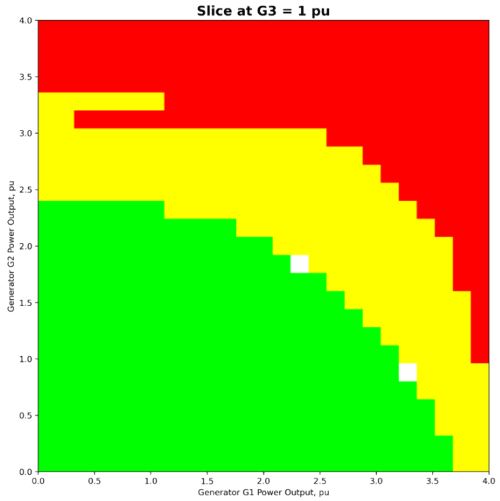

Figure A-1. Local Optima in 2D Slice of (3 color) Feasibility Space Using First Reward Objective.

An attempt to address the issue of these local optima was to design a second reward objective that gives more reward signal to feasible states as compared with any nearby non-feasible ones.

A.2. Second Reward Objective

The second reward objective rewards more for feasible states than non-feasible ones (i.e., those in the penalty or failure regions, see Table 3-1 in Section 3.3). The second reward objective adds full value (i.e., 1.0) to feasible states defined in Equation (A-1). Since the values defined in Equation (A-1) were already normalized, adding 1 and then re-normalizing ensures that feasible states will always have higher reward than non-feasible neighbors. The second reward objective is defined as follows.

$$r\left(S_j,S_i,A_i\right)=\frac{r^*\left(S_j,S_i,A_i\right)}{\max\limits_{i,j} r^*\left(S_j,S_i,A_i\right)} \tag{A.2}$$

$$r^*\left(S_j,S_i,A_i\right)=\begin{cases}\dfrac{f\left(S_j\right)l\left(S_j\right)}{\max\limits_{j} f\left(S_j\right)l\left(S_j\right)}+2 & \text{if } f\left(S_j\right)=1\\ \dfrac{f\left(S_j\right)l\left(S_j\right)}{\max\limits_{j} f\left(S_j\right)l\left(S_j\right)} & \text{otherwise}\end{cases} \tag{A.3}$$

The second reward object resulting in changing the location of local optimal near the boundary between the feasible and penalty regions. Using the second reward objective ensures that local optima near this boundary are inside the feasible region (see Figure A-2).

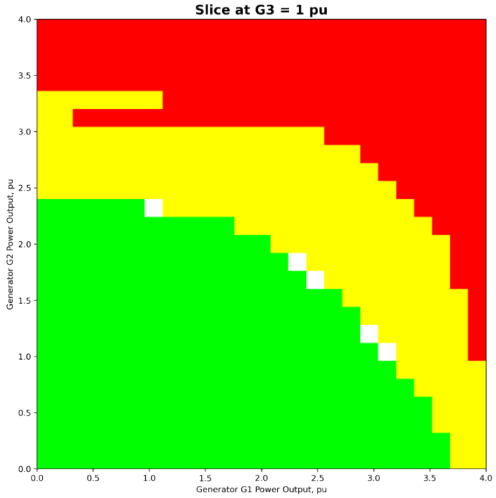

Figure A-2. Local Optima in 2D Slice of (3 color) Feasibility Space Using Second Reward Objective.

A.3. Results for First Reward Objective

Results using the first reward objective (see Appendix A.1) are presented in this section. Other than using the first reward objective, the RL agent was parameterized using parameters specified in Table 4-2 in Section 4.1. These results include penalized final load served, feasibility percentages and example 3D navigation (each in the following three sub-sections).

A.3.1. Penalized Final Load Served

Penalized final load served and the percentage of final load served are defined in Equation (9) and Equation (10), respectively, in Section 2.4.1. Histograms showing penalized final load served are shown in Figure A-3, and percentages of optimal load served are listed in Table A-1 for all three agents.

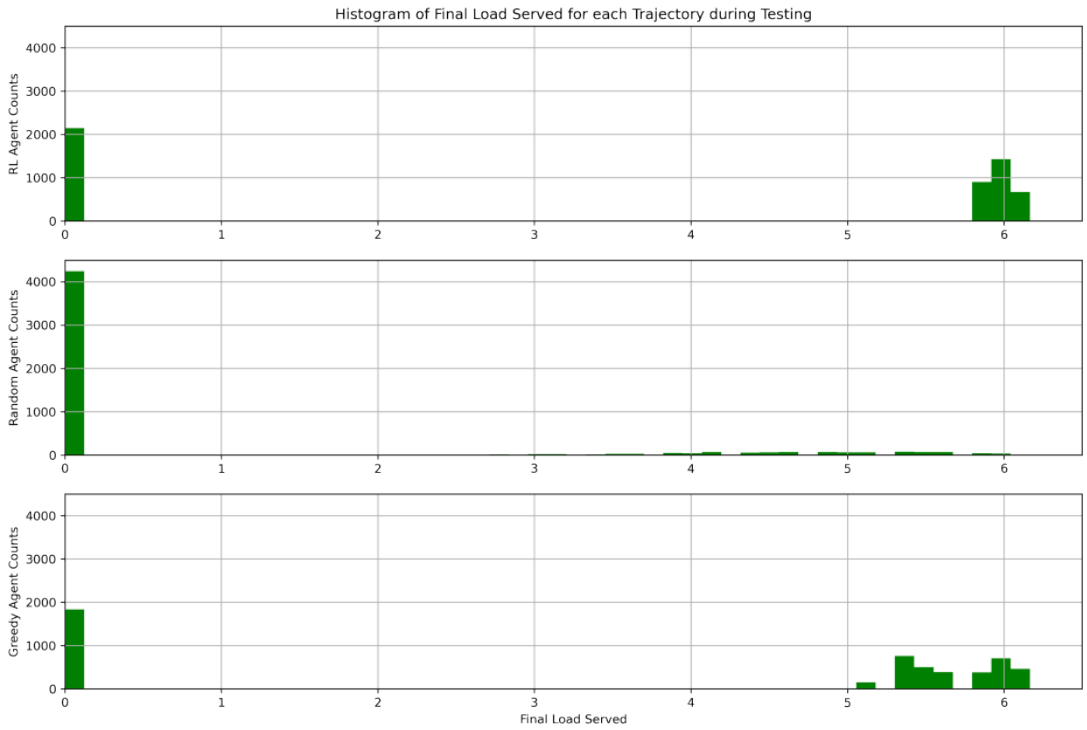

Figure A-3. Histograms showing penalized final load served during testing for each agent with first reward objective: RL (top), Random (middle) and Greedy (bottom), where the optimal final load is 6.167.

Figure A-3 demonstrates that agents are affected by local attractors outside and inside the feasible region (see Figure A-1). Local attractors outside the feasible region are responsible for all agents having difficulty in getting out of the penalty region (and into the feasible region), which is visible at the left in each histogram (penalized with 0 final load served). Local attractors inside the feasible region are responsible for the spread in both RL and Greedy histograms (top and bottom in Figure A-3).

Table A-1 Percentage of optimal final load served for each agent during testing (first reward objective)

Agent	Percentage of optimal final load served
Random	13.7
Greedy	59.8
RL	56.7

The results in Table A-1 further demonstrate the effect of the local attractors keeping both the learning and non-learning agents from reaching the feasible region and attaining a higher penalized final reward.

A.3.2. Getting into the Feasible Region

Percentage of feasibility, penalty or failure for the ending state of each test episode for each agent using the first reward objective are listed in Table A-2.

Table A-2 Percentage of final state feasibility, penalty or failure during testing for each agent (first reward objective)

Agent	Failure	Penalty	Feasible
Random	43.4	38.5	18.2
Greedy	0.0	35.4	64.6
RL	2.0	39.4	56.8

The results in Table A-2 are dominated by the local attractors just outside the feasible region (see Figure A-1). When the local attractors adjacent to the feasible region are removed, results are more reasonable (see Appendix A.4).

A.3.3. Example Navigation Trajectories

Figure A-4 shows three example starting states, one near the failure region in the upper right, one near the feasible region in the middle and one in a challenging region near the failure region in the lower right.

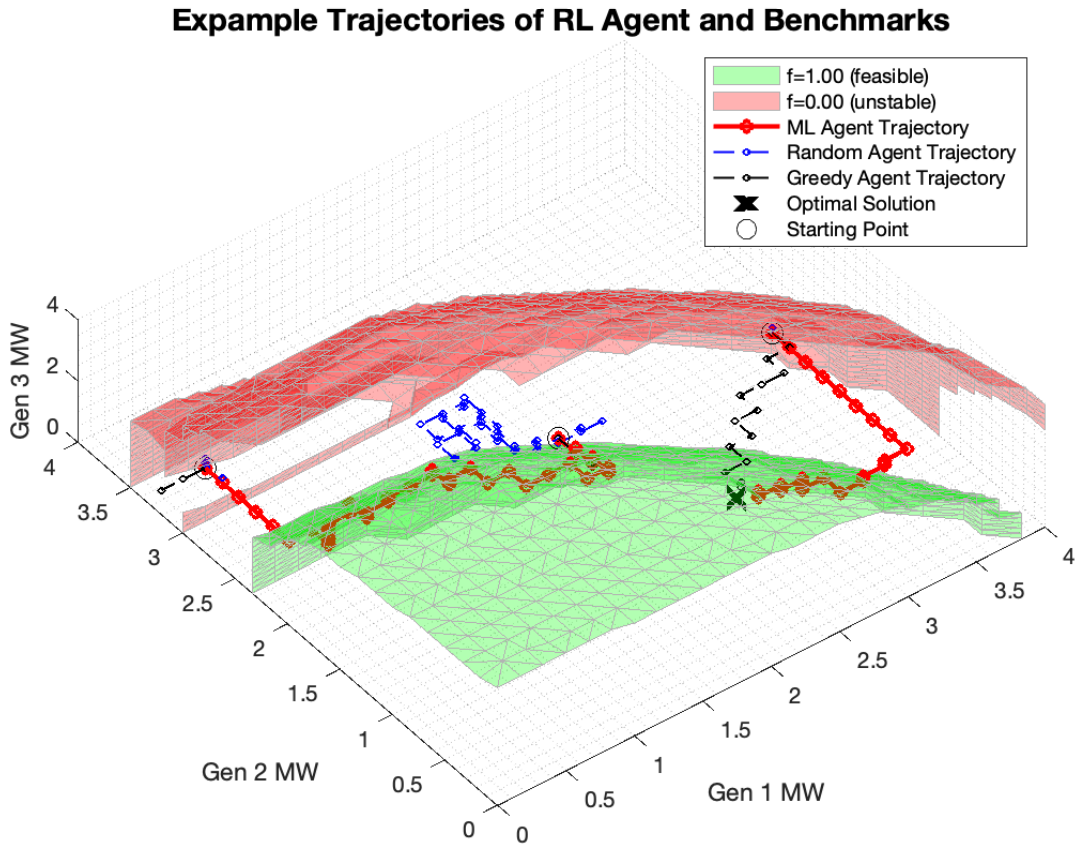

Figure A-4. Example navigation trajectories for each agent given three different starting states during testing (first reward objective).

As with example navigation using the third reward objective (see Section 5.3), the first starting state is not as challenging for the RL and Greedy agents to navigate, but the RL agent chooses a less

direct path to the feasible zone (bypassing any local attractors that might keep it from reaching the feasible region). The other starting states produce similar results as with the third reward objective (see Figure 5-2 in Section Example Navigation Trajectories).

A.4. Results for Second Reward Objective

Results using the second reward objective (see Appendix A.2) are presented in this section. Other than using the second reward objective, the RL agent was parameterized using parameters specified in Table 4-2 in Section 4.1. These results include penalized final load served, feasibility percentages and example 3D navigation (each in the following three sub-sections).

A.4.1. Penalized Final Load Served

Penalized final load served and the percentage of final load served are defined in Equation (9) and Equation (10), respectively, in Section 2.4.1. Histograms showing penalized final load served are shown in Figure A-5, and percentages of optimal load served are listed in Table A-3 for all three agents.

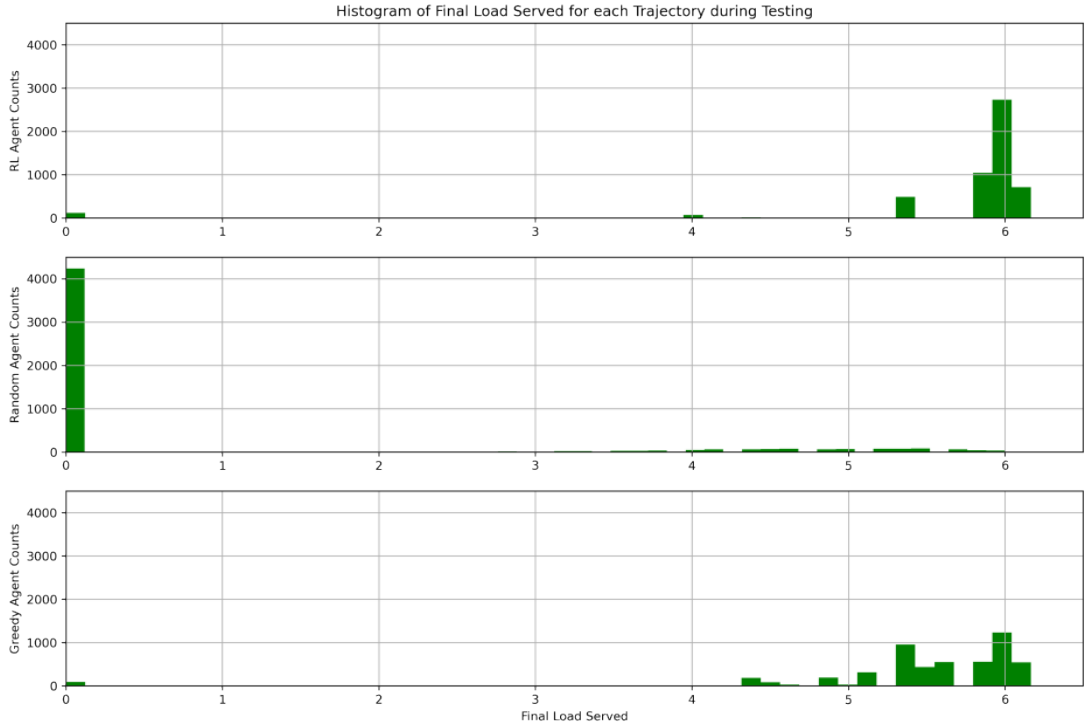

Figure A-5. Histograms showing penalized final load served during testing for each agent with second reward objective: RL (top), Random (middle) and Greedy (bottom), where the optimal final load is 6.167.

Figure A-5 demonstrates that RL and Greedy agents are not as drastically affected by local attractors outside the feasible region (see Figure A-2). Local attractors outside the feasible region are still responsible for all agents when they cannot escape the penalty region (into feasible region), which is visible at the left in each histogram (penalized with 0 final load served), drastically reduced effect for both RL and Greedy agents as compared to the first reward objective (see Figure A-3). Local attractors inside the feasible region are responsible for the spread in both RL and Greedy histograms (top and bottom in Figure A-5).

Table A-3 Percentage of optimal final load served for each agent during testing (second reward objective)

Agent	Percentage of optimal final load served
Random	13.8
Greedy	89.4
RL	93.3

The results in Table A-3 further demonstrate the lack of the local attractors keeping both the learning and non-learning agents from reaching the feasible region and attaining a higher penalized final reward (see Table A-1 for comparison). It is at this point using the second reward objective that the RL agent outperforms the Greedy agent, due to its training and the reduced effect of local attractors inside the feasible region, whereby it can achieve higher penalized final load served.

A.4.2. Getting into the Feasible Region

Percentage of feasibility, penalty or failure for the ending state of each test episode for each agent using the second reward objective are listed in Table A-4.

Table A-4 Percentage of final state feasibility, penalty or failure during testing for each agent (second reward objective)

Agent	Failure	Penalty	Feasible
Random	44.1	37.7	18.2
Greedy	0.0	1.7	98.3
RL	2.2	0.1	97.7

The results in Table A-4 are no longer dominated by the local attractors just outside the feasible region (see Figure A-2).

A.4.3. Example Navigation Trajectories

Figure A-6 shows three example starting states, one near the failure region in the upper right, one near the feasible region in the middle and one in a challenging region near the failure region in the lower right.

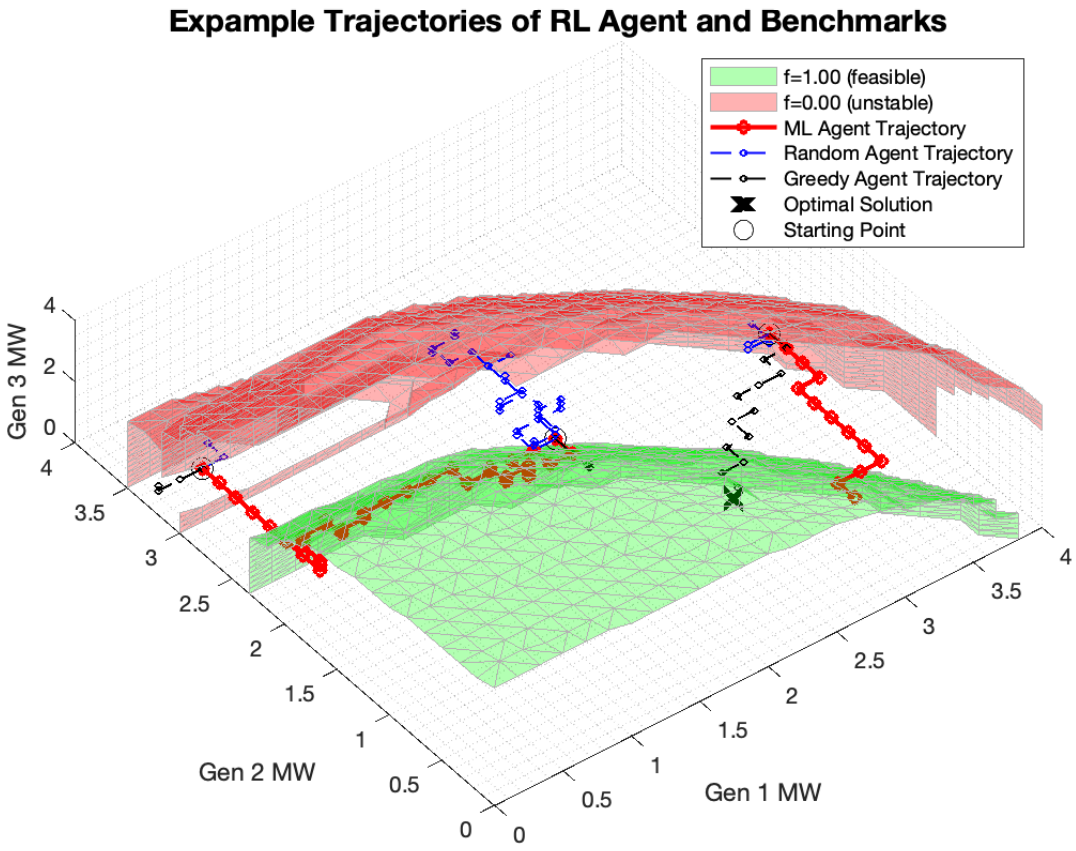

Figure A-6. Example navigation trajectories for each agent given three different starting states during testing (second reward objective).

This figure (Figure A-6) shows that the RL agent can be affected by local attractors inside the feasible region, as is visible with the first starting point (upper right in Figure A-6). In this case (using the second reward objective), the RL does get trapped by a local attractor in the feasible region, which keeps it from achieving a final load served closer to the optimal (the black X in Figure A-6). The other starting states produce similar results as with the third reward objective (see Figure 5-2 in Section 5.3).

A.5. Average Results for Multiple Trained RL Agents

In this section, we provide average results from five RL agents trained separately using the same model parameters with different random seed conditions. These agents were trained using three different high performance deep learning computing nodes: 1) dual 24-core Xeon Platinum 8168 2.7 GHz processors, 32 GB shared RAM and 2 Tesla Volta V100 GPU; 2) Lambda Scalar with dual 64-core AMD 7003 processors, 512 GM shared memory and 4 NVIDIA A100 GPUs; as well as 3) dual socket Xeon Gold 6130 2.1GHz processors, 768 GB shared memory and 4 Tesla Volta V100 NVIDIA GPUs.

We also capture results from five separate Random agents as well as five separate Greedy agents for results in this section. Note that the Greedy agents are fully deterministic, thus they produce

identical results. RL agents used for these results were parameterized identically to those described in Appendix A.4, including use of second reward objective and 15,000 episodes during training.

A.5.1. Penalized Final Load Served

Penalized final load served and the percentage of final load served are defined in Equation (9) and Equation (10), respectively, in Section 2.4.1. Histograms showing penalized final load served averaged across all 5 RL agents are shown in Figure A-7 (note that the y-axis counts are 5 times higher than those shown in Figure A-5 in Appendix A.4.1), and percentages of optimal load served are listed in Table A-5 for all three agents.

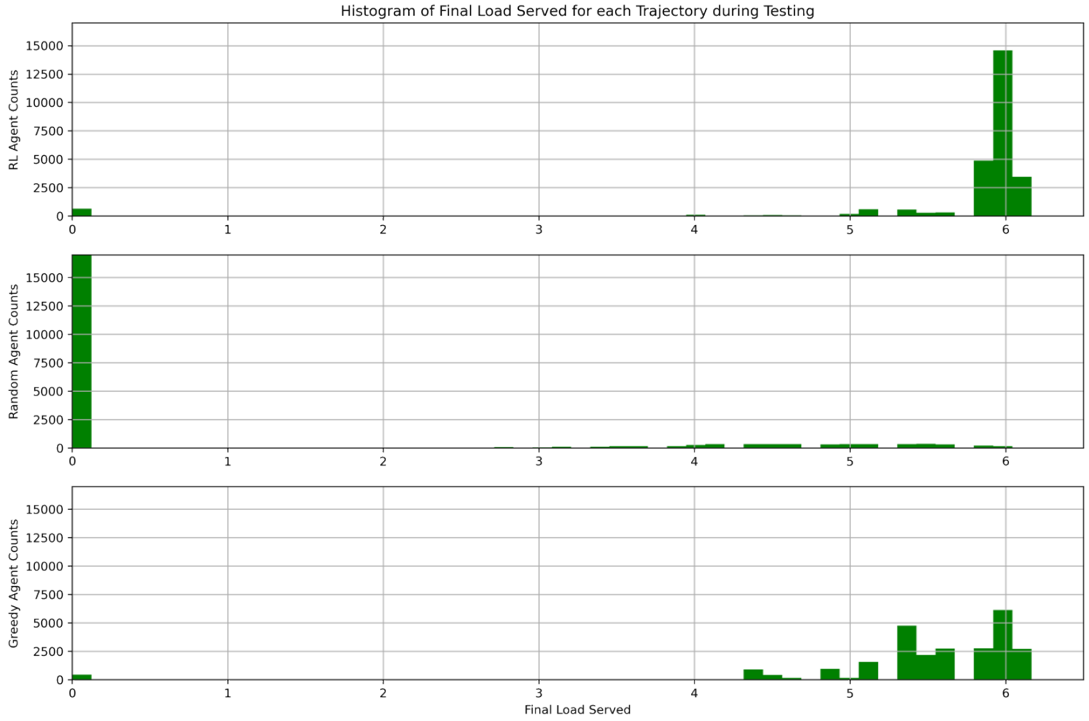

Figure A-7. Histograms showing penalized final load served during testing for each agent with second reward objective: RL (top), Random (middle) and Greedy (bottom), where the optimal final load is 6.167.

Figure A-7 demonstrates that results from five different agents of each type, Random, Greedy and RL, are not drastically different from a single agent's results (compare Figure A-5 with Figure A-7).

Table A-5 Average percentage of optimal final load served across five agents of each type during testing (second reward objective)

Agent	Average percentage of optimal final load served
Random	13.7
Greedy	89.3
RL	93.4

The results in Table A-5 are not drastically different from those for a single agent of each type, listed in Table A-3 in Appendix A.4.

A.6. Results from Training using more Training Episodes

In this section, we provide results for the RL agent in comparison with the Random and Greedy agents, where the RL agent is trained longer. In these results the RL Agent is trained using 150,000 episodes. These results are provided for all three reward objectives, but they are not drastically different that those provide for 15,000 episodes.

A.6.1. Results from 150,000 Episodes (first reward objective)

These results were obtained using the first reward objective (see Appendix A.3). Results for penalized final load served are shown in Figure A-8 and Table A-6, and those for feasibility are listed in Table A-7.

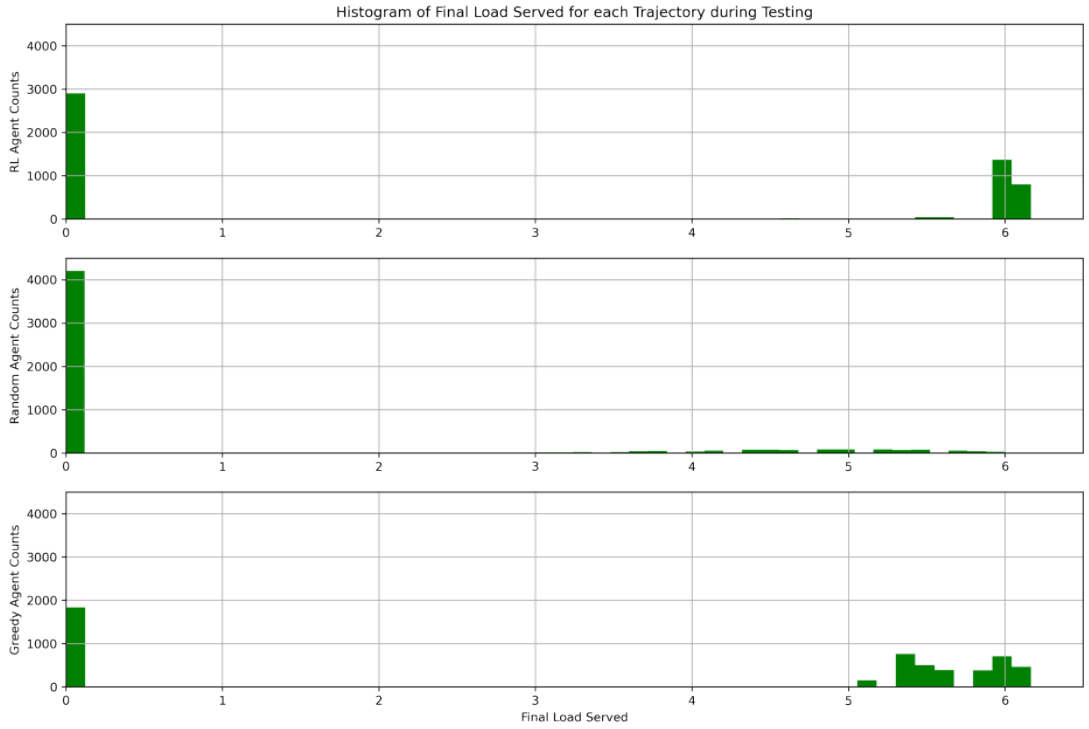

Figure A-8. Histograms showing penalized final load served during testing for each agent with first reward objective: RL (top) trained using 150,000 episodes, Random (middle) and Greedy (bottom), where the optimal final load is 6.167.

Table A-6 Percentage of optimal final load served for each agent during testing (first reward objective)

Agent	Percentage of optimal final load served
Random	14.2
Greedy	59.8
RL	43.0

Table A-7 Percentage of final state feasibility, penalty or failure during testing for each agent (first reward objective)

Agent	Failure	Penalty	Feasible
Random	42.3	38.8	18.9
Greedy	0.0	35.4	64.6
RL	2.6	53.4	44.0

A.6.2. Results from 150,000 Episodes (second reward objective)

These results were obtained using the second reward objective (see Appendix A.4). Results for penalized final load served are shown in Figure A-9 and Table A-8, and those for feasibility are listed in Table A-9.

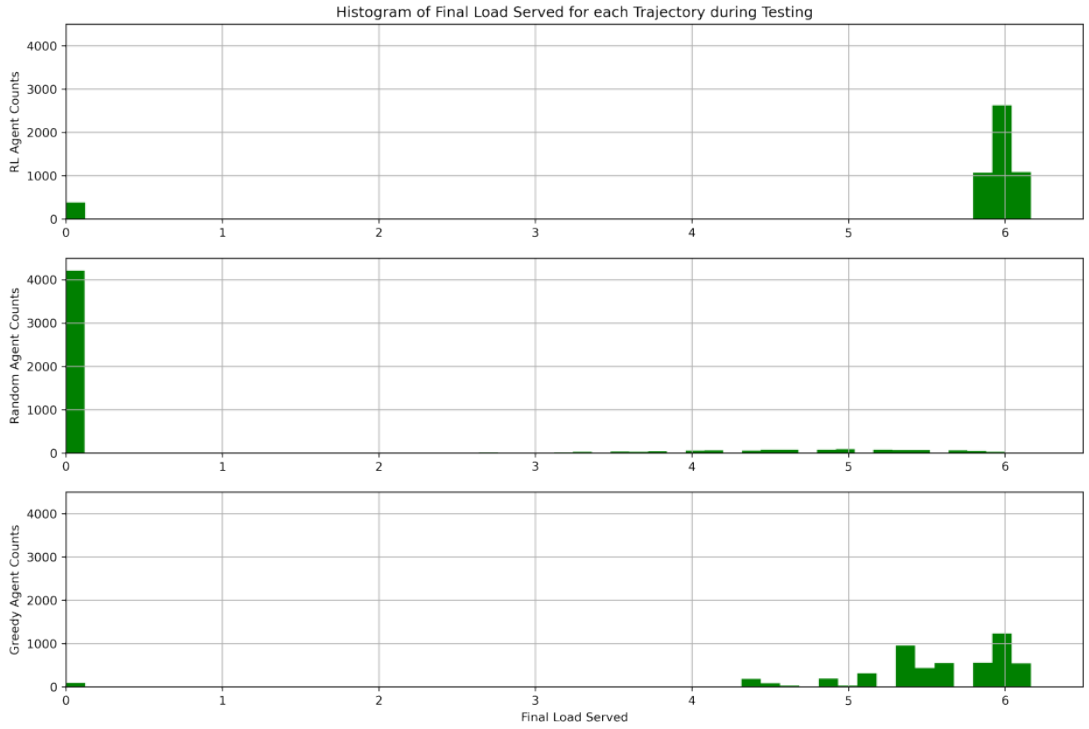

Figure A-9. Histograms showing penalized final load served during testing for each agent with second reward objective: RL (top) trained using 150,000 episodes, Random (middle) and Greedy (bottom), where the optimal final load is 6.167.

Table A-8 Percentage of optimal final load served for each agent during testing (second reward objective)

Agent	Percentage of optimal final load served
Random	14.3
Greedy	89.4
RL	90.1

Table A-9 Percentage of final state feasibility, penalty or failure during testing for each agent (second reward objective)

Agent	Failure	Penalty	Feasible
Random	42.9	38.3	18.8
Greedy	0.0	1.7	98.3
RL	7.4	0.0	92.6

A.6.3. Results from 150,000 Episodes (third reward objective)

These results were obtained using the third reward objective (see Section 2.1). Results for penalized final load served are shown in Figure A-10 and Table A-10, and those for feasibility are listed in Table A-11.

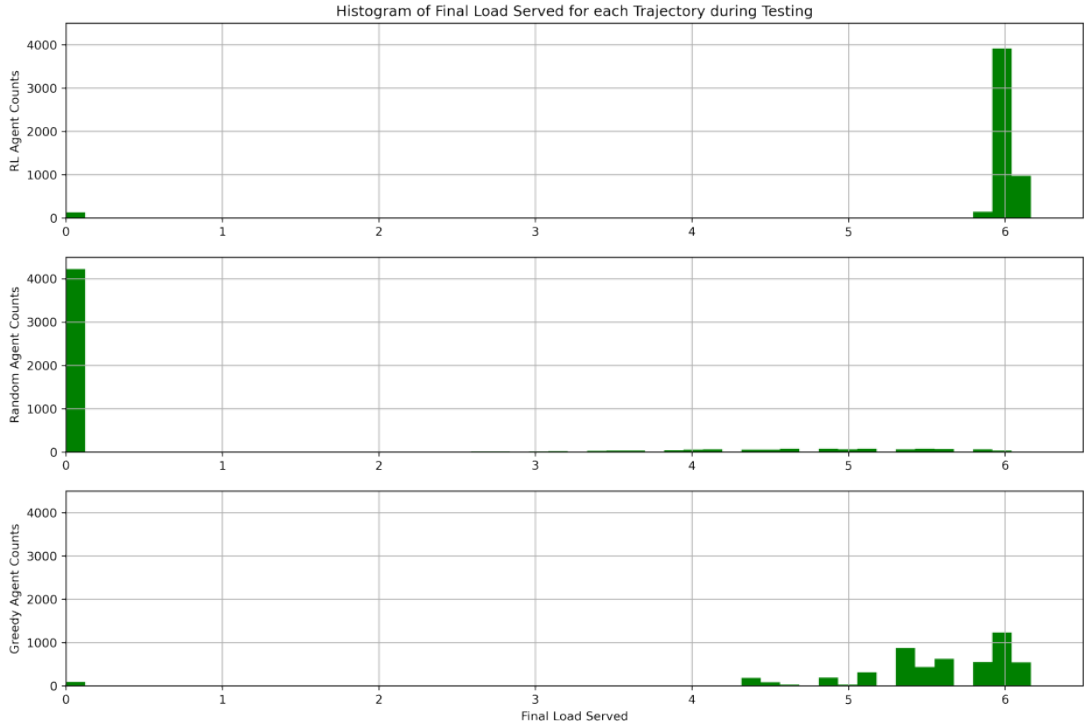

Figure A-10. Histograms showing penalized final load served during testing for each agent with third reward objective: RL (top) trained using 150,000 episodes, Random (middle) and Greedy (bottom), where the optimal final load is 6.167.

Table A-10 Percentage of optimal final load served for each agent during testing (third reward objective)

Agent	Percentage of optimal final load served
Random	14.1
Greedy	89.4

Agent	Percentage of optimal final load served
RL	95.2

Table A-11 Percentage of final state feasibility, penalty or failure during testing for each agent (third reward objective)

Agent	Failure	Penalty	Feasible
Random	42.3	39.2	18.5
Greedy	0.0	1.7	98.3
RL	2.6	0.0	97.4

A.7. Results using Prioritized Initial State Choice (during training)

In this section, we provide results for the RL agent in comparison with the Random and Greedy agents, where the RL agent is trained using prioritized initial state choice. This prioritization was done to focus RL agent training on those episodes that result in failure final states, to reduce them. Instead of using a uniform random choice for picking starting states during training, we employ prioritization to focus on those episodes that previously led to failure. Thus, any episode that leads to failure during training is marked such that during succeeding training rounds, it is 1000 times more likely to be chosen than those that did not result in failure. Results are provided using the third reward objective for 15,000, 150,000 and 1,500,000 training episodes.

A.7.1. Results from 15,000 Episodes (third reward objective)

These results were obtained using the third reward objective (see Section 2.1) utilizing prioritized initial state choice. Results for penalized final load served are shown in Figure A-11 and Table A-12, and those for feasibility are listed in Table A-13.

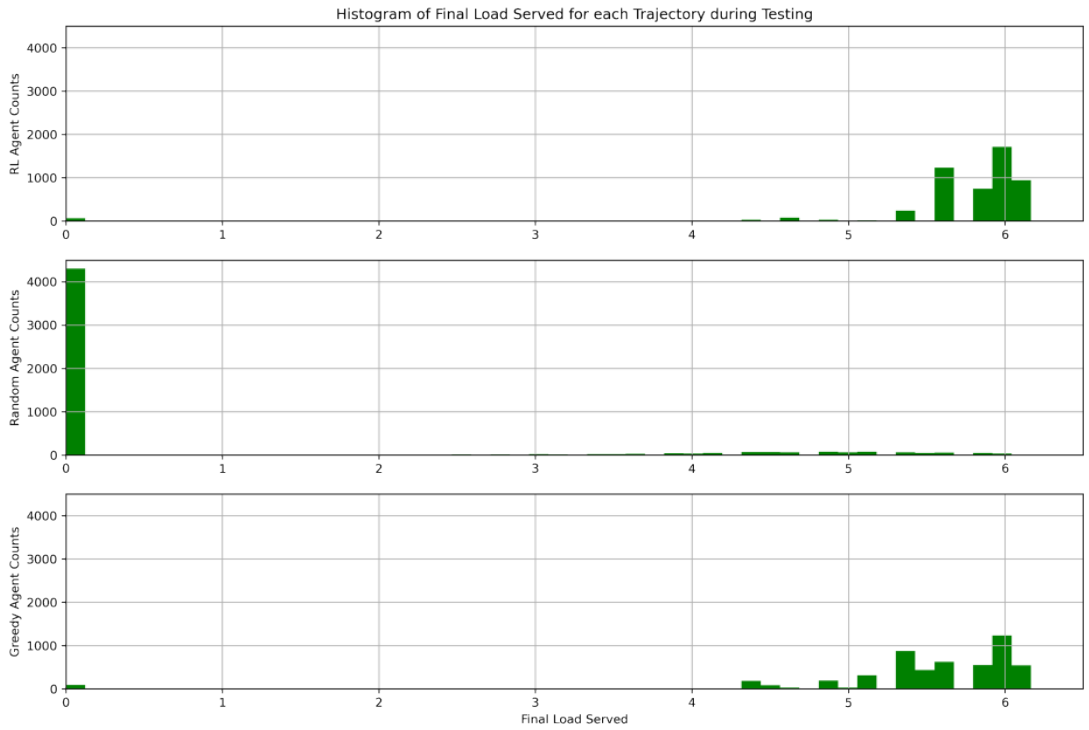

Figure A-11. Histograms showing penalized final load served during testing for each agent with third reward objective: RL (top) trained using 15,000 episodes and prioritized initial state choice, Random (middle) and Greedy (bottom), where the optimal final load is 6.167.

Table A-12 Percentage of optimal final load served for each agent during testing (third reward objective with prioritized initial state choice)

Agent	Percentage of optimal final load served
Random	12.8
Greedy	89.4
RL	92.9

Table A-13 Percentage of final state feasibility, penalty or failure during testing for each agent (third reward objective with prioritized initial state choice)

Agent	Failure	Penalty	Feasible
Random	44.1	39.0	16.9
Greedy	0.0	1.7	98.3
RL	1.3	0.0	98.7

A.7.2. Results from 150,000 Episodes (third reward objective)

These results were obtained using the third reward objective (see Section 2.1) utilizing prioritized initial state choice. Results for penalized final load served are shown in Figure A-12 and Table A-14, and those for feasibility are listed in Table A-15.

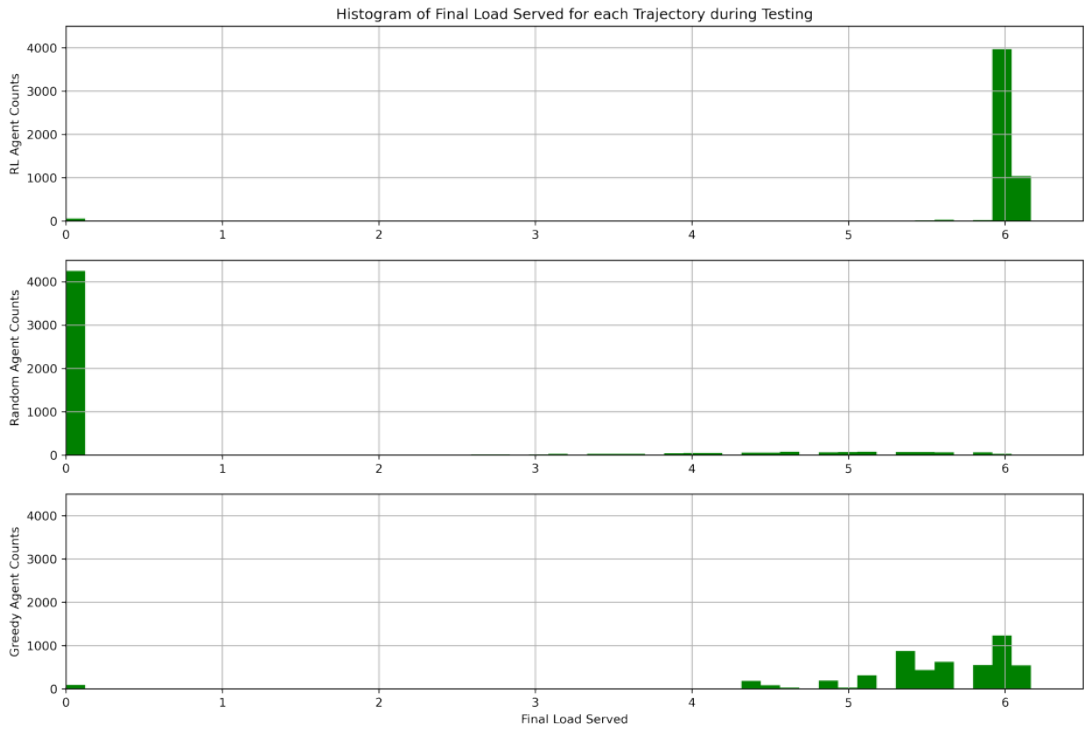

Figure A-12. Histograms showing penalized final load served during testing for each agent with third reward objective: RL (top) trained using 150,000 episodes and prioritized initial state choice, Random (middle) and Greedy (bottom), where the optimal final load is 6.167.

Table A-14 Percentage of optimal final load served for each agent during testing (third reward objective with prioritized initial state choice)

Agent	Percentage of optimal final load served
Random	13.6
Greedy	89.4
RL	96.4

Table A-15 Percentage of final state feasibility, penalty or failure during testing for each agent (third reward objective with prioritized initial state choice)

Agent	Failure	Penalty	Feasible
Random	42.9	39.1	18.0
Greedy	0.0	1.7	98.3
RL	0.9	0.2	98.9

A.7.3. Results from 1,500,000 Episodes (third reward objective)

These results were obtained using the third reward objective (see Section 2.1) utilizing prioritized initial state choice. Results for penalized final load served are shown in Figure A-13 and Table A-16, and those for feasibility are listed in Table A-17.

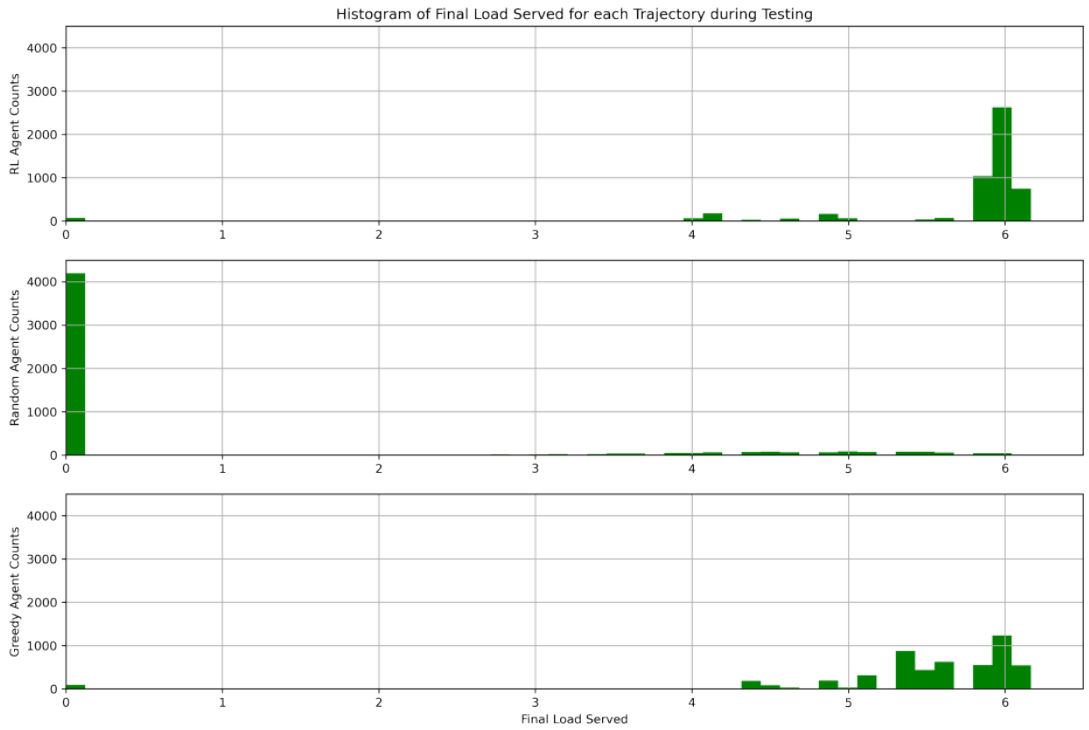

Figure A-13. Histograms showing penalized final load served during testing for each agent with third reward objective: RL (top) trained using 1,500,000 episodes and prioritized initial state choice, Random (middle) and Greedy (bottom), where the optimal final load is 6.167.

Table A-16 Percentage of optimal final load served for each agent during testing (third reward objective with prioritized initial state choice)

Agent	Percentage of optimal final load served
Random	14.4
Greedy	89.4
RL	92.8

Table A-17 Percentage of final state feasibility, penalty or failure during testing for each agent (third reward objective with prioritized initial state choice)

Agent	Failure	Penalty	Feasible
Random	43.5	37.4	19.1
Greedy	0.0	1.7	98.3
RL	1.1	0.3	98.6

It is likely these last results demonstrate some amount of overtraining effect.

APPENDIX B. MATLAB® MULTI-PERIOD OPTIMIZATION

```
%% Navigating stability margins (defined by 3rd order poly regressions)
% Solution uses multi-period non-linear optimization
% R Guttromson
close all
figs=1;% 1=yes, 0=no subplot figures
if figs==1
    figure;fig1=gcf;
end
rte=[];
startx=[ -3]; %IC for multi period path to global optimal
starty=[ 4]; %IC for multi period path to global optimal
figure;fig2=gcf;
for runs = 1:length(startx)
%% Make margin function, m1
[x,y] = meshgrid(-5:.1:5,-5:.1:5);
z=ones(101,101);
z(50:55,50:55)=5;z(30:40,30:40)=2;z(70:80,70:90)=-1;
if figs==1
    figure(fig1);subplot(4,1,1);surf(x,y,z);grid on;
end
% find m1 in closed form using poly regression
x1=x(1:end)';
x2=x1.^2;
x3=x1.^3;
y1=y(1:end)';
y2=y1.^2;
y3=y1.^3;
xo=ones(length(x1),1);
z=z(1:end)';
x=[xo x1 x2 x3 y1 y2 y3];
% cm1(1)is constant coef, cm1(2:4) are x coeffs, cm1(5:7) are y coeffs
cm1=mvregress(x,z);
[xx,yy] = meshgrid(-5:.1:5,-5:.1:5);
zz=cm1(1)+ cm1(2).*xx + cm1(3).*xx.^2 + cm1(4).*xx.^3 + cm1(5).*yy +
cm1(6).*yy.^2 + cm1(7).*yy.^3;
if figs==1
    figure(fig1);subplot(4,1,2);surf(xx,yy,zz,'LineWidth',.1);grid
on;xlabel('x');ylabel('y');
end
v=0:.02:2; %define values of level curves for contour plot (selected
somewhat arbitrarily for visual effect)
figure(fig2);contour(xx,yy,zz,v,'.-k');grid
on;xlabel('x');ylabel('y');hold on;

%% find max(cm1)
cm1fcn = @(x)[cm1(1) + cm1(2)*x(1) + cm1(3)*x(1)^2 + cm1(4)*x(1)^3  +
cm1(5)*x(2) + cm1(6)*x(2)^2  + cm1(7)*x(2)^3];
lb = [-5;-5];ub = [5;5];
obj = @(x) -cm1fcn(x)

%Initial Condition
```

```
xo=[3,-3];
A = [];
b = [];
Aeq = [];
beq = [];
[xf1,fval] = fmincon(obj,xo,A,b,Aeq,beq,lb,ub)
figure(fig2);plot(xf1(1),xf1(2),'ko','MarkerSize',10,'MarkerFaceColor','bl
ack');
text(xf1(1)-0,xf1(2)+.5,"\bf max(Margin 2)",'FontSize',14)

% note that max(z)=-fval

%% make m2 in closed form using regression
[x,y] = meshgrid(-5:.1:5,-5:.1:5);
z=ones(101,101);
z(70:80,30:40)=5;
if figs==1
    figure(fig1);subplot(4,1,3);surf(x,y,z);grid on;
end
x1=x(1:end)';
x2=x1.^2;
x3=x1.^3;
y1=y(1:end)';
y2=y1.^2;
y3=y1.^3;
xo=ones(length(x1),1);z=z(1:end)';x=[xo x1 x2 x3 y1 y2 y3];
% cm1(1)is constant coef, cm1(2:4) are x coeff, cm1(5:7) are y coeff
cm2=mvregress(x,z);
z(1:101,1:101)=0;
z(70:101,70:101)=5;

[xx,yy] = meshgrid(-5:.1:5,-5:.1:5);
zz=cm1(1)+ cm2(2).*xx + cm2(3).*xx.^2 + cm2(4).*xx.^3 + cm2(5).*yy +
cm2(6).*yy.^2 + cm2(7).*yy.^3;
if figs==1
    figure(fig1);subplot(4,1,4);surf(xx,yy,zz);grid
on;xlabel('x');ylabel('y');
end
v=0:.02:2; %define values of level curves for contour plot (selected
somewhat arbitrarily for visual effect)
figure(fig2);contour(xx,yy,zz,v,'r');

%% find max(cm2)
cm2fcn = @(x)[cm2(1) + cm2(2)*x(1) + cm2(3)*x(1)^2 + cm2(4)*x(1)^3  +
cm2(5)*x(2) + cm2(6)*x(2)^2  + cm2(7)*x(2)^3];
lb = [-5;-5];ub = [5;5];
obj = @(x) -cm2fcn(x);

%Initial Condition
xo=[2,2];
A = [];
b = [];
Aeq = [];
```

```
beq = [];
[xf2,fval] = fmincon(obj,xo,A,b,Aeq,beq,lb,ub)
figure(fig2);plot(xf2(1),xf2(2),'ro','MarkerSize',10,'MarkerFaceColor','re
d');
text(xf2(1)-2,xf2(2)+.5,'\bf max(Margin 1)','FontSize',14)

% note that max(z) = -fval

%% find global optimal
% since constraints are normalized, find the max of cm1+cm2,
% obj max(cm1+cm2)
% nl const cm1>0, cm2>0
% bounds [x,y] in [-5,5]

% Create an anonymous objective function using cm1 and cm1 functions
% https://www.mathworks.com/help/optim/ug/nonlinear-constraints.html
%https://www.mathworks.com/company/newsletters/articles/tips-and-tricks-
combining-functions-using-anonymous-functions.html

obj = @(x) -cm1fcn(x) - cm2fcn(x); %find max(obj)=> min(-obj)

% Nonlinear inequality constraints
cineq = @(x)[-cm1fcn(x); % Const 1: -cm1obj<0
             -cm2fcn(x)];% Const 2: -cm2obj<0
ceq = [];% use cineq since I dont want to overwrite c vector (poly coefs)
nlcon = @(x) deal(cineq(x),ceq);

lb = [-5;-5];ub = [5;5];
IC=[0,0];%Initial Condition
A = [];
b = [];
Aeq = [];
beq = [];
[X,fval] = fmincon(obj,IC,A,b,Aeq,beq,lb,ub,nlcon)

% note that max(z) = -fval

%% Find Path from xo to xf
% state variables:
%  x(1)  x(2)  x(3)  x(4)  x(5) x(6) x(7) x(8) x(9) x(10) x(11)
%  x1    y1    x2    y2     x3   y3   x4   y4   x5   y5    d
% parameters: xo,yo,xf,yf
%
% note we are maximizing z ==> minimizing -z
xo=[startx(runs),starty(runs)];
%xo=[2,3]; % initial state- user determined
xf=X; % final state is the global optimal solution
%plot(xo(1),xo(2),'k>','MarkerSize',10,'MarkerFaceColor','white');
%figure(fig2);plot(xo(1),xo(2),'b>','MarkerSize',8,'MarkerFaceColor','whit
e');
figure(fig2);plot(X(1),X(2),'bo','MarkerSize',8,'MarkerFaceColor','blue');
```

```
lb = [-5;-5;-5;-5;-5;-5;-5;-5;-5;-5;-5];
ub = [5;5;5;5;5;5;5;5;5;5;inf];
Aeq=[0,0,0,0,0,0,0,0,0,0,1];
beq=[pdist([xo;xf])/6*1.25];% SETS VALUE FOR d automaically.  It can be
set to any value > pdist/6
%Aeq=[];
%beq=[];
A = [];
b = [];
IC=[1,1,1.5,1.5,1.7,1.7,2,2,2.5,2.5,2];

% It is not always assumed that one may wish to restrict the gradient-
% depends on the user's needs
%                x(1)   x(2)  x(3)  x(4)  x(5) x(6) x(7) x(8) x(9) x(10)
x(11)
%                x1     y1    x2    y2     x3   y3   x4   y4   x5   y5
d

eqidx=0;%equation number index
k=1; % [0..1] percent that the objective is minimizing gradient.  1-k is
the percent the objective is maximizing margin
for n=[1 3 5 7 9] %x index for points 1 through 5

    eqidx=eqidx+1;
    gcm1{eqidx} = @(x) cm1(2) + 2*cm1(3)*x(n) + 3*cm1(4)*x(n)^2 + cm1(5) +
2*cm1(6)*x(n+1) + 3*cm1(7)*x(n+1)^2; %gradient of cm1 with respect to
p1(x,y)
    gcm2{eqidx} = @(x) cm2(2) + 2*cm2(3)*x(n) + 3*cm2(4)*x(n)^2 + cm2(5) +
2*cm2(6)*x(n+1) + 3*cm2(7)*x(n+1)^2; %gradient of cm1 with respect to
p1(x,y)
end

obj = @(x)[(1-k)*(-cm1fcn(x)- cm2fcn(x)) + k*(gcm1{1}(x) + gcm1{2}(x) +
gcm1{3}(x) + gcm1{4}(x) + gcm1{5}(x) + gcm2{1}(x) + gcm2{2}(x) +
gcm2{3}(x) + gcm2{4}(x) + gcm2{5}(x))];

ceq = @(x)[(x(2)  - xo(2))^2  + (x(1)  - xo(1))^2 - x(11)^2; % dx^2 + dy^2
- d^2 = 0
           (x(4)  - x(2))^2   + (x(3)  - x(1))^2  - x(11)^2; % dx^2 + dy^2
- d^2 = 0
           (x(6)  - x(4))^2   + (x(5)  - x(3))^2  - x(11)^2; % dx^2 + dy^2
- d^2 = 0
           (x(8)  - x(6))^2   + (x(7)  - x(5))^2  - x(11)^2; % dx^2 + dy^2
- d^2 = 0
           (x(10) - x(8))^2   + (x(9)  - x(7))^2  - x(11)^2; % dx^2 + dy^2
- d^2 = 0
           (xf(2) - x(10))^2  + (xf(1) - x(9))^2  - x(11)^2];% dx^2 + dy^2
- d^2 = 0
      cineq = [];% use cineq since I dont want to overwrite c vector
(poly coefs)
nlcon = @(x)deal(cineq,ceq(x));
[X,fval] = fmincon(obj,IC,A,b,Aeq,beq,lb,ub,nlcon)
```

```
rte=[rte; xo(1) xo(2) ; X(1) X(2) ; X(3) X(4) ; X(5) X(6) ; X(7) X(8) ;
X(9) X(10) ; xf(1) xf(2)];

end
% markers: <,>,h,x,o,p,s,d,^,v,.,*
%title({'Maximimizing Stability Margin Across Two NL Constraints';'Using
Non-Linear Muliti-Period Optimization'})
xlabel('Control Variable 1 (e.g. Gen 2 MW)'); ylabel('Conrol Variable 2
(e.g. Gen 2 MW)');
xticks([-4 -3 -2 -1 0 1 2 3 4]);xticklabels([0 1 2 3 4 5 6 7 8]); %relabel
axis easier than changing actual x,y parameters
yticks([-5 -4 -3 -2 -1 0 1 2 3 4 5]);yticklabels([0 1 2 3 4 5 6 7 8 9
10]);
figure(fig2);plot(rte(2:end,1),rte(2:end,2),'b>','MarkerSize',10,'MarkerFa
ceColor','blue');plot(rte(1,1),rte(1,2),'b>','MarkerSize',10,'MarkerFaceCo
lor','blue');hold off;
hold;
%plot(startx,starty,'O','MarkerSize',20,'LineWidth',10,'MarkerEdgeColor','
Blue')text(startx-.25,starty+.5,"\bf IC",'FontSize',14)
text(startx-.5,starty+.5,"\bf IC",'FontSize',14);
text(xf(1)-2.2,xf(2)+.5,"\bf Global Maximum",'FontSize',14)
```

APPENDIX C. SPECTRAL CLUSTERING OF MINI-WECC CONTROL DIMENSIONS

We utilized spectral clustering to aggregate the original control dimensions from the Mini-WECC model, consisting of 34 generators and 19 loads, into a reduced set. The process for using this method to reduce dimensionality is shown in **Error! Reference source not found.**.

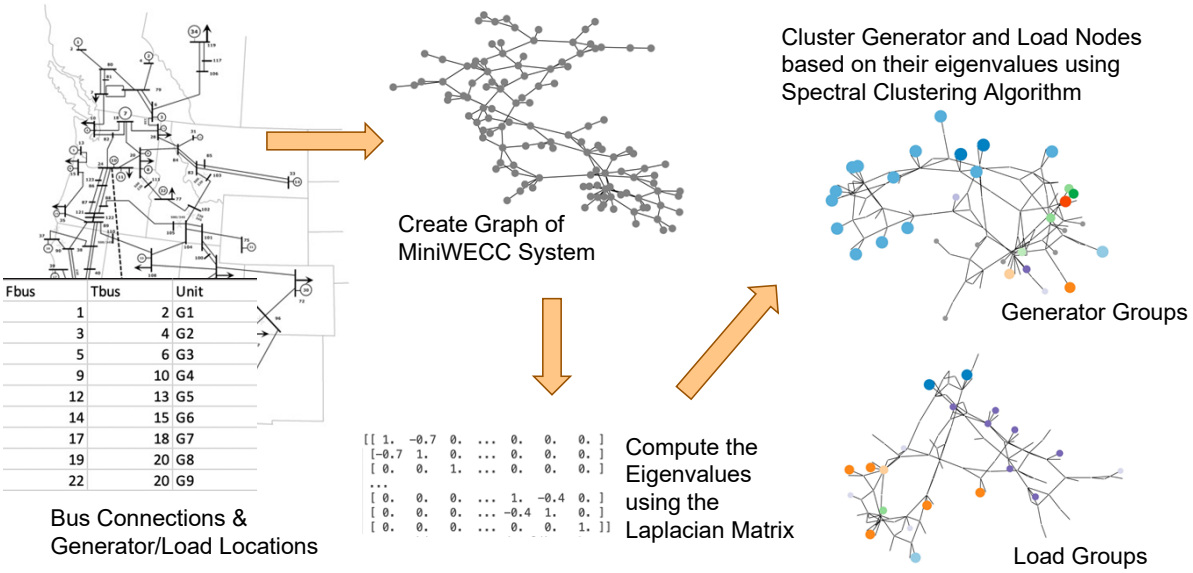

Figure A-14. Dimensionality Reduction Process using Spectral Clustering.

This reduction process resulted in identifying 9 groups of generators along with 8 loads for a reduction from 53 to 17 dimensions.

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Stephen Verzi	05522	sjverzi@sandia.gov
Ross Guttromson	08813	rguttro@sandia.gov
Asael Sorensen	05521	ahsoren@sandia.gov
Technical Library	01911	sanddocs@sandia.gov

This page left blank

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.