MULTI-CRITERIA DECISION MAKING USING REINFORCEMENT LEARNING AND ITS APPLICATION TO FOOD, ENERGY, AND WATER SYSTEMS (FEWS) PROBLEM

by

Aishwarya Deshpande

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

Department of Computer and Information Science at IUPUI

Indianapolis, Indiana

December 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Snehasis Mukhopadhyay, Chair

Computer and Information Science

Dr. Mihran Tuceryan

Computer and Information Science

Dr. Yuni Xia

Computer and Information Science

Approved by:

Dr. Shiaofen Fang

I dedicate this thesis to my amazing parents, Vikram Deshpande and Shital Deshpande, who have been a constant source of support and encouragement during the challenges of graduate school and life. I love you! My grandmother, "Aai," has never spared any efforts to keep me happy. I will see you soon! I also dedicate this work to my second grandmother, "Aajji," who has stood by my side in all my achievements and failures. Special thanks to my Dad and my friends who took out time and helped me with my defense presentation and proofreading. Lastly, I want to remember my paternal grandfather, "Baba," who taught me to live responsibly. I cherish all the letters that you wrote to me on my every birthday. May you find peace and endless books to read in Paradise!

ACKNOWLEDGMENTS

I would like to thank Dr. Snehasis Mukhopadhyay, who saw potential in me to work for him as a research assistant and guided me throughout my thesis. I would also like to thank all my committee members for their willingness to serve on my committee and for helping me with my thesis. Lastly, I would like to thank Dr. Meghna Babbar-Sebens for giving me the opportunity to work on the INFEWS project as a research team member. This work is an interagency partnership between USDA-NIFA and the National Science Foundation (NSF) on the research program, Innovations at the Nexus of Food, Energy, and Water Systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

MCDM: Multi-criteria decision making

FEWS: Food, energy, and water systems

OR: Operational research

WSM: Weighted sum model

WPM: Weighted product model

AHP: Analytical hierarchical process

ML: Machine learning

AI: Artificial intelligence

MANETs: Mobile ad-hoc networks

QoS: Quality of service

RF: Random Forest

MDP: Markov decision process

RL: Reinforcement learning

DNN: Deep neural networks

DQN: Deep neural networks and Q learning

ARL: Average reward reinforcement learning

ABSTRACT

Multi-criteria decision making (MCDM) methods have evolved over the past several decades. In today's world with rapidly growing industries, MCDM has proven to be significant in many application areas. In this study, a decision-making model is devised using reinforcement learning to carry out multi-criteria optimization problems. Learning automata algorithm is used to identify an optimal solution in the presence of single and multiple environments (criteria) using pareto optimality. The application of this model is also discussed, where the model provides an optimal solution to the food, energy, and water systems (FEWS) problem.

Keywords – Multi-criteria decision making, reinforcement learning, learning automata, pareto optimality

CHAPTER 1. INTRODUCTION

Napoleon Bonaparte once said, “Nothing is more difficult, and therefore more precious, than to be able to decide.” Multi-criteria decision making (MCDM) is all about making decisions from a set of options. At one point or another in our everyday lives, we make choices. For example, for a person, buying a new phone may depend on its features like camera quality, screen size, battery life, etc. Choosing an apartment may depend on its rent, location safety, etc. However, people do not make the right decisions every time, which may result in consequences. Most people usually consider cost as the main criteria and make decisions based on that. A person buying a cheaper phone may be comfortable with its consequences, including low battery life, low camera quality, etc. Whereas, if a person buys a cheaper apartment but isn’t located in a safe neighborhood, then it is surely considered a bad decision. Therefore, when the stakes are high, it is essential to carefully weigh the criteria and evaluate the best decisions possible. This research aims to develop an MCDM model that helps prevent bad decisions by critically evaluating the criteria and considering the weights of criteria and tradeoffs between them. The tradeoffs are studied using the phenomenon of pareto optimality. Pareto optimality is a tradeoff where a certain amount of sacrifice is involved in one parameter to achieve a certain advantage in the other. In this chapter, an introduction to MCDM and its background is discussed, followed by the research problem, aims and objectives, its significance, and limitations.

1.1 Background study

MCDM has widely grown to be a part of the Operational Research (OR) over the years. OR models are commonly used for decision making in the presence of multiple criteria. Many techniques and approaches have developed over time to carry out MCDM. The earliest model used for decision making was the Weighted sum model (WSM), developed by P C Fishburn in 1967. It was widely used because of its simplicity [1]. Soon enough, a better version than WSM was developed called Weighted Product Model (WPM) developed by Miller and Starr [2] [3]. Later, a technique named Analytical hierarchical process (AHP) was introduced by Saaty in 1977 and has been popular ever since [4].

1.2 Research problem

These techniques, however, have their own limitations. They usually work with a predefined problem, i.e., decision making with single or multiple goals in the presence of multiple criteria. These criteria are gathered based on subjective judgments by researchers, statistical analysis of the data, and proposed theories. In today's world, the complexity of real-life problems is increasing rapidly, and it will not be satisfied because the knowledge and experience those researchers carry are limited. Also, the statistical analysis and the proposed theories cannot suffice the need for real-world problems as they are more specific and do not work with minimal criteria. Another critical issue these methods have not addressed is improvement planning in problem-solving. The existing methods work on finding better alternatives based on predefined constraints and resources. The business world is a dynamic and rapidly changing environment in which these methods with predefined constraints will not satisfy the needs. Many industries work on big data and require a decision making model that understands the data, can study its patterns based on the historical data, and make predictions based on it to provide more accurate results. This makes the existing research inadequate for fulfilling such requirements of industries. In order to overcome these shortcomings, decision making models using hybrid techniques that involved machine learning were introduced [5].

1.3 Research aims and objectives

This study aims to use a Machine learning (ML) algorithm to achieve our goal. ML is a branch of artificial intelligence (AI) that helps the systems to learn and improve from experience with minimal human intervention. [6] It learns from the input data, understands its pattern, and helps make better decisions in the future. There are multiple types of ML algorithms, namely supervised, unsupervised, semi-supervised, and reinforcement learning algorithms. Supervised learning learns from a known training dataset and gives output values based on the previous values using prediction analysis. Unsupervised learning learns from an unknown dataset and analyzes the hidden structures in the data. Semi-supervised learning falls between supervised and unsupervised learning. It learns from both labeled data and unlabeled data and helps improve learning accuracy. In this study, we will be using a reinforcement learning algorithm for developing an MCDM model. Reinforcement learning is a method that requires a feedback signal for the agent to learn

and identify the optimal action. In this paper, our first objective is to study learning automata algorithm and implement it using agents with congruent goals. Our second objective is to carry out reinforcement learning using learning automata in identical and multiple environments. Our final objective is to implement our model on food, energy, and water systems (FEWS) problem.

1.4 Significance and limitations

MCDM using machine learning is a need of the hour because the industrial world is rapidly growing and changing with everyday developments in technology. More and more criteria are observed and gathered which makes it difficult for humans to work as the data is large. Therefore, many industries are relying less on humans and shifting towards machine learning techniques for more accurate results. Some industries are even opting for fully automated systems that require machine learning algorithms that learn and act from its errors and avoid them in the future.

This model comes with its limitations. Multiple runs can overload the model, affecting the results as it will not always converge to the optimal action. The separation between the data values is needed if they are close to each other. It is challenging to work with multiple decision-makers as the complexity of assigning weights to the criteria according to the decision maker's preference increases.

1.5 Contribution

In the world of people trying to make a decision in the presence of conflicting criteria where the goal is to satisfy both sides of the party, a decision making model that carefully calculates the importance of criteria and provides an optimal solution is needed. The main contribution of this thesis is an MCDM model that helps in making accurate decisions based on the reward probabilities of every action. A learning automata-based reinforcement learning algorithm is proposed which can be helpful in multi-criteria problems. The model is implemented on an NSF funded project called Innovation at the nexus of FEWS, and the results of this implementation prove that this model can be used on large-scale issues.

1.6 Structure outline

In Chapter 1, the context of the study has been introduced. Background study of the existing research and their problems have been discussed. The aims and objectives of the research have been identified. Chapter 1 concludes by discussing the significance and limitations of this study. In Chapter 2, existing literature will be reviewed to identify better machine learning approaches for decision making in a dynamic and rapidly changing environment. In Chapter 3, the theoretical framework of the MCDM model will be introduced. It will explain in detail all the experiments that were carried out. Chapter 4 will showcase the experimental results and discuss and evaluate them. Chapter 5 will discuss the implementation of the model on the FEWS problem, along with the experiments and their results. The study will be concluded in Chapter 6.

CHAPTER 2. LITERATURE REVIEW

The need for MCDM has grown tremendously over the last two decades. This literature review consists of papers, articles, and books explaining the methodologies, their contribution, and applications closely related to our model. It describes the advantages and their limitations which gives us an idea of how to create a better model that can overcome the shortcomings of the traditional approaches. In this model, we use a reinforcement learning algorithm that uses learning automata to carry out decision making. We also used the pareto optimality phenomenon to find the optimal solution by understanding the tradeoff between the criteria. Many articles that use reinforcement learning in their applications due to its advantages. Pareto optimality has also been studied and proved helpful in MCDM problems. In this chapter, we will cover relevant research that proposed MCDM approaches using reinforcement learning and performed better than the traditional approach.

2.1 Multicriteria Reinforcement Learning Based on a Russian Doll Method for Network Routing

The aim of this study is developed by authors of [6] and implemented in Mobile Ad-Hoc Networks (MANETs) for network routing optimization. It uses pareto optimality for decision making and reinforcement learning for finding an optimal routing option that consists of optimized parameters. A Russian doll method is used to find the optimal solution using reinforcement learning. It is very difficult to assign weights to the criteria according to the decision maker's preference. So, the parameters are set beforehand and fed to the Russian doll method in the form of nested boxes. These are Quality of Service QoS parameters which consist of hop count, delay, bitrate, etc. Just like a Russian doll which has multiple smaller size dolls inside it, they use a set of nested boxes to define the criteria space. These boxes are calculated by a Cartesian product and the criteria are represented by vectors. The concept of pareto optimality is used for identifying pareto optimal set of vectors on the criteria space. More boxes make it easier to control the choice of optimal solutions accurately. The number of boxes and their shapes is decided by the number of parameters and their values are determined by the threshold values of the parameters. The pareto set of optimal vectors can be accessed on the boxes using weighted Chebyshev distance.

The set of criteria vectors are compared using a set of rules to find the best criteria vector. Once the set of pareto optimal vectors is identified, they carry out the task of finding the best route, which is a path between the source and destination node. The set of optimal parameters are evaluated at the destination node, and to satisfy these QoS requirements, the router needs to select the best neighbor node by comparing all the neighboring nodes consisting of the multicriteria vectors using the Russian doll method. Reinforcement learning comes into the picture at the source node, where it updates the parameters for every packet till it converges to the optimal route. This study proved that their proposed method works better than the traditional MCDM approaches and makes it easier to optimize as it does not require interaction with the decision-maker.

2.2 A Deep Reinforcement Learning Based Multi-Criteria Decision Support System for Optimizing Textile Chemical Process

The textile industry is increasing globally, and this study proposes a decision support system for optimizing textile chemical processes. Manufacturing textile consists of interconnected processes which consist of multiple important criteria which need to be taken into consideration for the decision making process. As the number of criteria increases, the complexity to deal with them also increases, leading to finding an optimal MCDM approach that can carry out decision making. To achieve this, they combine Random Forest (RF) model and a traditional MCDM Analytical Hierarchical Process (AHP) model. Markov decision process (MDP) is used to describe the textile manufacturing process, and a deep reinforcement learning algorithm is used for its optimization.

RF model is used for predicting the unknown performance of the textile manufacturing process, which is done with the help of multiple weighted regression trees. Later, the AHP model is used for decision making, which carries out a pairwise comparison method to convert a multi-criteria problem into a single objective optimization problem that consists of a hierarchy formed based on criteria weights or priorities. The reinforcement learning (RL) algorithm works on a set of actions that are selected by learning the environment states and the reward associated with that action. However, this study uses a deep reinforcement learning algorithm as a simple MDP problem that is not feasible for large-scale decision-making issues. This deep reinforcement learning algorithm consists of deep neural networks (DNN) combined with reinforcement learning

based on Q learning RL algorithm named DQN, which is used as a decision support system in this study.

The proposed framework is implemented on the textile color fading ozonation process to evaluate its performance. Color fading ozonation is a finishing process that gives a washed-out look, especially on denim, using an ozone gas because of its less environmental impact as it does not use a water bath for color fading. This process is an MCDM problem with criteria such as water, temperature, *pH*, and time which is fed to an RF model as input parameters to learn and predict the output parameters, namely color depth, lightness, and chromatic components. AHP is used to determine the weights of the criteria that are acceptable as a good quality fabric by creating a pairwise comparison matrix. DQN algorithm is used for finding an optimal solution from the weighted criteria by choosing an action with the highest performance index. [7]

2.3 Dynamic preferences in multi-criteria reinforcement learning

The aim of this study is to propose a multi-criteria reinforcement learning algorithm that finds an optimal policy in the presence of conflicting criteria, taking into consideration the agent's time-varying preferences. For example, a clothing brand has a goal of producing quality clothes while keeping the selling cost low to make it affordable for the customers. This can be termed as conflicting criteria as it is difficult to satisfy both goals. Over time, when the clothing brand's market value increases, the manufacturer might want to sell even better-quality clothes but for a costlier price. This shows how the preferences of the agent changes with time. The method proposed in this study is implemented on Buridan's ass problem and a network routing problem. [8]

The Average Reward Reinforcement Learning (ARL) approach is used to assign average fixed weights to the criteria, and which helps the numerous criteria act as a scalar function where the weights are reward probability. Markov's decision process (MDP) helps in making the model learn quicker as MDP has the ability to remember the policies learned in the past. This proves very helpful when the preferences change with time, and the weights are updated accordingly as the MDP does not learn from scratch when the new weights are assigned; instead, it simply tries to make the existing optimal policy better.

The framework starts with dividing the rewards into k types and assigning a weight vector $w = (w_1, w_2, w_3, \ldots, w_k)$ to it, and multiplying it with the rewards assigned with every action of type i to calculate weighed gain. The next task is to maximize the weighted gain using an optimal policy called the undominated policy, while the other policies are called dominated policies. The dynamic multi-criteria reinforcement learning algorithm learns these undominated optimal policies and stores them for future use. Weighted gain is maximized using an optimal policy that maximizes the inner product. This inner product is assigned with an average reward vector, and new policies are learned and updated using vector reinforcement learning, and these policies are stored in the set of optimal policies.

There are endless such articles based on MCDM using reinforcement learning and pareto optimality that can be studied and used in future research. We used the learning automata algorithm for learning agents in reinforcement learning to achieve our goals. The next chapter explains all the algorithms used in putting together our model and describes the methodology used in detail.

CHAPTER 3. METHODOLOGY

Multi-criteria decision making MCDM uses machine learning to carry out the algorithms that understand the data and give the desired output. The output is an optimal solution that is identified for a set of solutions using an MCDM model. There are various MCDM models that are researched and implemented in numerous applications. In our model, we use a reinforcement learning algorithm to learn the data using learning agents. The data is studied from a feedback response which also guides in choosing an optimal action. The methodology that we used to achieve the research aims and objectives is explained in this chapter, along with the algorithms used to form the MCDM model. The first algorithm explained is the reinforcement learning algorithm which is used for decision making. The second algorithm explained is learning automata which is used for learning agents in reinforcement learning. Pareto optimality also plays an important role in this model as it gives an optimal decision in the presence of conflicting criteria. After the algorithms are explained, the algorithms are implemented in the experiments. All the experiments that we carried out are explained in detail that contains the dataset used along with the equations that are updated for getting better results from the learning agents.

3.1 Reinforcement learning

The need for intelligent systems has increased tremendously in today's world as every user has become more demanding for high-precision results. Almost every appliance at home or at the office works on an intelligent system for problem-solving or decision making. The goal of an intelligent system is to learn and behave just like a human being which can be very useful as it does the work of a human for a human. This system has applications in all the domains ranging from a data company to the automobile industry to the medical field. Dynamic intelligence is required to carry out such complex tasks in any respective domain. Dynamic intelligence learns from existing knowledge and takes decisions based on the response of every action. It also has the ability to explore new knowledge with the help of existing knowledge and interactions with the environment. Machine learning (ML) is the solution to these demands. ML provides intelligent systems that learn from available knowledge to attain better understanding which will help to build knowledge. It learns from a feedback response which is in the form of rewards or penalties. There

are three types of machine learning, namely, unsupervised, supervised, and semi-supervised learning. We use reinforcement learning, which is a type of machine learning, in our methodology. It uses learning agents who learn on their own and perform tasks like classification, prediction, decision making, etc.

Reinforcement learning is a mix of dynamic programming and supervised learning. It performs learning from its past experiences as well as from the environment to take better actions in the future. This feature of reinforcement learning is called exploitation and exploration. The learning agent will select an action that it chose in the past, which has a higher reward to obtain more rewards, but at the same time, it also has to try new actions which might have even a higher reward. Obtaining more rewards by choosing the same action that proved effective in the past can be termed exploitation whereas exploration means finding new actions that have not been performed before to achieve a better understanding and to build the existing knowledge successfully. It is important to maintain the balance between exploitation and exploration to satisfy the need of providing better and accurate results.[9][10][11]

Reinforcement learning faces a challenge while identifying optimal actions in the presence of unpredicted real-time scenarios. This happens as there is no data available for learning when an unpredicted scenario is encountered. In this case, the learning agent takes a decision based on the experience that it registered in a similar scenario. The reinforcement learning algorithm is shown in Figure 1:

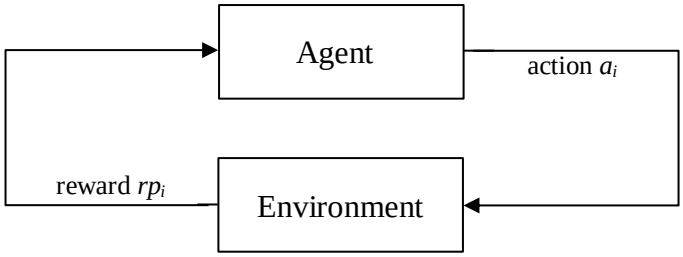

Figure 1. Reinforcement learning

In this figure, reinforcement learning is depicted where the learning agent selects action a_i from a set of actions and interacts with the environment which gives a response in terms of reward probability rp_i of the selected action. The agent registers this response and continues learning by discovering new actions and the rewards associated with them. The learning agent here uses the learning automata algorithm, which is explained in the next section.

3.2 Learning automata algorithm

Learning automata has proved very important in the artificial intelligence (AI) field as it consists of learning agents that learn on their own and are capable of making decisions just like humans. In this study, we use the learning automata algorithm researched by Kumpati S. Narendra & Mandayam A. L. Thathachar to achieve the goal of converging to the optimal action. Learning automata algorithm consists of a set of actions, an action probability list, and a reward probability list. The action probability list consists of the probability of choosing an action from a set of actions. The reward probability list consists of reward probabilities assigned to every action. The set of actions consists of a finite number of actions that act as an input, while the reward probability list acts as a response to the learning agent. Depending on the reward probability assigned to a particular action, the choice of action made by a learning agent is guided. If the reward probability is in favor of the optimal action, then the action probability is updated. The learning agent keeps on performing actions and updating its probabilities till it converges to the optimal action. As it is a learning agent, it learns the present and past responses of every chosen action, which helps it to perform better while choosing the next action.[12][13][14]

This algorithm basically performs a trial-and-error method that learns from its experience. For example, if a child touches a hot object for the first time, it gives a burning sensation, and the child starts crying. This shows that the response achieved from taking this particular action was in the form of a penalty and not a reward. The child will register the response and will avoid taking that action again. Here, the child acts as a learning agent. Every action has a reward probability which can be as per the user's needs. For example, if a user decides to buy a new phone and the criterion important to the user is the phone's battery life, then the reward probability assigned to the action of buying a phone with a longer battery life will be more than that given to others.

To get more acquainted with this learning algorithm, we implemented it on a toy dataset. The dataset consisted of ten actions, each having an action probability of 0.1. The reward probability list consisted of random values between the range 0-1. We picked two numbers at random and compared them with action probability and reward probability, and updated the action probability till one of the action probabilities became 1. The algorithm is explained using the steps as follows:

Table 1. Learning automata algorithm is described in steps mentioned in this table

Learning automata algorithm		
Step 1	:	A random number r_1 is picked from the range 0-1
Step 2	:	A running sum is calculated of the action probability ρ_i list
Step 3	:	r_1 is compared with it the running sum
Step 4	:	Once r_1 it's greater than the cumulative sum, action a_i associated at that position is chosen
Step 5	:	The reward probability rp_i of action a_i is selected for further calculation
Step 6	:	A random number r_2 is picked from the range 0-1
Step 7	:	r_2 is compared with rp_i
Step 8	:	**if** $rp_i > r_2$ **then** $\rho_i = \rho_i + \alpha(1 - \rho_i)$ $\rho_j = \rho_j - \alpha\rho_j$ for $j \neq i$
Step 9	:	The above steps are repeated till ρ_i converges to 1

This algorithm does not always converge to the right optimal action. The convergence can be controlled by adjusting the learning rate α. The value of alpha α is made smaller to avoid the algorithm converging to the wrong action. We use this algorithm for learning agents in our experiments. Initially, we used a single learning agent to achieve an optimal action in a single environment. Later, we worked with two agents, which performed mutually to achieve the same goal of identifying the optimal action. Lastly, we worked with multiple environments where we used two learning automata for each to perform pareto optimality for finding an optimal solution according to the decision maker's preferences. The phenomenon of pareto optimality is explained in the next section.

3.3 Pareto optimality

In the world of decision making, there are various ways to solve a multi-criteria decision making problems. One of them is by using Pareto optimality. Pareto optimality is basically a tradeoff where there is a certain amount of sacrifice is involved in one parameter to achieve a certain advantage in the other. Pareto optimality plays a significant role in decision making where multiple conflicting criteria are involved. Taking a thoughtful, intelligent decision is a difficult

task when many complex criteria are involved. In this case, machine learning is used which learns like a human being and provides results with high precision. In 2001, Sevan G. Ficici and Jordan B. Pollack developed an algorithm designed to help coevolutionary learning perform better with the help of pareto optimality. [15] In 2007, Sadan Kulturel-Konak and David W. Coit created a methodology that shows a relation between pareto optimality and single solution approaches.[15][16]

A real-life example of pareto optimality can be when both sides of the transaction are in benefit. For example, if a seller is selling a house for $70,000 and the buyer is willing to pay $80,000, then $75,000 will be a pareto optimal price which will benefit both the seller and the buyer. Another example can be a production or manufacturing industry that aims to produce goods of good quality and affordable for a commoner at the same time. These examples show conflicting criteria where a decision that satisfies the user is needed. In this study, we use pareto optimality to attain an optimal action in the presence of multiple environments.

To study pareto optimality, we considered two environments and one set of actions. The learning agents learn from both the environments to gain rewards associated with every action that the agents select at every instance. The agents carry out mutual learning where they learn from both the environments and update the action probabilities in a way that the optimal action is identified. The results can be improved by assigning weights to the criteria according to the decision maker's preferences. In the section below, all the experiments that we carried out are discussed in detail, which carries out decision making in identical and multiple environments.

3.4 Experiments performed

In this section, all the experiments that we carried out in the process of forming the MCDM model are discussed. The experiments were performed using multiple learning agents to carry out decision making in multiple environments using reinforcement learning and pareto optimality.

3.4.1 Experiment I: Learning automata

We implemented the learning automata algorithm on a toy dataset consisting of ten actions and updated the action probability based on the reward probabilities. The optimal action was identified by a learning agent that learned from the data and built knowledge about the actions and

their rewards. The step size α is varied from 0.5 to 0.001 to study the convergence of the learning agent. As the step size value is made smaller, the performance of the learning agent improves as it converges to an optimal action. To get more details on how the algorithm performs, we calculate the convergence speed and the accuracy of the learning agent.

3.4.2 Experiment II: Performance with various step size equations

This experiment studies the performance of agents corresponding to various step size equations in the presence of one environment. We implemented two learning agents within the code where both the agents are connected to the same environment (same reward probability list). Nevertheless, they generate different random numbers for both action selection as well as reward generation. We kept the learning rate α or step-size constant for experiment I and carried out the steps mentioned in the learning automata algorithm to identify an optimal action. For this experiment, the step size was not kept constant from iteration to iteration and was made proportional to the action probability of the other agent for the action chosen by the given agent. Following are the equations of step sizes of two agents used in the experiments performed to optimize the performance of the learning agents:

Experiment 1:

After carrying out the initial experiment to get acquainted with the learning automata algorithm, we changed the constant step size to an equation that calculates the inner product of the action probabilities ρ_1 and ρ_2 using the function np.inner.

$$\alpha = \alpha * \text{np.inner}(\rho_1 , \rho_2) \quad (1)$$

Experiment 2:

In this equation, we calculate the norm of the vectors in the denominator by importing the linear algebra NumPy library as LA.

$$\alpha = \alpha * \text{np.inner}(\rho_1 , \rho_2) / (\text{LA.norm}(\rho_1) * \text{LA.norm}(\rho_2)) \quad (2)$$

Experiment 3:

$$\alpha = \alpha * (2^{\rho_2[i]} - 1) \quad (3)$$
$$\alpha = \alpha * (2^{\rho_1[j]} - 1) \quad (4)$$

Experiment 4:

$$\alpha = \alpha * (1/3) * (2^{(2*\rho_2[i]-1)}) \quad (5)$$

$$\alpha = \alpha * (1/3) * (2^{(2*\rho_1[j]-1)}) \quad (6)$$

Experiment 5:

$$\alpha = \alpha * (1/7) * (2^{(3*\rho_2[i]-1)}) \quad (7)$$

$$\alpha = \alpha * (1/7) * (2^{(3*\rho_1[j]-1)}) \quad (8)$$

Experiment 6:

$$\alpha = \alpha * (1/15) * (2^{(4*\rho_2[i]-1)}) \quad (9)$$

$$\alpha = \alpha * (1/3) * (2^{(4*\rho_1[j]-1)}) \quad (10)$$

Experiment 7:

$$\alpha = \alpha * \rho_2 [i] \quad (11)$$

$$\alpha = \alpha * \rho_1 [j] \quad (12)$$

Experiments 3 to experiment 7 state equations of their respective agents. For e.g., Equation (3) is the step size used by agent 1, and equation (4) is the step size used by agent 2 where ρ_1 is the action probability of the first set of actions and ρ_2 is the action probability of the second set of actions.

Experiment 8:

Here, the step size a is constant for both the agents. The steps mentioned in the learning automata are carried out here where both the agents converge to their respective optimal action by carrying out isolated learning. From the above experiments and their performance, we picked the best simulation set up with respect to mutual learning, i.e., the best environment for which the effect of mutual learning was the most prominent among the experiments that we performed.

Experiment 9:

Instead of updating the action probabilities through mutual learning in all the iterations, we updated it in iteration *k* where *k* is greater than or equal to 1, with probability Q(*k*), where Q(*k*) is equal to $1/\sqrt{k}$. That is, initially for a lower value of k, the probability of updating through mutual learning is close to 1. However, as k becomes larger and larger, that probability approaches zero.

When mutual learning is not selected according to the probability (1- Q(*k*)), the action probabilities update as isolated learning with the constant step size. The previous best simulation was compared with the results of this experiment to conclude further the best way to carry out convergence that behaved similarly to the first experiment where only one learning agent was involved, and the learning performed was isolated learning.

3.4.3 Experiment III: Mutual learning with multiple environments

We set up a multi-criteria problem with two automata using a learning algorithm, but with two different environments (teachers). Firstly, we implemented a decoupled automata where each one performed isolated learning and converged to the optimal action (the one with the highest reward probability) in its respective environment. Then, we implemented coupled mutual learning algorithm to see which single action the coupled automata converged to as a team. Here, we implemented a pareto optimality algorithm to get the results. The two environments used are as follows:

Environment 1 has ten actions where action number 7 is the optimal action as it has highest reward probability. The reward probability list is,

reward_prob = [0.2, 0.3, 0.2, 0.1, 0.2, 0.6, 0.9, 0.5, 0.4, 0.1]

Environment 2 also has ten actions where action number 4 is the optimal action as it has highest reward probability. The reward probability list is,

reward_prob2 = [0.2, 0.3, 0.2, 0.9, 0.2, 0.6, 0.2, 0.5, 0.4, 0.1]

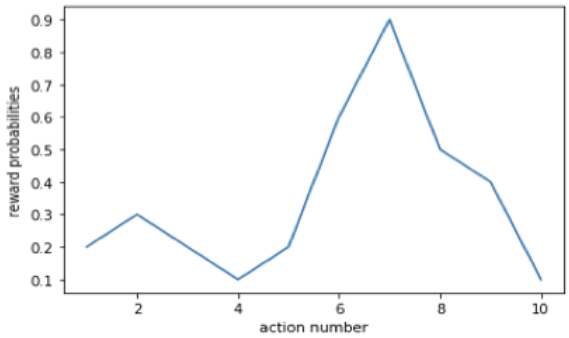

Figure 2. Environment 1

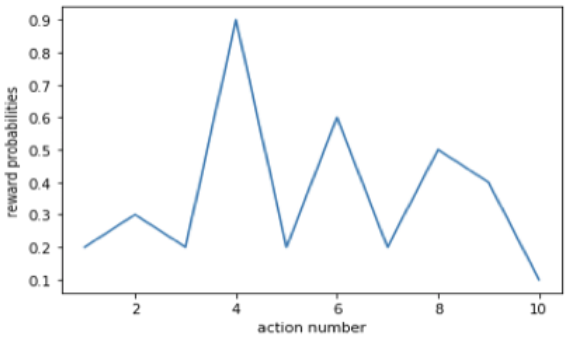

Figure 3. Environment 2

3.4.4 Experiment IV: Varying weight between self-learning and mutual learning

This experiment is carried out in multiple criteria where the goal of the experiment is to identify an optimal decision that favors the decision maker's preference. To make this happen, certain weights are assigned to each criterion and are varied to get a solution that is optimal and satisfies the decision-maker. The weight is varied between each criterion in a way that the agents perform isolated learning if the weight assigned to mutual learning is zero. The agents are decoupled and perform the learning automata algorithm in their respective environments, thereby giving two optimal solutions, one of each environment. When the weight assigned to isolated learning is zero, the agents perform mutual learning where both the agents are coupled and programmed to work as a team, thereby giving one optimal solution for multiple environments. We use pareto optimality here as the criteria involved are conflicting, and a solution that benefits the decision-maker is needed. As it works with conflicting criteria, it is impossible for a solution to exist that gives the best of both criteria. Thus, the optimal solution found contains a slight compromise on one of the criteria to get the best of the other.

Later, this experiment is implemented on the FEWS problem where two conflicting criteria are involved, and the pareto optimality phenomenon is used to make an intelligent decision.

CHAPTER 4. RESULTS

4.1 Experiment I: Learning automata

The first experiment that we performed was implementing the learning automata algorithm using one learning agent to identify the optimal action. The action probability list and reward probability list used are stated below. Here, the lists start with index 0, making action number 6 the optimal action with a reward probability of 0.9

$$\rho_i = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]$$

$$rp_i = [0.2, 0.3, 0.2, 0.1, 0.2, 0.6, 0.9, 0.7, 0.4, 0.1]$$

Table 2. Performance of learning agent using learning automata algorithm

α	Convergence time	Accuracy (%)
0.5	9.33	36
0.2	37.38	56.99
0.1	102.97	76
0.05	262.65	92
0.02	571.52	100
0.01	1185.15	100
0.005	2468.09	100
0.002	6198.11	100
0.001	12221.79	100

In the table above, the learning agent converges to the optimal action number 6 using 0.02 as the step size α value as the accuracy achieved with 0.02 is 100%. Speed parameter is calculated where the values indicate the number of iterations required to converge to the optimal action. Accuracy states the percentage of converging to the right action, which is the number of iterations converging to the right action from the total number of iterations required to converge to the right action.

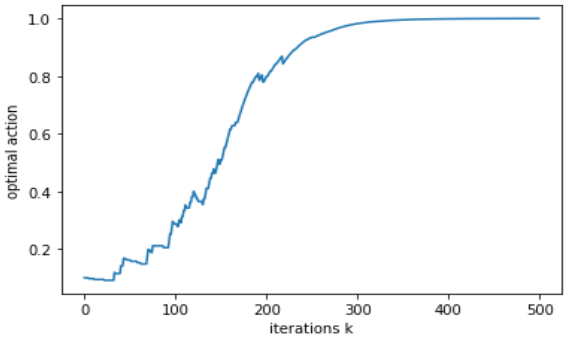

Figure 4. Action probability of the optimal action number 6 averaged over 500 iterations

Figure 4 shown above shows a graph drawn over 500 iterations showing the average value of the optimal action probability. The action probability of optimal action was initially 0.1 and went on updating till it was close to 1, which took around 300 iterations.

4.2 Experiment II: Performance with various step size equations

We carried out Experiment II, where we used two sets of actions, each having the same action probability in the beginning. Though there are two action sets, there is only one reward probability list common for both. We introduced two learning agents to work with multiple action sets, who learn and update the action probability that is made proportional to each other. Various step-size equations are carried out to improve the performance of the learning agents. To understand the performance better, we calculated the convergence speed and the accuracy of the model. Unlike in the learning automata algorithm, where the step size is kept constant, it is varied in this experiment. The results of the performance of the model using all the step size equations are discussed below.

4.2.1 Experiment 1

In this experiment, we calculated the inner product of both the action probabilities ρ_1 and ρ_2 of the learning agents, agent 1 and agent 2, respectively. The maximum step size (α) value was also calculated as the agents were coupled.

Table 3. Performance of agent 1 using equation (1)

α	Convergence time	Accuracy (%)
0.47	199.44	78
0.18	243.78	97
0.09	480.62	100
0.045	929.02	100
0.018	2235.9	100
0.008	4507.08	100
0.0049	9006.53	100
0.0017	22351.62	100
0.0008	44796.18	100

Table 4. Performance of agent 2 using equation (1)

α	Convergence time	Accuracy (%)
0.47	159.16	76
0.18	238.98	98
0.09	460.03	100
0.045	911.58	100
0.018	2218.58	100
0.008	4484.75	100
0.0049	8971.27	100
0.0017	22263.9	100
0.0008	44756.75	100

From the tables above, it is seen that the results of both the learning agents are very similar. Agents 1 and 2 give a hundred percent accuracy when the step size value used is 0.1. As the step size equations have action probabilities made proportional to the other agent.

4.2.2 Experiment 2

The performance of both the agents using equation 2 is shown below, where both the agents converge with 0.02 step size within 700 iterations.

Table 5. Performance of agent 1 using equation (2)

α	Convergence time	Accuracy (%)
0.49	123.33	67
0.19	224.48	76
0.09	298.85	88
0.049	355.9	99
0.019	680.78	100

Table 6. Performance of agent 2 using equation (2)

α	Convergence time	Accuracy (%)
0.49	162.91	53
0.19	206.23	72
0.09	293.68	83
0.049	396.9	97
0.019	669.99	100

Table 5. continued

α	Convergence time	Accuracy (%)
0.009	1299.3	100
0.0049	2433.45	100
0.0019	6318.67	100
0.0009	12335.34	100

Table 6. continued

α	Convergence time	Accuracy (%)
0.009	1268.23	100
0.0049	2557.7	100
0.0019	6117.89	100
0.0009	12285.77	100

From the above results, experiment 1 converges faster than experiment 2, and the accuracy also turns out better than that of experiment 2.

4.2.3 Experiment 3

We decided to carry out the remaining Experiment II with one more environment to understand the convergence better. The environments used to carry out the experiments explained below are stated as follows:

Environment1 = [0.2, 0.3, 0.2, 0.1, 0.2, 0.4, 0.9, 0.4, 0.2, 0.1]

Environment2 = [0.2, 0.3, 0.2, 0.1, 0.2, 0.6, 0.9, 0.7, 0.4, 0.1]

The step size equations used in experiment 3 to experiment 7 are carried out to study the performance of the agents when they are coupled. Later, the results of these experiments are compared with experiment 8, where the agents are decoupled. The comparison is carried out in order to see which step size equation used for mutual learning works more like the equation used for isolated learning. The results of both the agents' performance using environments 1 and 2 and comparing it with the results of experiment 8 are discussed below:

There are four tables describing the performance of two agents in two separate environments where tables 7 and 8 show results of both the agents in environment 1, whereas tables 9 and 10 show the performance of agents in environment 2.

Table 7. Performance of agent 1 using equation (3) environment 1

α	Convergence time	Accuracy (%)
0.49	52.22	92
0.19	116.18	99
0.09	231.74	100
0.045	443.7	100
0.017	1093.61	100
0.0089	2170.5	100
0.0044	4337.16	100
0.0017	10883.04	100
0.0008	21684.03	100

Table 8. Performance of agent 2 using equation (4) environment 1

α	Convergence time	Accuracy (%)
0.49	51.79	92
0.19	115.59	99
0.09	231.02	100
0.045	442.89	100
0.017	1093.88	100
0.0089	2170.27	100
0.0044	4336.03	100
0.0017	10884.67	100
0.0008	21692.33	100

Table 9. Performance of agent 1 using equation (3) environment 2

α	Convergence time	Accuracy (%)
0.49	50.07	65
0.19	129.2	79
0.09	248.73	86
0.045	498.3	92
0.018	1210.75	100
0.008	2378.34	100
0.0044	4740.85	100
0.0017	11868.19	100
0.0008	23686.02	100

Table 10. Performance of agent 2 using equation (4) environment 2

α	Convergence time	Accuracy (%)
0.49	50.8	65
0.19	128.8	79
0.09	247.59	86
0.045	498.23	92
0.018	1209.99	100
0.008	2379.08	100
0.0044	4740.65	100
0.0017	11861.85	100
0.0008	23681.1	100

From the tables above, it is seen that the convergence time to achieve 100% accuracy of agents in environment 1 is much less than that of the agents in environment 2. The results are

depicted in the form of graphs below, where figure 5 and 6 is the visualization of table 7 and table 8, and figure 7 and 8 is the visualization of table 9 and 10.

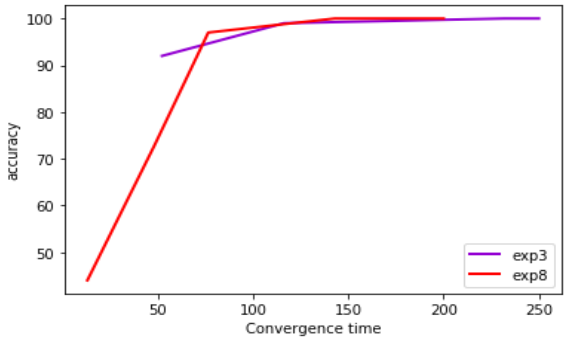

Figure 5. Performance of agent 1 using equation (3) environment 1 compared with agent 1 of experiment 8

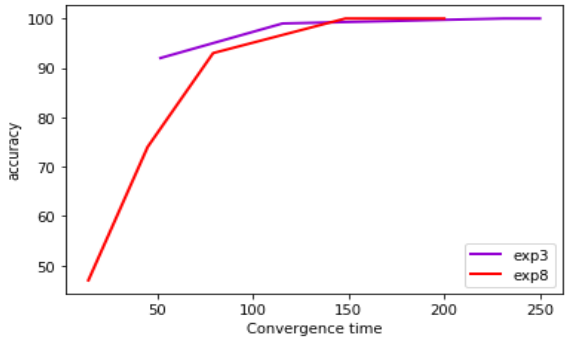

Figure 6. Performance of agent 2 using equation (4) environment 1 compared with agent 2 of experiment 8

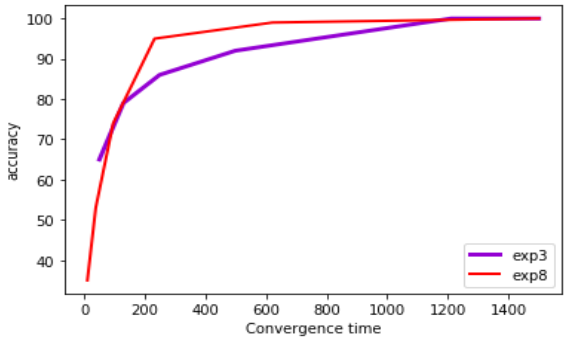

Figure 7. Performance of agent 1 using equation (3) environment 2 compared with agent 1 of experiment 8

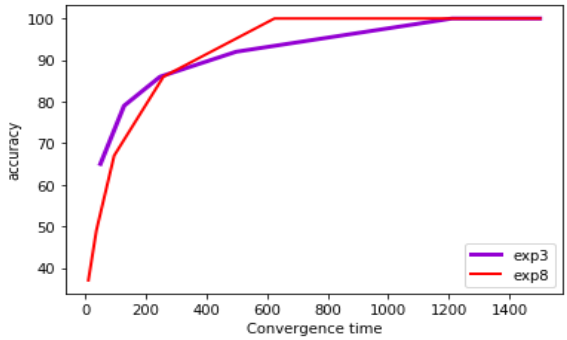

Figure 8. Performance of agent 2 using equation (4) environment 2 compared with agent 2 of experiment 8

From the figures above, it is seen that the purple line represents experiment 3 and the red line represents experiment 8. In the next section, the results of the remaining experiments are shown, just like the results of experiment 3. Instead of showing the results in the form of tables as well as graphs, only graphs have been included as they visualize the results better.

4.2.4 Experiment 4

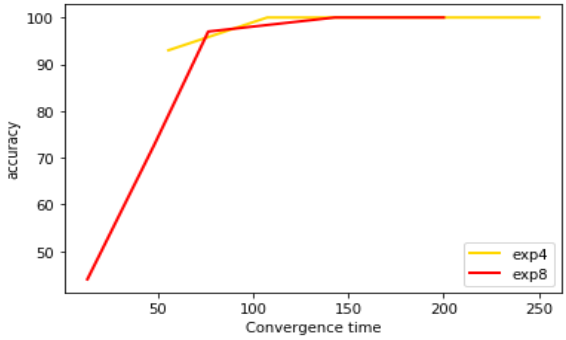

Figure 9. Performance of agent 1 using equation (5) environment 1 compared with agent 1 of experiment 8

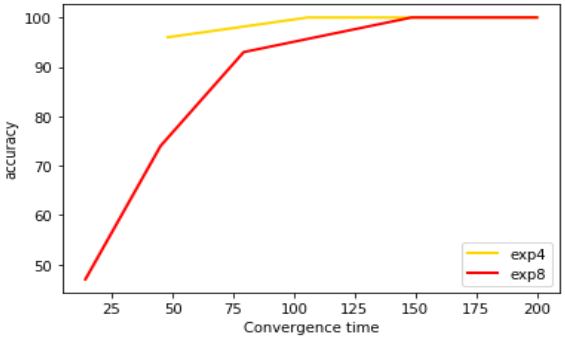

Figure 10. Performance of agent 2 using equation (6) environment 1 compared with agent 2 of experiment 8

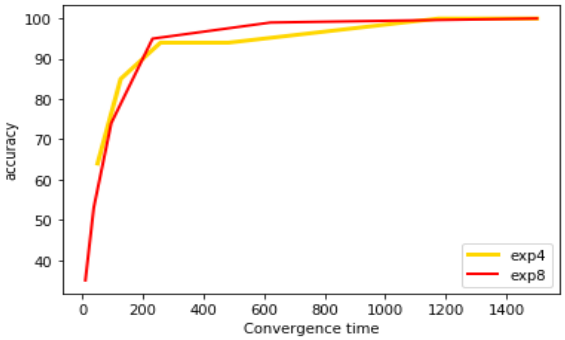

Figure 11. Performance of agent 1 using equation (5) environment 2 compared with agent 1 of experiment 8

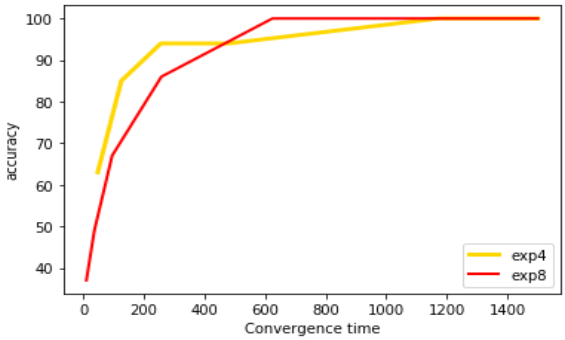

Figure 12. Performance of agent 2 using equation (6) environment 2 compared with agent 2 of experiment 8

4.2.5 Experiment 5

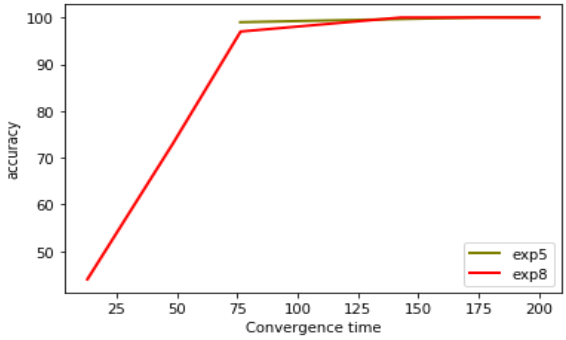

Figure 13. Performance of agent 1 using equation (7) environment 1 compared with agent 1 of experiment 8

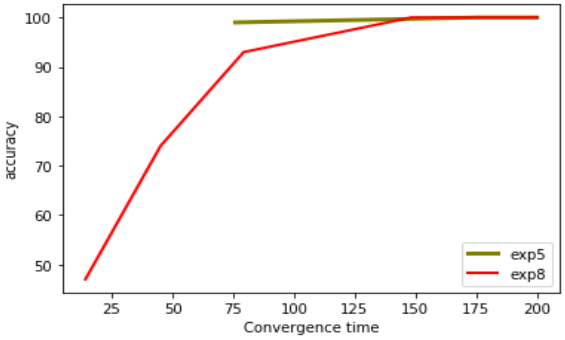

Figure 14. Performance of agent 2 using equation (8) environment 1 compared with agent 2 of experiment 8

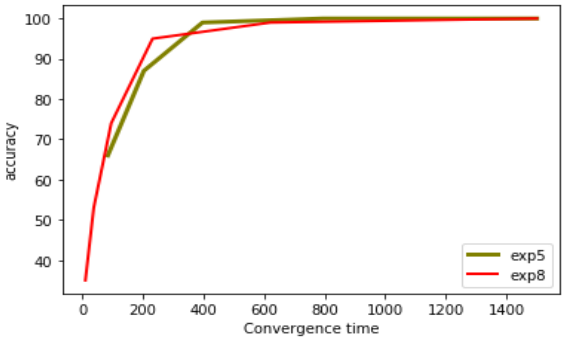

Figure 15. Performance of agent 1 using equation (7) environment 2 compared with agent 1 of experiment 8

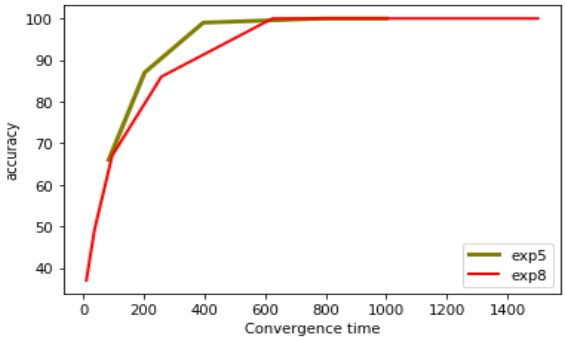

Figure 16. Performance of agent 2 using equation (8) environment 2 compared with agent 2 of experiment 8

4.2.6 Experiment 6

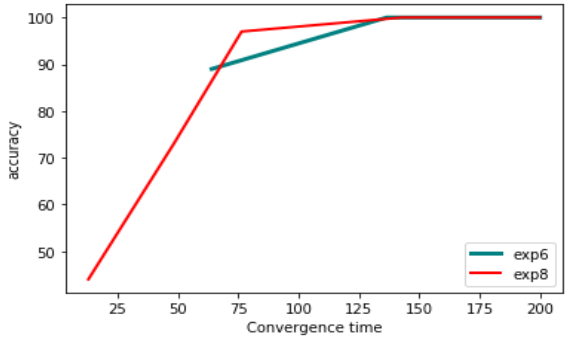

Figure 17. Performance of agent 1 using equation (9) environment 1 compared with agent 1 of experiment 8

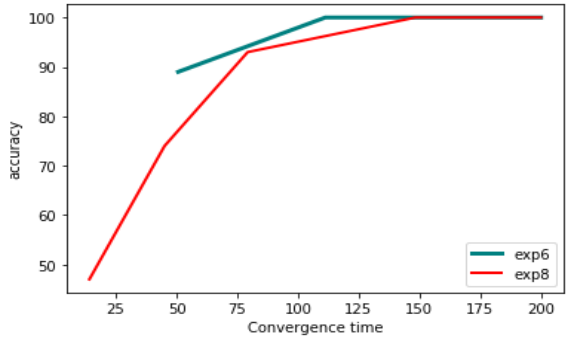

Figure 18. Performance of agent 2 using equation (10) environment 1 compared with agent 2 of experiment 8

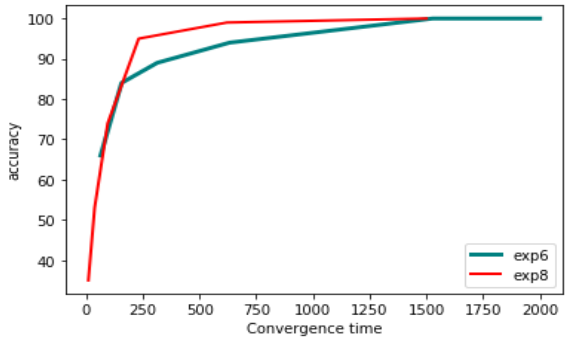

Figure 19. Performance of agent 1 using equation (9) environment 2 compared with agent 1 of experiment 8

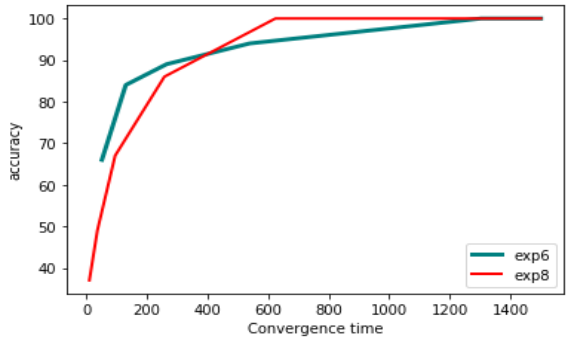

Figure 20. Performance of agent 2 using equation (10) environment 2 compared with agent 2 of experiment 8

From experiments 4 to 6, it's observed that maximum step size decreases and converges slower but with better accuracy.

4.2.7 Experiment 7

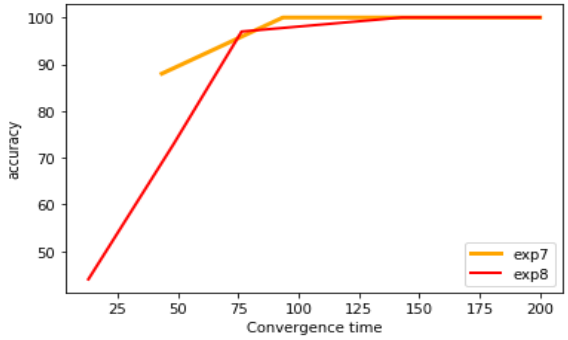

Figure 21. Performance of agent 1 using equation (11) environment 1 compared with agent 1 of experiment 8

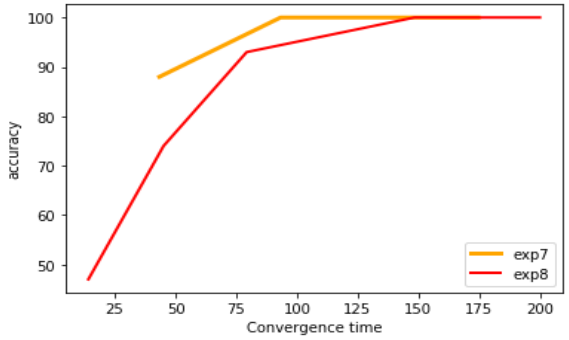

Figure 22. Performance of agent 2 using equation (12) environment 1 compared with agent 2 of experiment 8

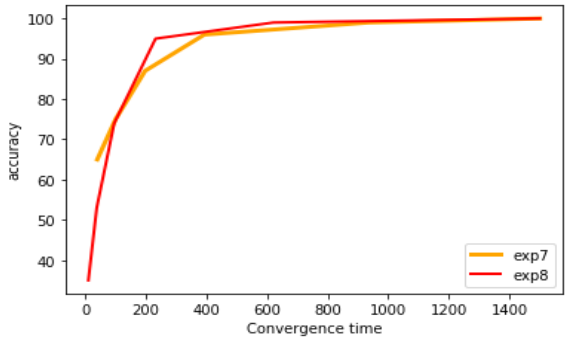

Figure 23. Performance of agent 1 using equation (11) environment 2 compared with agent 1 of experiment 8

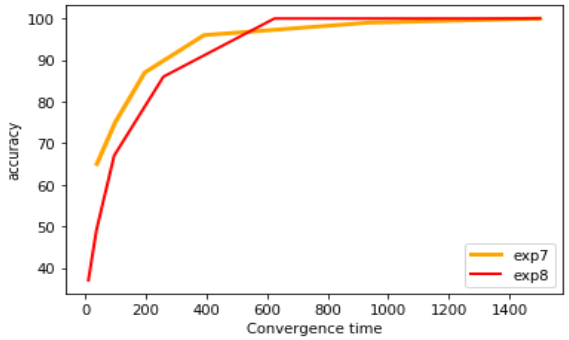

Figure 24. Performance of agent 2 using equation (12) environment 2 compared with agent 2 of experiment 8

4.2.8 Experiment 9

After performing eight experiments, the performance of learning agents for both the environments and all the step size equations were compared to realize that the step size equations 7 and 8 proved comparatively better than others for the tested environment. Results of experiment 5 display that the coupled model in experiment 5 learns and performs similar to experiment 8, which has a decoupled model. We went ahead with experiment 5 as the best simulation to be compared with experiment 9. The results of experiment 9 are as follows:

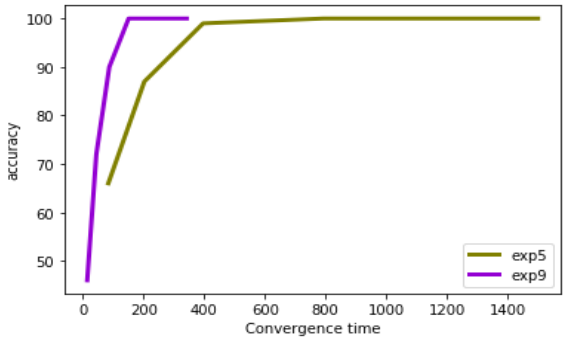

Figure 25. Comparison of experiment 5 with experiment 9

After the results of experiment 5 were compared with experiment 9, it showed a significant improvement in converging speed. Figure 25 shows that it took only around 150 iterations for experiment 9 to converge, unlike experiment 5, which took around 400 iterations to converge, which is more than double the number of iterations needed to converge for experiment 9.

4.3 Experiment III: Mutual learning with multiple environments

Figure 26 and Figure 27. show results of decoupled automata where the agents perform independently and converge to their respective optimal action in the presence of two environments, *reward_prob* and *reward_prob2*.

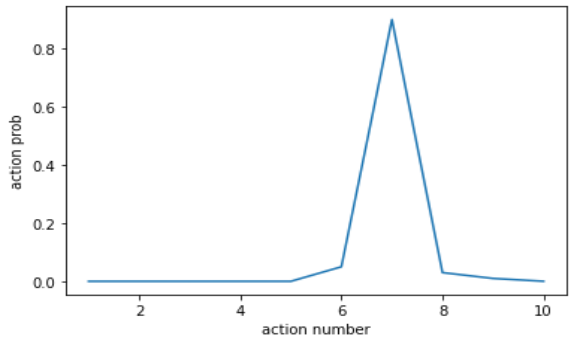

Figure 26. Agent 1

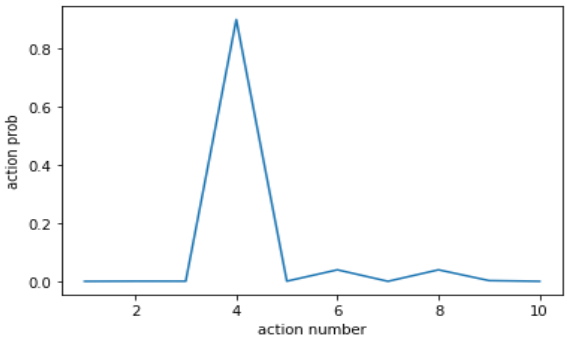

Figure 27. Agent 2

Here, agent 1 converged to action 7 with the highest reward probability, and agent 2 converged to action 4 with the highest reward probability. Figure 28 and Figure 29 show the results of coupled automata where the agents perform dependently as a coupled model and converge to action number 6, as a team. If either of the agents had chosen action with the highest probability,

it would have affected other agents and the overall decision. Even though the highest reward probability was not at action number 6; it was the optimal solution for both the agents working together. Thus, these two agents performed mutual learning and chose an optimal action using pareto optimality.

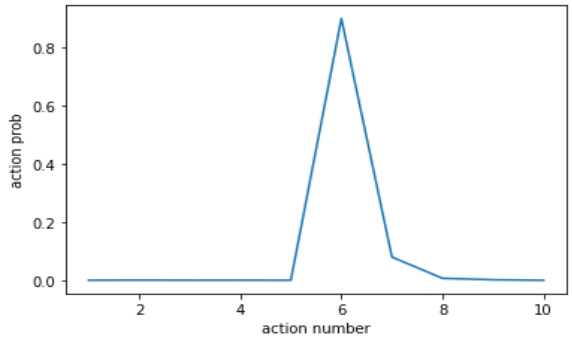

Figure 28. Agent 1

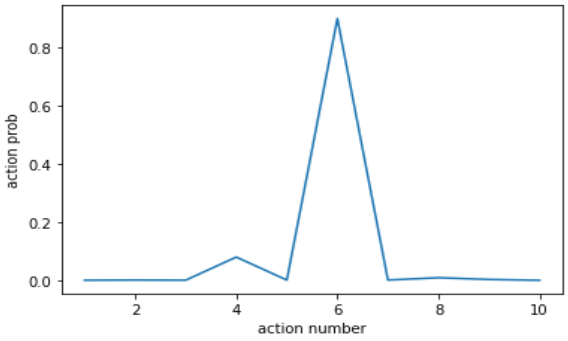

Figure 29. Agent 2

This experiment gives the optimal solution for coupled and decoupled automata. However, not always, it can be optimal, and for that, a leader-follower kind of situation is needed, where the two agents converge to the same action that is not Pareto optimal but is the leader's optimal action.

4.4 Experiment IV: Varying weight between self-learning and mutual learning

4.4.1 Experiment 1:

The weight is kept the same for both the agents and is varied from 0 to 1 to see where the transition takes place.

Table 11. Weight experiment 1 results

Weight	Convergence time (agent 1, agent 2)	Converged action (agent 1, agent 2)
0.0	917.75, 921.38	7, 4
0.2	1165.95, 1142.83	7, 4
0.4	1551.02, 1545.01	7, 4
0.6	2396.49, 2416.41	7, 4

Table 11. continued

Weight	Convergence time (agent 1, agent 2)	Converged action (agent 1, agent 2)
0.8	5982.12, 6661.75	7, 4
0.82	6923.0, 7701.84	7, 4
0.84	7332.4, 8770.88	6, 6
0.9	6145.1, 6190.37	6, 6
1.0	7939.11, 7936.59	6, 6

The learning agents perform isolated learning for the weight ranging from 0 to 0.82. Mutual learning is performed by the agents after the weight is varied from 0.84 to 1. The transition takes place at 0.84

4.4.2 Experiment 2:

Unlike the previous experiment, in this experiment, weight is kept constant at 0 for agent 2 and is varied from 0 to 1 for agent 1 to achieve a leader-follower situation.

Table 12. Weight experiment 2 results

Weight	Convergence time (agent 1, agent 2)	Converged action (agent 1, agent 2)
0, 0	899.54, 901.33	7, 4
0.2, 0	1153.79, 911.29	7, 4
0.4, 0	1539.36, 905.6	7, 4
0.6, 0	2320.69, 930.73	7, 4
0.8, 0	4922.33, 946.42	7, 4
0.9, 0	28768.41, 939.8	4, 4
1, 0	5178.3, 930.88	4, 4

The learning agents perform isolated learning for the weight ranging from 0 to 0.9. Mutual learning is performed by the agents after the weight is varied from 0.9 to 1. The transition takes place at 0.9, where agent 2 acts as a leader and agent 1 as a follower.

CHAPTER 5. APPLICATION TO FEWS PROBLEM

The NSF USDA funded INFEWS program is about accelerating innovation at the Nexus of Food, Energy, and Water Systems (FEWS). This program works with regional and local community farmers who are allotted water resources to cultivate crops. The farmers need to manage these resources in a way that not only produces good food but also considers its impact on the environment. This study aims to provide a multi-criteria decision-making (MCDM) model that helps identify a decision that satisfies the farmer's needs as well as keeps the environmental damage under control.[17][18]

5.1 Dataset

Table 13 shows the original dataset where Choice ID is the action set having total of 405 actions. Surface water amount, Groundwater amount, Crop ID, and Fertilizer ID are the decision variables. The details of decision variables are as follows:

Crop ID has three choices based on the prices of yield. C1 is $200/unit of yield; C2 is $500/unit of yield and C3 is $100/unit of yield. Three types of fertilizer ID are available to choose from. This decision variable acts as the energy in the FEWS problem. Irrigation amount is a combination of surface water (SW) and groundwater (GW) and has a total of 100 units. Profit and Environmental cost are the two criteria or environments of the FEWS problem. The original values of these criteria are shown in this table.

Table 13. Original Dataset

Choice ID	Surface water	Ground water	Crop ID	Fertilizer ID	Profit	Environmental Cost
1	10	10	1	1	3481.067993	3992.067993
2	10	10	1	2	2381.067993	4458.658867
3	10	10	1	3	2181.067993	2117.089341
4	10	10	2	1	26240.9532	3992.067993

Table 14. Processed Dataset

Choice ID	Surface water amount	Groundwater amount	Crop ID	Fertilizer ID	Profit	Environmental Cost	Profit	Environmental Cost
1	10	10	1	1	3481.067993	3992.067993	0.013953747	0.201557402
2	10	10	1	2	2381.067993	4458.658867	0.009544433	0.108235838
3	10	10	1	3	2181.067993	2117.089341	0.008742739	0.576566753
4	10	10	2	1	26240.9532	3992.067993	0.105185999	0.201557402
5	10	10	2	2	25140.9532	4458.658867	0.100776685	0.108235838
6	10	10	2	3	24940.9532	2117.089341	0.099974992	0.576566753

In Table 14, the last two columns have normalized values of profit and environmental cost in the range of 0 to 1: 1 being maximum profit and minimum environmental damage and 0 being minimum profit and maximum environmental damage.

5.2 Implementation

5.2.1 Experiment I: Reward shaping

Profit and environmental cost act as reward probabilities of two environments. The separation in these reward probabilities did not appear to be sufficiently high which could result in slow convergence. To avoid this, we applied Boltzmann Distribution on reward probabilities *d[i]* to get new reward probabilities $\bar{d}[i]$ using the formula:

$$\bar{d}[i] = \frac{e^{(\frac{d[i]}{T})}}{\sum_j e^{(\frac{d[j]}{T})}} \tag{13}$$

Here, T is a Temperature parameter.

Smaller values of T will amplify the separations, with $T \to 0$ in a limit corresponding to an impulse function whose 1 value is at *i*, which has the highest *d[i]*, with all the other *j's* corresponding to 0 after applying the transformation. Larger values of T will compress the separations, with $T \to \infty$, in a limit corresponding to a flat transformed value where the reward probability $\bar{d}[i]$ for all *i's* will be 1/N, where N is the number of actions.

5.2.2 Experiment II: Varying weights between mutual and self-learning

We conducted an experiment where we varied the relative weights between self-learning and mutual learning and studied the convergence. When the weight given to mutual learning is zero, each agent performs isolated learning and converges to their respective optimal action, whereas when the weight given to mutual learning is one, both the agents converge to the same action using pareto optimality. We varied the weight parameter from 0 to 1 to see where the transition takes place. Also, if the weights assigned to the agents are asymmetrical, the one having more weight to mutual learning acts as a leader and the other as a follower. Here, both the agents converge to the same action, which is the leader's optimal action.

5.3 Results

5.3.1 Experiment I: Reward shaping

For T = 0.05, it gave a reasonable separation in $\bar{d}$ value of the best action from the next best one. Fig.12. shows the original profit values, which are not well separated, whereas, in Fig.13. profit values are well separated, and negligible values are discarded.

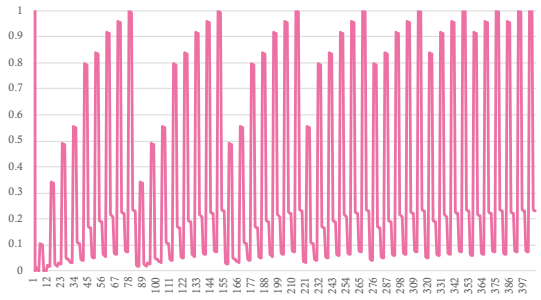

Figure 30. Profit values before reward shaping

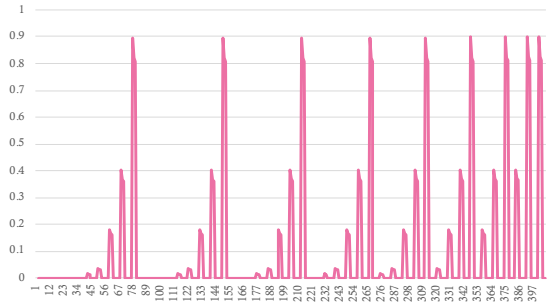

Figure 31. Profit values after reward shaping

The reward probabilities of the second criteria were already well separated, so it did not require reward shaping. The highest reward probability was 0.5, so we changed it to 0.9 and other values using normalization.

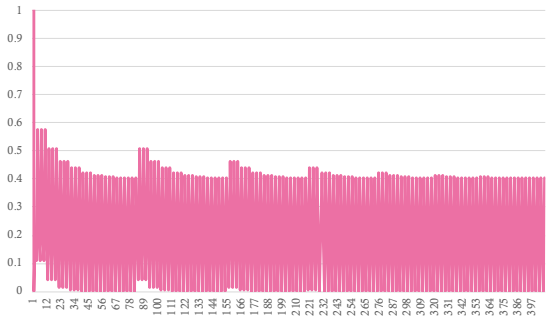

Figure 32. Environment cost before normalizing

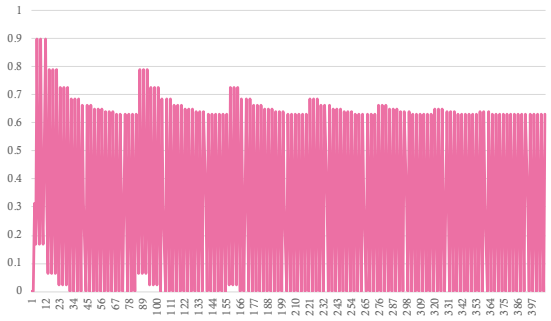

Figure 33. Environment cost after normalizing

5.3.2 Experiment II: Varying weights between mutual and self-learning

Table 15 shows the results of the weight experiment where the weight column shows the weight varied from 0 to 1. The second column shows which action both the agents converge to, and the last column shows the reward probability of the converged action. The agents performed isolated learning till the weight given was 0.92. Mutual learning took place at 0.94 where both the agents converged to action 78 with reward probabilities 0.8 and 0.62 respectively. Figure 34 shows a graph visualization of Table 15.

Table 15. FEWS weight experiment results

Weight	Converged action(agent1, agent2)	Reward probability(agent1, agent2)
0.0	400, 3	0.9, 0.9
0.2	76, 6	0.89, 0.9
0.4	346, 3	0.89 ,0.9
0.6	148, 6	0.89, 0.9
0.8	346, 3	0.89, 0.9
0.9	391, 6	0.89, 0.9
0.92	391, 6	0.89, 0.9
0.94	78, 78	0.8, 0.62
0.96	213, 213	0.8, 0.62
1.0	213, 213	0.8, 0.62

Note: agent 1 is profit, and agent2 is the environmental cost

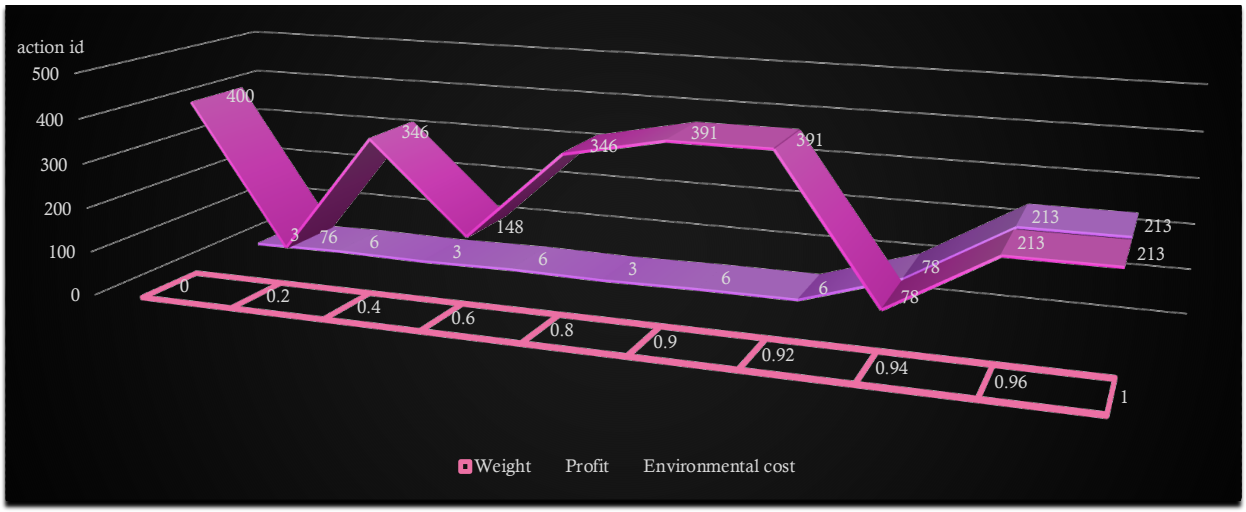

Figure 34. Mutual learning between two agents

Varying the weight between two agents helped in finding optimal solutions. The solutions can be calculated according to the actor's needs. If the agent wants a profit-oriented solution, the weight can be assigned in a way that the learning automata go through a leader-follower situation. Here, the mutual learning weight assigned to the profit agent should be more than that of the environmental cost agent so that the profit agent can act as a leader and environmental cost as a follower.

CHAPTER 6. CONCLUSION AND FUTURE SCOPE

In this paper, a reinforcement learning approach is used to form an MCDM model which helps in making better decision choices that aim to satisfy decision makers' preferences. The model uses a learning automata algorithm, which includes a learning agent that learns from its experience and from rewards and penalties provided in the form of feedback by the environment. The learning agent chooses an optimal solution based on the existing knowledge and by building new knowledge by exploring new actions and the rewards associated with them. We were able to succeed in finding an efficient algorithm that satisfies our requirement of maximizing its performance based on the rewards and finding a pareto optimal solution. We achieved this by using multiple learning agents that learn from each other in the presence of multiple environments and update their respective action probabilities to identify the optimal action.

We successfully implemented our model on the FEWS problem to achieve more accurate decision making when conflicting criteria were involved. The conflicting criteria used were profit and environmental cost which, were assigned with weights according to the decision maker's preference. Keeping the decision maker's priorities in the head, our model gave a pareto optimal solution that satisfied the requirements. The results of the FEWS problem prove that this model can be used as a decider or optimizer in large-scale multi-criteria problems. In the future, we plan on working on more complex criteria and increasing the number of actors to gain more understanding of the potential of this algorithm.

REFERENCES

[1] P. C. Fishburn, Additive Utilities with Incomplete Product Set: Applications to Priorities and Assignments, Operations Research Society of America (ORSA) Publication, Baltimore, MD, 1967.

[2] P.W. Bridgman, Dimensional Analysis, Yale University Press, New Haven, CN, 1922.

[3] Miller, D.W.; M.K. Starr (1969). Executive Decisions and Operations Research. Englewood Cliffs, NJ, U.S.A.: Prentice-Hall, Inc

[4] T. L. Saaty, The Analytic Hierarchy Process, McGraw-Hill International, New York, NY, 1980.

[5] Gwo-Hshiung Tzeng, Kao-Yi Shen, *"New Concepts and Trends of Hybrid Multiple Criteria Decision Making"* pp. 23-33, 2017

[6] Alain Petrowski, Farouk Aissanou, Ilham Benyahia, Sebastien Houcke, "Multicriteria Reinforcement Learning Based on a Russian Doll Method for Network Routing" *2010 5th IEEE International Conference Intelligent Systems. IEEE, 2010.*

[7] He, Zhenglei, et al. "A deep reinforcement learning based multi-criteria decision support system for optimizing textile chemical process." *Computers in Industry* 125 (2021): 103373.

[8] Sriraam Natarajan and Prasad Tadepalli. 2005. "Dynamic preferences in multi-criteria reinforcement learning." *In Proceedings of the 22nd international conference on Machine learning (ICML '05)*. Association for Computing Machinery, New York, NY, USA, 601–608.

[9] Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." Journal of artificial intelligence research 4 (1996): 237-285.

[10] Szepesvári, Csaba. "Algorithms for reinforcement learning." Synthesis lectures on artificial intelligence and machine learning 4.1 (2010): 1-103.

[11] Parag Kulkarni *"Reinforcement and Systemic Machine Learning for Decision Making"* pp. 1-17, 2012

[12] M. A. L. Thathachar and P. S. Sastry, "A New Approach to the Design of Reinforcement Schemes for Learning Automata" *IEEE Transactions on Systems, Man, and Cybernetics, Vol. Smc-15, No. 1, January/February 1985*

[13] K. S. Narendra and M. A. L. Thathachar, *"Learning Automata: An Introduction."* Prentice-Hall, 1989

[14] Kumpati S. Narendra and Kannan Parthasarathy, "Learning Automata Approach to Hierarchical Multiobjective Analysis" *IEEE Transactions on Systems, Man, and Cybernetics, Vol. 21, No. 1, January/February 1991*

[15] Ficici, Sevan G., and Jordan B. Pollack. "Pareto optimality in coevolutionary learning." *European Conference on Artificial Life*. Springer, Berlin, Heidelberg, 2001.

[16] Kulturel-Konak, Sadan, Abdullah Konak, and David W. Coit. "Multiobjective metaheuristic approaches to reliability optimization." *Computational Intelligence in Reliability Engineering.* Springer, Berlin, Heidelberg, 2007. 37-62.

[17] Uslu, Suleyman, Yefeng Ruan, and Arjan Durresi. "Trust-based decision support system for planning among food-energy-water actors." Conference on Complex, Intelligent, and Software Intensive Systems. Springer, Cham, 2018.

[18] Piemonti, A. D., Babbar-Sebens, M., Mukhopadhyay, S., & Kleinberg, A. "Interactive genetic algorithm for user-centered design of distributed conservation practices in a watershed: An examination of user preferences in objective space and user behavior." Water Resources Research, 53(5) (2017): 4303-4326.