Dynamic Economic Emissions Dispatch with Thresholded Lexicographic Ordering

Conor F. Hayes

M.Sc. Data Analytics

College of Engineering & Informatics
National University of Ireland, Galway

Supervised by:
Dr. Enda Howley & Dr. Patrick Mannion

August 22, 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Abstract

Reinforcement Learning (RL) is a very important area in the study of Machine Learning. In RL, autonomous agents traverse an environment and learn from experience. This concept can be extended to Multi-Agent Reinforcement Learning (MARL). In MARL multiple autonomous agents traverse a problem space and again learn from experience. The goal of this thesis is to outline how RL and MARL agents behave in an environment and to implement and test these techniques on a number of Multi-Objective Reinforcement Learning (MORL) problems. MORL considers problems that have more than a single-objective. The majority of research into RL and MARL considers single-objective problems. The majority of real world problems have more than one objective.

This research applies RL and MARL techniques to the Dynamic Economic Emissions Dispatch (DEED) problem. DEED is a multi-objective problem where there are a number of fuel generators creating power for a specified population. The goal of the DEED problem is to service the population with power while minimizing the overall fuel cost and emissions.

The MARL techniques being applied to the DEED problem are Q-Learning and TLQ-Learning. Both techniques have been found to work well on single-agent RL tasks and MARL tasks. Once implemented, each algorithm will be bench-marked against existing algorithms that have been applied to the DEED problem.

1.2 Research Questions

This work aims to answer the following questions:

1. How can traditional reward structures from single-agent reinforcement learning be applied to a benchmark multi-agent reinforcement learning problem domain? (RQ1)

2. How can reward structures be applied to large multi-objective multi-agent reinforcement learning problem domains? (RQ2)

3. Is it possible to prioritise the optimisation of objectives in a multi object multi-agent reinforcement learning problem domain in lexicographic order? (RQ3)

Chapter 2

Background and Literature Review

2.1 Agents

In Machine Learning an agent can be defined as " anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators " [30] .
An agent is a software program that traverses an unknown environment and gains more information about that environment by completing defined actions and receiving rewards for those actions. A reinforcement learning agent can learn from past experiences and this leads to the agent eventually being able to define the most optimal path to the reward in the environment.

An agent has a given state in an environment. When an agent moves from one state to the next state it is known as an action. Each action updates the agent's state in the environment. An agent receives a reward for each action. This sequence of actions and changing of states is how an agent interacts with its environment. Figure 2.1 taken from Suton and Barto [32] illustrates how an agent interacts with its environment.

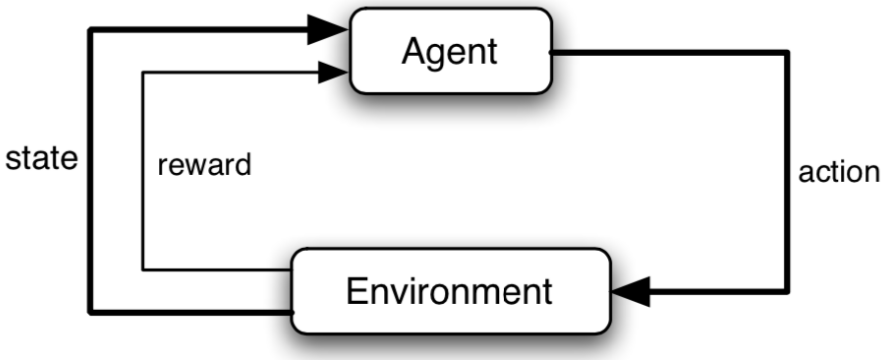

Figure 2.1: An agent interacting with its environment

An agent has four properties. Agents must be able to sense information about their environment, reason based on that information, act upon that information, act upon the environment, and receive feedback from the environment regarding their actions.

Often single-agents are used as solutions to certain problem domains. Certain problems are too complex for a single-agent and instead multiple agents are used. A multi-agent system (MAS) is a system composed of multiple interacting intelligent agents.

In MAS multiple autonomous agents act independently in a common environment. MAS can have a single-objective or multiple objectives. In multi-objective MAS the agents can be cooperative or competitive, giving rise to multi-objective multi-agent systems (MOMAS) [21]. A co-operative MAS is when agents work together to maximize their reward in a common environment. In co-operative MAS each agent has a common goal or desired outcome. In competitive MAS agents will have different goals and individual agents seek only to maximize their own gains [14].

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning pardigram, where a computer agent learns how to complete a task based on experience. An agent can be described as the learner or decision maker in an environment [32].

RL uses a similar approach to system design as used in MAS. In all RL problems there is an agent or agents that traverse an environment with the goal of maximizing their reward. In general it can be hard to define the boundary from where an agent ends and where the environment begins. It is stated that anything outside of the agent's control is part of the environment [32].

An RL agent learns how to behave in an environment usually without any prior knowledge about how to behave.

RL has been applied to many problem domains including traffic light control (Arel et al. [2] and Mannion et al. [19]) , robotics (Levine et al. [17]), optimisation problems (Mannion et al. [21], Mason et al. [20]), gaming (Mousavi et al. [24]) and air-traffic control (Britain et al. [7])

2.2.1 Reinforcement Learning Principles

Before defining a RL framework, the following main principles must be understood:

Online vs. Offline Learning - Online Learning performs learning directly on the problem instance, while offline learning uses a simulator of the environment. Offline learning is a beneficial way to run many instances to learn the environment. Often for an agent or agents to learn an environment they need hundreds, if not thousands, of iterations. Offline learning is an effective way to complete this [39].

Credit Assignment - It is important to know if the outcome of an action will be good or bad. Often the effect of an action can be delayed, the true outcome of an action or series of actions might not be fully realised until a later time. The initial actions of an RL agent have a significant impact on the overall outcome of the reward for an agent. From

the initial actions of the agent to the agent arriving at the terminal state and receiving the maximum reward, there exists a significant number of decisions to take place. Some actions, such as the initial actions, have more weight on whether or not the agent will reach the terminal state and achieve the maximum reward. How to assign an appropriate award for actions that have more significance is known as the temporal credit assignment problem [39].

Exploration vs. Exploitation Trade-off - In RL the agent learns the environment by trial and error through interaction. An agent performs an action and the environment changes state, the agent then receives feedback or a reward. The agent does not receive any feedback as to whether the action taken was the right action to take or not. At a point in time the agent will have learned a policy with a certain performance. There could be potential improvements to this policy. The only way for the agent to know whether or not there is a possible improvement to the current policy is to test various actions and evaluate their results. This could lead the agent to find a sub-optimal policy or the agent finding a policy more optimal than the current one. The only way for the agent to find this information is to explore its environment. The agent should also exploit what it already knows. The agent needs to find a balance between exploration and exploitation. Finding balance is known as the exploration-exploitation problem [39].

Goals and Rewards - The goal or purpose of an agent is formulated in terms of a special scalar signal known as a reward, r. The agent's goal is to maximize the total amount of reward it receives. Rewards can be both positive and negative numbers. Ideally, the agent will want to maximize the amount of reward it receives. Reward formulation is task specific and designing rewards can vary depending on the task. Rewards should be formulated so agents only receive the maximum reward for completing a task, such as winning a game. Generally, an agent should not be rewarded for completing sub goals. If that were the case, the agent would simply try to obtain the immediate reward and ignore the long-term reward. The idea of rewards should be used as a way of communicating to the agent what goal it should achieve, not how it should achieve the goal [32].

2.2.2 Markov Decision Processes

Markov Decision Processes (MDPs) are often used in single-agent RL problems. MDPs consist of states, actions, transitions between states and a reward function definition.

A set of States S are defined as the finite set $\{s^1, ..., s^N\}$ where the size of the state space is N [39]. The current state of an agent is the current situation an agent holds in the state action space. Usually in an RL problem environment the agents next state is limited to the states reachable by the actions that are defined for the agent in that environment.

The set of Actions A is defined as the finite set of $[a^1, ..., a^K]$ where K is the size of the action space. Actions for an agent are problem specific. For example, in a simple gridworld problem, the actions for an agent are limited by the environment. An agent in

a given state has only four actions it can perform. An agent can only move north, south, east or west [39].

When an agent is in state s and wants to complete action a, this means that the agent will transition from s to new state s'. The transition function T is defined as $T : S \times A \times S \rightarrow [0, 1]$. This equation means the probability of ending up in state s' given action a and current state s is denoted by $T(s, a, s')$ [39]. This defines the transition function.

The reward function defines the reward given to an agent for performing some action in a state. The state reward function is defined as $R : S \rightarrow \Re$ and it specifies the reward obtained in states. Therefore an MDP consists of a tuple of $< S, A, T, R >$

An agent in an environment acts based on the policy for that environment. A policy π determines how an agent will behave in a given environment. A policy outlines the actions for an agent to take in a given state. As mentioned earlier, an agent must make a trade off between exploration and exploitation in order to learn the optimum policy for a given environment. An agent can assume a learned policy is optimal, but if the agent has not traversed all states in the environment this cannot be true. In order for the agent to find the optimal policy, π^*, an agent must explore all states in the given environment. There are a number of algorithms that attempt to solve the exploration-exploitation problem and these will be discussed later [39].

In order to compute the optimal policy, π^*, MDPs use value functions. Value functions return a value denoting the discounted sum of future reward that can be expected for an agent in a given state. The value of a state s while following a policy π is the expected return when starting in s and following π thereafter. $V^\pi(s)$ creates the following equation:

$$V^\pi(s) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s \right\} \tag{2.1}$$

where γ is a discount factor. The discount factor influences the agent's behaviour. γ is a value between 0 and 1. A value for γ close to 1 ensures the agent is seeking to maximize a long term reward. A γ value close to 0 ensures an agent is seeking to maximize a short term reward [32].

Solving a task in RL, means finding a policy that achieves a large reward over time. Value functions define a partial ordering over policies. A policy π is defined to be better than or equal to another policy π' if the policy's expected return is greater than or equal to policy π. There is at least one policy that is always better than or equal to all other policies, known as the optimal policy, π_* [32]. Using π_* it is possible to define the optimal $state - valuefunction$, v_*, as

$$v_*(s) = \max v_\pi(s) \tag{2.2}$$

Optimal policies also share the same *optimal action-value function*, q_*, as

$$q_*(s, a) = \max q_\pi(s, a) \tag{2.3}$$

This gives the expected return value for taking action a in state s while following an optimal policy. Therefore we can write q_* in terms of v_* as follows:

$$q_*(s, a) = E[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a]. \tag{2.4}$$

2.2.3 ϵ - greedy Algorithm

RL agents must make a trade off between exploration and exploitation. ϵ - greedy will be used as a potential solution in later sections to this problem. A greedy action is an action where the estimated value of that action is the greatest compared to all other actions available [32]. An agent will always take the action with the greatest value or reward when acting greedily. An agent who takes a greedy action is said to be exploiting, while an agent who take a non greedy action is said to be exploring.

To ensure that the agent can both explore and exploit an environment ϵ - greedy is used. ϵ is used to define a small probability that the agent will select a random action out of the actions available to it. This increases the chances of an agent exploring the environment compared to if the agent constantly exploited the environment. The ϵ - greedy strategy selects the action with the highest expected value with probability 1 - ϵ, or randomly selected actions with the remaining probability ϵ.

2.2.4 Q - Learning

Q-Learning is a form of model-free RL. Model-free RL does not learn the transition function, model-based RL does learn the transition function. Model-free RL provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains [38]. Q-Learning is defined by the following equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.5}$$

In this case, the learned action-value function Q, directly approximates q_*, the optimal action-value function, independently of the policy being followed. Q-Learning converges to the optimum policy in a given environment even if the agent is acting sub optimally [38].

Q-Learning values for all state-action pairs are often initialized to zero. The values for $Q(s_t, a_t)$ will be set to zero for all values of t initially. Q-values are generally stored in tabular form known as a Q-table. Each state-action pair is represented in this table. Each entry represents a discrete value for each state-action pair. The Q table then acts as a look-up table for an agent. This aids the agent in selecting what action to take when in a given state. The agent generally takes the action with the highest Q-value out of the available actions. For simple RL problem domains a Q table works well. As problems grow in complexity and size so too does the Q table and can cause the 'curse of dimensionality' to become a problem. The 'curse of dimensionality' makes Q-learning an extremely computationally expensive task [38]. In this scenario learning an environment is still possible but it may take an unacceptable amount of time.

Often a function approximator is used as a solution to the problem created by large environments and the computational limits from Q tables. The function approximator generalizes or estimates values of the state-action pairs. A Q-function is used to store and retrieve estimates of the utility of the state-action pairs [18].

Neural Networks (NNs) have displayed excellent performance as function approximators. Applying NNs to RL problems has created a new field of research known as Deep Reinforcement Learning, leading to impressive advances in the capabilities of RL. For example RL learning to play Atari games [23].

2.3 Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) is an expansion on traditional single-agent RL that was seen in Section 2. In MAS there are multiple agents operating in a common environment. When going from single-agent RL to MARL, there is a generalization of the MDP. In the multi-agent case, MDP is generalized to stochastic games (SGs) or matrix games (MGs).

2.3.1 Matrix Games

A matrix game (MG) is a simple framework to deal with single shot games in MAS. A single shot game is where multiple agents have only a single state. In MGs each agents depends on each agents actions and not just on its own action. An MG can be defined as a tuple $(n, A_{1...n}, R_{1...n})$. Where n is the number of Agents. A_i is the action for each agent and where R is the reward for each agent. In MGs each agent's reward function depends on all the actions of the other agents and not just on its own actions.

Each agent's reward structure can be represented by an n-dimensional matrix. Using this concept, a strategy plays a very similar role to a policy. A strategy, σ, determines the way an agent decides on a MG. A joint strategy is a collection of strategies, one per agent. For every joint strategy there is a reward for each individual agent, which can be defined in terms of the rewards for each individual action.

An agent's strategy is deemed a best-response strategy if it achieves the highest possible reward. A Nash equilibrium, is a collection of strategy's for all agents. A Nash equilibrium is when no agent can do better by changing strategies, given that all other agents continue to follow the equilibrium strategy. It can be said that strategy's, s_1 and s_2 are in Nash equilibrium in a MAS if $agent_i$ plays s_1, $agent_j$ can do no better than play strategy s_2. If $agent_j$ plays s_2, $agent_i$ can do no better than play strategy s_1. Neither agent has any incentive to deviate from the Nash equilibrium as each agent will not receive any better reward than what they are getting currently if they take any other action [25]. All MGs have a least one Nash equilibrium. There are three types of games MGs can be classed as.

Zero-Sum Games - Are two player games, games with two agents, where the reward for one of the agents is always symmetric to the reward of the other agent.

Team Games - These have an n number of agents but the reward for each agent is the same for every joint action.

General Sum Games - These are all types of MG. This term is used only when a MG cannot be classed as a zero-sum game [25].

2.3.2 Stochastic Games

A stochastic game is a tuple $(S, A_1, ..., A_n, f, p_1, ..., p_n)$ where n is the number of agents, S is the finite set of environment states, A_i, $i = 1$, ..., n are the set of actions available to the agents. f is the state transition probability function and p_i are the reward signals of the individual agents. A SG can be thought of as an extension of MGs and MDPs. SGs are a super-set of both MGs and MDPs. An MDP can be thought of as a SG where the number of agents is equal to 1. While an SG can be thought of as an MG when the number of states is equal to 1.

In the multi-agent case, state transitions are the result of the joint action of all the agents. The next state of the system depends on the joint actions of all the agents. The policies followed by each agent also joined together to form one policy for the system to create a joint policy. In a SG the rewards of each agent depends on the joint action, which in turn depends on the joint policy.

SGs can be fully co-operative, fully competitive and mixed. In a fully co-operative SG the agents have the same reward function. The goal of each agent is to maximize the common discounted reward. In fully competitive SG the agents have opposing goals, these forms of SG are also known as zero sum games. A more technical definition of how these games behave is that one agents reward is the negative of another agents reward. Each agents goal is to maximize its own utility. Zero sum games have a unique Nash equilibrium [5].

For SGs a best response policy for an agent is one that is optimal with respect to some joint policy of the other agents. A Nash equilibrium for SGs is a collection of policies, one for each agent, so all of the policies are best response policies and no agent can do better by changing its policy, assuming all other agents continue to play their Nash equilibrium policies [25]. SGs can be classed similarly to MGs. The difference is that for a SG to be a zero-sum game, all states must define a zero-sum matrix game.

Finally, SG feature co-operative and competitive elements, hence these types of SG are known as mixed SGs for example Robo-Cup soccer.

2.3.3 Approaches to Solve Stochastic Games

The following approaches can be used to learn a stochastic game.

Q-learning - As mentioned earlier, Q-learning is an RL technique. Q-learning can be used to learn a stochastic game in certain circumstances. A SG is equivalent to a MDP from the point of view of one agent when the other agents are playing stationary policies. Any algorithm that can learn the optimal policy for a MDP is perfectly suited for a SG. Q-learning has been used on MARL problems with success. Q-learning for MARL is only suited to 'simple' problems. When the state space is large, Q-learning requires a lot of memory and computation and this approach becomes extremely inefficient for problems with a large state space [26].

Joint Action Learners - Joint Action Learners (JALs) are intended to solve MGs. JALs learn the value of their own actions with respect to other agents [9]. JALs are different from MDP methods. JALs learn Q-values based on joint actions rather than an agent's own actions. JALs can fully observe their own state and the state of other agents. Q-learning and JALs are the same except for when it comes to the action space. When deciding which action to take, if the Q-functions for all agents were co-coordinated, the agent would always take the action with the highest reward when the agent is not exploring [9]. There is no guarantee all agents are learning at the same rate or another agent is learning. It was proposed to use a value function to be used instead of a Q-value when deciding what action to take in this circumstance [9].

Nash-Q - Nash Q-learning is a technique that adapts Q-learning to the multi-agent context. Nash Q-learning recognizes joint actions rather than individual actions. Nash Q-learning's Q function becomes $Q(s, a^1..., a^n)$ for n agents. Nash Q-learning also is guaranteed to converge to a Nash equilibrium, this was proved in Hu et al. [13].

Hu et al. describes Nash Q-learning values as " the expected sum of discounted rewards when all agents follow specified Nash equilibrium strategies from the next period on " [13]. Q-learning and Nash Q-learning are very similar in a lot of ways, but differ in regards to use of the Q-values of the next state to update those on the current state [13]. In Nash Q-learning the agent updates with future Nash equilibrium pay offs, while a Q-learning agent updates with its own maximum payoff. There needs to be a way for the agent to learn the Nash equilibrium payoffs. To do this, each agent must observe its own reward and the rewards of all other agents.

A stage game is a game with one period while a stochastic game is a game with many periods [13]. Initially, all Q-values are set to 0 at timestep 0, $Q_0^i(s, a^1..., a^n)$, where n is the number of agents and i is the number of agents. Q^i is used to denote the Q-value for each agent. At each timestep, t, each agent, i, observes the current state and takes its action. The agent observes its own reward, the reward of all other agents, actions taken by all other agents, and the new state. The agent calculates the Nash equilibrium for the stage game and updates its Q-values using the following equation [13]:

$$Q_{t+1}^i(s, a^1, ..., a_n) = (1 - \alpha_t)Q_t^i(s, a^1, ..., a^n) + \alpha_t[r_t^i + \beta NashQ_t^i(s')] \tag{2.6}$$

where

$$NashQ_t^i(s') = \pi^i(s') \cdots \pi^n(s') \cdot Q_t^i(s') \tag{2.7}$$

2.4 Multi-Objective Reinforcement Learning

Most RL research has focused on single-objective problems. In reality most problems have multiple objectives. Real-world problems can have conflicting objectives, for example, in the autonomous vehicle domain, an agent in charge of driving the vehicle will have to take into account speed vs. safety. Certain trade offs need to be made to establish an optimal solution in these problem domains. There has been little research into multi-objective reinforcement learning (MORL) and this thesis will focus on MORL and applying it with MARL to a dynamic economic emissions distribution problem (DEED).

The aim of a MORL algorithm is to produce compromises between the multiple objectives of a task. This introduces the idea of Pareto dominance. Pareto dominance looks at multiple solutions and can determine which is the set of solutions which Pareto dominate all others. For example, one solution Pareto dominates another solution if it is superior on all objectives. A solution weakly dominates if it dominates on at least one objective and at least equal on all other objectives. Lastly, two solutions are incomparable if each is superior than the other on at least one objective [35].

In order to extract the best solutions, we can take solutions that are dominant or incomparable with every other solution. This resulting set of solutions is know as the Pareto front [35]. In order to evaluate MORL approaches certain quality indicators are used. The hypervolume indicator is a type of quality indicator used to evaluate MORL [36]. The hyperplane indicator measures the spread of a given set of solutions. Mannion describes that " diversity and accuracy of any set of solutions produced by an algorithm can easily be evaluated, by comparing its hypervolume with that of the the non dominated solution produced by a competing algorithm, or with that of the true Pareto front of the application domain " [18]. The hypervolume indicator can be used to describe the best performance obtainable by an agent.

Another distinction between single-objective and multi-objective problems is the reward. In MORL the reward is a vector rather than a scalar, as seen in single-objective RL. A MDP can be extended to a MORL problem, this leads to Multi-Objective MDPs (MOMDP) [29]. In MOMDP there is a reward function $\mathbf{R} : S \times A \times S \rightarrow R^d$, this describes a vector of d rewards. SGs can also be applied to the multi-objective domain, generating MOSG [18].

This leads to an alteration on the value function. The following equation is generated:

$$V^{\pi}(s) = E^{\pi}[\sum_{k=1}^{\infty} \gamma^k \mathbf{r}_{t+k+1} | \pi, s_t = s] \tag{2.8}$$

where r_t is the vector of rewards received at time t and $V^{\pi}(s)$ describes the multi-objective value of a state [29].

2.5 Deep Reinforcement Learning

In recent years there have been significant advances in Computer Vision due to training machine learning models using Neural Networks (NN). NN are composed of simple processing elements known as nodes. Each node has the ability to pass its activation to another node through a connection; each connection has an associated weight. A NN has multiple layers made up of nodes. There are three sections to each NN, an input layer, hidden layer(s) and an output layer. The input layer is the intersection of raw input and the beginning of an NN. The output layer is the output of the NN. There can be an unlimited number of hidden layers. Once an NN goes beyond one hidden layer it is said to be a Deep Neural Network (DNN).

Each node in all hidden layers contains an activation function. Below describes how a unit computes its activation.

$$y = sigmoid(\sum_j w_j x_j) \tag{2.9}$$

. where w_j is the weight of the jth incoming connection to the unit, x_j is the activation passed from the jth connection. Sigmoid corresponds to a linear activation function, the equation is outlined below:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.10}$$

All activation functions need to be differentialable. Some use linear functions while others use non-linear functions. In recent years sigmoid has been very popular, it is common to see ReLU and Tanh activation functions implemented in DNNs.

The back-propagation algorithm is the most common training algorithm used on NN. The goal of back propagation is to minimize the output error between the NN and the training data. This error can be measured using the Mean Squared Error equation. The backpropagation algorithm then uses this error to adjust the weights at each layer. After a certain amount of training, often thousands of iterations, the error should be low enough to make a good prediction against test data.

NNs are function approximators. NNs have been used in the past with RL with excellent results. Tesauro's TD-Gammon [33] is a famous example where a NN was trained to beat a computer at the game backgammon. A NN with weights θ used with RL is known as a Q-Network.

A Q-network [23] can be trained by minimizing a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i:

$$L_i(\theta_i) = E_{s,a\sim p(\epsilon)}[(y_i - Q(s, a; \theta_i))^2] \tag{2.11}$$

where $y_i = E_{s'\sim\epsilon}[r + \gamma \max_{a'} Q(s', a'; Q_{i-1})|s, a]$ is the target for iteration i and $p(s, a)$ is a probability distribution over sequences s and actions a [23]. The output depends on the weights of the network. To be able to accurately obtain weights to minimize the

error, the differential of the loss function is needed. Differentiating the loss function with respect to the weights gives the following equation:

$$\nabla_{\theta_i} L_i(\theta_i) = E_{s,a\sim p(.);s'\sim\epsilon}[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))\nabla_{\theta_i} Q(s, a; \theta_i)]. \quad (2.12)$$

The Deep Reinforcement Learning approach outlined above is used in Silver et al. [23]. Silver et al. applied Deep Reinforcement Learning to get RL agents to learn to play Atari games. In this process Silver et al. used a technique known as experience replay. This is where the agents' experiences at each time step are stored, $e_t = (s_t, a_t, r_t, s_{t+1})$ in a data set, $D = e_1, ..., e_n$. During the inner loop of the algorithm, Q-learning updates were applied to samples of experience. The remainder of the algorithm is outlined by Silver et al. " After performing experience replay, the agent selects and executes an action according to an ϵ-greedy policy. Since using histories of arbitrary length as inputs to a neural network can be difficult, the Q-function instead works on fixed length representation of histories produced by a function ϕ " [23]. This algorithm is known as Deep Q Learning. When this algorithm is used with Convolutional Neural Networks it is known as Deep Q-Networks (DQN).

The DQN algorithm performed well on several games beating other RL algorithms. Deep RL has been applied to many problem domains such as autonomous driving (O'Kelly et al. [27] and Lange et al. [16]), energy (Anderson et al. [1] and Glavic et al. [12]), finance (Brandt et al. [6] and Deng et al. [10]) and computer vision (Jie et al. [15] and Caicedo et al. [8]). Possibilities exist to apply Deep RL to many other problem domains.

2.6 Dynamic Economic Emissions Dispatch

Dynamic Economic Emissions Dispatch (DEED) is a problem first introduced by Basu [4]. In DEED a number of electrical generators must be run in order to provide power for a population while minimizing both cost and emissions. This problem has received attention in recent years and Particle Swarm Optimization techniques [22] and MARL [21] techniques have been applied to this problem with good results.

Basu's version of DEED represents itself as a multi-dimensional optimisational problem with each dimension representing a generator at a given time. In order to simulate the DEED problem the following equations have to be defined.

Cost: The following equation represents the fuel cost for each generator. This equation is known as the fuel cost function and it takes into account the valve point loading [37].

$$f_1 = \sum_{m=1}^{M}\sum_{n=1}^{N}[a_n + b_n P_{nm} + c_n(P_{nm})^2 + |d_n sin\{e_n(P_n^{min} - P_{nm})\}|] \tag{2.13}$$

where M = 24, the number of hours. N = 10 the number of generators, a_n, b_n, c_n, d_n and e_n are the cost coefficients associated with each generator n, P_{nm} is the power output from generator n at time m, and P_n^{min} is the minimum permissible power output of generator n.

Emissions: In order to model emissions outputted from each generator another equation needs to be defined. SO_2 and NO emissions coming from each generator are defined by the equation f_2

$$f_2 = \sum_{m=1}^{M}\sum_{n=1}^{N}[\alpha_n + \beta_n P_{nm} + \gamma_n(P_{nm})^2 + \eta \exp \delta P_{nm}] \tag{2.14}$$

where α_n, β_n, γ_n, η_n and δ_n are the emission coefficients associated with each generator n.

Power: Basu states that " the total real power generation must balance the predicted power demand plus the real power losses in the transmission lines, at each time interval over the scheduling horizon " [4]. This generates the following constraint:

$$\sum_{n=1}^{N} P_{nm} - P_{Dm} - P_{Lm} = 0 \tag{2.15}$$

Mannion et al. defined the total power output in a given hour must be equal to the sum of power demand and transmission losses [21]. Equation 2.15 can be modified to define the following:

$$\sum_{n=1}^{N} P_{nm} = P_{Dm} + P_{Lm} \qquad \forall m \in M \tag{2.16}$$

where P_{Dm} is the power demand over hour m and P_{Lm} is the transmission loss over hour m. Equation 2.16 will be used from here to define total power.

Constraints: There are two constraints set by Basu [4]. These constraints are for real power operating limits and generating unit ramp rate limits.

Real power operating limits are defined as the following:

$$P_n^{min} \leq P_{nm} \leq P_n^{max} \tag{2.17}$$

This equation defines the maximum and minimum power ranges a generator can operate within. The maximum or minimum power an generator can output.

Ramp rate limits are defined by the following equations:

$$P_{nm} - P_{n(m-1)} \leq UR_n \tag{2.18}$$

$$P_{n(m-1)} - P_{nm} \leq DR_n \tag{2.19}$$

The ramp rate limits determine the maximum or minimum possible power output of a generator from one hour to the next [21]. These equations determine the range in which a generator's power output can fluctuate over time.

For equations 2.17, 2.18 and 2.19 the following parameters are defined: P_n^{min} and P_n^{max} refer to the minimum and maximum power output of each generator, P_{nm} is the power output for $n \in N$ and $m \in M$, and UR_n and DR_n are the ramp up and ramp down limits for generator n.

Determination of Generation Levels: Mannion et al. describes that the first generator is treated as a slack generator. This is used to ensure that the power constraint set in equation 2.16 is satisfied [21]. Therefore the power outputs of the other generators are set directly and the slack generator then is able to generate more energy to meet the demands if required. The power output of the slack generator for a single hour, P_{1m}, may be calculated by the following:

$$P_{1m} = P_{Dm} + P_{Lm} - \sum_{n=2}^{N} P_{nm} \tag{2.20}$$

The transmission loss P_{Lm} is a function of all the generators, including the slack generator, and can be defined with the following equation:

$$P_{Lm} = \sum_{n=2}^{N}\sum_{j=2}^{N} P_{nm}B_{nj}P_{jm} + 2P_{1m}(\sum_{n=2}^{N} B_{1n}P_{nm}) + B_{11}(P_{1m})^2 \tag{2.21}$$

where B is the matrix of transmission line loss coefficients [4]. Therefore, by expanding Equation 2.20 and rearranging with respect to Equation 2.21, the following equation can be defined:

$$0 = B_{11}(P_{1m})^2 + (2\sum_{n=2}^{N} B_{1n}P_{nm} - 1)P_{1m} + (P_{Dm} + \sum_{n=2}^{N}\sum_{j=2}^{N} P_{nm}B_{nj}P_{nm} - \sum_{n=2}^{N} P_{nm}) \tag{2.22}$$

The loading of the slack generator at each hour can be found by solving the quadratic Equation 2.22 with simple algebra. All required values for the cost coefficients, emission coefficients, ramp limits, generator capacity limits, power demands and transmission line loss coefficients can be found in the work of Basu [4].

2.7 DEED as a Multi-objective Stochastic Game

In order to apply MARL techniques to the DEED problem, it is necessary to create a version of DEED that is a multi-objective stochastic game. Mannion et al. has completed significant work converting the DEED problem to a multi-objective stochastic game. To do this it is necessary to convert the problem into a sequential decision making problem where each hour m is equal to a separate time step [21].

Since this problem will use multiple agents, an agent will be used per generator. There will be an agent for each directly controlled generator. In DEED there are 10 generators but only 9 are directly controlled since the slack generator, when $n = 1$, has its values set via an equation. There will be 9 agents used where $i = (2, ..., 10)$. Each agent will set the power output P_{nm} of its generator $n = i$ at every timestep m.

In order to measure the system utility of each timestep, a new emissions and cost function needs to be derived. A new function f_c^L computes the local cost for each generator, n, over hour m. This equation is defined below:

$$f_c^L(n, m) = a_n + b_n P_{nm} + c_n(P_{nm})^2 + |d_n sin\{e_n(P_n^{min} - P_{nm})\}| \tag{2.23}$$

Therefore the following can be derived for the global cost function f_c^G for all generators over hour m:

$$f_c^G(m) = \sum_{n=1}^{N} f_c^L(n, m) \tag{2.24}$$

A new function f_e^L computes the local emissions for each generator, n, over hour m. This equation is defined below:

$$f_e^L(n, m) = E(a_n + b_n P_{nm} + \gamma_n(P_{nm})^2 + \eta \exp \delta P_{nm}) \tag{2.25}$$

where E = 10 the emissions scaling factor, chosen so the magnitude of the local emissions function f_e^L matches that of the local cost function f_c^L. It follows that the global emissions function f_e^G for all generators over hour m is:

$$f_e^G(m) = \sum_{n=1}^{N} f_e^L(n, m) \tag{2.26}$$

The next state for each agent needs to be defined. This is defined as a vector that contains the change in power demand since the previous timestep, ΔP_D, and the previous power output of the generator n, P_{nm}. The change in power demand at time m is calculated as [21]:

$$\Delta P_{Dm} = P_{Dm} - P_{D(m-1)} \tag{2.27}$$

Therefore the state vector for agent i (controlling generator n) at time m is:

$$s_{im} = [\Delta P_{Dm}, P_{n(m-1)}] \tag{2.28}$$

The action chosen by agent i at each timestep determines the power output of the generator n it controls. The power outputted by each generator has to fall in line with the

constraints set in equation 2.17. Therefore the actions for agent i are in the range of the following:

$$A_i = \{P_n^{min}, ..., P_n^{max}\} \tag{2.29}$$

At any hour m, when the ramp limits in equations 2.18 and 2.19 are imposed, an agent's action set is constrained to:

$$A_{im} = \{P_{n(m-1)} - UR_n \geq P_n^{min}, ..., P_{n(m-1)} - UR_n \leq P_n^{max}\} \tag{2.30}$$

The power limits and the ramp limits for the slack generator must be taken into consideration. Mannion et al. developed a global penalty function f_p^G based on the static penalty method developed by Smith et al. [31]. This function captures the violations of these constraints:

$$f_p^G(m) = \sum_{v=1}^{V} C(|h_v + 1|\delta_v) \tag{2.31}$$

$$h_1 = \begin{cases} P_{1m} - P_1^{max} & \text{if } P_{1m} > P_1^{max} \\ P_1^{min} - P_{1m} & \text{if } P_{1m} < P_1^{min} \\ 0 & \text{otherwise} \end{cases} \tag{2.32}$$

$$h_2 = \begin{cases} (P_{1m} - P_{1(m-1)}) - UR_1 & \text{if } (P_{1m} - P_{1(m-1)}) > UR_1 \\ (P_{1m} - P_{1(m-1)}) + DR_1 & \text{if } (P_{1m} - P_{1(m-1)}) < -DR_1 \\ 0 & \text{otherwise} \end{cases} \tag{2.33}$$

where V is determined by the number of constraints handled in the given problem. In this case $V = 2$ as there are two constraints that could be violated. Slack generator power and the ramp limits are the two constraints that could potentially be violated.

$C = 10E6$ is the violation constant, h_v is the violation of each constraint, and $\delta_v = 0$ if there is no violation in a given constraint and $\delta_v = 1$ if the constraint is violated. The violation constant $C = 10E6$ was selected so the output of the penalty function will have a similar magnitude to that of the cost function f_c^G [21].

It is important to note the penalty function is an additional objective that will need to be optimized in conjunction with the cost and emissions functions [21].

2.8 Thresholded Lexicographic Ordering

Thresholded Lexicographic Ordering (TLO) is a concept introduced by Gabor et al. [11]. TLO is the idea that an agent can learn in a multi-objective environment by prioritising or selecting a particular objective first and minimising this objective to a threshold. Once this threshold is reached the agent can then carry on minimising the next objective to the specified threshold.

The goal of TLO is to learn a Pareto optimal solution in less episodes than traditional Q-learning. TLO focuses on each objective until a desired solution, if possible, is reached.

The order in which each objective is prioritized can have either a positive or negative effect on what solution the agent learns.

The first implementation of TLO was by Vamplew et al. [34]. Vamplew et al. applied TLO with Q-learning called thresholded lexicographic Q-learning (TLQ-learning). Vamplew et al. states that " this algorithm is designed for problems where one objective must be maximised, subject to satisfying constraints on the other objectives (such as maximizing factory production while maintaining a required safety level) " [34]. TLQ-learning is essentially an action selection algorithm, where action selection is performed by applying a combination of thresholding and lexicographic ordering to the objective values of the available actions, as follows:

- Let n denote the number of objectives, labeled from $1..n$
- Let A denote the set of available actions
- Let C_j be the threshold value (minimum acceptable value) for objective j, as defined by the constraint for that objective (note: objective n will be unconstrained, hence $C_n = +\infty$)

$$CQ_{s,a,j} \leftarrow min(Q_{s,a,j}, C_j) \tag{2.34}$$

In state s, the greedy action a' is selected such that superior($CQ_{s,a'}$,$CQ_{s,a}$, 1) is true $\forall$ a $\in A$ where superior($CQ_{s,a'}$, $CQ_{s,a}, i$) is recursively defined as [34]:

Algorithm 1: TLQ-learning Action Selection

```
if CQ_{s,a',i} > CQ_{s,a,i} then
    return true;
end
if CQ_{s,a',i} = CQ_{s,a,i} then
    if i = n then
        return true;
    else
        return superior(CQ_{s,a'} , CQ_{s,a}, i + 1);
    end
else
    return false;
end
```

The algorithm above is what was used by Vamplew et al. [34]. The algorithm was applied to the Deep Sea Treasure Problem. The Deep Sea Treasure Problem is a multi-objective benchmark problem. This problem consisted of a grid of 10 rows and 11 columns. The agent controls a submarine searching for undersea treasure. There are multiple treasure

locations, and each location has a different treasure value. There are two objectives, the first is to minimise the time taken to find treasure and the second is to maximise the amount of treasure received. A visualisation of this problem can be seen in below in figure 2.2, this is taken from Vamplew et al. [34].

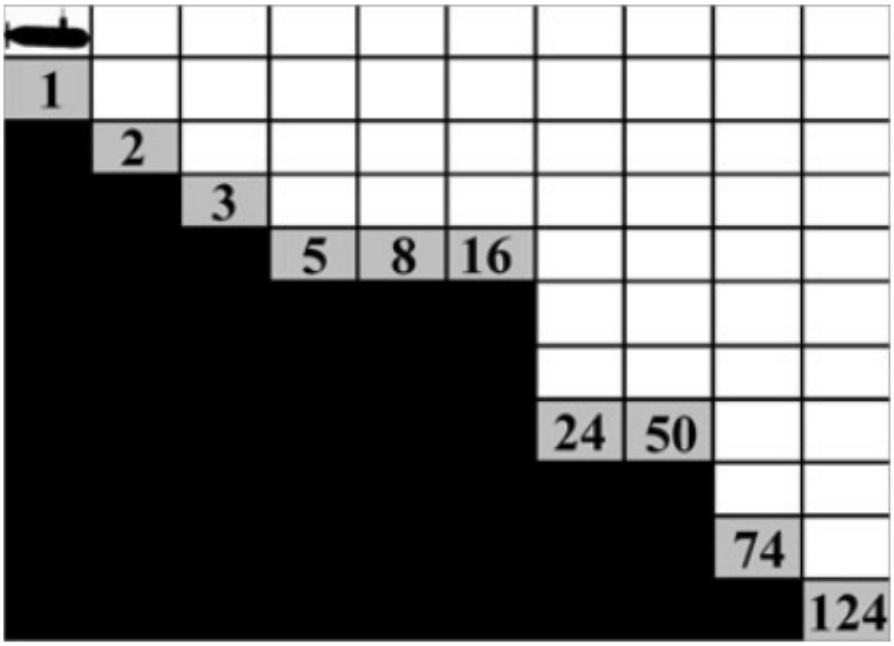

Figure 2.2: Deep sea treasure problem visualisation

Vamplew et al. [34] performed a number of experiments. First standard Q-learning was used followed by TLQ-learning with treasure being optimised first followed by time (TLQ-learning (Treasure)) and finally TLQ-learning with time being optimised followed by treasure (TLQ-learning (Time)). The results for standard Q-learning showed that Q-learning reached a Pareto optimal solution, the results also showed that optimising time first in TLQ-learning (Time) produced a poor solution. TLQ-learning (Treasure) reached a Pareto optimal solution before standard Q-learning. In this instance TLQ-learning (Treasure) produced the best solution as it reached a Pareto optimal solution in less episodes than standard Q-learning. It is important to note that this TLQ-learning (Treasure) in this instance was only tested on a single-agent multi-objective problem domain.

Chapter 3

Multi-Agent single-objective Study

Chapter 3 evaluates multiple Q-learning agents' behaviours in a single-objective domain. Chapter 3 outlines how reward shaping techniques such as global, local and difference rewards allow multiple agents to learn the specified domain and find Pareto optimal solutions. The difference reward is expected to produce the best results.

Section 3.1 presents an evaluation of the difference, global and local rewards in a multi-agent single-objective RL benchmark problem. Each reward structure rewards each agent differently. An agent's behaviour depends on the reward it receives. Using a MARL benchmark problem allows for the evaluation of each reward structure and gives insight into a reward's effect on the learning rate of each agent.

3.1 Multi-Night Problem

S.Proper et al. proposed the multi-night problem [28], which is a variant of the El Farol Bar Problem (Arthur 1994) [3]. The El Farol Bar Problem describes a congestion problem. In New Mexico, USA, there was a bar known as the El Farol Bar. The bar played Irish music every Thursday night. This was a popular event and the bar was often congested. Arthur hypothesized if it was possible for people to figure out when to attend the bar without communicating with each other so the bar was not too full or too empty.

In this version of the problem, each agent has to determine which day of the week to attend a bar. If too few agents attend the bar on a given night or too many agents attend on a night, the enjoyment of the attending agents drops. The goal is to find an optimal attendance for each night.

The El Farol Bar Problem is an example of a stateless RL problem. In a stateless RL problem, an agent has no state. An episode ends after one action, since an agent only selects which night to attend the bar. The agent then receives a reward and the process repeats for a specified duration. Each agent has only a single-objective, which is to maximize its own reward.

3.1.1 Reward Shaping

In order to quantify the systems performance, S.Proper et al. defines a reward G. G is a function of the joint action of all the agents in the system z [28], which is defined in the following equation:

$$G(z) = \sum_{day=1}^{n} x_{day} e^{\frac{-x_{day}}{C}} \tag{3.1}$$

G(z) defines the global reward and is also used to quantify the system performance. For an agent receiving the global reward, each agent receives the same reward, G(z). This can lead to selfish behaviour by the agents and can lead the system into receiving a poor reward. For example, if all agents predict an empty bar, they will all attend and receive a poor reward. If all agents predict a full bar, no agent will attend and the agents will receive a poor reward.

In order to counteract this issue, S.Proper et al. defines a difference reward. A difference reward defines an agent's contribution to the system, allowing an agent to predict what effect its actions will have on the system performance. The difference reward is defined by the following equation:

$$D_i(z) = G(z) - G(z - z_i) \tag{3.2}$$

The difference reward can be defined as the following:

$$D_i(z) = x_{dayi} e^{\frac{-x_{dayi}}{C}} - (x_{dayi} - 1) e^{\frac{-(x_{dayi}-1)}{C}} \tag{3.3}$$

where x_{dayi} is the total attendance on the day selected by agent i.

Finally, a local reward was defined. The local reward is a reward created to reward an agent for its contribution to the state of the system alone. This means that an agent only gets rewarded for its own actions, therefore the agent has no knowledge of how the other agents are acting and therefore can only act to enhance its own reward.

The local reward is defined by rewarding an agent by the following:

$$L_i(z) = x_{dayi} e^{\frac{-x_{dayi}}{C}} \tag{3.4}$$

3.1.2 Action Selection

For the multi-bar problem there is a small action space defined. The actions for this problem reflect the nights of the week. There are seven possible actions for each agent to select, and these actions reflect which night of the week the agent decides to attend the bar.

$A = \{0, 1, 2, 3, 4, 5, 6\}$ with each action representing a day of the week. Each agent selects actions from A using the ϵ-greedy strategy.

3.1.3 Applying MARL

Q-learning agents were tested on the multi-bar night problem using the global, difference and local reward structures, previously defined in Section 3.1.1 by S.Proper et al. [28]. The rewards for this problem are scalars and do not receive any further scalarisation.

The G reward steps are outlined below:

Algorithm 2: Multi night problem with G reward

```
initialise Q-values: Q(a)=0;
for episode = 1 in numOfEpisodes do
    for i = 1 in numOfAgents do
        choose a, using ε − greedy ;
    end
    evaluate global reward G ;
    for i = 1 in numOfAgents do
        set R = G ;
        update Q(a) value
    end
end
```

The L reward steps are outlined below:

Algorithm 3: Multi night problem with L reward

```
initialise Q-values: Q(a)=0;
for episode = 1 in numOfEpisodes do
    for i = 1 in numOfAgents do
        choose a, using ϵ − greedy ;
    end
    for i = 1 in numOfAgents do
        evaluate local reward L_i ;
    end
    for i = 1 in numOfAgents do
        set R_i = L_i ;
        update Q(a) value
    end
end
```

The D reward steps are outlined below:

Algorithm 4: Multi night problem with D reward

```
initialise Q-values: Q(a)=0;
for episode = 1 in numOfEpisodes do
    for i = 1 in numOfAgents do
        choose a, using ϵ − greedy ;
    end
    for i = 1 in numOfAgents do
        evaluate local reward D_i ;
    end
    for i = 1 in numOfAgents do
        set R_i = D_i ;
        update Q(a) value
    end
end
```

Algorithm 5: Multi night problem with G reward

```
initialise Q-values: Q(a)=0;
for episode = 1 in numOfEpisodes do
    for i = 1 in numOfAgents do
        choose a, using ϵ − greedy ;
    end
    evaluate global reward G ;
    reduce ϵ by multiplication with epsilonDecayRate ;
    reduce α by multiplication with alphaDecayRate ;
    for i = 1 in numOfAgents do
        set R = G ;
        update Q(a) value
    end
end
```

The L reward steps with decay rates are outlined below:

Algorithm 6: Multi night problem with L reward with decay rates

```
initialise Q-values: Q(a)=0;
for episode = 1 in numOfEpisodes do
    for i = 1 in numOfAgents do
        choose a, using ϵ − greedy ;
    end
    for i = 1 in numOfAgents do
        evaluate local reward L_i ;
        reduce ϵ_i by multiplication with epsilonDecayRate ;
        reduce α_i by multiplication with alphaDecayRate ;
    end
    for i = 1 in numOfAgents do
        set R_i = L_i ;
        update Q(a) value
    end
end
```

The D reward steps with decay rates applied are outlined below:

Algorithm 7: Multi night problem with D reward with decay rates

initialise Q-values: Q(a)=0;
for *episode = 1 in numOfEpisodes* **do**
    for *i = 1 in numOfAgents* **do**
        choose a, *using* $\epsilon - greedy$;
    end
    for *i = 1 in numOfAgents* **do**
        evaluate local reward D_i ;
        reduce ϵ_i *by multiplication with* epsilonDecayRate ;
        reduce α_i *by multiplication with* alphaDecayRate ;
    end
    for *i = 1 in numOfAgents* **do**
        set $R_i = D_i$;
        update $Q(a)$ *value*
    end
end

3.1.4 Experimental Procedure

S.Proper et al. set the parameters used in multi-night problem experiments [28]. In the multi-night problem, the agents learn for 4,000 episodes using 168 agents. Each episode consists of a single action by each agent, making this a single-shot problem. The learning parameters for all agents are set as follows: $\alpha = 0.1, \gamma = 1, \epsilon = 0.05$. S.Proper et al. [28] used the same values.

There are two experiments executed. The first experiments use an α decay rate set to 0.99 and an ϵ decay rate set to 0.99. The decay rate is multiplied by the initial ϵ and α values after each episode. The resulting value is set as the ϵ and α value, reducing the and α values over time, leading to fewer random actions taken by the greedy strategy. Applying a decay rate ensures each agent converges to a solution in fewer episodes than in Q-learning without decay rates.

The second set of experiments do not apply any decay rate and instead leave α and ϵ as their values from initialisation. The experiment begins with the same level of randomness in the ϵ greedy strategy. Generally, the experiment will not converge to a definite solution as early as an experiment using decay rates.

3.1.5 Experimental Results

All plots show a 10 point average based on the mean of 20 statistical runs, with each run containing 4000 episodes. The plots describe the overall system performance over each episode, measured as G(z), for each reward structure on the y-axis. G(z) is calculated as follows for each reward structure:

$$G(z) = \sum_{day=1}^{n} x_{day} e^{\frac{-x_{day}}{C}} \tag{3.5}$$

In plot A, Figure 3.1, the difference reward, D, converges to a stable policy at the same time the local reward, L, does. The global reward, G, converges to a more stable policy after 500 episodes.

L performs poorly in this scenario, as expected, due to L encouraging each individual agent to maximise its own reward. L acting to only maximise its own reward creates a more competitive environment. G and D perform much better, given their rewards are shaped to encourage cooperation, since an agent's actions affect the entire system.

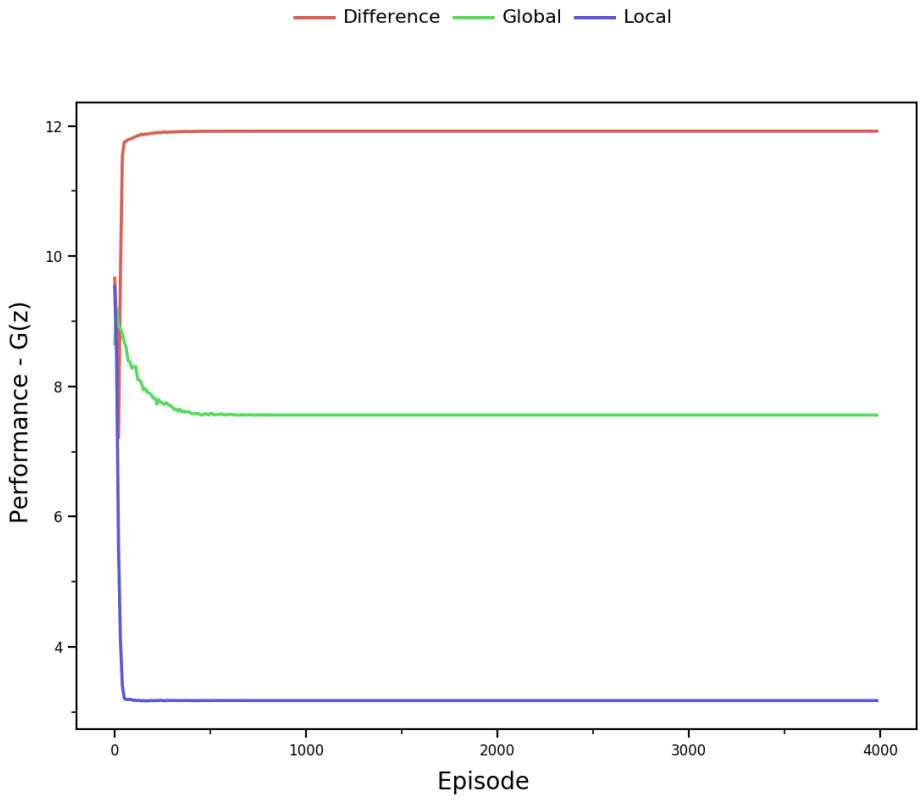

Figure 3.1: (a) Multi bar problem results with α and ϵ decay rates

Figure 3.1, shows quick convergence from all rewards. Convergence on each line is visible when the line becomes flat and smooth, showing the system has converged to a policy.

The policy may not be an optimal policy, but this the result of using decay rates.

In Figure 3.2, the D reward performs best out of the three reward structures tested, followed by the G reward, then the L reward. The main difference in Figure 3.2 is no decay rates have been applied to any of the reward structures, hence there are more jumps and dips visible in each line on the plot.

The G reward shows the most change when compared to the G reward in Figure 3.1. In Figure 3.1, the G reward performs better from Episode 0 to Episode 1000 when it eventually converges. In Figure 3.2 the G reward performs better as the episodes increase. Since the ϵ value in this plot does not decay, the agent has more chance of taking a random action, potentially allowing the agent to learn new solutions over time.

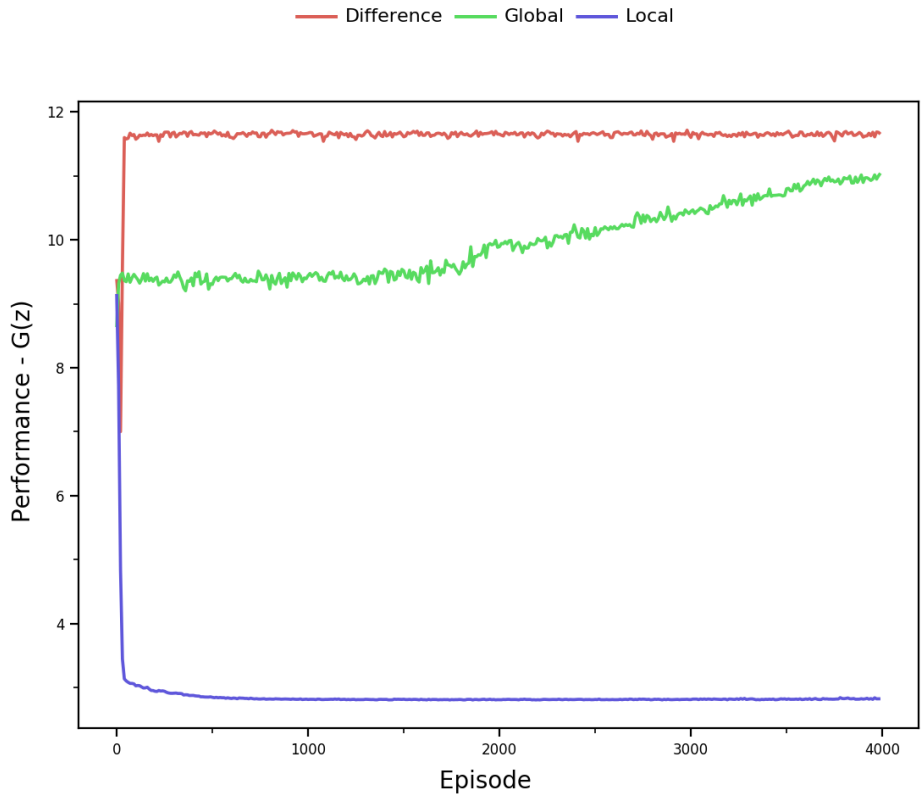

Figure 3.2: (b) Multi bar problem results with no decay rates

When looking at the D reward, it is clear agents can learn a near optimal solution very quickly. The L reward can learn a solution quickly, but the learned solution is not an optimal solution. In both experiments the D reward learns a near optimal solution in less than 500 episodes. In Figure 3.2, the D reward learns a good solution in fewer than 500 episodes and refines the solution throughout the remaining episodes while keeping a performance of less than 12.

The G reward can also learn good policies, but not as quickly as D learns good policies. In Figure 3.2, as the episodes increase, the G reward's performance also increases.

In both experiments it is apparent the difference reward, D, performs the best out of the benchmark reward structures; the global reward, G, performs second best and the local reward, L, performs the worst.

The plots of the learning curves presented indicate the learning speeds of all three reward structures.

3.1.6 Discussion

This study evaluated the effectiveness of three reward structures widely used in RL and applied them to a multi-agent domain. This experimental work shows and benchmarks how each different reward structure behaves and learns. This work showed how the D reward can increase learning speeds when compared to agents who learn with the G reward. This work has also highlight the shortcomings of the L reward and how the L reward's greedy nature significantly reduces an agent's ability to learn.

Chapter 3 shows in a stateless multi-agent single-objective environment the D reward can learn a good solution quickly, while the G can also learn a good solution. However, the G reward needs significantly more time to learn. The L reward struggles to learn a good solution over time.

The state and action space required for this environment is small compared to real-world problems. Applying the D reward, G reward and L reward to this problem domain provides a much-needed expectation benchmark when implementing these reward structures in a multi-objective multi-agent RL environment. The results in Chapter 3 can be used as a reference for future problems to determine expected behaviour in an environment with a significantly larger state and action space.

Using decay rates throughout experiments for the multi-night problem provides a benchmark for how each reward structure behaves while the agents α and ϵ decay after each episode, which impacts the solution on which each agent converges. The decay rates impact how quickly an agent converges on a solution, while also heavily impacting the solution on which the agent converges. Using this method ensures quick convergence but compromises the quality of the learned solution, which is apparent when comparing the results of the G reward in Figures 3.1 and 3.2.

Chapter 4

Multi-Agent Multi-Objective Study

The typical reward structures and their expected performance for a multi-agent RL problem have been established in a multi-agent RL benchmark problem in Chapter 3. Chapter 4 will attempt to expand the reward structures presented in Chapter 3 to a multi-agent multi-objective RL problem domain. The problem chosen to apply these techniques is the Dynamic Economic Emissions Dispatch (DEED) problem. The DEED problem is an established problem domain that was proposed by Basu [4] and since has been studied using Genetic Algorithms by Basu [4], Particle Swarm Optimisation by Mason [22] and Reinforcement Learning by Mannion [18].

4.1 Dynamic Economic Emissions Dispatch (DEED)

As outlined in Section 2.6 the DEED problem consists of a series of electricity generators that must operate in order to meet a specified demand to ensure a population has sufficient electricity. Mannion notes, " Generator scheduling is a complex task due to many different factors including: unpredictable fluctuations in demand: power loss within the transmission lines; and varying efficiency level, power limits and ramp limits among generators in the same plant " [18] [4].

The constant fluctuations in energy and fuel costs raise the need to operate each generator in a cost effective manner. It is also desirable to minimise the harmful emissions produced from each generator during operation. The overall goal of DEED is to minimise both cost and emissions, but the objectives are in opposition to each other, which makes this a multi-objective problem.

The traditional format of DEED introduced by Basu [4] was presented in Section 2.6, this describes the DEED problem before it is reformatted into a MOSG. Mannion reformatted the problem to become a MOSG [18]. DEED as a MOSG was introduced in Section 2.7. Section 2.7 outlines the reward structures and equations to formulate the DEED problem.

4.1.1 Applying MARL

Multiple Q-learning agents were tested in the DEED MOSG defined in Section 2.7. DEED contains 9 generators that need to be controlled with one additional generator, known as the slack generator. The 9 generators are each controlled by an agent, while the slack generator's energy generation is dependent on the sum of the generators controlled by each agent and the power demand. The equations to calculate the emissions and cost for the slack generator are defined in equations 2.20 and 2.22.

Using the three reward structures defined in Section 3.1, it is possible to measure the performance of the agents in this problem domain. The local, L, global G, and difference rewards will be used in this problem domain. The three reward structures mentioned will be used to minimise the cost, emissions and penalty violations of each agent.

The equation for cost is defined in equation 2.13. The equation for emissions is defined in equation 2.14. The penalty violations are calculated by equation 2.31, while also violations are subject to the constraints defined in equations 2.32 and 2.33.

For DEED the G reward is defined by taking the sum of all generator costs, emissions and violations. This reward is given to all agents in the system per episode or hour, m.

$$G(z) = f_c^G(m) + f_e^G(m) + f_p^G(m) \tag{4.1}$$

This reward is then given to each of the 9 agents.

Algorithm 8: DEED problem with G reward

```
initialise Q-values: Q(s,a)=0;
for episode = 1 in numOfEpisodes do
    for hour = 1 in numOfHours do
        for agent = 1 in numAgents do
            choose a, using ε − greedy ;
            set s = agents state ;
        end
        calculate G reward ;
        for agent = 1 in numAgents do
            get agents nextState ;
            set r = G reward ;
            set s' = agents next state ;
            update Q(s, a) with r and s'
        end
    end
end
```

The D reward is defined by taking the sum of all generator costs, emissions and violations. Once calculated, the agent's contribution to cost, emissions and violations is subtracted from the sum of each (cost, emissions and penalty violations). The D reward for each agent can be defined as the following:

$$D_i(z) = G(z) - G(z - i) \tag{4.2}$$

G(z-i) is calculated using three counterfactuals for cost, emissions and violations [21]. These counterfactuals for cost, emissions and violations for an agent i are calculated by assuming the agent did not choose a new power output value in the timestep. This means the power output of generator n did not change [21].

The counterfactual for cost is calculated by the following: .

$$f_c^{G(z-i)} = \sum_{\substack{n=1 \\ n \neq i}}^{N} f_c^L(n, m) + f_c^L(i, m-1) \tag{4.3}$$

The counterfactual for emissions is calculated by the following: .

$$f_e^{G(z-i)} = \sum_{\substack{n=1 \\ n \neq i}}^{N} f_e^L(n, m) + f_e^L(i, m-1) \tag{4.4}$$

The output of the counterfactual version of $f_p^{G(z-i)}$ of the penalty function f_p^G is calculated using equation 2.31, with the term $P_{1m}^{(z-i)}$, which is used instead of P_{1m} in equations 2.32 and 2.33. $P_{1m}^{(z-i)}$ is calculated by the following:

$$P_{1m}^{(z-i)} = P_{Dm} + P_{Lm} - \sum_{\substack{n=2 \\ n \neq i}}^{N} P_{nm} - P_{i(m-1)} \tag{4.5}$$

A reward is generated for each individual agent and each agent receives its corresponding reward.

Algorithm 9: DEED problem with D reward

```
initialise Q-values: Q(s,a)=0;
for episode = 1 in numOfEpisodes do
    for hour = 1 in numOfHours do
        for agent = 1 in numAgents do
            choose a, using ε − greedy ;
            set s = agents state ;
        end
        for agent = 1 in numAgents do
            calculate D reward for each agent ;
        end
        for agent = 1 in numAgents do
            get agents nextState ;
            set r = D reward ;
            set s' = agents next state ;
            update Q(s, a) with r and s'
        end
    end
end
```

The L reward, also written as L_i is based on the utility of the system that agent i can observe directly. The L reward is generated for each agent based on the agent's contribution to the system. An agent is rewarded for its own actions, which can cause agents to act in their own interest and ignore global system performance.

The L reward only considers the agent's local cost and emissions.

$$L_i = f_c^L(i, m) + f_e^L(i, m) \tag{4.6}$$

Where i is the corresponding agent and m is the hour. There are no local penalty violations. The L reward is subject to only cost and emissions.

Algorithm 10: DEED problem with L reward

```
1  initialise Q-values: Q(s,a)=0;
2  for episode = 1 in numOfEpisodes do
3      for hour = 1 in numOfHours do
4          for agent = 1 in numAgents do
5              choose a, using ε − greedy ;
6              set s = agents state ;
7          end
8          for agent = 1 in numAgents do
9              calculate L reward for each agent ;
10         end
11         for agent = 1 in numAgents do
12             get agents nextState ;
13             set r = L reward ;
14             set s' = agents next state ;
15             update Q(s, a) with r and s'
16         end
17     end
18 end
```

4.1.2 Scalarisation Of Objectives

Mannion et al. defined the reward structures for each objective. The cost, emissions and violations are combined for each reward structure to which they apply. The reward signals for L_o, G_o and D_o are combined for each objective $o \in O$. Each of the reward structures is scalarised using two types of scalarisation, linear scalarisation (+) and hypervolume scalarisation (λ) [21]. The agents receive one of the scalarised reward signals while learning: $D(+)$, $D(\lambda)$, $L(+)$, $L(\lambda)$, $G(+)$ or $G(\lambda)$.

The reward structure for linear scalarisation is defined by the following equation:

$$R_{+} = -\sum_{o=1}^{O} w_o f_o \tag{4.7}$$

The reward structure for hypervolume scalarisation is defined by the following equation:

$$R_{\lambda} = -\prod_{o=1}^{O} f_o \tag{4.8}$$

where w_o is the objective weight, f_o is the objective function (global, difference or local, where appropriate) and the generic R is replaced by L, G or D. The objective weight used as $w_c = 0.225$, $w_e = 0.275$ and $w_p = 0.5$. Mannion et al. [21] chose those values by completing parameter sweeps to maintain a good balance between objectives while learning. In the case of the L reward $O = 2$, where there is no local penalty function. For D and G, $O = 3$. All rewards are assigned as negatives, as all objectives must be minimised [21].

4.1.3 Action Selection

Each agent has a large action space to explore, as already outlined in equation 2.29 and equation 2.30. Mannion found in initial experimental work, using these action parameters, the quality of policies learned was highly variable [21].

Mannion et al. [21] attributed high variability to the action space for each agent having a different size. For example, using a discretisation level of 1MW, the smallest action space has 46 actions while the largest has 321 actions when using the generator operating limits from Basu's work [4].

To overcome the high variability of the policies learned, Mannion et al. created an abstraction A^* of the action space. In A^* each agent has a set of 101 possible actions [21], $A^* = \{0, 1, 2, ..., 100\}$. Each action represents a different percentage value of the operating range of the generator.

The power output from generator n for action a_i^* is calculated as:

$$P_n = P_n^{min} + a_i^*(\frac{P_n^{max} - P_n^{min}}{100}) \tag{4.9}$$

where $i = n$. Using the action abstraction A^* ensures each agent is still subject to the ramp limits outlined in equations 2.17, 2.18 and 2.19. Each agent selects an action from A^* using an ϵ - greedy strategy.

4.1.4 Experimental Procedure

Experiments were conducted on two variations of the DEED MOSG. In the normal version of the problem, defined by Mannion et al. [21], the agents learn for 20,000 episodes with each episode having 24 hours. The second version of the DEED MOSG also runs for 20,000 episodes, but after 10,000 episodes the demand profile changes from the standard demand profile used by Basu [4], Mason [22] and Mannion [21] to a demand profile that represents a sine wave. The purpose of the second set of experiments is to test the robustness of the agents' abilities to respond to disturbances in the DEED domain.

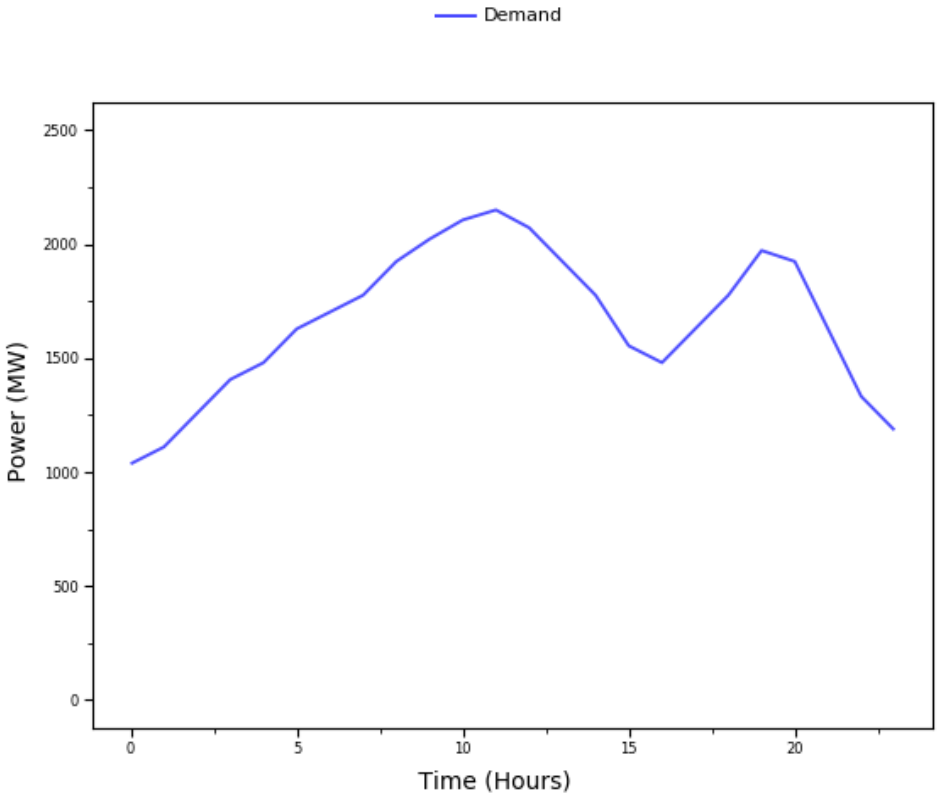

Figure 4.1: Power demand over 24 hours

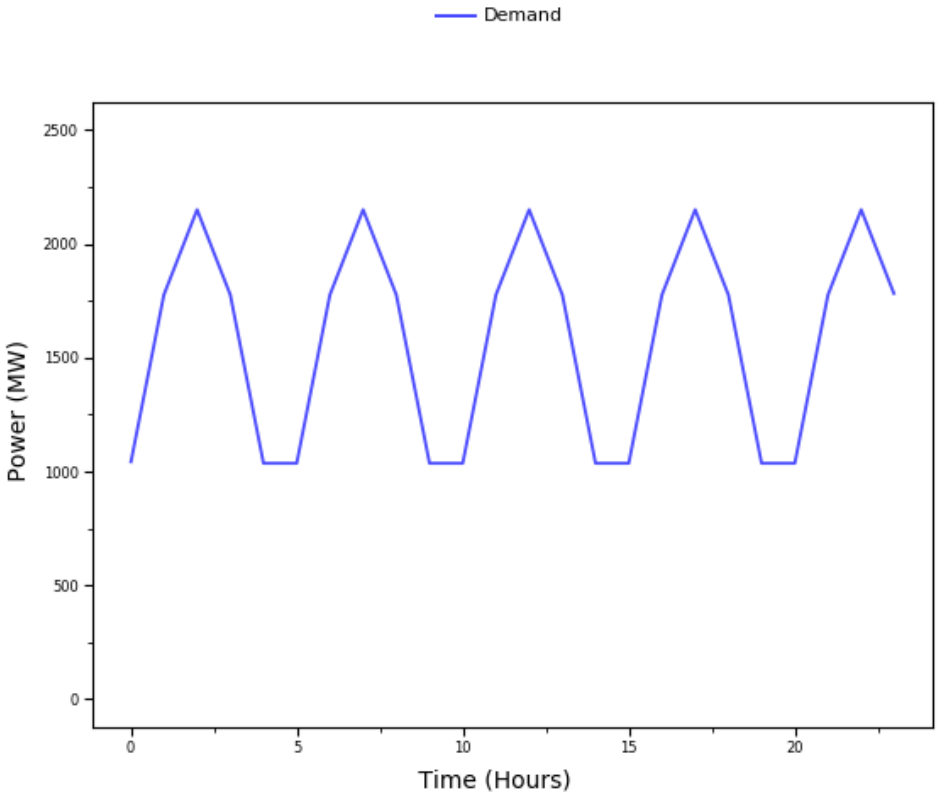

Figure 4.2: Power demand with sine wave

Figure 4.1 displays the demand profile used by Mannion [21], Basu [4] and Mason [22]. Figure 4.2 shows the sine wave demand profile which is passed to the agents after 10,000 episodes. An exact sine wave cannot be generated for this problem domain. Due to the power limitations of the generators and the restrictions applied to the generators via ramp limits, the actual power demand does not perfectly resemble a sine wave.

The learning parameters for all agents are as follows: $\alpha = 0.10$, $\gamma = 0.75$ and $\epsilon = 0.05$. Mannion et al. [21] set the learning parameters using parameter sweeps.

4.1.5 Experimental Results

The first set of results presented will be for the standard version of the DEED problem. All plots show a 200 point moving average across the 20 statistical runs. All claims of statistical significance are supported by two-tailed t-tests assuming unequal variances, with $p = 0.05$ selected as the threshold for significance.

In Section 3.1 the results showed the difference (D), global (G) and the local (L) reward structures and how each reward performed on a basic benchmark problem. The difference reward performed the best, while the global reward did not perform as well as the difference reward, and the local reward performed the worst of the three. Using the results presented in Section 3.1 it is expected the difference reward will generate the best results for the DEED MOSG problem, with the global reward generating the second best results, and the local will generate the worst results.

According to Mannion, " The plots of learning curves for the cost objective give an indication of the relative learning speeds and stability of performance of each of the approaches tested " [18].

As expected in figure 4.3 and figure 4.4, the L reward performs poorly. As already mentioned in Section 3.1 the L reward encourages agents to act greedily. Agents are not concerned with the state of the system; they are only concerned with maximising their own reward.

D and G rewards both learn good solutions in the DEED problem domain. With linear and hypervolume scalarisations, D and G learn good policies immediately. G learning a good policy immediately slightly contradicts what Mannion et al. [21] and Mannion [18] have reported. The difference in results could be due to a different degree of discretisation when defining the problem environment.

The results obtained in Section 4.1.5 confirm the outcomes from Chapter 3.

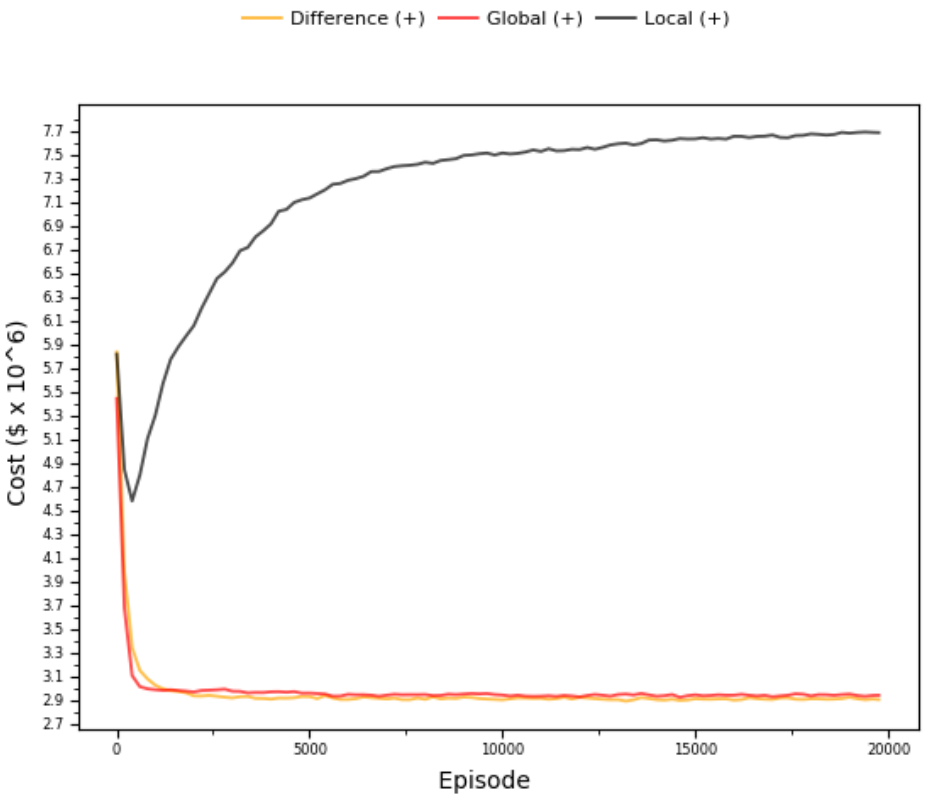

Figure 4.3: Cost with linear scalarisation

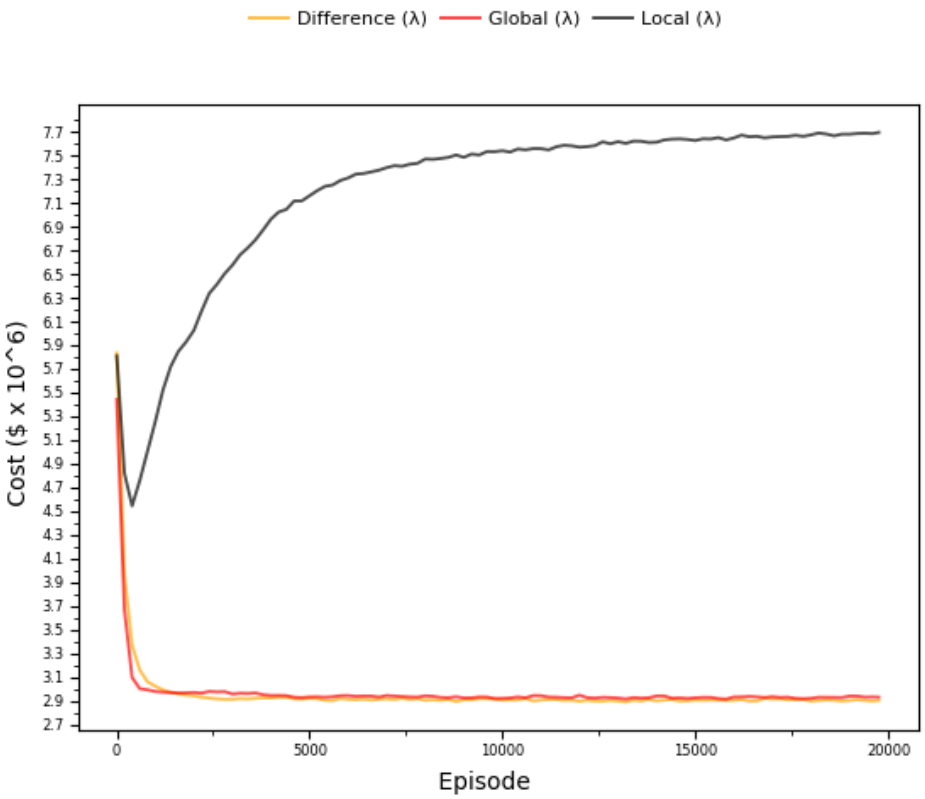

Figure 4.4: Cost with hypervolume scalarisation

	Cost ($ x 10^6)	Emissions (lb x 10^5)
L(λ) (*Mannion*)	4.1149	17.6606
L(+) (Mannion)	4.1127	28.8266
L(λ)	7.6970	49.9000
L(+)	7.6826	197.0350
G(λ) (*Mannion*)	2.7607	3.9788
G(+) (Mannion)	2.7647	3.9098
G(λ)	2.9296	7.9937
G(+)	2.99787	12.0287
D(λ) (*Mannion*)	2.6748	3.8980
D(+) (Mannion)	2.6641	3.3255
D(λ)	2.8870	6.4354
D(+)	2.9398	17.3012
PSO-AWL (Mason)	2.5463	2.9455
NSGA-II (Basu)	2.5226	3.0994

Table 4.1: DEED average final performance

The differences in the mean final performance of $D(+)$ and $G(+)$ were not found to be statistically significant for cost ($p = 0.4697272$), emissions ($p = 0.521542775$) or penalty violations ($p = 0.340942122$). There is no difference in the mean final performances of $D(\lambda)$ and $G(\lambda)$ in regards to cost ($p = 0.257168$) and emissions ($p = 0.130995$). The D(λ) penalty violations are statistically better than the G(λ) ($p = 3.24332x10^{-6}$) .

The differences in the mean final performance of D(+) and G(+) for cost and emissions differ from the results presented by Mannion et al. [21]. Mannion et al. presented results showing the $D(+)$ reward is statistically better than the $G(+)$ reward for cost and emissions. Mannion et al. did not present any results for penalty violations.

Mannion et al. also presented the $D(\lambda)$ reward as statistically superior to the $G(\lambda)$ reward. The differences in the results presented in Section 4.1.5 and the results Mannion et al. presented could be explained by the level of discretisation used. Mannion et al. seems to have used a different level of discretisation, allowing for results closer to an optimal solution. It might be possible to obtain more accurate results for each reward structure upon further investigation into the configuration of the DEED environment, in tandem with performing more statistical runs.

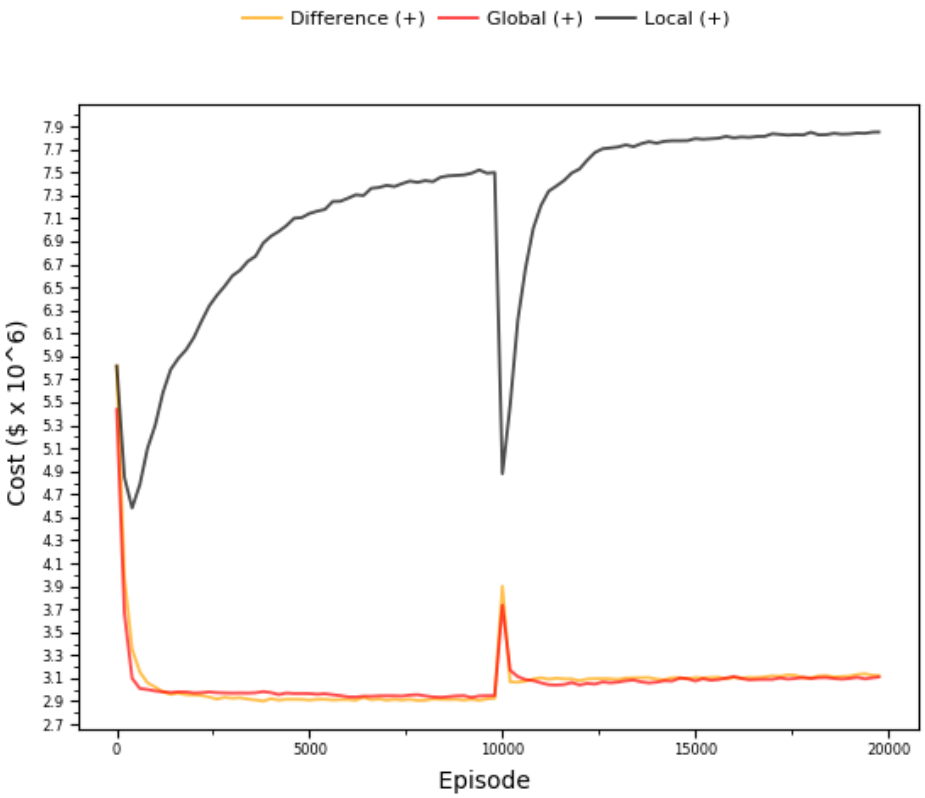

Figure 4.5: Cost with linear scalarisation with sine wave

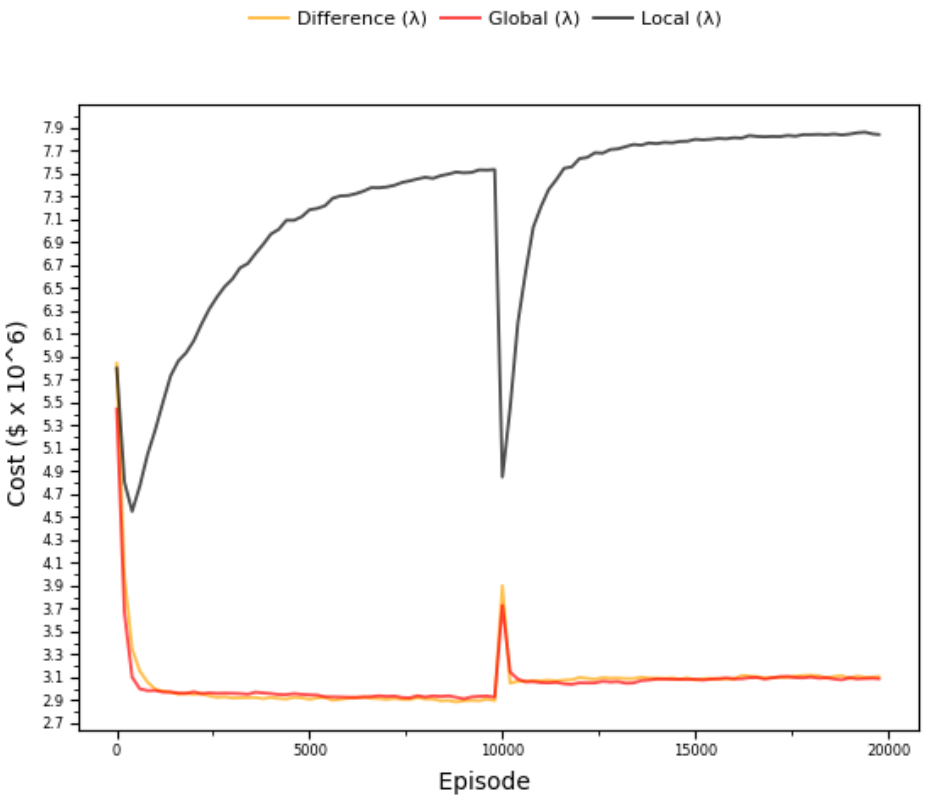

Figure 4.6: Cost with hypervolume scalarisation with sine wave

Figure 4.5 and 4.6 show the results from the second set of experiments, where after 10,000 episodes a sine wave power demand is passed to the agents. The introduction of the sine wave after 10,000 episodes is a disturbance to each agent and each agent must then attempt to learn the new power demand. The goal of this experiment is to see if the agents are robust enough to adapt to a change in their environment. The change in power demand is displayed in Figure 4.2 and is a representation of a sine wave, given the constraints of the environment.

In Figure 4.5 the L reward performs poorly before and after the sine wave is introduced. Both G and D rewards learn a good solution almost immediately. At 10,000 episodes, a large spike in the cost can be seen and, almost immediately, the G and D rewards learn a good solution.

In Figure 4.6 the L reward performs poorly. Both G and D rewards learn a good solution almost immediately. At 10,000 episodes, once the sine wave power demand has been introduced, a large disturbance in the cost can be seen. The G and D reward learn a good solution instantly, demonstrating the robustness of both of these reward structures.

In the second set of experiments, with another power demand being introduced, the robustness of the G and D rewards is visible in both Figure 4.5 and Figure 4.6. The ability of both the G and D reward structures to learn and adapt to unforeseen issues reaffirms how applicable these reward structures are to real world problems.

4.1.6 Discussion

Chapter 4 explores a multi-objective multi-agent real world problem domain. The DEED domain was reformatted, as per Mannion et al. [21], to allow for the application of multi-agent multi-objective reinforcement learning. The goal of Chapter 4 was to reproduce the problem domain as per Mannion et al. and to replicate the performance from the difference, local and global reward structures while analysing the robustness of the reward structures. Chapter 4 evaluates the reward structures using two different scalarisation techniques, hypervolume and linear.

The results in Section 4.1.5 show the local reward performing poorly, while the global and the difference rewards perform well. Both difference and global rewards learn good policies very quickly, while both scalarisation techniques seem to have little effect on the performance of the reward structures. This chapter's findings differ slightly from Mannion et al.'s [21] findings.

Mannion et al. [21] reported the difference reward performing much better than the global and local rewards. Mannion et al. [21] used a different degree of scalarisation when creating the agent environment, which could explain why the results are not exactly similar to results Mannion et al. [21] present. The difference, D, and global, G, reward structures with linear scalarisation have produced results comparable to other previously published attempts to solve the DEED problem domain, including NSGA-II (Basu) [4], Particle Swarm Optimisation (Mason) [22] and MARL (Mannion) [21] [18].

In future work it would be worthwhile to optimise the level of discretisation used in the DEED problem domain. It could be possible to enhance the results received for all objectives after more experimentation with the levels of discretisation. In future work, it would be worth experimenting with applying Deep Q learning (DQN) to the DEED problem domain. DQN uses function approximation and it could be beneficial to see how agents using DQN can adapt to unseen changes in their environment.

Chapter 5

Multi-Agent Multi-Objective Study With Thresholded Lexicographic Ordering

In Chapter 4 a multi-agent multi-objective problem domain, DEED, was defined and explored through the difference, global and local reward structures. Thresholded Lexicographic Ordering (TLO) was first hypothesised by Gabor et al. [11] and was first implemented in a single-agent multi-objective environment by Vamplew et al. [34]. Chapter 5 explores TLO and its applicability to a multi-agent multi-objective environment, such as DEED. The goal of Chapter 5 is to implement multiple TLO RL agents in the DEED environment.

5.1 DEED with Thresholded Lexicographic Ordering

The version of DEED used is the same version used in Mannion et al. [21] and in Chapter 4. To ensure that TLO agents can be deployed, adjustments have been made to how the rewards are structured.

The version of DEED used still consists of a series of electricity generators that must operate in order to meet a specified demand to ensure a population has sufficient electricity. It was shown in Chapter 4 that MARL is a useful pardigram to apply to this problem domain, as it is more robust when compared to traditional methods like PSO [4]. Since the demand placed on electrical generators for a population can fluctuate highly over periods of time within a season, MARL is an ideal pardigram to use on this problem domain.

As mentioned in Chapter 4, the overall goal of DEED MOSG is to minimise cost, emissions and penalty violations. Using TLO agents, it should be possible to minimise each objective individually.

5.1.1 Reward Shaping

Reward shaping for TLQ-learning is different from the approach used in the DEED MOSG. In Chapter 4 the rewards were simply defined as the sum of all objectives (cost, emissions and penalty violations). Each reward structure was simply a scalar. For TLQ-learning the objectives will be minimised independently of each other. In TLQ-learning the rewards need to be represented as a vector. The rewards are represented as a $3X1$ vector, where each element in the vector is located based on the order in which it is to be minimised. The reward R can be defined as the following:

$$R = [objective_1, objective_2, ..., objective_n] \tag{5.1}$$

Representing the reward as a vector leads to the Q table for each agent being reconfigured. Normally the size of a Q table is defined by $Q(s, a)$, where s is the number of states and a is the number of actions. For TLQ-learning the size of a Q table is defined by $Q(s, a, o)$, where s is the number of states, a is the number of actions, and o is the number of objectives.

5.1.2 Thresholds

Thresholds in TLQ-learning are defined as the goal value for each objective. A threshold is specified for each objective, with the exception of the final objective . The system attempts to reach the specified threshold for each objective that has a threshold. The last objective does not receive a threshold. The system attempts to minimise the final objective to the best of its ability. Threshold, T, is stored as a vector and can be defined as follows:

$$T = [goalValue_1, goalValue_2, ..., goalValue_{n-1}] \tag{5.2}$$

where n is the number of objectives and $goalValue$ is the desired outcome for each objective.

For example, in the Deep Sea Treasure problem domain there are two objectives, treasure and time. If the goal is to have a treasure of 24, the threshold vector, T, would be defined as $T = [24]$ and time would be minimised to the best of the system's ability.

5.1.3 Action Selection

The action space for TLQ-learning still faces the same issues outlined in Section 4.1.3. TLQ-learning uses the same action abstraction as Mannion et al. defined [21], A^*. Each agent has 101 possible actions, $A^* = \{0, 1, 2, ..., 100\}$, where each action represents a different percentage value of operating range for each generator.

A different algorithm is used to select each action. In Q-learning the action with the max Q-value is considered the best action to take. In TLQ-learning there is a specific algorithm designated for action selection. The algorithm will be presented in the next section.

5.1.4 Multi-Agent TLO Action Selection

TLQ-learning has only ever been applied to a multi-objective single-agent problem domain. Therefore, a novel approach to maintaining thresholds must be used when expanding to multi-agent problems. In order to ensure each agent is able have its own fair interaction with the thresholds, a subtraction method is used. This method ensures each agent knows their desired action falls within the boundaries of the threshold. Each agent uses the TLO action selection method, selecting actions within the thresholds.

Multiple actions will be under the thresholds. Once multiple actions fall within the thresholds, agents must use another method to determine which action to take; this will be discussed more later. Once the agent has determined which action to take, the action's reward will be subtracted from the overall threshold for that objective. Each agent will repeat this process. In an ideal system the threshold will reach 0 or reach a positive state for each objective after each agent has been able to perform an action.

The action selection for TLQ-learning makes use of a comparative algorithm. An agent will see if its desired action is less than the threshold for the first objective, if not, the action will be discounted and the agent will test the next action. If the next action is less than the threshold for the first objective, the action will be compared to the threshold for the second objective. If the action is not less than the second objective's threshold, the action will be kept until a better action presents itself. If no better action presents itself, the agent will select the action saved. This process repeats over all actions and objectives, except for the last objective, which is optimised to the best of the system's ability.

Both the global and difference reward structures select actions using the described method. This method often leads to agents discovering multiple optimal actions, which is known as a tie-breaker scenario. In a tie-breaker scenario, the system considers each action to be as optimal as the next. The main difference between the global and difference reward structures is how the agent selects its action in a tie-breaker scenario.

The algorithm for TLQ-learning action selection is outlined below; this algorithm is also outlined in Section 2.8:

- Let n denote the number of objectives, labeled from $1..n$
- Let A denote the set of available actions
- Let C_j be the threshold value (minimum acceptable value) for objective j, as defined by the constraint for that objective (note: objective n will be unconstrained, hence $C_n = +\infty$)

$$CQ_{s,a,j} \leftarrow min(Q_{s,a,j}, C_j) \tag{5.3}$$

In state s, the greedy action a' is selected such that superior($CQ_{s,a'}$,$CQ_{s,a}$, 1) is true $\forall\ a \in A$ where superior($CQ_{s,a'}$, $CQ_{s,a}, i$) is recursively defined as [34]:

Algorithm 11: TLQ-learning Action Selection

```
if CQ_{s,a',i} > CQ_{s,a,i} then
    return true;
end
if CQ_{s,a',i} = CQ_{s,a,i} then
    if i = n then
        return true;
    else
        return superior(CQ_{s,a'} , CQ_{s,a}, i + 1);
    end
else
    return false;
end
```

In a TLQ-learning tie-breaker scenario the global reward simply selects one action, at random, from the list of potential optimal actions. The global reward relies on, when selecting a random action in a tie-breaker scenario, each action already being below a threshold. Therefore, selecting an action at random should only have a positive impact on the system.

5.1.5 Applying MARL

Multiple TLQ-learning agents were tested in the DEED MOSG defined in Chapter 4. DEED contains 9 generators that need to be controlled, with one additional generator that is known as the slack generator. The 9 generators are each controlled by an agent. The slack generator's energy output is dependent on the sum of the generators controlled by each agent and the power demand. The experiment deploys 9 TLQ-learning agents to control each generator.

To measure the performance of each TLQ-learning agent in this problem domain, two reward structures are defined. The TLO Global reward and the TLO Difference reward are used to measure the agent's performance.

The rewards for TLQ-learning differ from what was presented for standard Q-learning in the DEED problem domain. Rewards are defined as vectors that contain cost, emissions and penalty violations. The global, $TLO\ G$, reward can be reformatted to the following:

$$TLO\ G(z) = [f_c^G(m), f_e^G(m), f_p^G(m)] \tag{5.4}$$

It is important to note the reward shape is subject to change in respect to which objective is being optimised first. In the equation above, cost is placed first in the vector and will be optimised first in the TLO pardigram. For example, if emissions need to be optimised first, the reward should be designed to place the emissions value in the first position in the $TLO\ G$ vector. The rest of the objectives will be located in the vector based on the order of their desired optimisation.

Algorithm 12: TLO DEED problem with G reward

```
initialise Q-values: Q(s,a, j)=0;
initialise Thresholds ;
for episode = 1 in numOfEpisodes do
    for hour = 1 in numOfHours do
        for agent = 1 in numAgents do
            choose a, using ϵ − greedy and TLO Action Selection Algorithm;
            set s = agents state ;
            update Thresholds based on action a
        end
        calculate G reward vector for each agent ;
        for agent = 1 in numAgents do
            get agents nextState ;
            set r = G reward vector ;
            set s′ = agents next state ;
            update Q(s, a, j) with r and s′
        end
    end
end
```

The TLO difference reward, $TLO\ D$, takes a different approach to handling the tie-breaker scenarios. When an agent operating with the TLO reward enters a tie-breaker scenario, the agent does not take a random action. Instead, the agent references the standard Q-learning difference reward Q-table. The agent gathers all optimal actions and checks the difference reward Q-learning Q table for each of these actions. The action that returns the highest Q-value is deemed the most optimal action to take in the tie-breaker scenario. The only difference between the $TLO\ D$ and the $TLO\ G$ reward structures is how the agent selects an action in a tie-breaker scenario. The difference reward, $TLO\ D$, can be defined as the following:

$$TLOD(z) = [f_c^G(m), f_e^G(m), f_p^G(m)] \tag{5.5}$$

As mentioned earlier, the reward structure is subject to change based on the desired ordering of objectives.

Algorithm 13: TLO DEED problem with D reward

```
initialise Q-values: Q(s,a, j)=0;
initialise Thresholds ;
for episode = 1 in numOfEpisodes do
    for hour = 1 in numOfHours do
        for agent = 1 in numAgents do
            choose a, using ε − greedy and TLO Action Selection Algorithm;
            set s = agents state ;
            update Thresholds based on action a
        end
        calculate D reward vector for each agent ;
        for agent = 1 in numAgents do
            get agents nextState ;
            set r = D reward vector ;
            set s' = agents next state ;
            update Q(s, a, j) with r and s'
        end
    end
end
```

5.1.6 Scalarisation Of Objectives

Mannion et al. [21] defined a set of reward structures and scalarisations for the DEED problem domain. For TLO DEED the linear scalarisation technique is not applied. The hypervolume scalarisation is also not applied. No scalarisation techniques are applied in this implementation of DEED. Using scalarisation with TLQ-learning agents adds complexity to the problem domain. If the reward structures are scalarised, it becomes difficult for each agent to have a fair interaction with the thresholds. With no scalarisation, the reward structures for TLQ-learning agents can be defined by the following equation:

$$R = [-o_1, -o_2, ..., -o_n] \tag{5.6}$$

where o is an objective (cost, emissions, penalty violations) and n is the number of objectives. For all reward structures in this domain, $n = 3$. There is potential to develop new scalarisations techniques and apply existing scalarisation techniques to use with TLQ-learning.

5.1.7 Experimental Procedure

Experiments were conducted on the variation of the DEED MOSG presented by Mannion et al. [21]. In the DEED domain the TLQ-learning agents learn for 20,000 episodes with each episode having 24 hours. There is only one experiment used in this section and the purpose of this experiment is to examine how TLQ-learning agents behave in a multi-agent multi-objective environment. This is the first time TLQ-learning has been used in a MARL environment.

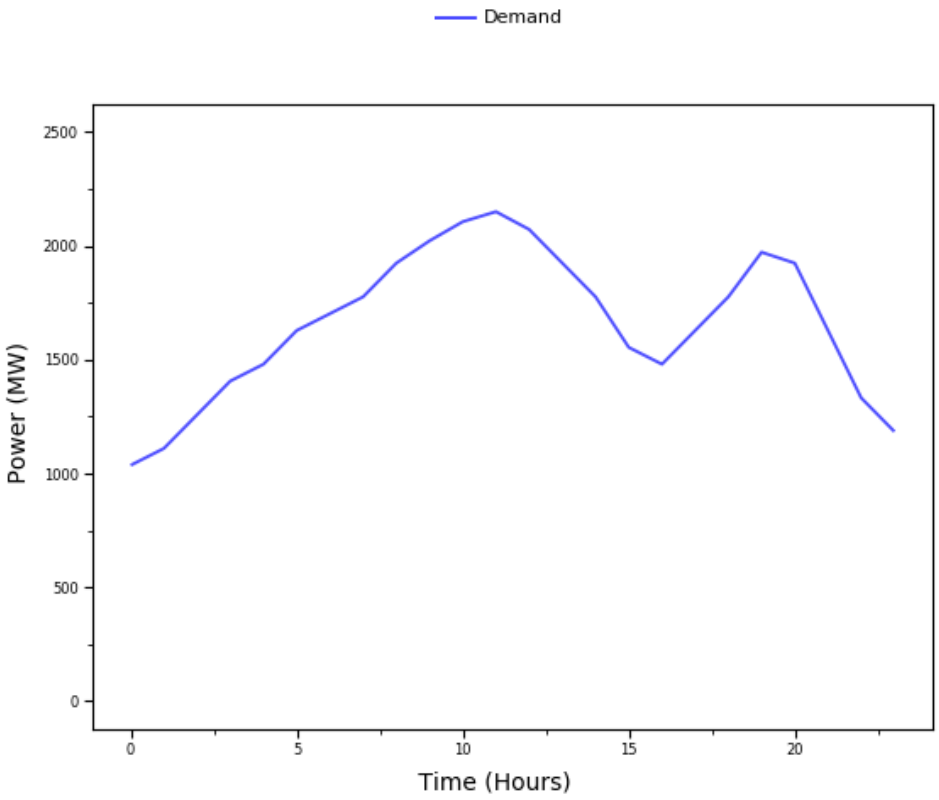

Figure 5.1: Power demand over 24 hours

Figure 5.1 displays the demand profile used by Mannion [21], Basu [4] and Mason [22]. The learning parameters for all agents are as follows: $\alpha = 0.10$, $\gamma = 1.0$ and $\epsilon = 0.05$. The γ was set from Vamplew et al. [34]; this is a requirement for the TLQ-learning agents to learn and to be able to interact with the thresholds and the Q table. Mannion et al. [21] set the α and ϵ values through parameter sweeps.

During parameter sweeps performed for the TLQ-learning agents, optimal performance

was achieved by optimising the objectives in the following order: penalty violations, cost, emissions. To ensure optimal performance the threshold vector, T, was defined as follows:

$$T = [-10000000, -2800000] \tag{5.7}$$

The values specified are for penalty violations and cost. Using a goal value larger than -10000000 for penalty violations achieved sub-optimal solutions. Using a goal value larger than -2800000 also led to sub-optimal solutions. The agents will minimise emissions to the best of their ability. The values specified in vector T were found to be optimal during parameter sweeps.

5.1.8 Experimental Results

All plots show a 200 point moving average across the 20 statistical runs conducted. All claims of statistical significance are supported by two-tailed t-tests assuming unequal variances, with $p = 0.05$ selected as the threshold for significance.

Before analysing the plots generated, it is important to note the results from Section 4.1.5 are included with the results of this section. The goal of this section is to compare the standard Q-learning difference, local and global reward structures' performances with TLQ-learning reward structures. The reward structures presented in each figure in this section contain the following: difference, global, local, TLO global and TLO difference.

For the TLQ-learning reward structures, the experiments were set up to minimise the objectives in the following order: penalty violations, cost, emissions. The plots generated include both penalty violations and cost.

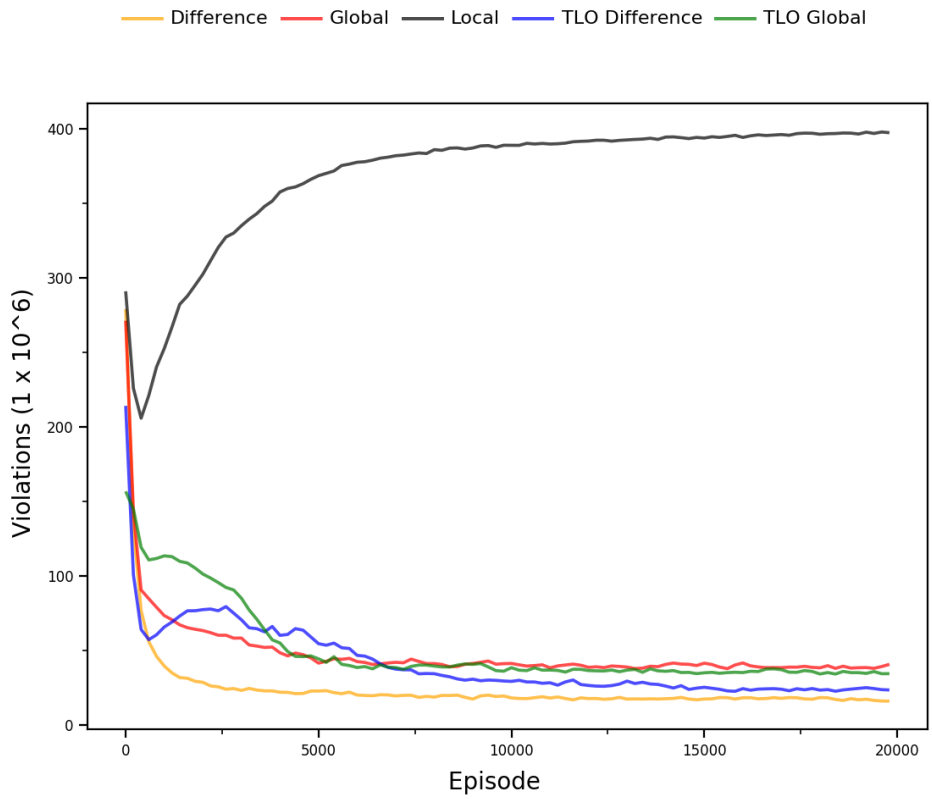

Figure 5.2: Penalty Violations with TLQ-learning

Figure 5.2 presents the graph for penalty violations. Local, difference and global Q-learning agents are included to compare the performances between Q-learning and TLQ-learning agents. As seen in previous plots, the local reward performs poorly, which is expected. The standard global and difference rewards quickly learn good solutions. The new reward structures of TLO global and TLO difference perform well and learn good solutions. In terms of penalty violations, the TLO global reward performs better than the standard global reward. The TLO difference reward does not outperform the standard difference reward. The TLO difference reward does perform better than the standard global reward and the TLO global reward. There is still significant room to improve and enhance the TLO difference reward.

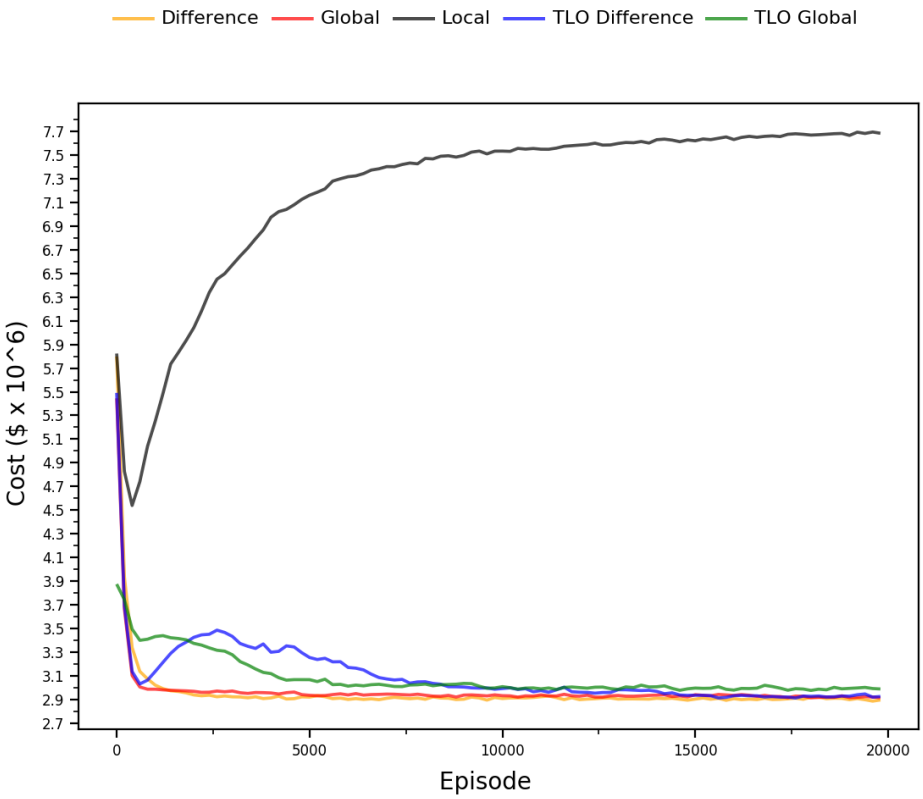

Figure 5.3: Cost with TLQ-learning

In Figure 5.3 the graph for the cost objective is presented. Cost is the second objective to be minimised in TLQ-learning. Figure 5.3 shows a graph in which all agents learn an optimal solution with little difference between the reward structures. Mannion et al. [21], Basu [4] and Mason [22] obtain much lower cost values, which are presented in Table 4.1. There seems to be a limitation as to how well an agent can minimise the cost objective, which could be caused by the level of discretisation used.

	Penalty Violations (1 x 10^6)	Cost ($ x 10^6)
G	40.5757	2.9147
TLO G	34.3277	2.9887
TLO D	23.3842	2.9253
D	15.9003	2.8958

Table 5.1: TLO DEED average final performance

Table 5.1 displays the average final performance of the global, difference, TLO global and TLO difference rewards. In terms of violations, TLO G performs better than G. However, TLO D performs better than both TLO G and G reward structures in regards to violations. D performs better than all rewards in relation to violations and cost.

The threshold for statistical significance is $p = 0.05$. Statistically, the average final performance of the TLO G and the standard G reward structures showed no difference in cost ($p = 0.0777103$) or violations ($p = 0.859243$). The D reward is statistically better that the G reward for cost ($p = 0.011668$) and penalty violations ($p = 6.60645 \times 10^{-9}$). The D reward is statistically better than the TLO G reward for cost ($p = 0.000814$) and penalty violations ($p = 7.69156 \times 10^{-6}$). The TLO D reward and the standard G reward showed no statistical difference for cost ($p = 0.3921$), but TLO D was statistically better than the standard G for penalty violations ($p = 0.001898$). The D(λ) reward was statistically more significant than the TLO D reward, in regards to penalty violations ($p = 0.006605$) and cost ($p = 0.01031$). There was no statistical significance between TLO D and TLO G for cost ($p = 0.530487$), but for penalty violations TLO D was statistically better than TLO G ($p = 0.01627$).

5.1.9 Discussion

This section uses a multi-objective multi-agent problem to evaluate the performance of TLQ-learning. Using the DEED problem, TLO difference and TLO global reward structures were created and deployed in the problem domain. The goal of TLQ-learning is to try and solve the need to prioritise the optimisation of certain objectives. This section reported that both the TLO D and and TLO G were successful at performing better than the standard G reward. In terms of cost, all reward structures performed to the environmental limit for this implementation of DEED. The level of discretisation used is most likely the reason for the above limitation.

The results show the difference reward, D, achieved the best overall performance in this problem domain and produced results comparable to other previously published attempts at solving this problem domain, including NSGA-II (Basu) [4], PSO (Mason) [22], MARL (Mannion) [21].

In future work, the TLO difference reward could be reformulated to produce better results. It would be constructive to reformat the TLO global reward not to use random actions as a tie-breaker solution; this could potentially generate better results. It would also be worthwhile to test the robustness of TLQ-learning agents to determine if they could perform as well as, or better, than the results generated in Section 4.1.5.

Chapter 6

Conclusion

The results of this thesis have been presented and this chapter concludes with an overview of the main contributions and limitations of this thesis. As already stated in Chapter 1, Section 1.2, this thesis explores the following research questions:

1. How can traditional reward structures from single-agent reinforcement learning be applied to a benchmark multi-agent reinforcement learning problem domain? (RQ1)

2. How can reward structures be applied to large multi-objective multi-agent reinforcement learning problem domains? (RQ2)

3. Is it possible to prioritise the optimisation of objectives in a multi-objective multi-agent reinforcement learning problem domain in lexicographic order? (RQ3)

Following the investigations in each chapter of this thesis, these questions may be answered as follows:

1. Chapter 3 confirmed traditional reward structures from single-agent RL can be applied to multi-agent RL and these reward structures can learn good solutions in this expanded domain.

2. The study undertaken in Chapter 4 showed that the reward structures applied in Chapter 3 can also be applied to multi-objective MARL. Chapter 4 shows these reward structures perform well with multiple objectives and they can learn good policies in this domain.

3. Chapter 5 introduced the application of TLO difference and TLO global reward structures into MARL for the first time. In this chapter the TLQ-learning agents were able to learn good policies.

6.1 Summary of Contributions

The main contributions of this thesis are the following:

1. Chapter 3 provided analysis of how traditional reward structures from the single-agent RL domain can be applied to problem domains in the multi-agent domain. Chapter 3 shows these reward structures can be applied with great success and their performance is not hindered by the presence of multiple agents. This confirmed the work of Proper et al. [28].

2. Chapter 4 provided analysis on how the reward structures introduced in Chapter 3 can be applied to a multi-objective domain. This chapter shows how MARL with multiple objectives can use these reward structures and still expect the same general performance as seen in Chapter 3. This confirms these reward structures can scale from single-agent and single-objective domains to multi-agent and multi-objective problem domains. This confirms the work of Mannion et al. [21].

3. Chapter 5 shows, for the first time, how TLQ-learning can be applied to MARL. Both the TLO difference and TLO global reward structures were applied to the DEED problem domain with the goal of minimising the objectives in lexicographic order. The TLO global and TLO difference rewards performed better at this task when compared with the standard global reward. Although there is no statistical significant difference between the TLO global and the standard global reward, there is a statistical difference between the TLO difference and the standard global reward. The standard difference reward performed much better when compared to the TLO global, TLO difference and standard global reward. Consequently, there is future potential to work with reward shaping in the TLQ-learning domain to create new rewards that could potentially perform better than the standard difference reward.

   Limitations in this domain exist due to the implementation of the DEED problem. There is also potential to expand this implementation and expanding could generate better results on the TLO difference and TLO global reward structures. Applying TLQ-learning to MARL expanded on the work of Vamplew et al. [34] and applied the work of Vamplew et al. [34] to MARL for the first time.

6.2 Impact

This thesis will impact future research. Chapter 5 illustrates how there is significant room to implement further solutions for the TLQ-learning domain. There is also room to improve the rewards created in Chapter 5. With the growing popularity of MORL and MORL's association with real world problems the need to opimtimise objectives in lexicographic order will need to be explored further. Due to the work presented in this thesis TLQ-learning has been shown to perform well in the multi-agent multi-objective domain this is something that has not been explored before.

6.3 Limitations

Despite the contributions outlined above, this work does have limitations, which are outlined below:

6.3.1 Statistical Runs

Due to time constraints and constraints on computational power, the experiments in Chapter 3 and Chapter 4 only used 20 statistical runs. Ultimately, 100 runs should be used. In future work it would be valuable to perform more statistical runs which could show statistical significance.

6.3.2 Discrete States and Actions

Chapter 3 and Chapter 4 present application domains with a finite set of states and actions which were represented discretely. As problem domains grow in size, in terms of states and actions, it becomes problematic to represent these domains discretely. There are many real-world problems which need to be represented continuously and, therefore, there is a limitation of how these techniques in RL can be applied. If RL could be used, with some form of function approximation applied, this could potentially allow for RL to be used in a continuous problem domain. An example of an RL algorithm which uses function approximation is Deep Q-learning (DQN) [23].

6.3.3 Multi-Objective Domains

In the DEED problem defined in Chapter 4 and Chapter 5 there are three objectives defined. In real-world problems there are often more than three objectives that need to be optimised. The reward structures defined in these chapters would need to be evaluated against more complex and realistic problem domains.

6.4 Future Work

Following the investigations in this thesis, there are a number of promising avenues for future research and potential to improve the performance of the current work being presented:

6.4.1 TLQ-learning Reward Shaping

The TLQ-learning reward structures created in this thesis perform well. There is significant room to improve on both reward structures presented. In particular, the tie-breaker scenarios for TLQ-learning need to be evaluated. There is significant room to expand on the current solution. Currently the agents take a random action [34], but there is room to explore using other potential solutions to determine a more optimal action. There is also significant room to expand on the TLO difference reward. If further investigations take place into both of these reward structures, there is potential to increase performance beyond the work presented in this thesis.

Also, it would be productive to apply Deep RL to the DEED problem domain, specifically, applying Deep RL with TLO. Using these two techniques, there is potential to create a reward structure that performs better than the standard difference reward across all objectives.

6.4.2 Further Investigation Into Discretisation

As already mentioned in this thesis, the level of discretisation used in Chapter 4 and Chapter 5 needs to be refined. In the DEED problem domain the agents reached a limit in their ability to minimise certain objectives. Mannion et al. [21] did not reach the limitation that was found in this thesis. If investigated correctly, a new level of discretisation could lead to even better performance from the TLO global and TLO difference reward structures, as well as an increased performance for all other reward structures in this domain. A new level of discretisation could allow the agents to lower the values of cost, emissions and penalty violations, similar to the values in the works of Basu [4], Mason [22] and Mannion et al. [21].

6.4.3 Further Investigation Into Scalarisation

Chapter 4 uses linear and hypervolume scalarisation with Q-learning agents. Chapter 5 does not use any form of scalarisation on TLQ-learning agents. It would be constructive to experiment with applying scalarisation techniques to TLQ-learning reward structures. It could be beneficial to apply the standard linear and hypervolume scalarisations to TLQ-learning reward structures. Further investigation into scalarisation for reward structures in the DEED domain could lead to better results.

6.4.4 Correlation Of Objectives

Chapters 4 and 5 present results generated from operating MARL in the DEED domain. The objectives for DEED are cost, emissions and penalty violations. It became apparent there is a correlation between each of the objectives in DEED. Minimising a single objective also lead to the other objectives producing results closer to zero. A result for cost less than 2.9 also produced low results for emissions and penalty violations. A result for cost greater than 2.9 caused the emissions and penalty violations to increase significantly. It would be beneficial to perform an R^2 Correlation test on the objectives of DEED. It would also be constructive to apply the Q-learning and TLQ-learning reward structures to a problem domain with objectives that are not correlated.

6.5 Final Remarks

MORL is an area of research which continues to grow in importance due to the number of real world problems exhibiting multiple objectives. This thesis explores the potential for traditional reward structures to be applied to multi-agent multi-objective domains, while completing additional research in an attempt to target issues arising in MORL. While further work is necessary to convincingly show the performance enhancement associated

with TLQ-learning, this thesis has shown how the traditional reward structures associated with single-agent single-objective RL problems can be scaled and applied to multi-agent multi-objective reinforcement learning problem domains without any performance issues.

Bibliography

[1] R. Anderson, A. Boulanger, W. B. Powell, and W. Scott. Adaptive stochastic control for the smart grid. *Proceedings of the IEEE*, 2011.

[2] I. Arel, C. Liu, T. Urbanik, and A. Kohls. Reinforcement learning-based multi-agent system for network traffic signal control. *IET Intelligent Transport Systems (Volume: 4 , Issue: 2)*, 2010.

[3] W. Arthur. Inductive reasoning and bounded rationality. *Vol. 84, No. 2, Papers and Proceedings of the Hundred and Sixth Annual Meeting of the American Economic Association, pp. 406-411*, 1994.

[4] M. Basu. Dynamic economic emission dispatch using nondominated sorting genetic algorithm-ii. *International Journal of Electrical Power Energy Systems Volume 30, Issue 2, February 2008, Pages 140-149*, 2007.

[5] M. Bowling and M. Veloso. Rational and convergent learning in stochastic games. *Proceedings of the 17th international joint conference on Artificial intelligence - Volume 2 Pages 1021-1026*, 2001.

[6] M. W. Brandt, A. Goyal, P. Santa-Clara, and J. R. Stroud. A simulation approach to dynamic portfolio choice with an application to learning about return predictability. *The Review of Financial Studies*, 2005.

[7] M. Brittain and P. Wei. Autonomous air traffic controller: A deep multi-agent reinforcement learning approach. *arXiv:1905.01303*, 2019.

[8] J. C. Caicedo and S. Lazebnik. Active object localization with deep reinforcement learning. *International Conference on Computer Vision*, 2015.

[9] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. 1998.

[10] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

[11] Z. Gabor, Z. Kalmar, and C. Szcpcsvari. Multi-criteria reinforcement learning. *International Conference on Machine Learning*, 1994.

[12] M. Glavic, R. Fonteneau, and D. Ernst. Reinforcement learning for electric power system decision and control: Past considerations and perspectives. *The 20th World Congress of the International Federation of Automatic Control*, 2017.

[13] J. Hu and M. Wellman. Nash q-learning for general-sum stochastic games. 2003.

[14] P. Jant Hoen, K. Tuyls, L. Panait, S. Luke, and J. La Poutré. An overview of cooperative and competitive multiagent learning. *International Workshop on Learning and Adaption in Multi-Agent Systems*, 2005.

[15] Z. Jie, X. Liang, J. Feng, W. F. Jin, X andLu, and S. Yan. Tree-structured reinforcement learning for sequential object localization. *Conference on Neural Information Processing Systems*, 2016.

[16] S. Lange, M. Riedmiller, and V. nde. Autonomous reinforcement learning on raw visual input data in a real world application. *Conference on Neural Information Processing Systems*, 2012.

[17] L. Levine, F. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 2016.

[18] P. Mannion. Knowledge-based multi-objective multi-agent reinforcement learning. *PhD Thesis, NUI, Galway*, 2017.

[19] P. Mannion, J. Duggan, and E. Howley. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. *Springer International*, 2016.

[20] P. Mannion, E. Howley, J. Duggen, and K. Mason. Applying multi-agent reinforcement learning to watershed management. *Adaptive and Learning Agents workshop*, 2016.

[21] P. Mannion, K. Mason, S. Devlin, E. Howley, and J. Duggan. Multi-objective dynamic dispatch optimisation using multi-agent reinforcement learning. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2015.

[22] K. Mason. Avoidance techniques neighbourhoodtopologies in particle swarm optimisation. master's thesis. *MSc Thesis, NUI, Galway*, 2015.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *Conference on Neural Information Processing Systems Deep Learning Workshop*, 2013.

[24] S. Mousavi, P. Mannion, M. Schukat, and K. Howley. Applying q(λ)-learning in deep reinforcement learning to play atari games. *Adaptive and Learning Agents workshop*, 2017.

[25] G. Neto. From single-agent to multi-agent reinforcement learning: Foundational concepts and methods. 2005.

[26] T. Nguyen. A multi-objective deep reinforcement learning framework. *arXiv:1803.02965*, 2018.

[27] M. O'Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi. Scalable end-to-end autonomous vehicle testing via rare-event simulation. *Conference on Neural Information Processing Systems*, 2018.

[28] S. Proper and K. Tumer. Multiagent learning with a noisey global reward signal. *AAAI Conference on Artificial Intelligence*, 2018.

[29] D. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A survey of multi-objective sequential decision-making. *Journal Of Artificial Intelligence Research, Volume 48, pages 67-113*, 2013.

[30] S. Russell and P. Norvig. Artificial intelligence: A modern approach. 2001.

[31] A. Smith, D. Coit, T. Baeck, D. Fogel, and Z. Michalewicz. Penalty functions. 2000.

[32] R. Sutton and A. Barto. Reinforcement learning: An introduction. 1998.

[33] G. Tesauro. Temporal difference learning and td-gammon. 1995.

[34] P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning, Volume 84, Issue 1–2, pp 51–80*, 2010.

[35] P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning, Volume 84, Issue 1–2, pp 51–80*, 2011.

[36] K. Van Moffaert and A. Nowe. Multi-objective reinforcement learning using sets of pareto dominating policies. *Journal of Machine Learning Research*, 2014.

[37] D. Walter and G. Sheble. Genetic algorithm solution of economic dispatch with valve point loading. *IEEE transactions on Power Systems*, 1993.

[38] C. Watkins and P. Dayan. Q-learning. 1992.

[39] M. Wiering and M. van Otterlo. Reinforcement learning and markov decision processes. 2012.