DYNAMIC SPECTRUM ACCESS UTILIZING NEURAL NETWORKS AND PARTICLE SWARM OPTIMIZATION IN COGNITIVE RADIOS

By

CAROLINA ARBONA

Bachelor of Science in Electrical Engineering

Bachelor of Science in Computer Engineering

Oklahoma State University

Stillwater, OK

2012

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2018

DYNAMIC SPECTRUM ACCESS UTILIZING

NEURAL NETWORKS AND PARTICLE SWARM

OPTIMIZATION IN COGNITIVE RADIOS

Thesis Approved:

Dr. George Scheets

Thesis Adviser

Dr. Gary Yen

Dr. Weihua Sheng

ACKNOWLEDGEMENTS

First, I am thankful for my advisor, Dr. George Scheets, who stuck with me through all the ups and downs over the years and for staying on after retirement to see my thesis through. Without his support and review this work would never have been completed. Dr. Scheets was the most influential professor in my college education as he offered the majority of the communications based courses that I was interested in. Some of the best advice I've ever gotten was 'keep it simple', and it came from him.

I would also like to acknowledge Dr. Gary Yen, whose course in computational intelligence was one of the most interesting I've ever taken. I still remember elements from the lectures despite it being 4 years ago. His passion for the subject seeped through in every lesson and into this thesis.

My mother, Gloria Ossa, deserves all my love and gratitude. My mother insisted I pursue my masters and thesis to completion and was understanding of my not coming to visit her often enough. A mother's love is unconditional; so much so that she proofread chapters 1 and 2 for grammar & flow even though English is her second language and her background is in nursing. I would also like to acknowledge the home cooked meals she'd bring me when visiting.

I would like to recognize Ian Finley for proofreading the more technical chapters. I am forever grateful that Ian designed and built me a new computer after he identified my old one was not up to the task of running the experiments in this thesis. I appreciate Ian's thoughtful questions that would often help get my gears moving again after being stuck as well as his patience and positive attitude that kept me going when I wanted to quit.

Another acknowledgment goes my co-worker and friend, Lars Fetzek, for being of curious mind and offering to proof-read chapters 1-3 in exchange for being allowed to read my research.

Finally, I am thankful for my friends & family who were understanding of my schedule and appreciative of my co-workers who were kind and accommodating whenever I needed to take vacation for school.

Acknowledgements reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

Name: CAROLINA ARBONA

Date of Degree: JULY, 2018

Title of Study: DYNAMIC SPECTRUM ACCESS UTILIZING NEURAL NETWORKS AND PARTICLE SWARM OPTIMIZATION IN COGNITIVE RADIOS

Major Field: ELECTRICAL ENGINEERING

Abstract: In recent years, spectrum policy has shifted and has become more accepting of dynamic spectrum access. Technology improvements such as software defined radio and cognitive radio has allowed research to explore more into this field. Software-defined radio has the ability to switch the transmission parameters, such as modulation schemes; Cognitive Radio extends software defined radio by giving it a brain to decide which of those parameters to choose. Spectrum management can be broken down into four steps: spectrum sensing, spectrum decision, spectrum sharing and spectrum mobility. This paper will assume that the spectrum sensing is already known and is only interested in the spectrum decision aspect. The author introduces a hybrid approach of an Artificial Neural Network (ANN) which is trained by another biologically inspired algorithm: Particle Swarm Optimization (PSO) for the spectrum decision aspect. ANNs were chosen due to their universal function approximation capabilities. PSO was chosen to train the ANN given its fast convergence and ability to search a complex and non-continuous state-space. The PSO-ANN hybrid technique will be compared to game theoretic 'Regret-Based Learning' procedures. This thesis seeks to gain a better comprehension of learning schemes used in the creation of the cognitive engine and shed knowledge on the decision capabilities through data derived from a decentralized system. This research was able to find acceptable PSO-ANN configurations that boasted improved utilities with respect to 'Regret Tracking'. These improved outcomes were reached faster, with convergence occurring quickly. This phenomenon occurred in both static and dynamic situations and for congested and non-congested environments. Regret tracking has faster computational speeds for small simplified networks but scales very poorly for more system channels or more channels allowed per secondary users, with both speed and system memory being of concern. Regret tracking was found to require evaluation and therefore sensing of every option whereas PSO-ANN only evaluates a handful of particle options. The decrease sensing needs of PSO-ANN make it much more practical for implementation. Due to all the limitations in regret tracking that PSO-ANN overcomes, it should be a preferred choice when working with dynamic spectrum access.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER I

I. INTRODUCTION

In this era of information and mobility, society has adapted and developed a need for interconnectedness and communication. With the advent of new technologies in process automation, commerce, safety, and more, both private industry and government agencies that require connectivity were experiencing 'spectrum drought'. To better understand this situation, here defined as the "scarcity of a seemingly infinite resource", a review of spectrum management policies is needed. For the purposes of this thesis only United States policies will be covered, understanding that other countries have other agencies and may have developed other methods.

In the United States two federal agencies are responsible for spectrum management policies. One of them is the Department of Commerce's National Telecommunications and Information Administration (NTIA), which allocates spectrum for government use. The other one is the Federal Communications Commission, FCC, which currently manages spectrum by assigning frequencies to non-governmental license holders.

Historically, the FCC has used many means to assign spectrum. From 1934 to 1984, the FCC allocated these bands through comparative hearings where the parties claimed why they needed the license. Then, from 1984 to 1993 the method changed to utilize lotteries where qualified applicants were pooled together and the winners were randomly selected. After 1993 the FCC moved to competitive bidding where participants bid for the spectrum through auctions and the highest bidder wins [1].

In general, FCC policies have been driven by the technology of the day as well as the needs of consumers and government agencies. Overall, these methods were static and are now known to be inefficient; and moreover, the allocations were set to last for long periods of time.

On May 28, 2004 the US Government Accountability Office released a report in which it showed the results of a study on spectrum efficiency. This report recognized that many technologies were geared towards improving the quality of communications or throughput by using smaller amounts of available spectrum; furthermore, it cited the importance and encouraged investment in new technologies as follows:

"Technologies like software-defined cognitive radios can be adapted to operate in virtually any segment of spectrum and, in the future, may be able to adapt to real-time conditions and make use of underutilized spectrum in a given location and time. [2]"

In July of 2012 yet another report came out, this one from the President's Council of Advisors on Science and Technology (PCAST) [3]. PCAST labeled the auction system as inefficient and identified gains in data transmission by utilizing shared spectrum access that co-exists with legacy infrastructure. The PCAST report recommended working with the existing agencies to free up spectrum for sharing and encouraged research into spectrum management technology and practice.

The United States government listened to these recommendations [4, 5] and several "beachfront" frequency bands were identified and freed to ameliorate the situation of spectrum scarcity and stimulate the economy. Additionally, the FCC is now motivating research into spectrum sharing and the use of white spaces in order to maintain international leadership in innovation.

1.1 Motivation

Researching the spectrum decision portion of cognitive radio for use in opportunistic spectrum access is an intriguing optimization problem due to the many real world applications.

According to Zhao and Sadler [6], Dynamic Spectrum Access can be classified into three categories. First, 'Dynamic Exclusive Use Model', in which the licensed user can sublet resources, for which economy and market values may play a role. Second, 'Open Sharing Model', in which all users have equal priority in sharing the spectrum. And third, 'Hierarchical Access Model', in which the idle spectrum is shared so long as interference to license holders is minimal.

The FCC and NTIA may be open to changing policies to any or all of the above categories depending on the original license holder and its needs. A cell phone company looking to divvy up its spectrum at base stations could follow a hierarchical access model for users who pay for higher or lower speeds. The same company could instead use a dynamic exclusive use model in which the prime carriers sublet to smaller carriers. 'Internet of Things" (IOT), could follow an open sharing model on an unlicensed band or a hierarchical access model on allocated bands.

John Polson, an engineer with Bell Helicopter Textron states that all of the pieces for making the leap from software radios to cognitive radios already exist. Such pieces include location services, spectrum sensing and analysis, regulations databases, etc. The biggest challenge is designing clever algorithms to take available information and make decisions about where to operate. [7]

This thesis presents a novel approach to spectrum decision in dynamic spectrum access that applies neural networks trained with Particle Swarm Optimization. PSO-ANN was applied to a network model obtained from a published article [8] and a chapter in the book 'Cognitive Wireless Communication Networks' [9]. This model was chosen due to its decentralized collaborative approach and high-level nature. PSO-ANN is then compared to the current solutions provided for that model to further research into this complex optimization problem.

CHAPTER II

II. REVIEW OF LITERATURE

This chapter will delve into the theoretical framework behind this thesis as well as related works within these fields. The first topic considered will be cognitive radios and spectrum management. Afterwards, a brief overview of computational intelligence will be provided along with some tie-ins to game theory that are essential in multi-agent systems. Following these concepts, this chapter will focus on a subset of computational intelligence, namely neural networks. This chapter will first cover the theory, then provide an overview of the traditional training methods. This will clarify why an alternative is worth exploring. Finally, this chapter will pivot to Particle Swarm Optimization, which is the training method proposed for the neural network.

The related work will first cover the research in [8, 9] that provided the starting point for this thesis. In addition, the model proposed in that research will be used in this thesis, as well as the algorithm, as comparison for the neural network particle swarm hybrid proposed. This topic will be further expanded upon in chapter 3. A few other recent DSA works are covered before moving onto computational intelligence. The beginnings of PSO-ANN and the evolution towards a more generalized and robust algorithm is covered. The section concludes with a mention of a company presently doing research in this field.

2.1 Dynamic Spectrum Access and Cognitive Radios

Cognitive radios have emerged as part of a solution to spectrum scarcity. A software-defined radio can reconfigure transmission parameters via software. Cognitive radio is built on the software-defined radio platform but with awareness of its surroundings and the ability to learn. Cognitive radios rearrange transmission parameters intelligently and autonomously; in layman's terms, it is a software defined radio with a brain. Some other things a cognitive radio must do include managing its resources as well as sharing the spectrum with other users.

This thesis focuses on Opportunistic Dynamic Spectrum Access in cognitive radio, which is also known as spectrum overlay. Opportunistic access occurs when a spectrum hole is sensed and used by an unlicensed user. Spectrum holes are sometimes referred to as white or gray spaces. The FCC definition of a white space is that where portions of the radio spectrum may not be used for significant periods of time; these spaces may be temporal or geographic. An example of a temporal white space is a band that lies unused at night but is active throughout the day. An example of a geographic white space is a band that is used heavily in urban areas but is unused in rural areas. The definition of a gray space is "areas where emissions exist but could accommodate additional users without raising the overall noise level in a band to a level unacceptable to incumbent users – to increase spectrum efficiency." [10] An example of a spectrum hole can be seen in the figure below.

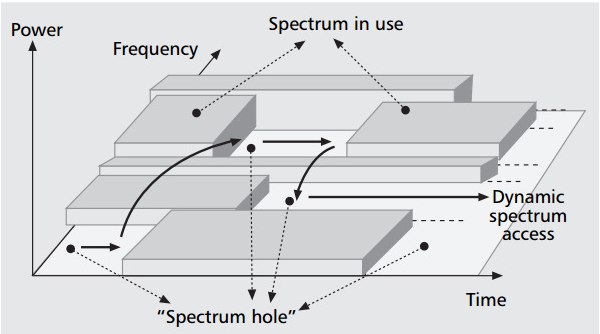

Figure 1: Spectrum Hole Concept [11]

Those with access or a license to the spectrum are known as primary users. These have typically paid for the use of this resource and the corresponding assignment has been made by a regulating entity. Users that are un-licensed and do not have guaranteed rights to a portion of the spectrum but have a need to communicate will be referred to as secondary users. Because primary users own the resource, it is extremely important that they achieve all the communications desired and that they are unaffected by the actions of secondary users. The premise is that if a secondary user is on a band and it senses transmission from a primary user, said user must immediately evacuate and resume transmission elsewhere; this is known as pre-empting. For the purpose of this research, all secondary users are assumed to have the same access rights and thus cannot pre-empt each other.

According to I. F. Akyildiz, et al [11], there are four major steps to a cognitive radio and spectrum management; these are defined below:

- Spectrum Sensing: Consists of monitoring the spectrum to determine the availability of white spaces and the operation patterns of users. Sensing will determine interference and channel characteristics. This stage is paramount to avoiding collision with a primary user.
- Spectrum Decision: In this step, the radio learns the characteristics of certain bands, as determined in the spectrum sensing stage. It then decides what parameters to reconfigure in order to broadcast more efficiently. The decision may involve multiple transmission bands or parameters. A cognitive radio may have multiple configurations for simultaneous use depending on its hardware buildout.
- Spectrum Sharing: It is important to be aware that there may be many secondary users who also need access. Greedy behaviors may result in more interference and possibly jamming.

  The first consideration is the network. In some cases, there may be a centralized model where all secondary users report to a central entity such as a base station or spectrum

broker. In such a case, the central authority allocates and shares between all of the users. Another option is an independent or decentralized model where each secondary user decides how to share the spectrum without external input. This situation presents a challenge, as it can lead to an overall destructive environment if the user acts too greedily.

The next consideration is the need for cooperation. In a cooperative environment, various secondary users may communicate with each other to share information and intentions. In a non-cooperative environment, there are savings from message exchanges that result in less power/resource utilization. Again, this is at the expense of explicit information sharing and coordination.

The last consideration is the sharing technique. In Spectrum Overlay, only white and gray spaces are utilized. In Spectrum Underlay, spread spectrum is used so that the noise levels seen by primary users are tolerable.

- Spectrum Mobility: The final step is to be able to move when the need arises, temporally or spatially. A temporal example is the detection of and pre-empting by a primary user. Another temporal example is when transmission conditions such as channel qualities, change for better or for worse. As mentioned before, the radio must also efficiently accommodate the quick changes in spectrum and characteristics that may result from motion in space.

As can be seen, there are many factors that must be taken into account to automate spectrum allocation. This thesis will focus specifically on computational intelligence for the spectrum decision step in an independent model. The spectrum sensing portion will be assumed and simplified. Spectrum sharing will occur through a fitness function that balances greedy behavior and decentralized collaboration. Finally, this thesis will consider a dynamic case with varying transmission conditions to better appreciate how well the method performs for spectrum mobility.

2.2 Computational Intelligence

Computational Intelligence is an all-encompassing term for non-organic creatures (machines, computer programs, etc.) that learn from data or trials. Sometimes problems can be solved through intricate decision trees and logical instructions, known as hard coding. In recent years there has been a creative explosion of soft-coding methods to address the problems that cannot be readily solved by a perfect set of instructions provided by a human. These methods may use probability, statistics and nature-inspired motifs. A few examples:

- Neural Networks are very popular and are based on the concept of interconnected neurons/perceptrons that mimic a biological brain.
- Particle Swarm Optimization bases its search for a solution within a state space upon the efficient behavior of bird flocking patterns.
- Genetic Algorithms utilize the concept of survival of the fittest. Parameters or genes are passed from parents to children in the search for better solutions.

There are many more such algorithms, as this is not meant to be an exhaustive list.

Although soft-coding is a very powerful tool, it is also known to be quite computationally intense. The argument can be made, however, that computing power is growing exponentially. Thus this branch of computing is still worth researching.

These algorithms are adept at recognizing patterns, searching a data space, and making decisions. In order to utilize those functionalities, a suitable model must be obtained. Sometimes models are simple but sometimes more advanced situations with multiple players and multiple interests require more complex models.

Game theory studies multi-agent environments where rational players choose strategies with varying degrees of competition or collaboration. Game theory can take day to day situations and

simplify them to ‘games’: zero sum games, prisoner’s dilemma, etc. Intuitively it can be noted that game theory may have great potential in the cognitive radio field. This intuition is demonstrated in an IEEE Spectrum magazine article by K.J. Ray Liu [12] from the University of Maryland. In this article, Liu goes over several game-theoretic approaches to cognitive radio, including a wireless-networking version of the aforementioned prisoner’s dilemma.

Game theory has many procedures based upon expected values and decision trees that are mathematically proven to converge to equilibria, a bit more aligned with hard-coding sprinkled with stochastics. Machine learning is very good at analyzing and classifying data but, as mentioned, could be improved for the multi-agent environments. As such, machine learning can benefit from game theoretic models and game theory could explore soft-coding techniques used in machine learning. By putting these two methods together, great strides could potentially be made towards furthering computational intelligence in cognitive radio.

Supervised learning is when previous knowledge of the problem is given. The system outputs are compared to those expected and an error is calculated and used iteratively to adjust parameters. Conversely, unsupervised learning is used when dealing with an unlabeled dataset. Usually, unsupervised algorithms train to group the data by likeness or distance. In reinforcement learning, a possible solution will attempt the problem but will not be told the correct way to do it as there is not a dedicated training set for it. Instead, the solution will be rewarded if it did well according to a fitness function. Supervised learning seeks to minimize the error while reinforcement learning is training to maximize the reward. In reinforcement learning, an explicit desired solution is not provided, identified or labelled as in supervised learning. There has been significant research towards supervised learning, but because training data may not always be available in real-world scenarios, this research will train ‘online’ through reinforcement learning of a cognitive radio’s fitness function. The proposed solution is a hybrid approach, of an artificial neural network (ANN) trained with particle swarm optimization (PSO).

2.2.1 Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs) were inspired by the human brain and try to mimic its biology by creating a group of interconnected neurons or perceptrons. They are best used when a linear relationship between inputs and outputs doesn't exist or isn't readily observable, as they are known for being universal function approximators. ANNs consist of different layers in which data are manipulated. There are three different types of layers: input, hidden and output. Most of the calculations are executed in the hidden layers and are not shown to the final user. Shallow neural networks are those with only one or two hidden layers and 'deep neural networks' describe ANNs with many hidden layers. The more layers in an ANN, the better it is able to handle non-linearity but also the higher the risk to over-fit data.

Perceptrons are the units or individual neurons that make up the network. In feedforward ANNs, perceptrons are connected through signals coming in from the layer strictly before it. An example is shown below. Please note the number neurons per layer may vary. In recurrent ANNs, a signal can come from any neuron and closed paths are often observed when depicting these. Though very powerful, recurrent ANNs and other advanced ANNs are more difficult to train and will not be the subject of discussion in this project.

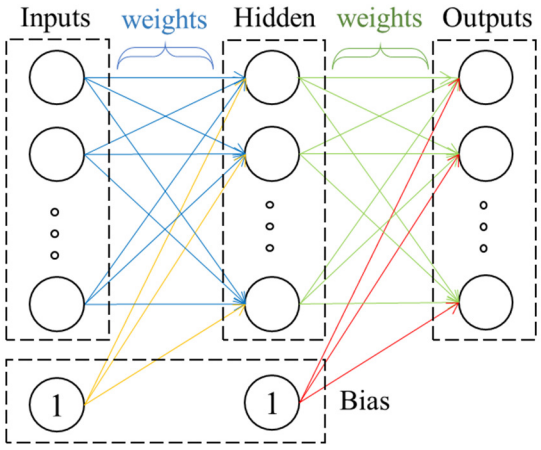

Figure 2: Simple representation of a 3-layer Feed-Forward ANN

A perceptron provides a single output, *the state of activation,* that it determines from a weighted sum of its inputs via its transfer function. As shown in figure 3, some popular transfer functions include hard limiter, logarithmic (log-) sigmoid, hyperbolic tangent (TanH-) sigmoid, and rectified linear unit (ReLU).

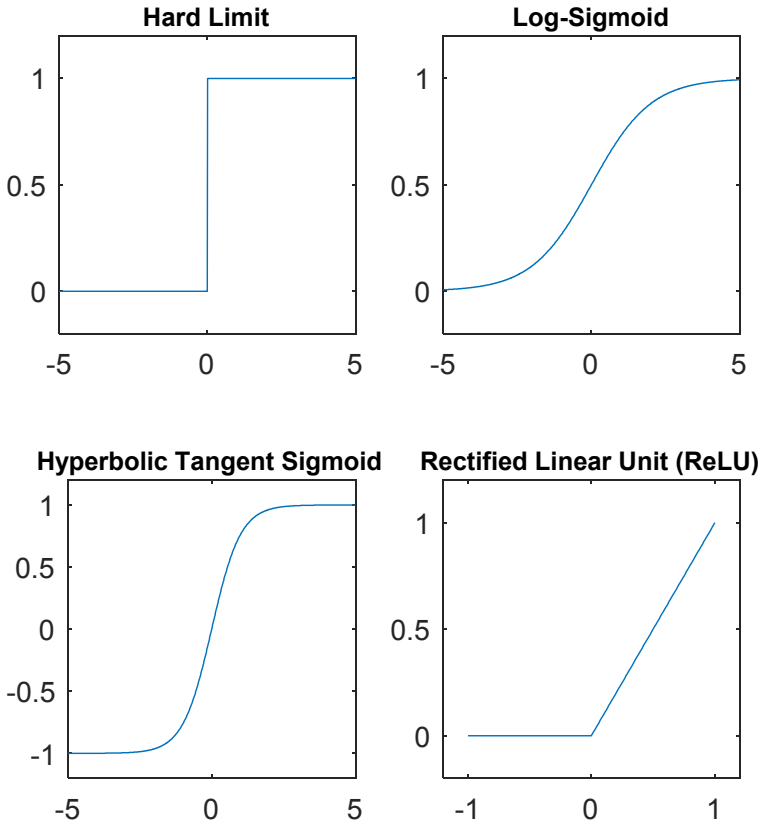

Figure 3: Popular Transfer Functions used in perceptrons

Function Name	Mathematical Expression
Hard Limit	Y = 1 if X > 0; Y = 0 otherwise;
Log-Sigmoid	$Y = 1/(1+e^{-X})$;
TanH-Sigmoid	$Y = (e^{X} - e^{-X}) / (e^{X} + e^{-X})$;
Rectified Linear Unit (ReLU)	Y = max(X, 0);

Table 1: ANN Transfer Functions' Mathematical Expressions

The hard limit function has a jump at the zero point, and therefore has a discontinuous derivative. Log-sigmoid and TanH-sigmoid are similar except that the former has an output range of (0,1) and the latter has an output range of (-1, 1); compared to log-sigmoid, the hyperbolic tangent can be thought of as a 'taller S' [13]. Both the log-sigmoid and hyperbolic tangent have continuous derivatives but are not particularly well behaved: near zero the derivative is quite high and towards the plateau of the 'S' the derivative is infinitesimally small; this leads to the vanishing or exploding gradient problem. Some problems, such as classification, are better suited towards sigmoid whereas others prefer the zero-centered output provided by hyperbolic tangent. Many developers prefer ReLU, because it is more simple and thus less computationally intense than the others. The ReLU derivative is mostly well behaved, though some have sought to improve upon ReLU by providing variants that have a non-zero derivative for $x<0$. Many more transfer functions exist, but this thesis considers only some of the more popular and simple ones. If computational resources are not an issue, it is worth exploring some more powerful ones [14].

ANNs require training of the weight values for the various inputs. A popular training mechanism is back propagation. Back propagation calculates an error or cost function and then propagates it backwards from outputs to inputs, adjusting the weights as it goes. It makes the adjustments based upon gradient descent and therefore may have issues such as the above-mentioned vanishing or exploding gradient, putting strain on the activation functions to be chosen. Traditional training methods may require pre-processing of the inputs to filter out those with correlation and to generalize. This is time consuming (even for an expert) but if not done, the ANN will take longer to train.

Research in neuroevolution is helping to overcome the shortcomings in gradient descent training algorithms. In this research field, ANNs are trained by evolutionary algorithms such as genetic algorithms, particle swarm, ant colony optimization, etc. One of the benefits of neuroevolution is the use of an algorithm that does not require a differentiable transfer function (let alone a 'well-

behaved' continuous derivative). Another advantage is the ability to learn the topology, best fitness function selection, and input features. By removing human intervention, the creation of ANNs can be more automatic [15]. This self-learning is an exciting step towards artificial intelligence.

Neural networks have been used with great success in many applications. Natural language processing uses a variety of neural network types to interpret what humans mean in speech and written language. Image processing and computer vision have advanced significantly by the use of convolutional neural networks. The application of neural networks to economic prediction and forecasting has been extensively studied, especially regarding stock markets and cryptocurrencies. In recent years, there has been an explosion of research in neural networks and reinforcement learning for decision making in game play, self-driving cars and automata/robotics.

However, ANNs should not be considered a panacea. These tend to be more computationally intense and may be too much effort if a problem can be solved with a straightforward algorithm. Evolutionary ANNs should also not be utilized indiscriminately or on all parameters. If a researcher has *a priori* knowledge that they can express in code, then it is worthwhile to do so. By reducing the state-space to be searched, a model can be trained with fewer iterations and less computation. Consequently, this thesis will make use of available information to pre-process the inputs and produce the ANN topology shown in chapter three. The evolutionary algorithm will be used for weight training only. However, if this method needs to be generalized for another problem/model with less *a priori* knowledge, then the groundwork for extensive training exists.

2.2.2 Particle Swarm Optimization

PSO is an optimization algorithm inspired by the patterns observed in the flocking of birds and other wildlife. It is a simple yet powerful technique that boasts fast computation time and good scalability while being easy to comprehend. This technique initializes a pre-defined number of particles with random initial velocities to make up the population and search the state space. Each particle contains positional data from the objective space and will map onto the decision space through the use of a fitness function. Each particle does parallel computations to search for the global optimum and will update its position after each iteration according to its velocity, per the formulas:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

$$v_i(t+1) = a * v_i(t) + b * r_1\left(x_{pb}(t) - x_i(t)\right) + c * r_2\left(x_{gb}(t) - x_i(t)\right)$$

Variable	Meaning
a	Inertia Coefficient
b, c	Acceleration coefficients
x_{ib}	position of personal best
x_{gb}	position of global best.
r1, r2	uniformly distributed RV from (0, 1)

Table 2: PSO Variable Explanations

In some variations, the values of a, b, and c above may progress linearly or exponentially over the span of time, sometimes expressed as an 'annealing' factor. For example, the inertia coefficient could be closer to one at the beginning and near zero at the end for exploration. Alternatively, the acceleration coefficients may start near zero and end near one to improve exploitation of solutions found. In yet other variations, the global best is replaced by a local best. The local best is from a

smaller group of particles than the whole population as used by the global best. The local best may look at the k-nearest neighbors or follow a topology such as ring, star, etc.

Particle swarm is exciting because it overcomes the problem of getting stuck at local minima or local maxima, which tends to occur for other searching algorithms such as gradient descent. The use of the random variables r1 and r2 introduces an element of uncertainty to better explore the objective space. The initial velocity also provides some "inertia" to the particle to keep exploring even when plateauing on minima or maxima. A graphical explanation of the position update is shown in the following figure.

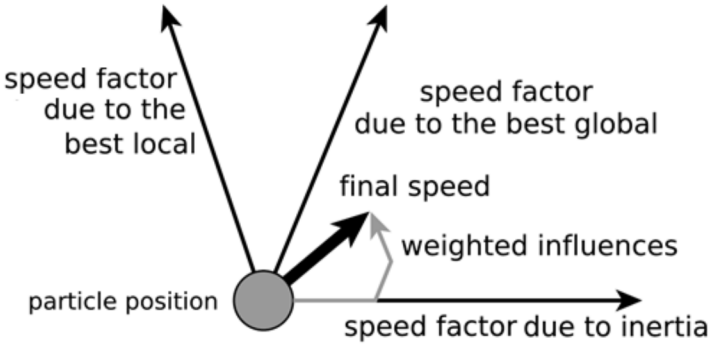

Figure 4: PSO Velocity Vector Analysis.

When the position is updated, the new fitness values are calculated. If any of the new positions have generated new personal bests, then these are stored accordingly. The same is done with the global best. The algorithm will come to an end when the stopping conditions are met. Usually stopping criteria are a maximum number of iterations or convergence, but CPU resources or the developer's patience may also influence stopping.

On occasion a set of particles may be prematurely converging to local minima or maxima as a result of initialization or the attraction towards the local/global bests. In that case, it is useful to 'mutate' the worst performers by re-initializing them with new positions and velocities to continue the search.

In summary, PSO utilizes random variables, inertia speed and mutations for exploration. Global/personal best position tracking is used for exploitation of good solutions and convergence. The figure below illustrates PSO in flowchart form.

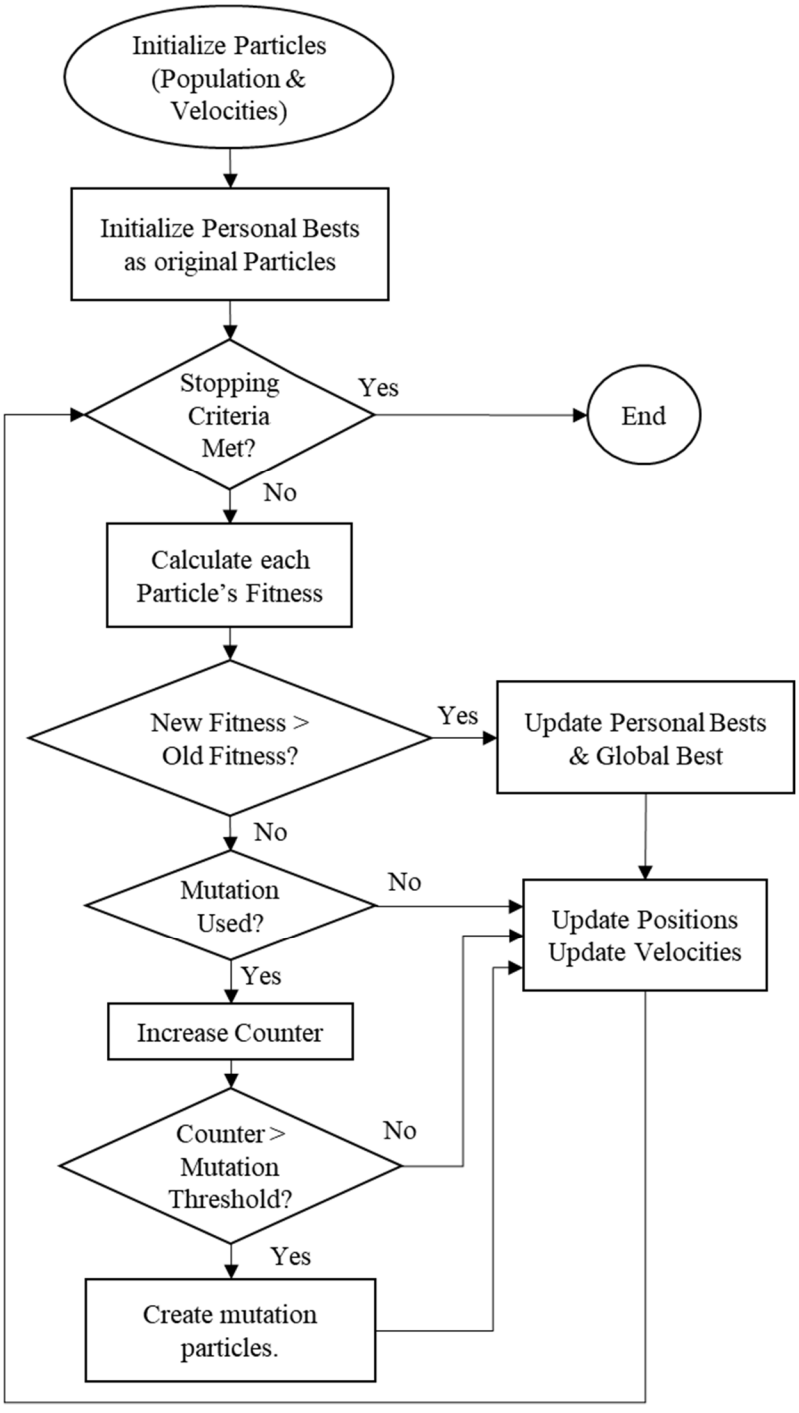

Figure 5: Particle Swarm Optimization (PSO) Flowchart

2.3 Related Work in Dynamic Spectrum Access (DSA)

In 2009, Michael Maskery [8] et al published a paper on dynamic spectrum access using spectrum overlay. The 2009 paper [8] builds upon the work published by Maskery in Chapter 11 of a cognitive networks book [9]. Based on a carrier sense multiple access (CSMA) system and a Hidden Markov Model (HMM) to estimate channel contention and effective throughput, this insight is used to create a local utility that each radio can compute which balances its personal needs (competition) and system needs (cooperation). The competitive part of the local utility rewards the user for meeting the demand. The cooperative components will penalize for achieving in excess of its demand and for the degradation of channel quality as measured by collisions. As each secondary user selfishly strives to maximize its local utility, it also works in decentralized collaboration with other secondary users maximize the system global utility. The decision-making algorithms he provides are game theoretic and stochastic in nature. Chapter three, Methodology, will cover this in greater depth.

The government has been working through their Defense Advanced Research Projects Agency (DARPA) with private companies such as Raytheon and Shared Spectrum Company to develop the neXt Generation (XG) radio [16]. DSA research continues in 2018 with the NOAA awarding a new contract for an engineering study and analysis of 1675-1680 MHz for sharing [17].

Private industry has been studying cognitive radio and has provided a physical radio prototype [18] and modular software to accompany it [19]. The ‘DSA network’ module of the software package is based on collaborative negotiation between radios to share sensing and decision data.

IEEE Standards has a group for Dynamic Spectrum Access Networks (DYSPAN) that hosted a conference in July 2018. So far, the DYSPAN group worked to standardize terminology, best practice for interference and coexistence, access layer protocols, building blocks for decision making, sensing databases, etc [20].

2.4 Related Work in Computational Intelligence

In 2003, Ribeiro and Schlansker [21] applied neural networks with particle swarm training to reactive power control research. Their work refines the ANN weights but leaves a foundation for generalizing the algorithm and optimizing topology and transfer functions. Their research problem of reactive power injection for power control requires different ANN architecture according to the system requiring the power control and therefore is a good candidate for evolutionary ANNs.

In 2015, an article was published by Garro and Vázquez [22] describing research conducted into designing the topology and selecting the transfer function and synaptic weights. This generalized algorithm was applied to supervised training data from many research fields and showed good results for sigmoidal or Gaussian transfer functions. Many different ANN topologies emerged from the research. Overall PSO for hyper-parameter training and weight training has proven promising.

In 2016, Changzhi Wang et al [23] applied PSO to a feedforward neural network for the purpose of indoor RFID localization with supervised training data. They found that the approach converged quickly and achieved higher fitness values than some of the competitors such as a genetic algorithm trained feedforward neural net or traditional positioning algorithms.

Evolutionary neural networks have transcended the theoretical research barrier and are now being studied and considered by companies. Per a blog post earlier this year [24], Uber (a ride-share company) is studying the reinforcement learning aspect of evolutionary science.

In 2017, neural networks were applied to the spectrum sensing in DSA. ANNs and their pattern recognition capabilities provided an appeal over the Hidden Markov Model (HMM) as they do not require statistical models or extensive knowledge of the system [25].

CHAPTER III

III. METHODOLOGY

This thesis focuses on the 'Hierarchical Access Model' (in particular spectrum overlay) for dynamic spectrum access. In spectrum overlay, the secondary users opportunistically use spectrum holes (channels sensed unused by primary users). Consider

First, a few clarifications: physical radio characteristics are not used, which allows the research to be applied to other models and problems. Thus, '*channels'* henceforth will be a broad term, that could just as well signify bandwidth, spread-spectrum codes, etc. Channel interference is taken to be minimal and as such sensing errors are disregarded. By omitting physical radio characteristics, this research overlooks the signal processing side of spectrum sensing. This high-level approach facilitates generalization and portability to a wide range of applications. A decentralized collaborative approach is used as there are no infrastructure needs to prevent putting this theoretical research into practice.

This research furthers the works from Michael Maskery's high-level and decentralized collaboration model as proposed in [8], [9]. The first two sections of this chapter will describe Maskery's model for static and dynamic environments and the developments thereof in the present research. Maskery's algorithms will be used as a basis of comparison for evaluating the effectiveness of the PSO-ANN hybrid approach described in the last two sections of this chapter. ANN was chosen to provide a fast output to the radios and particle swarm was chosen for weight training to deliver quick results and a generalized algorithm that can be reused on other models.

3.1 Static System

The opportunistic model will rely upon independent agents, which means that no central authority prescribes the actions of the radios. All of the users will act selfishly but may collaborate if doing so improves the chances of getting what they want. This is comparable to a real-life situation in which the traffic light is out and drivers must decide who gets to cross the intersection.

In order to make the decision as to which channels to access, each individual radio must know the following:

Variable	Meaning
Ch	Number of channels
$Y_n \in \{0,1\}^{Ch}$	Primary Usage Pattern
C_n	Quality Vector. (Quality Bits per Channel)
d_n^l	Demand level of cognitive radio
m^l	Maximum Number of Channels that the radio may occupy.

Table 3: Known Parameters

Where 'n' denotes the current time interval and 'l' is the secondary user. Each user only knows its own demand and channel restrictions. The secondary usage pattern of others (X_n^{-l}) is unknown.

Example: If there are 6 channels, of which a user can only occupy 2 at once and the other knowns are: Primary usage pattern, Yn = (1 1 0 0 0 0); Quality, C_n = (1 1 1 1 1 1) kbps; demand, d = 1.5kbps; Then the following usage patterns would all be valid: (0 0 0 0 1 1), (0 0 0 1 0 1), (0 0 1 0 0 1), (0 0 0 1 1 0), (0 0 1 0 1 0), (0 0 1 1 0 0).

Even with this trivial example, a straightforward solution to usage pattern (X_n) is not known. To make a decision, a user would need to learn from the environment and infer the best outcome.

The system as designed by Maskery [8], [9] utilizes 'Carrier Sense Multiple Access' (CSMA) in which several users share spectrum via channel sensing. Each decision period *n* is divided into *k* equal CSMA sub-slots. Figure 6 shows an example of k=10 sub-slots.

Decision Period *'n'*	1										2									
CSMA Subslot *'k'*	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10

Figure 6: CSMA Sub-slots per Decision Period.

For each channel chosen during decision period *(n)* and at the start of a sub-slot *(k),* a secondary user (*l*) will generate independent & uniformly distributed back-off times: $\tau_k^l \in (0, \tau\_max)$. Let δ be the time it takes between sensing the channel idle and switching the parameters to transmit. A channel capture will occur for user *l* when $\tau^l < \tau^m + \delta$, for all users $m \neq l$. Per Maskery, the probability of *l* capturing a channel *i* with *N* competing users during time *n* and sub-slot *k* is:

$$P(l \text{ captures channel } i \text{ during } n,k) = \begin{cases} \left(1 - \frac{\tau_k^l(\mathrm{i}) + \delta}{\tau_{\max}}\right)^{N_n^l(\mathrm{i})-1}, & \tau^l(\mathrm{i}) \leq \tau_{\max} - \delta \\ 0, & \tau^l(\mathrm{i}) > \tau_{\max} - \delta \end{cases} \quad [8, 9]$$

The figure below shows a decision period with 3 CSMA sub-slots. When the shortest back-off time is shorter than the next by more than δ, the user with the lowest back-off time captures and transmits on the channel. Alternatively, when the two back-off times are less than δ apart, both radios sense the channel is idle and transmit on it, thereby causing a collision of signals. This is similar to the analogy of traffic at an intersection, except that wireless communication has no brakes. Therefore, 'routes' with low-traffic intersections are better.

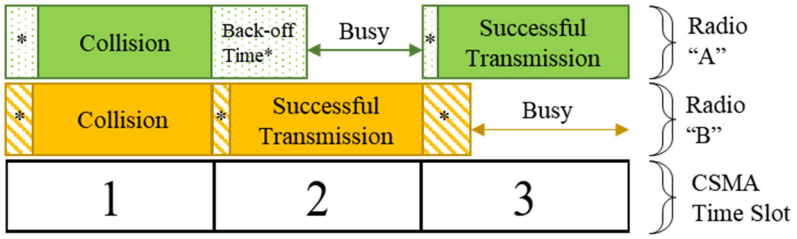

Figure 7: Channel Capture Options: Collisions, Failures and Successes

For the example illustrated in Figure 7, the radios can keep the records shown in Table 4 and will know the back-off times at which each event occurred.

User	Success	Failure	Collision
Radio "A"	1, $\tau_3{}^a$	1, $\tau_2{}^a$	1, $\tau_1{}^a$
Radio "B"	1, $\tau_2{}^b$	1, $\tau_3{}^b$	1., $\tau_1{}^b$

Table 4: Channel Capture Record-Keeping

In his work, Maskery provides a means to infer the number of users competing with user *l* on a channel from the success, failure and collision statistics and their respective back-off times. The inference expands on the probability of *l* capturing the channel during *(n, k)* as provided before. The channel contention equation is difficult to solve, so he uses numerical approximations for his final formula. Maskery admits that although accurate on average, his formula has a large error in special cases. Maskery then proposes using the channel contention formula as a seed for Newton-Raphson iteration, Newton-Raphson's method being necessary due to the approximation that had been made for the difficult-to-solve equation.

The author did at first encode the channel contention estimate as was stated in the article but found it very imprecise. This thesis is a study of computational intelligence techniques for spectrum decision and not a study of spectrum sensing. Hence, it was decided to assume some other means of knowledge or enhanced spectrum sensing and thus each radio in this study will know exactly how many users are competing against '*l*' for each channel: $N_n^l(\mathrm{i}) = \sum_{j \neq l} X_n^j(i)$. Thus, there is no estimation or uncertainty; *N* is the sum of all other players vying for the channel.

In game-theoretic terms, this knowledge will avoid a game of imperfect information. This assumption is appealing in consideration of the amount of knowledge derived from the quantity of users. (An error in *N* propagates significantly to other variables.) The adoption of this

assumption renders the performance of the compared decision-making algorithms insensitive to variances in sensing. Maskery makes the same assumption in his paper as is explained later in the chapter.

Through more probabilistic derivations, Maskery provides the ability to calculate the expected throughput (successful attempts) and collisions per channel '*i*' in decision period '*n*'.

- Throughput [8, 9]:

$$R_n^l(\mathrm{i}) = \begin{cases} \frac{X_n^l(\mathrm{i})}{1+N_n^l(\mathrm{i})}(1-\frac{\delta}{\tau_{max}})^{1+N_n^l(\mathrm{i})}, & N_n^l > 0 \\ 1, & N_n^l = 0 \end{cases}$$

- Collisions [8, 9]:

$$Q_n^l(\mathrm{i}) = \frac{X_n^l(\mathrm{i})\delta}{\tau_{max}} + \frac{X_n^l(\mathrm{i})}{1+N_n^l(\mathrm{i})}[1-(\frac{\delta}{\tau_{max}})^{1+N_n^l(\mathrm{i})} - \left(1-\frac{\delta}{\tau_{max}}\right)^{1+N_n^l(\mathrm{i})}]$$

The throughput and collisions of the formulas above correspond to the expected proportion of CSMA slots captured or determined to be useless per decision period, consistent with Figure 7. (Although Figure 6 showed an example of 10 CSMA slots per decision period, Maskery uses 20 CSMA slots per decision period in his study.)

All of the users on the channel should be taken into account for calculations of throughput and collisions; this is reflected in the formulas by the term $(1 + N_n^l(\mathrm{i}))$; where again, N_n^l stands for the number of users competing against *'l'* for the channel and the addition of 1 accounts for user *'l'*.

Another assumption being made is that δ and τ_{max} are significantly smaller than the total CSMA time slot such that the throughput is affected only by channel captures. Consequently, the effect of sensing the channel at the start of each CSMA period is not factored into the throughput calculation.

The collision and throughput formulas provided by Maskery were validated to see the progression of the performance trackers as N_n^l grew. The calculations of the validation table assumed $\delta/\tau\_max = 0.1$, but the pattern holds for other δ/τ_max values.

As seen in Table 5, the throughput acts as expected and decreases as N_n^l grows. The collisions stay almost static with a slight decrease as users grow, which is counter-intuitive. The expectation is that collisions should grow as users grow. For that reason, an alternate 'collisions' formula was pursued, instead of the one provided by Maskery.

There are finite states for an attempted channel: Either it was captured by one of the users or there was a collision: $R_n^l(\mathrm{i}) * \left(1 + N_n^l(\mathrm{i})\right) + Qnew_n^l(\mathrm{i}) = 1$. This mathematical expression takes into account the equal likelihood of channel capture and the requirement that probabilities sum to 1.

Isolating the collision term of the prior mathematical expression and substituting the throughput for its formula yields a new collisions calculation that will be used in this study instead:

- Collisions

$$Qnew_n^l(\mathrm{i}) = \begin{cases} X_n^l(\mathrm{i}) * [1 - \left(1 - \frac{\delta}{\tau_{max}}\right)^{1+N_n^l(\mathrm{i})}], & N_n^l > 0 \\ 0, & N_n^l = 0 \end{cases}$$

A collision rate that satisfies this equation is shown by column *Qnew*.

N+1	R	Q	Qnew
1	1	0	0
2	0.41	0.19	0.18
3	0.24	0.19	0.28
4	0.16	0.19	0.34
5	0.12	0.18	0.41
6	0.09	0.18	0.47

Table 5: Throughput and Collision Validation

Per the first row of Table 5, for one user on a channel, a throughput of one and no collisions are expected. The second row shows two users on the channel: 41% of attempts go to user 1, another 41% to user 2, and 18% of attempts are wasted due to collisions. The other rows similarly work out, with small rounding errors for the 2 decimal places shown. Being satisfied with the results, the Qnew formula superseded Q in all future calculations. To be clear, all further mentions to Q will refer to the new formula just derived.

From the performance measurements, a global system utility can be derived. The global utility will be used to track the overall system health in the graphs of chapter 4. Unfortunately, the individual radios cannot compute the global utility function from the parameters that are known to them. Therefore, local utility functions calculated from known parameters will be created to help maximize the global utility function.

- Global Fitness Function:

$$U(X_n) = \min_{l=1,\ \ldots,\ L} \left(min\left(\frac{C_n^T * R_n^l(X_n)}{d^l},\ 1 \right) \right)$$

The global fitness is determined by the worst performing user in the system. For this, the proportion of demand satisfied for each radio 'l' will be found by calculating the quality * throughput / demand. It does not matter if the ratio is higher than what was needed, so it gets capped at 1. The minimum is taken to identify the worst performing user. The global fitness will then be bounded from [0, 1].

As can be seen, the global fitness function requires knowing the demand and the usage pattern of all users. For that reason, it cannot be calculated by the individual users.

To construct an effective local utility, the calculations must use only knowledge that the user has. Intuitively, each user should attempt to meet but not exceed its demand. It has also been made clear that each user should avoid congested channels. Therefore, the local fitness function as obtained and provided by Maskery is a combination of rewards and penalties:

- Reward for reaching demand:

$$u^l[0]\left(X_n^l\right) = min\left(\frac{C_n^T R_n^l}{d^l},\ 1\right)$$

- Penalty for achieving excess rate and greedy behavior; to avoid congestion:

$$u^l[1]\left(X_n^l\right) = -\frac{1}{d^l}\left(C_n^T R_n^l - \left(d^l + \beta\right)\right)^+$$; β: Acceptable amount of excess

- Penalty for degradation of channel quality due to collisions; to avoid interference and jamming:

$$u^l[2]\left(X_n^l\right) = -\frac{1}{\sum_k C_n(k)} \sum_{i:\hat{N}^l(i)>0} \frac{C_n(i) Q_n^l(i)}{N_n^l(i)}$$

- Final Local Utility Function, a weighted sum of the above component utilities:

$$u^l\left(X_n^l\right) = \max\{u^l[0]\left(X_n^l\right) + \alpha_1 u^l[1](X_n^l) + \alpha_2 u^l[2]\left(X_n^l\right),\ 0\}$$

From the numerical results of his article, Maskery suggests the system performs well for values $(\alpha_1, \alpha_2) = (0.2, 1.8)$ in the final local utility defined above.

In attempting to maximize their local utility, the radios will also be working in a decentralized collaborative manner to improve the system performance. Similarly to the global fitness function, the local fitness function is bounded from [0,1].

Example: What follows below is a decision period output from the code. Let a decision period denote 20 CSMA attempts per channel. The rows of each matrix correspond to secondary users and the matrix columns correspond to the channels. 6 secondary users and 10 channels were used to mimic Maskery's graphs (to be seen in chapter four). Some of the calculations are partial elements of formulas previously presented; such partial calculations are explained in the table.

<table>
<tr><td colspan="5">Quality Vector (C_n) =
1 3 1 1 2 1 2 1 2 1</td><td colspan="5">User's Demand (d_n) =
(4 2 2 1 1 4) ^{T}</td></tr>
<tr><td colspan="10">Primary Usage Pattern (Y_n) = 0 0 0 0 0 0 1 1 0 0</td></tr>
<tr><td colspan="5">Secondary Usage Pattern (X_n) =
1 0 0 0 0 0 0 0 1 0
0 1 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 0
1 0 0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0 0 0</td><td colspan="5">Number of Users Competing with 'L' (N_n) =
1 1 1 1 1 1 0 0 1 1
2 0 1 1 1 0 0 0 2 1
2 1 1 1 1 1 0 0 1 1
1 1 1 0 1 1 0 0 2 1
2 1 0 1 1 1 0 0 2 0
2 1 1 1 0 1 0 0 2 1</td></tr>
<tr><td colspan="10">Expected Channel Throughput (R_n) = (before applying multiplication with X^l_n)</td></tr>
<tr><td>0.41</td><td>0.41</td><td>0.41</td><td>0.41</td><td>0.41</td><td>0.41</td><td>0</td><td>0</td><td>0.41</td><td>0.41</td></tr>
<tr><td>0.24</td><td>1.00</td><td>0.41</td><td>0.41</td><td>0.41</td><td>1.00</td><td>0</td><td>0</td><td>0.24</td><td>0.41</td></tr>
<tr><td>0.24</td><td>0.41</td><td>0.41</td><td>0.41</td><td>0.41</td><td>0.41</td><td>0</td><td>0</td><td>0.41</td><td>0.41</td></tr>
<tr><td>0.41</td><td>0.41</td><td>0.41</td><td>1.00</td><td>0.41</td><td>0.41</td><td>0</td><td>0</td><td>0.24</td><td>0.41</td></tr>
<tr><td>0.24</td><td>0.41</td><td>1.00</td><td>0.41</td><td>0.41</td><td>0.41</td><td>0</td><td>0</td><td>0.24</td><td>1.00</td></tr>
<tr><td>0.24</td><td>0.41</td><td>0.41</td><td>0.41</td><td>1.00</td><td>0.41</td><td>0</td><td>0</td><td>0.24</td><td>0.41</td></tr>
<tr><td colspan="10">Expected Collisions (Q_n) = (before applying multiplication with X^l_n)</td></tr>
<tr><td>0.19</td><td>0.19</td><td>0.19</td><td>0.19</td><td>0.19</td><td>0.19</td><td>0</td><td>0</td><td>0.19</td><td>0.19</td></tr>
<tr><td>0.27</td><td>0</td><td>0.19</td><td>0.19</td><td>0.19</td><td>0</td><td>0</td><td>0</td><td>0.27</td><td>0.19</td></tr>
<tr><td>0.27</td><td>0.19</td><td>0.19</td><td>0.19</td><td>0.19</td><td>0.19</td><td>0</td><td>0</td><td>0.19</td><td>0.19</td></tr>
<tr><td>0.19</td><td>0.19</td><td>0.19</td><td>0</td><td>0.19</td><td>0.19</td><td>0</td><td>0</td><td>0.27</td><td>0.19</td></tr>
<tr><td>0.27</td><td>0.19</td><td>0</td><td>0.19</td><td>0.19</td><td>0.19</td><td>0</td><td>0</td><td>0.27</td><td>0</td></tr>
<tr><td>0.27</td><td>0.19</td><td>0.19</td><td>0.19</td><td>0</td><td>0.19</td><td>0</td><td>0</td><td>0.27</td><td>0.19</td></tr>
<tr><td colspan="5">Global_U_temp = (before applying the 'min')
(0.30 2.00 0.41 1.41 2.00 .50) ^{T}</td><td colspan="5">Local Utilities =
(0 0.80 0.06 0.75 0.80 0.50) ^{T}</td></tr>
</table>

Table 6: System State example from code

In this example, channel qualities were taken from a uniform distribution of values within the subset {1, 2, 3} and demands from {1, 2, 3, 4} to mimic the original results shown in section 4.1.

It is comforting that the secondary usage patterns produced by the regret tracking algorithms of section 3.3 do not encroach upon the primary usage pattern. The secondary usage pattern shows that secondary user #1, corresponding to the first row, will to attempt channels #1 and #9 during this decision period of 20 CSMA attempts.

The matrix for *N* shows the number of users rivaling a user for that channel. User #1 is rivaling only one other user for channel #1. User #1 did not attempt to use channel #2, but if it had it would have expected to compete with one other user. This pattern holds for all other channels and all other users.

The values of throughput and collisions are as shown in Table 5. These are conditioned on the number of competing users shown in the N matrix. Note that Table 5 is indexed according to total users per channel (N+1) when reading and cross-verifying the values to this example that uses competition: N. For example, a user attempting to access a channel in competition against one other user, such as the case for user #1 and channel #1, would expect to access the channel 41% of the time. Had user #2 attempted to access channel #1, it would have competed against two other users and would expect to access the channel 24% of the time.

As mentioned in Table 6 above, the throughput and collisions matrices correspond to the formulas before multiplication with the usage pattern. This was done in the code to accommodate the decision-making algorithms- both regret tracking and PSO-ANN. Both will require evaluating options not taken in order to determine the future round. The multiplication with the usage must still occur in order to calculate the utility values which follow.

The 'Global_U_temp' matrix corresponds to each secondary user's contribution to the overall global utility function, or: $C_n^T R_n^l / d^l$.

For user #1 the calculation is:

$$\frac{(1\ 3\ 1\ 1\ 2\ 1\ 2\ 1\ 2\ 1) * (0.41\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0.41\ 0)^{\mathrm{T}}}{4} = \frac{0.41 + 2 * 0.41}{4} = 0.3075;$$

The throughput above now shows the multiplication with the usage pattern of user #1.

The local utilities correspond to the weighted sums of the subcomponents as explained before this numerical exercise. The local utility for user#1 can be calculated as follows:

- The first utility component, known as the reward for reaching the demand, is 0.3075. This value was previously calculated in the global utility example.
- The second component is the penalty for achieving excess rate. This value is zero, as no excess rate was achieved.
- The third component is the penalty for channel quality degradation. User #1 collides with user #4 19% of the time on channel #1. User #1 also collides 19% of the time with user #3 on channel #9.

  A penalty is expected and can be calculated with:

  $$-\frac{1}{\sum_k C_n(k)} \sum_{i:\widehat{N}^l(i)>0} \frac{C_n(i) Q_n^l(i)}{N_n^l(i)} =$$

  $$-\frac{\left(\left(1 * \frac{0.19}{1}\right) + 0 + 0 + 0 + 0 + 0 + 0 + 0 + \left(2 * \frac{0.19}{1}\right) + 0\right)}{1 + 2} = -0.19;$$

  In the above calculation, the usage pattern was applied to the collisions read from the 'Q' matrix of Table 6. Thus, the reason for the many zeros in the numerator. The denominator corresponds to the sum of the qualities being damaged by user#1.
- Finally, the local utility is: max{(0.3075 - 0.2*0 - 1.8*0.19), 0} = max{-0.03, 0} = 0;

The remaining users local utilities can be similarly verified from the formulas provided in this chapter. Such verification shows that users #1, #2, and #3 are penalized for their collisions in the local utility. Users #2 and #5 witness penalties for greediness and user #6 has no penalty.

User#2's local utility will be briefly reviewed to showcase the greedy penalty.

- The reward for reaching demand corresponds with:

  $$\min\left(\frac{[3*1+1*1]}{2}, 1\right) = \min(2,1) = 1;$$

- The greedy penalty for '*l*'=*2*:

  $$-\frac{1}{d^2}\left(C_n^T R_n^2 - \left(d^2 + \beta\right)\right)^+ = -\frac{[(3*1+1*1)-(2+0)]^+}{2} = -1;$$

  In this calculation, $\beta = 0$, which signifies zero tolerance for greediness.

- The penalty for channel quality degradation is zero, as this user had no collisions.
- Lastly, the final utility for user #2 is max{(1 - 0.2*1 - 0), 0} = max{0.8, 0} = 0.8;

These numerical examples were provided as instruction to the reader as well as to verify the code for the model. Additional verification is provided in chapter four.

3.2 Dynamic System

The dynamic system examined in this thesis will have 2 primary users, 10 channels, and 6 secondary users to mimic the data prepared by Maskery, who chose simple setups to demonstrate the algorithms. Two dynamic cases are analyzed, one with slow-varying parameters and one with fast-varying parameters.

In the slow varying case the parameters that may change are the demand levels of the 6 secondary users and the location of the 2 primary users. As such, the quantity of parameters changing is 8.

In the fast varying case all of the parameters from the slow varying case may change as well as the quality of the channels, which is allowed to fluctuate by +/- 10%. Because there are 8 available channels and there were already 8 varying parameters, the quantity of parameters changing is 16.

For each of these cases, the varying parameters do so independently of the others with probability of ρ (Greek letter 'ro'). Therefore, the expected duration time between a change, provided by Maskery in [8, 9], is: $T(\rho) = (1 - (1-\rho)^{\#params})^{-1}$

In order to simulate Maskery's results, the mean innovation time was taken from the dynamic graph in the numerical results. Then, the ρ of the table below was calculated by solving the above equation for ρ and inputting the mean innovation time (T). Finally dynamic system simulations (detailed in chapter 4) were run for each ρ as shown in the table.

Number of Varying Parameters = 8

T	1000	500	250	125	62.5	31.25	15.63	7.81	3.91	1.95
ro	0.0001	0.0003	0.0005	0.001	0.002	0.004	0.008	0.017	0.036	0.086

Number of Varying Parameters = 16

T	1000	500	250	125	62.5	31.25	15.63	7.81	3.91	1.95
ro	6.25E-05	0.0001	0.0003	0.0005	0.001	0.002	0.004	0.009	0.018	0.044

Table 7: Expected times between changes

3.3 Comparison Algorithms

The algorithms that follow in the subsections were either designed by Maskery or in turn used as a comparison in his work and stem from research done in the game theory field [8, 9]. Encoding these served as more than a performance comparison; the output graphs obtained were compared to those provided and helped diagnose if the model had been accurately represented and calculated.

Maskery first proposes his own game theoretic algorithm that is based on “regret tracking”; a notion in which each user seeks to minimize the regret of not having taken another option consistently. This method requires computation of every viable option whether taken or not taken during each decision-making interval. It calculates instantaneous regret values and recursively sums them so that the regret matrix will be a running average of the regrets over time. This method utilizes stochastics for the final decision; regrets help to scale the probabilities of taking a certain option, with a higher regret being more likely.

Other algorithms analyzed are 'best response' and 'fictitious play' which Maskery provided for his own comparison. These methods do not employ stochastics; they always choose the option that minimizes the regret. ‘Best response’ does not base its decision upon a running average of regrets, but rather upon the regret from the immediately previous round. ‘Fictitious play’ chooses the option that minimizes the running average regret.

Maskery acknowledges that the previous algorithms require knowing the utilities of channels not used. In turn, the utilities are calculated from the number of users on a channel. Using the channel contention method will require having attempted the channel to sense back-off times and outcomes. Maskery states that he assumes additional sensing resources for options not taken in the above algorithms but provides another ‘modified regret tracking’ method that does not use additional sensing resources. Instead, it infers the regrets from the existing data of options taken.

3.3.1 Regret Tracking

This algorithm will calculate the instantaneous regrets and recursively incorporate them into a running average regret. This average regret is used to compute the likelihood of choosing a solution; a higher regret corresponds to a higher likelihood of selection.

Denote $\boldsymbol{S}^l$ as the state space of valid solutions: $\boldsymbol{S}_n^l = \{\mathrm{x} \in \{0,1\}^{Ch} : x * Y_n = 0, \sum_{i \in Ch} x(i) \leq m^l\}$.

S^l is the number of elements in the state-space $\boldsymbol{S}^l$. Define a S^l x S^l instantaneous regret matrix as:

$$H_{jk}^l(X_n) = I\{X_n^l = j\}\,(u^l(k, X_n^{-l}) - u^l(j, X_n^{-l}))$$

Initialization is done by taking a random action X_0^l and then initializing the average regret, θ, as: $\theta_0^l = H^l(X_0)$. Then $X_1^l = argmax_k H_{jk}^l(X_0)$ and $\theta_1^l = H^l(X_1)$.

For n = 2, 3, … ; X_n^l is chosen with probability:

$$P(X_{n+1}^l = k \mid X_n^l = j, \theta_n^l = \theta^l) = \begin{cases} \max\{\theta_{jk}^l, 0\} / \mu\,, & k \neq j \\ 1 - \sum_{i \neq j} \max\{\theta_{ji}^l, 0\} / \mu\,, & k = j \end{cases}$$

Where $\mu > (S^l - 1)(u_{max}^l - u_{min}^l) = (S^l - 1)(1 - 0) = S^l - 1$; Therefore, $\mu = S^l$ was chosen to satisfy the inequality.

Next, the average regret is updated from the instantaneous regret as:

$$\theta_{n+1}^l = \theta_n^l + \varepsilon_n(H^l(X_{n+1}) - \theta_n^l).$$

$\varepsilon_n = 1/(n+1)$ is used for the static case or a constant in the dynamic case. As such, θ is a recursive calculation of the arithmetic average in the static case. In the dynamic case, θ is a recursive calculation of a moving average. As the regret of not having taken another action grows, so does the likelihood of changing to that decision.

3.3.2 Best Response

Best Response is based upon the instantaneous regret and so does not require calculation of θ. The usage pattern is chosen according to $X_{n+1}^{l} = argmax_k H_{jk}^{l}(X_n^l)$. This algorithm will change to a different usage pattern if that pattern would have yielded a better decision in the immediately preceding round. The steps are identical to those of the regret tracking algorithm, but use $\varepsilon_n = 1$.

3.3.3 Fictitious Play

Fictitious Play is similar to best response, but incorporates some memory to remedy the fallacy of assuming constant play between turns. Fictitious Play follows the regret tracking algorithm, with the exception of usage pattern selection. The decision behind the usage pattern follows the maximum of the average regret matrix: $X_{n+1}^{l} = argmax_k \theta_{jk}^{l}(X_n^l)$. Probabilities are not employed as the decision always goes to the highest average regret.

3.3.4 Modified Regret Tracking

Modified Regret Tracking extrapolates data from the options taken and uses it to compute the regret values for options not taken, with the aim of avoiding the need for additional sensing resources. Hence it doesn't require knowing the exact regrets for all values.

Modified regret tracking is similar to regret tracking except it substitutes the instantaneous regret calculation with an estimate that uses the probabilities as follows:

$$H_{jk}^{l}(X_n) = I\{X_n^l = k\} \left({p_n^l(j)}\Big/{p_n^l(k)} \right) u^l\left(k, X_n^{-l}\right) - I\{X_n^l = j\}\, u^l\left(j, X_n^{-l}\right)$$

On most rounds, X_n^l is chosen from the probability defined in the original regret tracking algorithm. However, on a few rounds as determined by a uniform random variable, the action is chosen from the state-space $\boldsymbol{S}^l$ with equal likelihood. This is done to introduce random exploration.

3.4 Neural Network Topology

As mentioned in previous chapters, PSO has the ability to optimize the creation of the artificial neural network (ANN), but if a priori knowledge of the problem exists, then it is wise to make use of it. To begin, only one hidden layer (h) was used. As will be seen in chapter 4, this topology was adequate for the problem. Additional experimentation was foregone given that adding more layers risked slowing down training, convergence and computational speeds. The specifics of the ANN used are shown in the figure and table that follow:

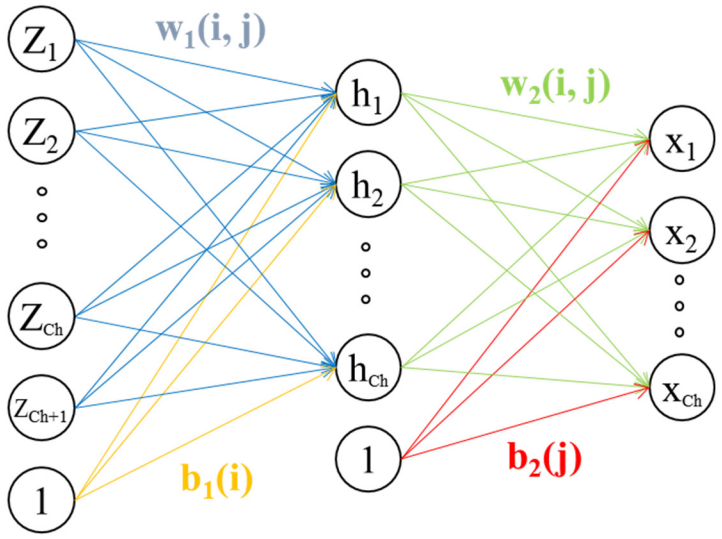

Figure 8: Neural Network Topology

Variables	Meaning	Size
$w_1(i,j)$	weight term from Z_j ⇨ h_i	*Ch x (Ch+1)*
$w_2(i,j)$	weight term from h_j ⇨ X_i	*Ch x Ch*
$b_1(i)$	Bias term going into h_i	*Ch x 1*
$b_2(i)$	Bias term going into X_i	*Ch x 1*
Ch	Number of Channels	Integer

Table 8: ANN Parameters

The inputs should be on a similar scale of magnitude as mismatches in scale risk, thereby creating an imbalance of input strength. For example, the channel quality (kbps ~ Mbps) would be an unfit choice of input alongside m^l. The argument can be made that the weights make up for the discrepancy, but this is only true if the weights are allowed a large range. Overall, it is best to scale the inputs to an appropriate level.

To bring channel quality to similar magnitude as m^l, it was decided to divide by the radio's demand for regularization. This can be conceptualized as Demand Satisfied (*DS*) per channel. To improve this concept, the throughput and channel availability ($\overline{Y_n}(i)$) were included:

$$DS_n^l(i) = R_n^l(i) * (\overline{Y_n}(i)) * C_n(i) \,/\, d_n^l; \quad \text{Where: } \overline{Y_n}(i) = \sim(Y_n(i))$$

The input matrix is then: $Z = [DS_n^l(1) \quad DS_n^l(2) \quad \dots \quad DS_n^l(Ch) \quad m^l]^T$. As defined before, $DS(i)$ is the demand satisfied per channel *i* and m^l is the maximum quantity of channels that the user may occupy.

The inputs are multiplied by the first layer weights and summed; then, the first level bias is subtracted. This value is fed to a transfer function and input to the second layer weights and second layer bias. From here, the top m^l outputs are taken as the channel usage pattern. The feed-forward process previously described can be computed with the following vector math:

$$h = \ w_1 * Z - b_1;$$

$$x = \ w_2 * S[h] - b_2;$$

$$X = \ max[x, m];$$

S[*] represents the transfer functions commonly used in ANNs previously explained in chapter 2. The outcomes from the Hard Limit, Log-Sigmoid, Hyperbolic Tangent Sigmoid, and Rectified Linear Unit (ReLU) functions are presented in chapter 4.

3.5 Particle Swarm

The particle swarm's position matrix is initialized with ANN parameters, namely the weights and biases for each layer. As such, the swarm actually consists of a network of neural networks. For visualization, an example swarm is shown in the figure below as a matrix with as many columns as there are particles and as many rows as there are number of elements. The number of elements per ANN is as defined in the previous section

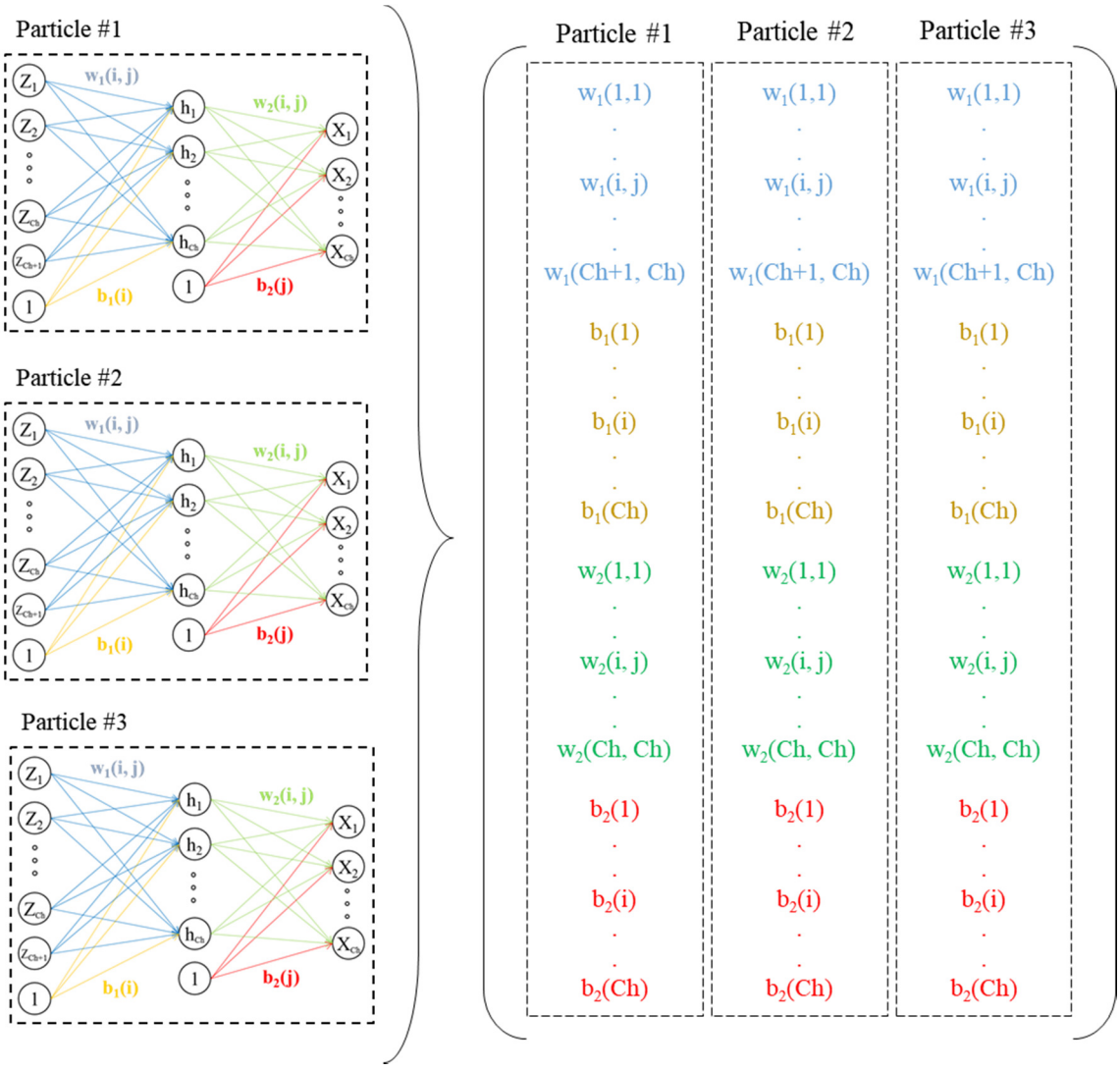

Figure 9: Example PSO-ANN Swarm Matrix

To begin, the ANN particles matrix is initialized with random values for the weights and biases. In order to apply the position update formula explained in chapter 2, a velocity matrix of the same size as the position matrix will be initialized.

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

Optimization of initialization parameters were evaluated and the results are presented in chapter 4. During initialization, the personal bests are set to the first iteration of ANNs in the particles matrix.

The input vector to the ANNs is created from the parameters provided in Maskery's model. The output layer (usage pattern) is calculated for each ANN particle by applying the input vector and feeding it forward. A local fitness function is calculated from the usage pattern. Then, each particle is compared to its personal best. If the new 'position' yielded a usage pattern with a higher fitness, then the personal best is updated with that new 'position'. These personal bests for each particle are in turn compared to the global best of the swarm. Similarly, the global best is updated if any of the other particles performed better.

The figure below shows example matrices for personal and global bests. For elegance of presentation, the weights and biases below were compressed to vector notation.

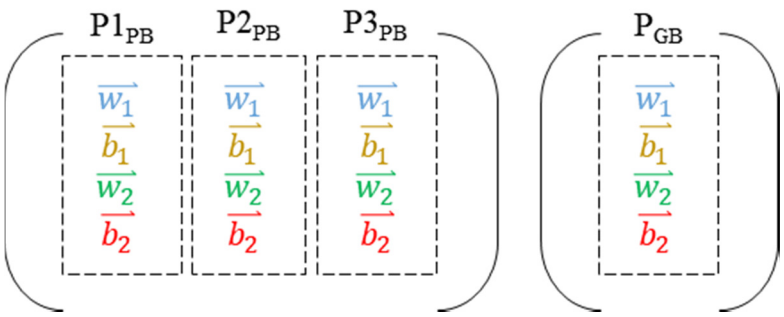

Figure 10: PSO-ANN Swarm Personal & Global Bests

The velocity was updated per the following formula (introduced in chapter 2) using the known personal and global bests.

$$v_i(t+1) = a * v_i(t) + b * r_1 \left(x_{pb}(t) - x_i(t)\right) + c * r_2 \left(x_{gb}(t) - x_i(t)\right)$$

The effects of varying the inertia coefficient (a) and the acceleration coefficients (b, c) are shown in chapter 4.

This research covers experiments with and without mutations. When the mutations were enabled, the global best was compared to the global best of the previous round. Each time these were equal, the counter increased. Otherwise, it was reset. When the counter was above the mutation threshold, the bottom-most particles were mutated, or re-initialized per the parameters used during the initialization.

For each iteration in which the PSO algorithm is run, it returns the global best as the chosen ANN to be used. The PSO-ANN algorithm is run for each secondary user considered.

CHAPTER IV

IV. FINDINGS

This chapter details the experiments dedicated to configuring the problem, the PSO-ANN algorithm and the comparisons to regret tracking algorithms. The tests were run over many decision periods to see the development of the algorithms. Then, they were averaged across several scenarios to be able to compare results more effectively.

The first experiment was done to verify the model as obtained from Maskery's publications [8], [9], essentially it is a comparison of copy-cat output graphs to those from the published articles.

From there, ANN parameters such as transfer function, initialization positions, and use of bias is observed. Then, a study of PSO Hyper-parameters is undergone, starting with the initialization velocity. After velocity, an analysis of mutation impact on performance was done and a preferred mutation interval was selected. This subsection ends with an optimization of coefficients a/b/c for the velocity update formula. This optimization is done both with and without mutations.

Once the PSO-ANN has been set up, a comparison between Maskery's methods and PSO-ANN ensues. The first comparison occurs in a congested environment; the next comparison has some bandwidth to spare. The final test is an evaluation of scalability of both maximum number of channels allowed per secondary user *(m)* as well as scalability of overall system channels *(Ch)*. The computational speeds provided correspond to an Intel® Core™ i5-7600K CPU @3.80GHz and will vary for other computers. The metric to note is instead the ratio of the speeds.

4.1 Model Verification

The graphs of the figure below show the performance for a static case with non-varying parameters. None of the regret tracking algorithms achieved quite as promised performance-wise; they are never the less close to what was expected. Moreover, the shapes of the curves do match in the equilibrium comparison.

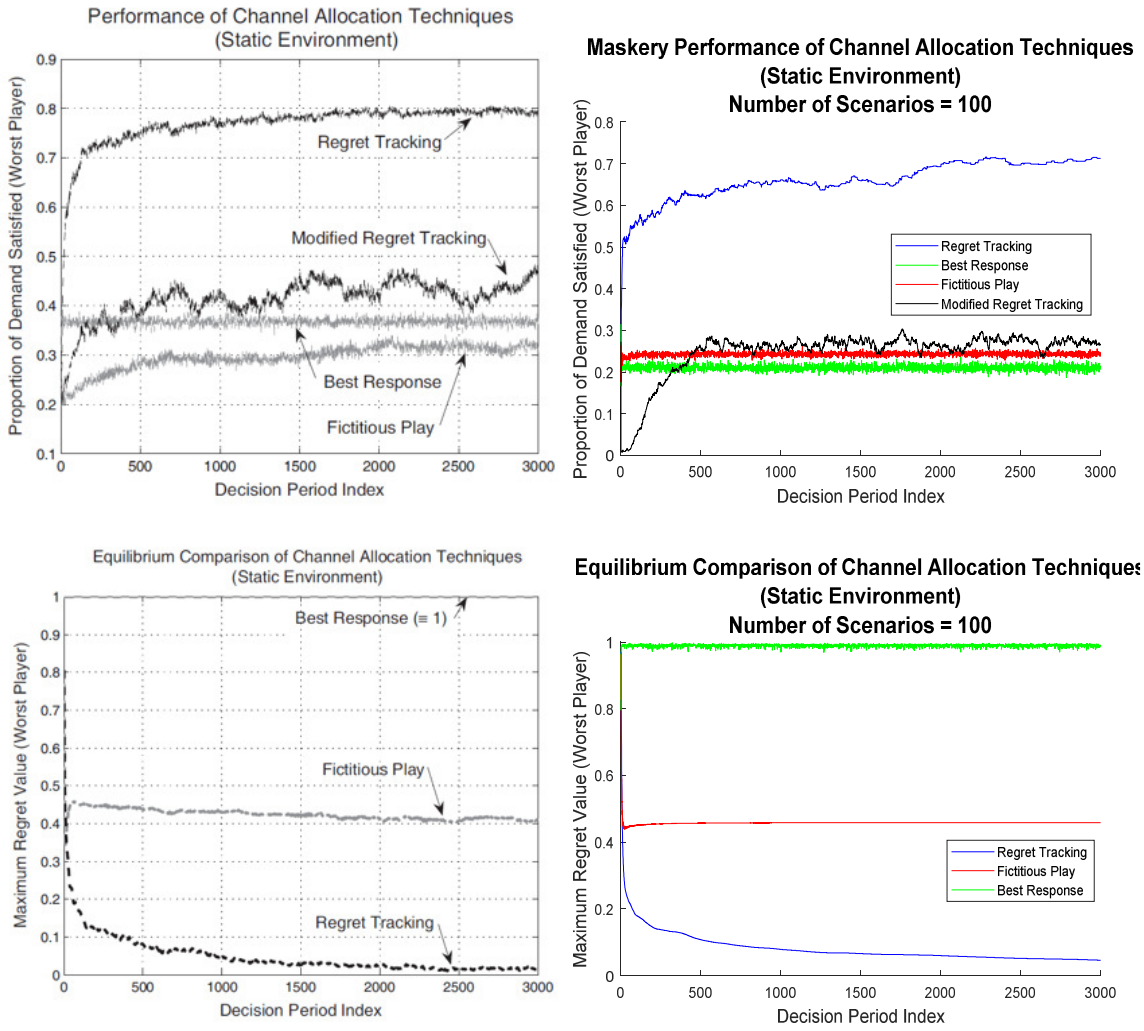

Figure 11: Published Results [8, 9] to Obtained Results – Static case

Modified regret tracking is not plotted in the equilibrium comparison graphs due to large un-normalized regret values as explained in [8, 9]. Additionally, this large un-normalized regret was yielding negative probabilities in order to satisfy the probability formula of chapter 3. So, the

regret was temporarily normalized before applying in the probability calculations. The large un-normalized regret values were stored for use across the iterations.

These observations from Figure 11 similarly hold true for the dynamic case that was executed below. Overall, the shapes of the copy-cat track with the publication, where lesser time between changes results in less utility. Modified regret tracking underperformed the most, but since regret tracking is the main focus of comparison, the remaining algorithms are left in for minimum benchmarking. The solid lines indicate slow varying primary users (w=8 changing parameters) and the dashed lines indicate fast varying (w=16 changing parameters) as explained in chapter 3.

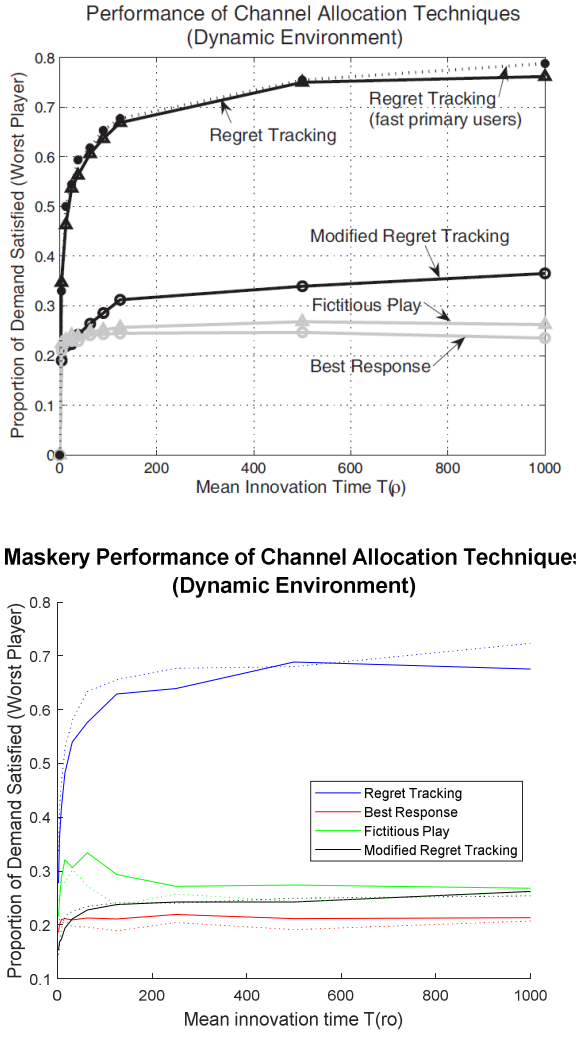

Figure 12: Published Results to Obtained Results – Dynamic case

4.2 ANN Initialization & Fitness Functions

Given a priori knowledge of the fitness functions and channels being bounded from [0, 1], it is logical to start with weights and biases around that range. Four transfer functions were tested with four different ranges of initialization PSO 'positions' (corresponding with ANN weights and biases). Solid lines indicate bias in use while the dashed lines of the same color do not use bias. It is difficult to discern a preferable weight and bias initialization strategy from the graphs but it is clear that Log-Sigmoid is the best transfer function for this problem. All but Log-Sigmoid show significant noise and variance despite the 100 scenario averaging. Both TanH and ReLU experience drop-offs in performance; although it's possible it could be from an un-perfected PSO. Based on these graphs, Log-Sigmoid is used as the transfer function for the PSO-ANN.

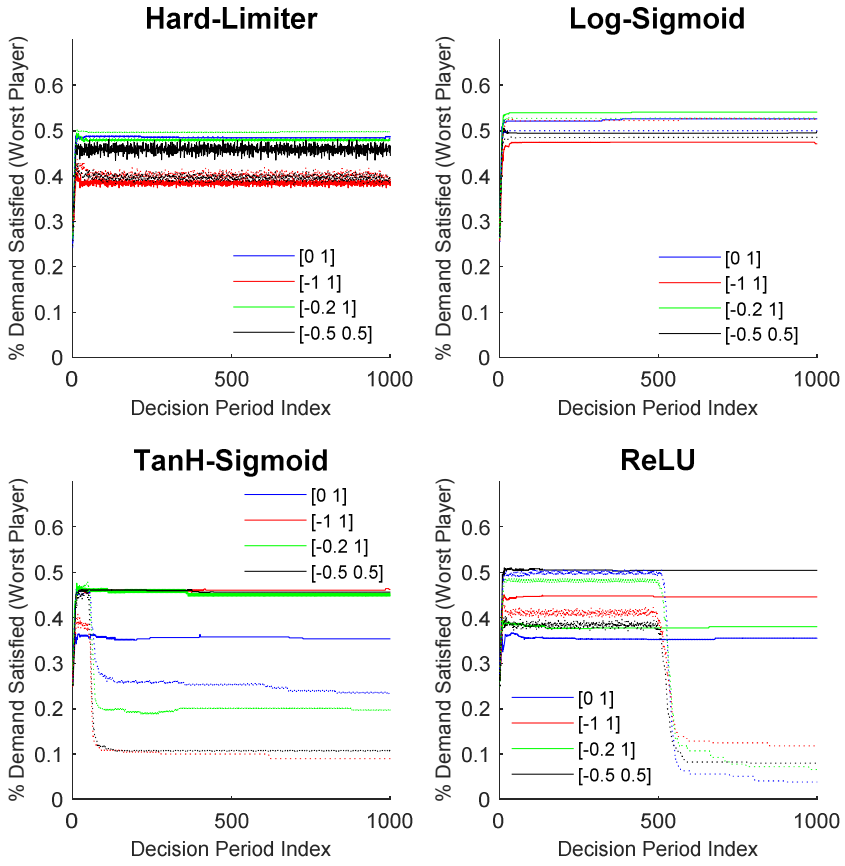

Figure 13: Performance of Different ANN initialization parameters

For these tests, an unrefined PSO with values of a=b=c=0.1 was used and initial velocities were taken from a uniform distribution in [-0.2 0.2]. To further hone the initial positions, more experiments were done. These experiments focus on the first 200 iterations to better see the initialization. Both static and dynamic cases were plotted as shown in the figures below.

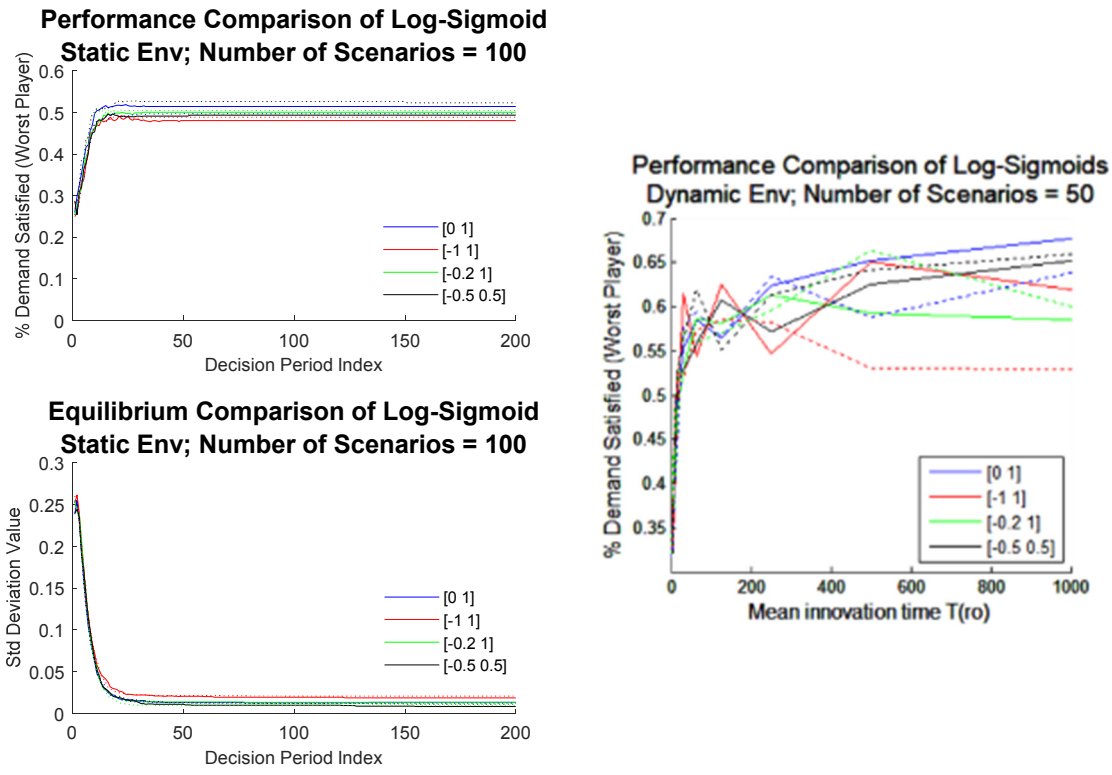

Figure 14: Weight & Bias Verification –Static & Dynamic cases

From the above curves, it was determined that the initialization range for 'positions' of weights and biases would be from the uniform random distribution of [0 1]. As explained before, the solid lines still indicate use of bias whereas dashed lines did not use bias. A strategy for bias was inconclusive and so it will continue to be studied in the following subsections.

4.3 PSO Hyper-parameters: Initial Velocity, Mutations and Coefficients

4.3.1 Velocity Training

The initialization velocity [V_{start}, V_{end}] was allowed to range from [-1, 1], with V_{start} <= V_{end}. The first experiment at 200 decisions was inconclusive. By the 200th decision, much of the benefit provided by the initialization velocity were negligible. Furthermore, the effect of the unrefined a/b/c coefficients could cloud or interfere with the results. For these reasons it was decided to focus on the first 10 decisions in the experiment below.

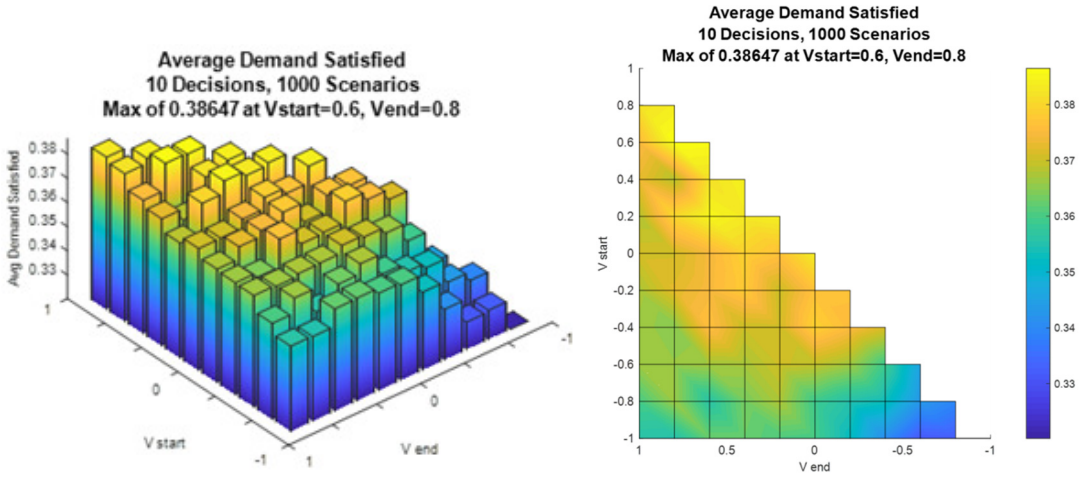

Figure 15: Initial Velocity Performance (3D & Top-Down views)

The heights on the left graph show the best outcomes received. Since the heights could be difficult to distinguish, colors and a top down view are provided as well. There are several adequate ranges but the absolute maximum occurred for the uniform random distribution within [0.6 0.8]. Consequently, this range was chosen to move forward.

4.3.2 Mutation Interval Training:

20% of the particle population was allowed to mutate whenever the global best had not improved in a certain amount of decisions. This threshold will be referred to as the mutation interval. The mutation interval was allowed the range: [0, 10, 20, … , 100] with an interval of 0 designating no mutations. The outcome is shown below.

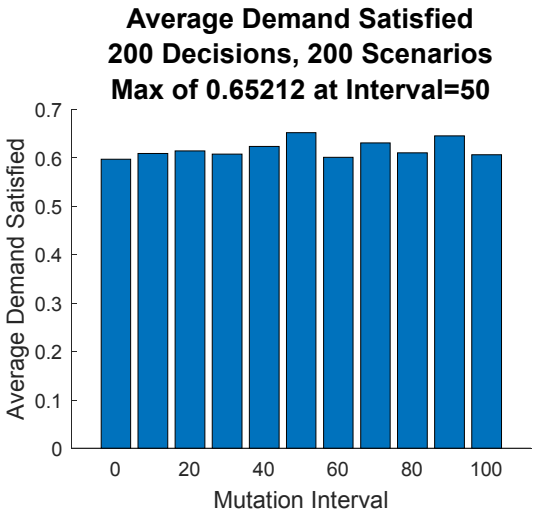

Figure 16: Mutation Interval to Demand Satisfied Comparison

As is readily observable, the results are overall inconclusive, with no pattern discernable. This is possibly as a result of unrefined PSO coefficients. Thus, the subsections that follow show the fine-tuning of a/b/c PSO coefficients with and without mutations. For the mutations case, an interval of 50 decisions was taken because it ranked highest in this experiment, in terms of average demand satisfied.

4.3.3 PSO Coefficients a/b/c without mutations

The experiments to refine the a/b/c coefficients evaluate and average the proportion of demand satisfied for different a/b/c values. This is similar to the velocity experiment but with an extra variable requiring optimization. The discrete range given to 'a' was [0.25, 0.5, 0.75, 1]; coefficients b and c were allowed the range [0, 0.2, 0.4, … , 2].

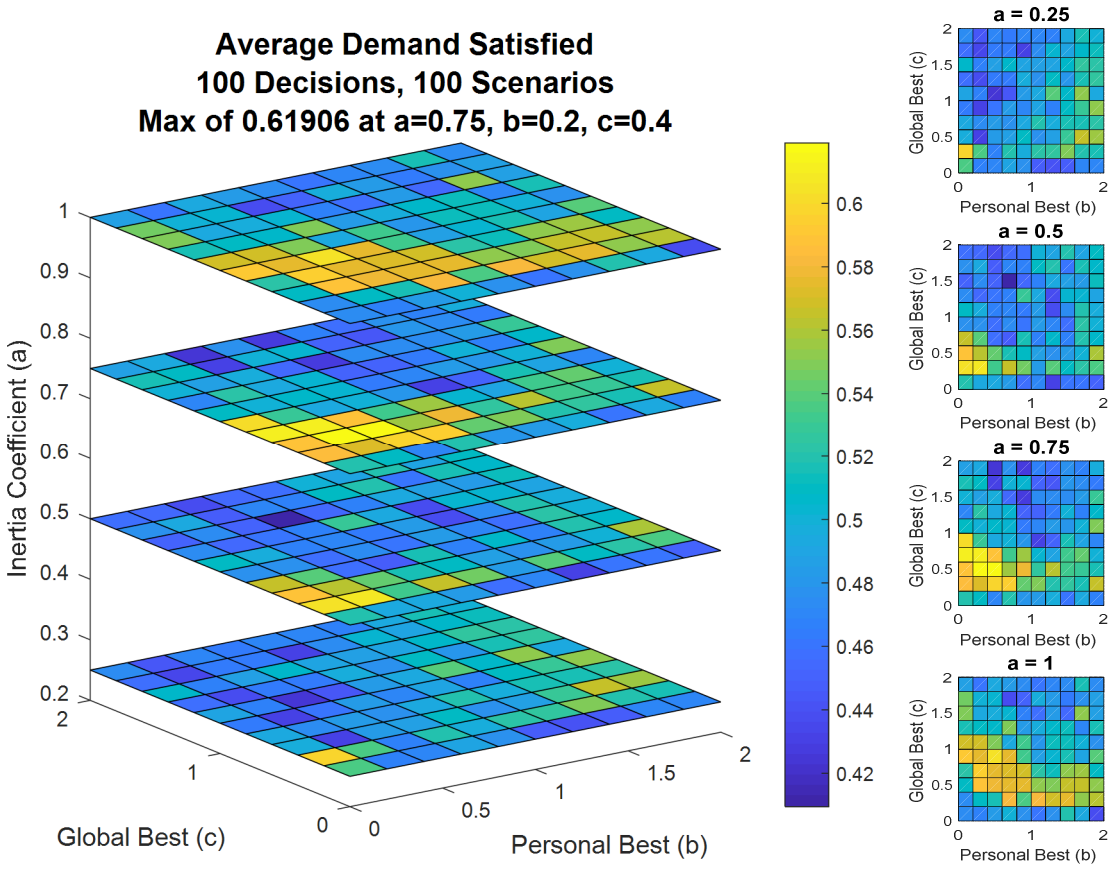

Figure 17: Comparison of PSO coefficients: a, b, c. (3D & Top Down view)

From the experiment above, several solutions proved viable, especially for the planes where a=0.75 and 1. The top 4 performing combinations of a/b/c were chosen to move forward to the verification experiments of the next page.

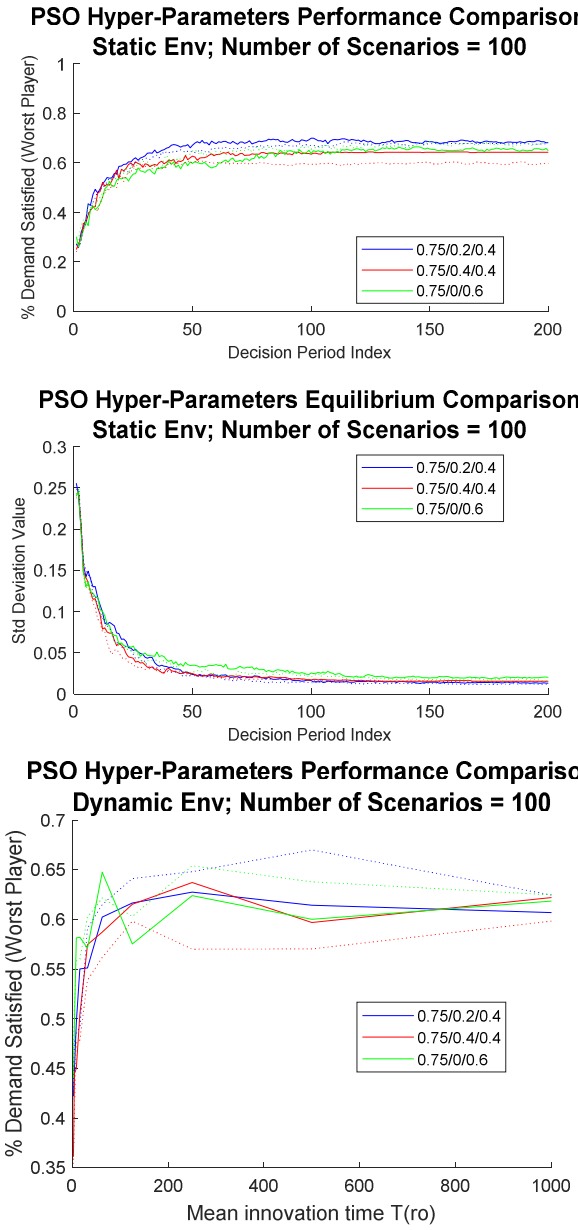

Figure 18: PSO Coefficients verification –Static & Dynamic cases

The above experiments show the performance of the different coefficients in static and dynamic cases. The solid lines use bias and the dashed lines do not. 0.75/0.2/0.4 with bias was chosen to move forward for its performance in the static case. 0.75/0.2/0.4 without bias was also chosen due to its performance in the dynamic case and runner-up status in the static case.

4.3.4 PSO Coefficients a/b/c with mutations

The exercise shown below is identical to that of the prior subsection except that mutations were enabled with an interval of 50 decisions.

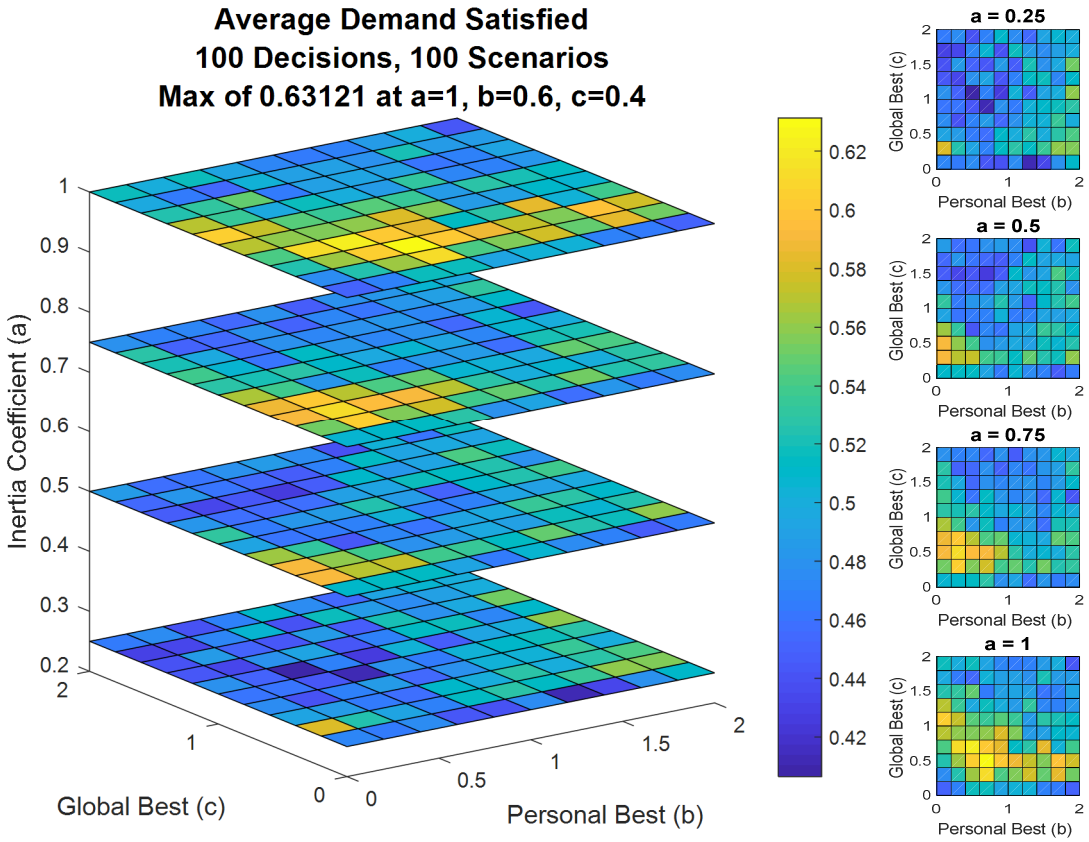

Figure 19: Comparison of PSO coefficients: a, b, c. (3D & Top Down view)

Observe that the mutations case favors the a=1 plane over the a=0.75 plane as opposed to the non-mutations case which favored a=0.75 over the a=1 plane. This is surprising because the ‘a’ coefficient corresponding to the inertial element of the velocity is usually associated with exploration as opposed to ‘b’ & ‘c’ who are exploitation. Since mutations are already introducing exploration, it was logical to assume less exploration would be needed from the coefficients as compared to the non-mutations experiment.

As in the prior subsection the four best outcomes moved forward to verification as shown next.

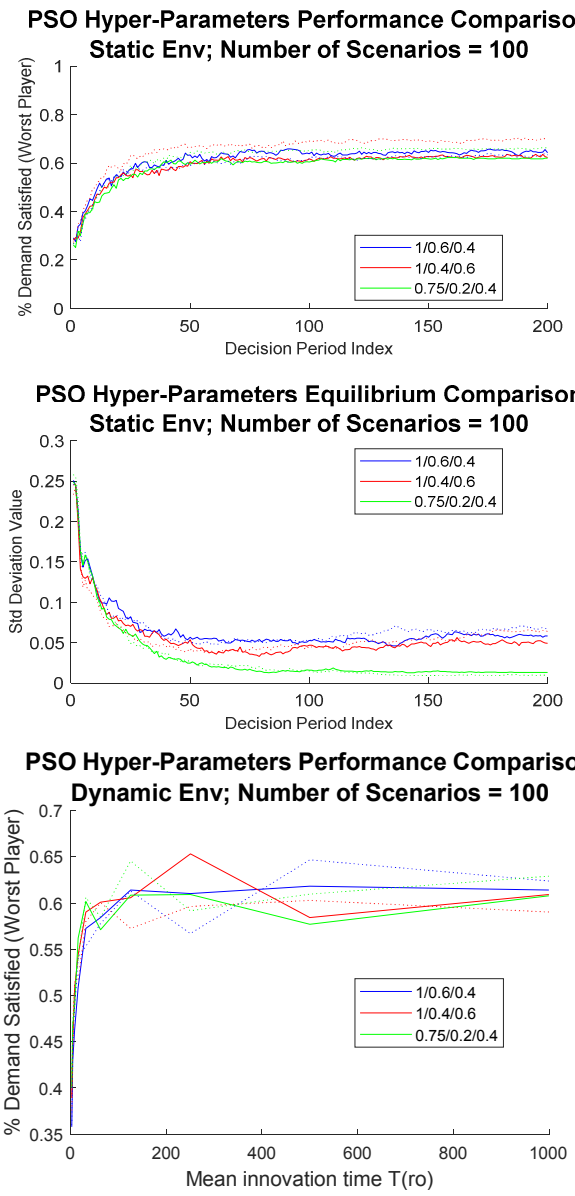

Figure 20: PSO Coefficients verification –Static & Dynamic cases

As was used before, the solid lines indicate presence of bias and dashed lines represent no bias. From this experiment, 1/0.4/0.6 without bias was chosen to move forward for its performance in the static case. 1/0.6/0.4 with bias was also chosen for its runner up performance in the static case and its stability in the dynamic case.

4.4 Comparison of Regret Tracking algorithms to PSO-ANN

From the prior subsections, the following combinations of PSO and ANN hyper-parameters are going to be used and are designated in the legends as shown in the table. Log-Sigmoid transfer function, initial position ranges in the uniform distribution of [0, 1] and velocity ranges from [0.6, 0.8] are common to all. Dashed lines now indicate fast users in the dynamic case (w=16).

Legend	PSO Coeff (a)	PSO Coeff (b)	PSO Coeff (c)	Bias Used
No Mut #1	0.75	0.2	0.4	Yes
No Mut #2	0.75	0.2	0.4	No
Mut #1	1	0.4	0.6	No
Mut #2	1	0.6	0.4	Yes

Table 9: Legends for PSO-ANN Parameters

The first experiment is a congested environment where demands are pulled from a uniform distribution on {1, 2, 3, 4} and qualities from {1, 2, 3}. This was provided by Maskery in his published works. According to statistics, the average demand expected per user is 2.5; the system demand expected is 15 with 6 secondary users. Similarly, the quality expected per channel is 2; the expected system quality is 16 from 10 channels of which 2 are occupied by primary users.

The second experiment modifies the first in that it allows the qualities to range the uniform distribution of {1, 2, 3, 4}. As such, the expected system quality is 20 to fulfill the expected system demand of 15. Therefore, the experiment occurs in a non-congested environment.

The third experiment evaluates the scalability of the regret tracking and PSO-ANN algorithms. It does so by increasing the maximum allowed channels per secondary user to 3 and 4. Then, it sets this value back to 2 and increases the available channels in the system to 15 and 20. The main focus of this experiment is computational complexity but performance graphs are provided also.

The performance graphs now include a fast-varying dynamic graph for ro = 0.05 and w=16.

4.4.1 Congested Environment

In this congested environment, PSO-ANN outperforms the regret tracking algorithms considerably. The worst player meets 60% of their demand within just 30~40 decisions using PSO-ANN, whereas regret tracking is nearing 60% around decision 200. All PSO-ANN combinations perform about equally well and converge rather quickly.

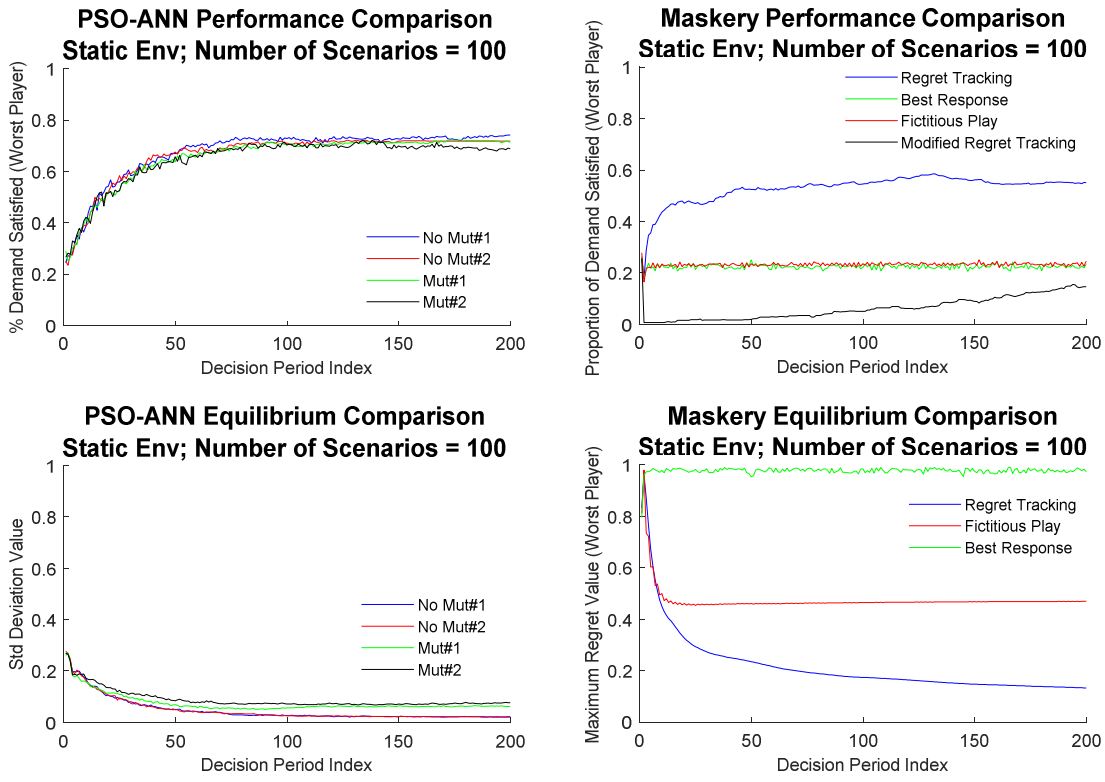

Figure 21: PSO-ANN & Maskery in static environment (congested)

PSO-ANN outperforms regret tracking in the dynamic cases of the following page also, with higher utilities being achieved. Although the fast-varying case of ro=0.05 is less than ideal it is understandable given the speed at which changes are occurring.

Regret tracking outperforms PSO-ANN in computational speeds, the latter being 1.79 times slower.

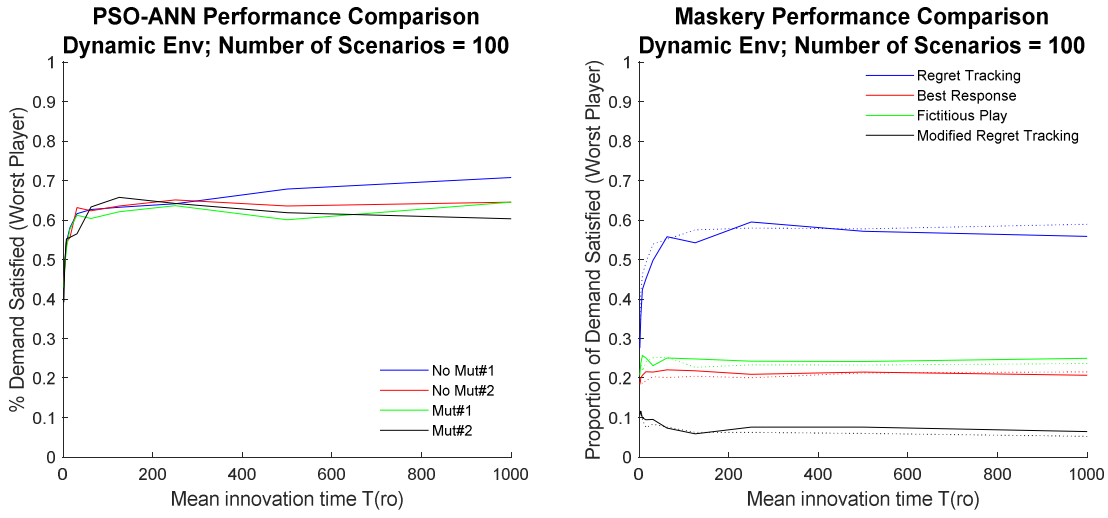

Figure 22: PSO-ANN & Maskery in dynamic environments (congested)

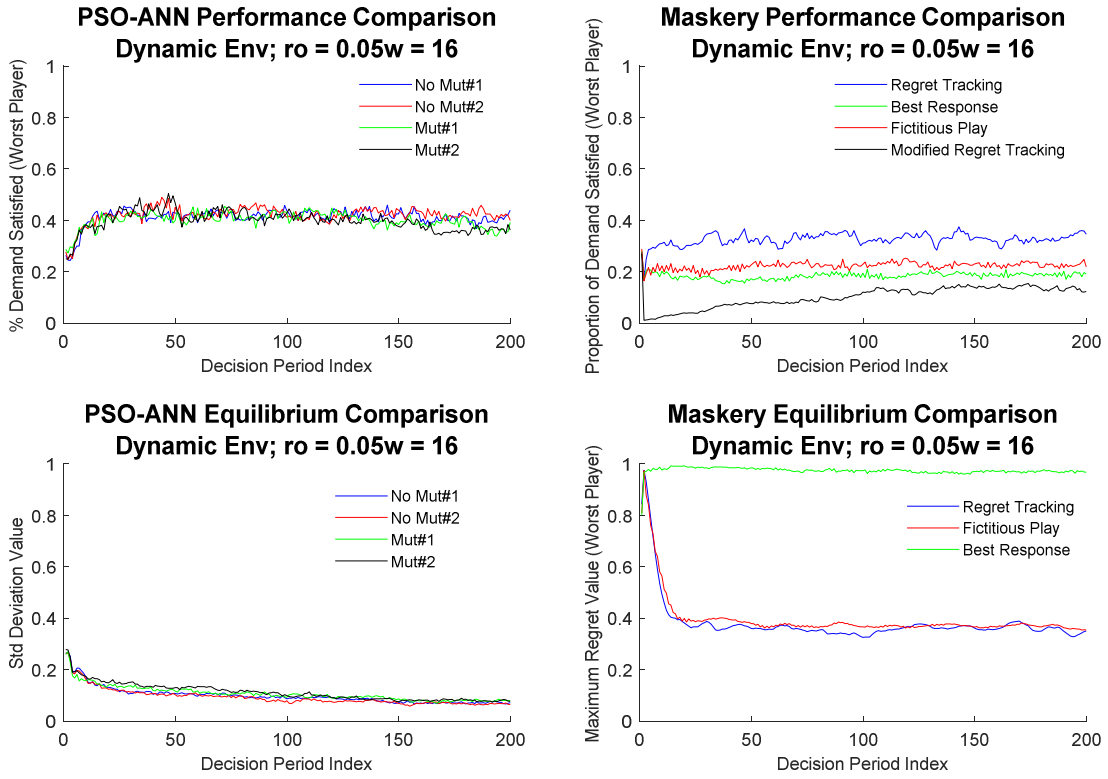

Figure 23: PSO-ANN & Maskery for ro=0.05 & w=16 (congested)

Computational Speed: 264 seconds for PSO-ANN & 143 seconds for Maskery. 1.79:1 ratio.

4.4.2 Non-Congested Environment

PSO-ANN outperforms regret tracking once again in the non-congested environment. It is even more prominent here, where PSO-ANN achieves at least 80% of the worst player's demand in about 30~40 decisions. Regret tracking is still at about 70% at 200 decisions in.

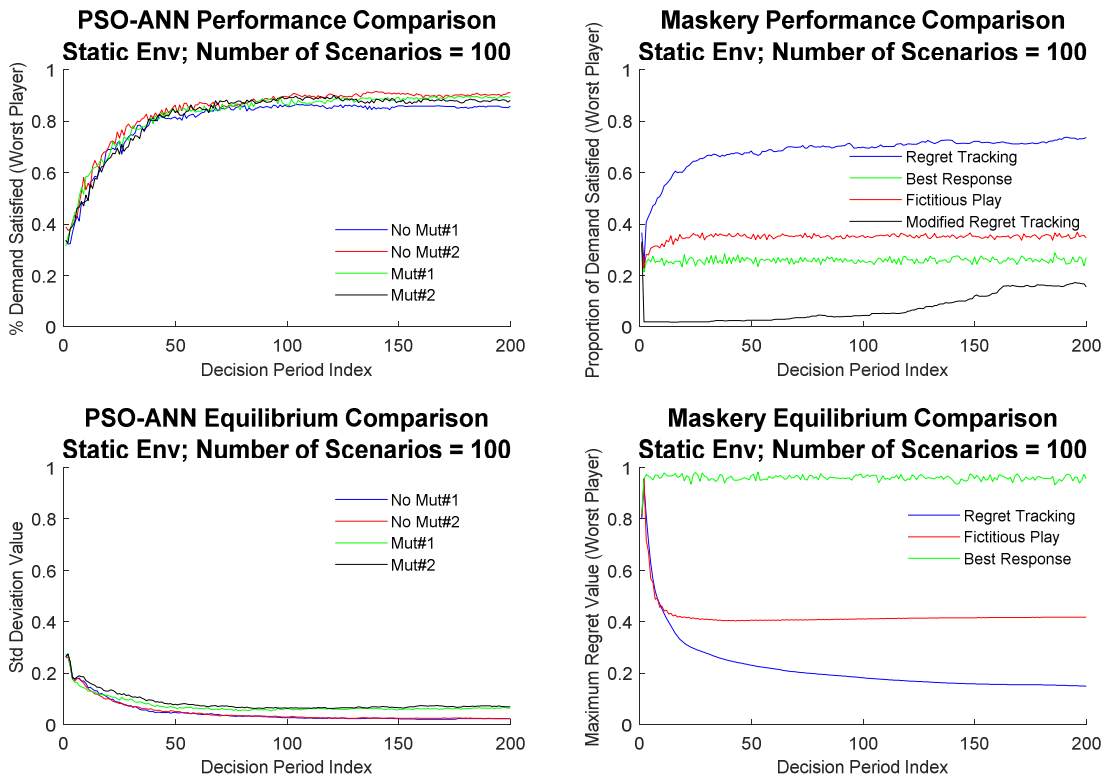

Figure 24: PSO-ANN & Maskery in a static environment (non-congested)

The same observation holds for the dynamic cases of the following page. PSO-ANN is at about 80% utility even for very fast changes. Regret tracking seems to max out at 70% for slower changes and does more poorly for lower T(ro). The last dynamic case has PSO-ANN at an average 60% where regret tracking is around 40%.

Regret tracking outperforms PSO-ANN in computational speeds once more, PSO-ANN being slower by a factor of 1.82.

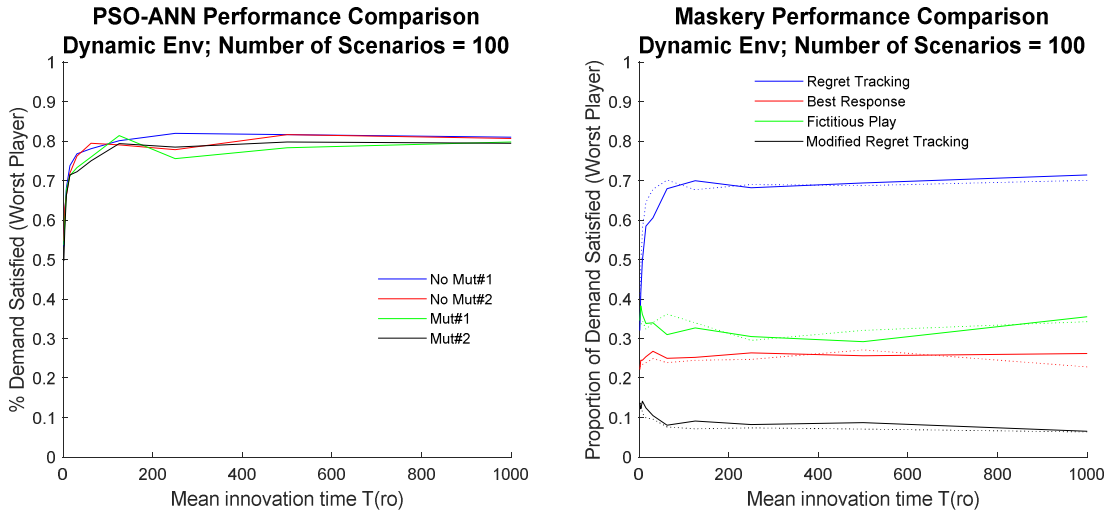

Figure 25: PSO-ANN & Maskery in dynamic environments (non-congested)

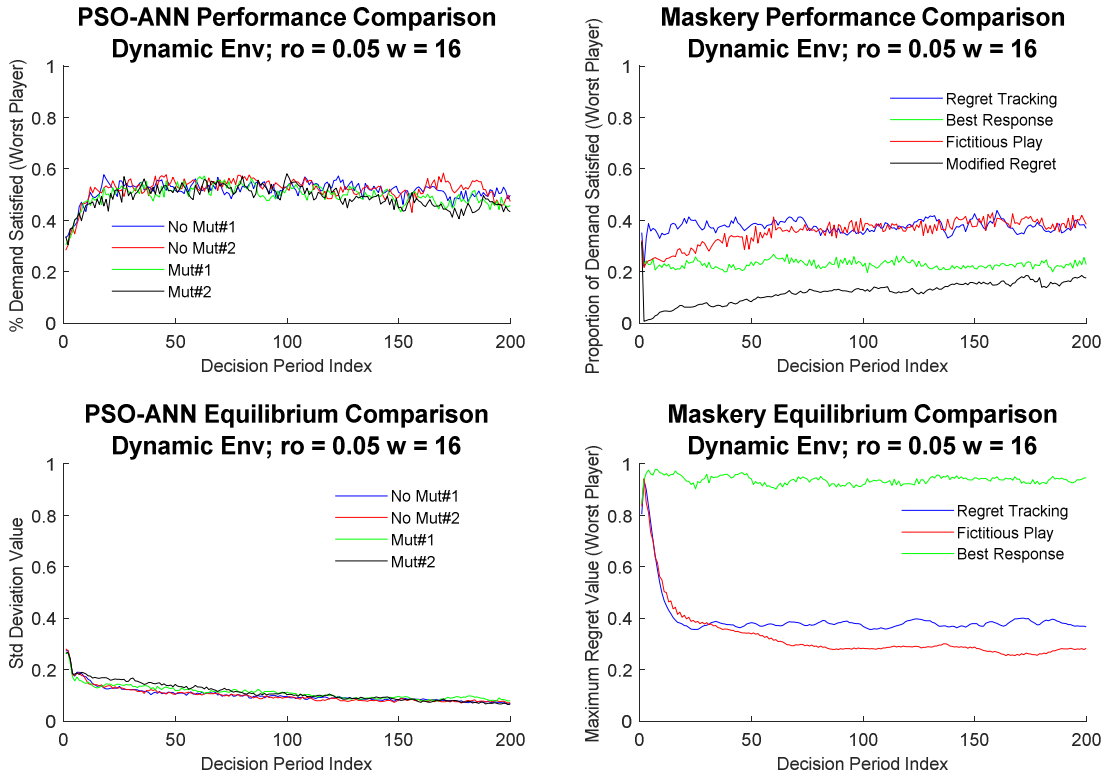

Figure 26: PSO-ANN & Maskery for ro=0.05 & w=16 (non-congested)

Computational Speed: 265 seconds for PSO-ANN & 145 seconds for Maskery. 1.82:1 ratio.

4.4.3 Evaluating scalability

Updating the maximum number of channels that the radio may occupy *(m)* from 2 to 3 yielded computational speeds of 253.99 seconds for PSO-ANN and 524.75 seconds for Maskery. Or a ratio of 1:2.08 in favor of PSO-ANN. Updating m = 4 and running only 5 scenarios instead of 100 yielded 19.38 seconds on PSO-ANN & 489.97 seconds for Maskery. A surprising 1:25 ratio in favor of PSO-ANN.

The graphs that follow correspond to m=3 and eventually both converge to their maximums from the m=2 non-congested case. It can be seen from the curve shapes that it takes both a few more decision periods to learn. As in the cases before, PSO-ANN outperforms regret tracking in utility, speed and convergence.

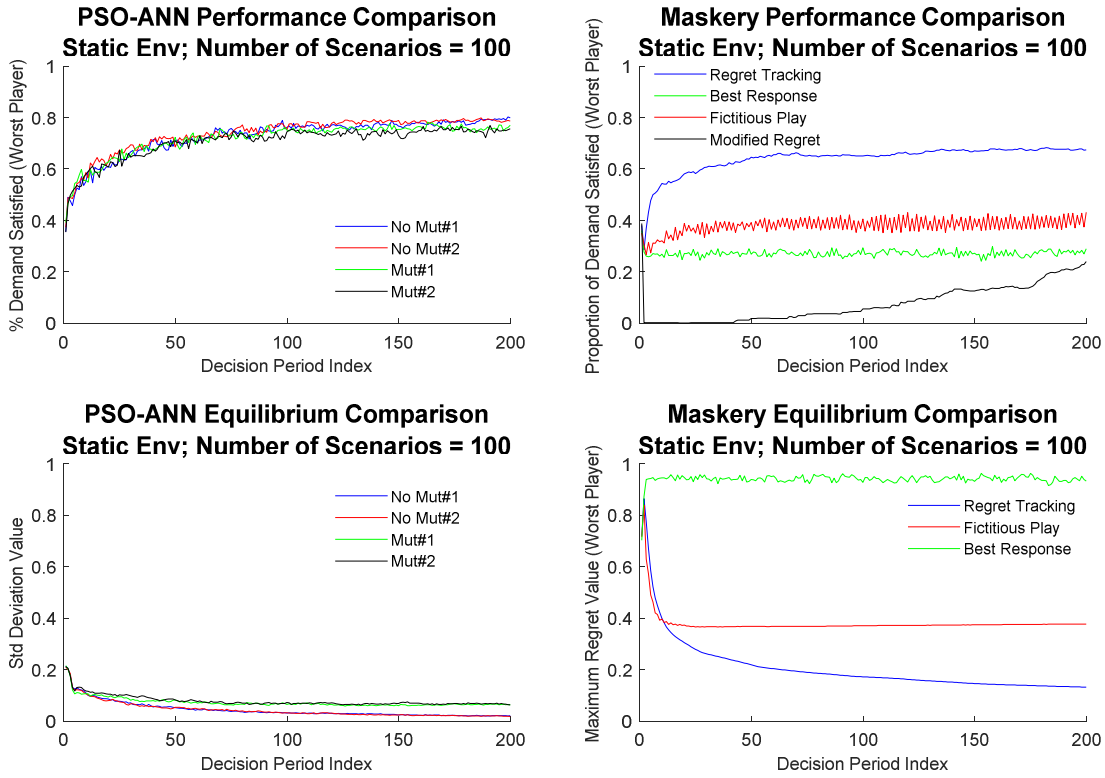

Figure 27: PSO-ANN & Maskery Scalability for *'m'* (static)

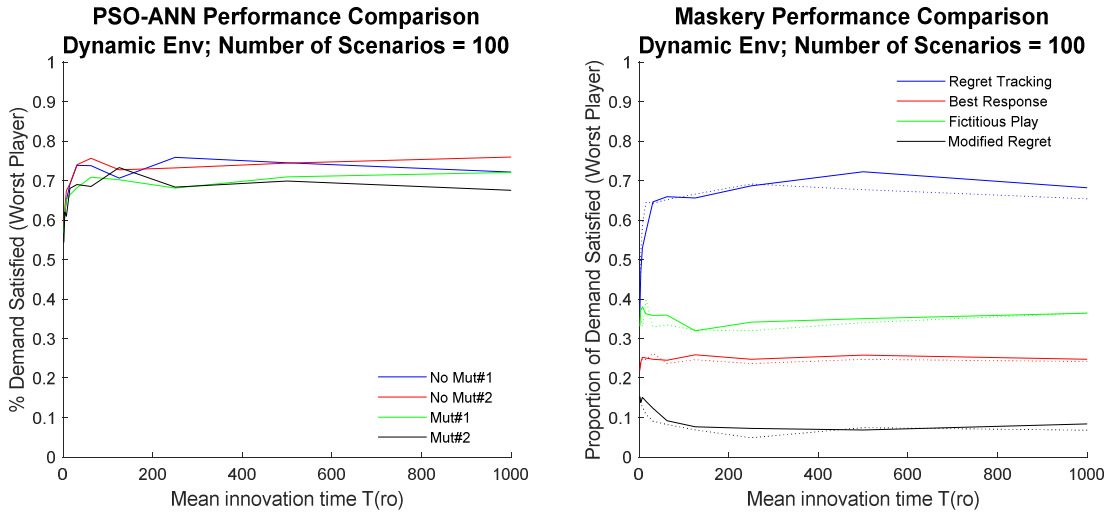

Figure 28: PSO-ANN & Maskery Scalability for *'m'* (dynamic)

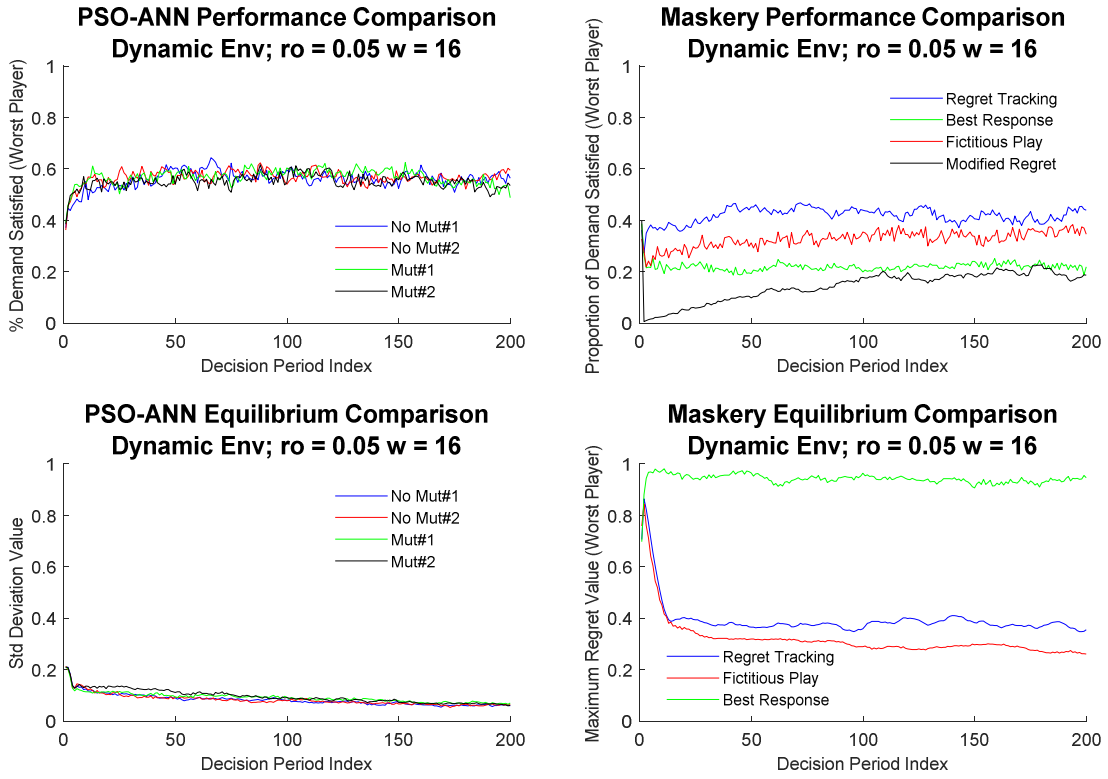

Figure 29: PSO-ANN & Maskery Scalability for *'m'* (ro=0.05)

The following experiments set *'m'* back to 2 but increased the total number of channels to 15. For this case the computational speeds were 355 seconds for PSO-ANN and 294 seconds for regret tracking. This is a small ratio of 1.19:1 in favor of regret tracking. When the number of channels increased to 20 the new speeds were 493.27 seconds for PSO-ANN and 895.70 seconds for Maskery. This corresponds to a 1:1.81 ratio now in favor PSO-ANN. A hybrid *'m'*=3 with 15 channels was attempted but could not run regret tracking due to memory constraints.

The graphs below are for 15 channels. PSO-ANN now hits 100% utility and does so quickly given all the quality available in the system. Surprisingly, regret tracking does not achieve 100% even at 200 iterations. PSO-ANN outperforms regret tracking in the dynamic cases also, providing 80% satisfaction in the fast varying case of ro=0.05 to Maskery's 60%.

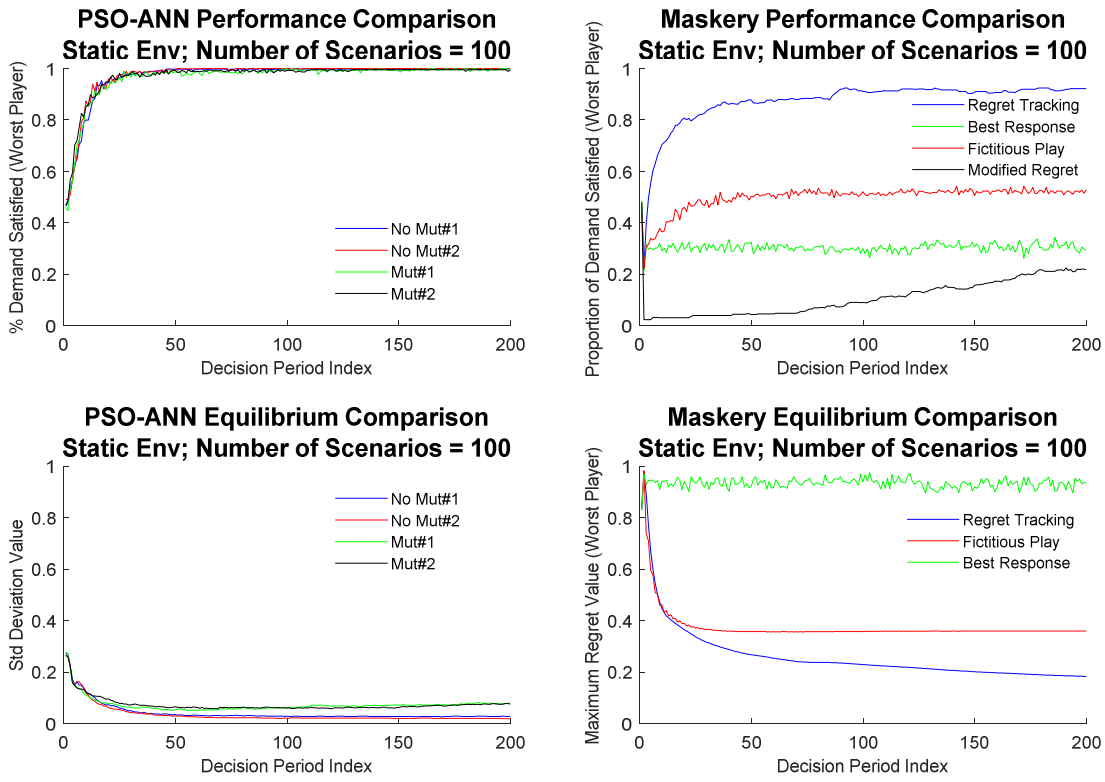

Figure 30: PSO-ANN & Maskery Scalability for *'Ch'* (static)

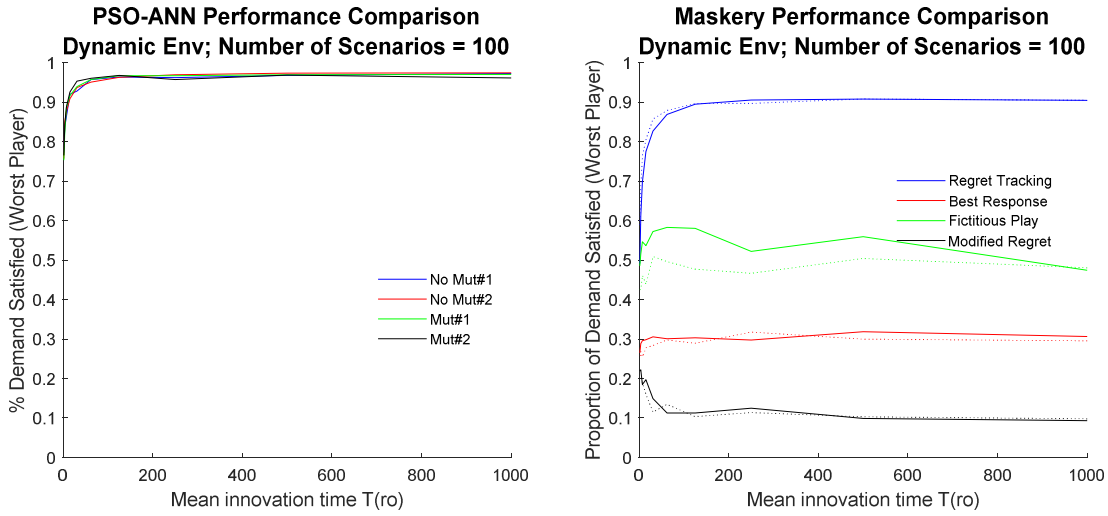

Figure 31: PSO-ANN & Maskery Scalability for *'Ch'* (dynamic)

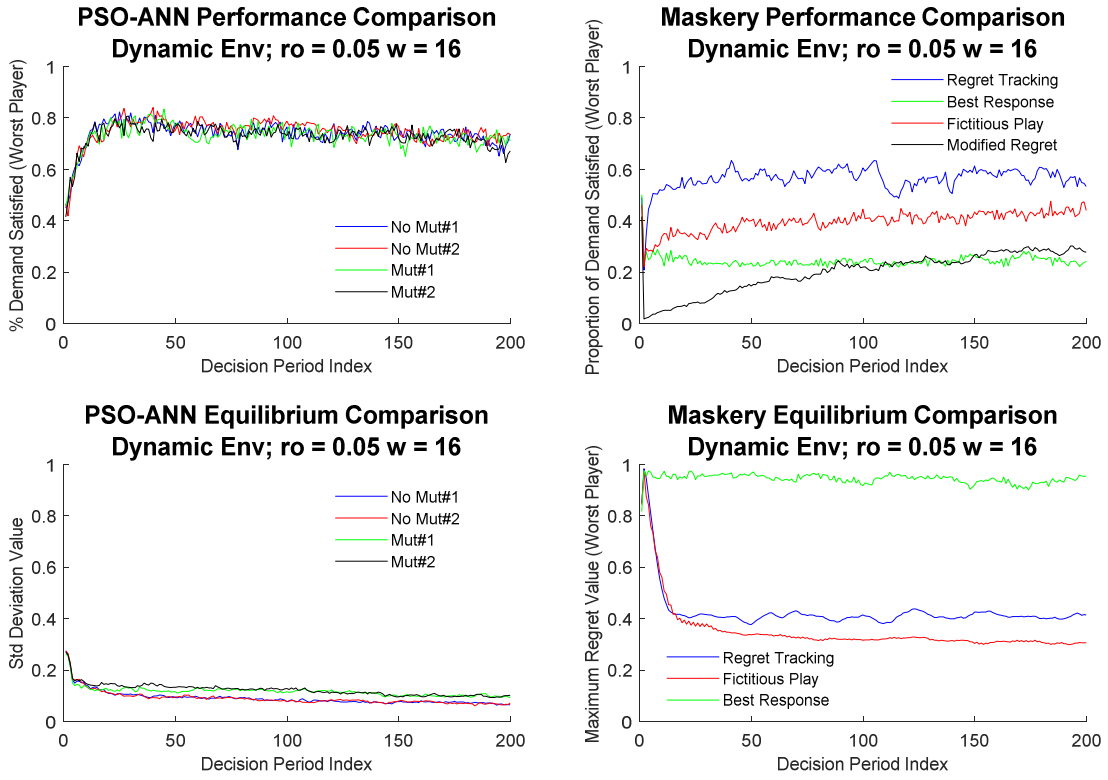

Figure 32: PSO-ANN & Maskery Scalability for *'Ch'* (ro=0.05)

CHAPTER V

V. CONCLUSION

From the findings it was concluded that compared to regret tracking algorithms, PSO-ANN has shown to converge faster and adapt to dynamic environments better. The proportion of demand satisfied was consistently higher in PSO-ANN for every test environment evaluated.

In small simplified networks the regret tracking algorithms compute decisions faster by simply computing every option. However, regret tracking struggles to scale as the problem grows, presenting speed and memory concerns. Regret tracking scales better to number of channels increasing than number of channels allowed per user. Within practical applications, system channel growth is much more likely than allowable channels per user growing, since the latter corresponds to adding transceivers. Nevertheless, the limitation exists and can be a vulnerability. PSO-ANN handles increases in channels per user without any computational burden but has some burden as the number of channels grow. This is due to the ANN weights being proportional to the number of channels. Overall, PSO-ANN is preferable in larger complex systems.

The regret tracking algorithm requires an all-knowing presence in order to compute every option, which is heavy on spectrum sensing. PSO-ANN only requires enough sensing to evaluate a handful of particle options. As mentioned in chapter 1, computers are getting stronger and smaller (more mobile) whereas adding more sensing could make it less mobile. For all these reasons PSO-ANN is strongly recommended over regret tracking for dynamic spectrum access.

5.1 Future Work

The recommended next step is to expand the particle swarm to incorporate some of the hyper-parameters of the ANN that were painfully tuned in chapter 4. As of now it the algorithm is optimizing the weights of an ANN whose topology was created with *a priori* knowledge. This expanded PSO could include the selection quantity of hidden layers, neurons per layer, and transfer function.

After generalizing the algorithm, more inputs could be included and the ANN could optimize itself to use them or not. An example of an additional input is computation time, to incentivize using the minimum amount of neurons and decrease complexity. Another input worth exploring is the hand-off time to switch parameters, this would help to inject stability and deter changes over minimal transmission improvements.

Once the model creation has been generalized by the PSO, it is worth expanding the model and fitness functions to include the hardware side for a physical radio, a specification sheet is provided in [18].

This research could be expanded to a centralized network where a hub or base station would monitor and direct the traffic. In this case, the hub could receive requests from several secondary users and determine the access and routing for each one. Some other systems that exhibit similar characteristics are Wi-Fi, internet of things, self-piloting vehicle, GPS route calculations, routers, fiber multiplexors, etc.

The last improvement recommended is expanding the fitness functions to include an economic analysis. Doing so would bring a better sense of reality to the problem versus the more theoretical and hypothetical nature of it at this stage.

REFERENCES

[1] United States General Accounting Office, "Spectrum Management: FCC's Licensing Approach in the 11, 18, and 23 Gigahertz Bands Currently Supports Spectrum Availability and Efficiency" *Report to Congressional Requesters*, GAO-13-78R, November 2012. [Online]

[2] United States General Accounting Office, "Spectrum Management. Better Knowledge Needed to Take Advantage of Technologies That May Improve Spectrum Efficiency." *Report to Congressional Requesters,* GAO-04-666, May 2004. [Online]

Available: https://www.gao.gov/new.items/d04666.pdf. [Accessed: June 3, 2018]

[3] President's Council of Advisors on Science and Technology, "Realizing the Full Potential of Government-Held Spectrum to Spur Economic Growth." *Report to the President,* July 2012. [Online]

Available: https://obamawhitehouse.archives.gov/sites/default/files/microsites/ostp/pcast_spectrum_report_final_july_20_2012.pdf. [Accessed: June 3, 2018]

[4] Barack Obama, "Presidential Memorandum: Unleashing the Wireless Broadband Revolution." *Memorandum for the Heads of Executive departments and Agencies.* June 28, 2010. [Online]

Available: https://obamawhitehouse.archives.gov/the-press-office/presidential-memorandum-unleashing-wireless-broadband-revolution. [Accessed: June 3, 2018]

[5] Jason Furman and John P. Holdren, “Making the Most of the Wireless Spectrum.” Blog. July 20, 2012. [Online]

Available: https://obamawhitehouse.archives.gov/blog/2012/07/20/making-most-wireless-spectrum [Accessed: June 3, 2018]

[6] Q. Zhao and B. Sadler, “A survey of dynamic spectrum access," IEEE Signal Processing Mag., May 2007. {pp. 2}

[7] Roy Rubenstein, “Radios Get Smart” Feb. 1, 2007 [Online]

Available: https://spectrum.ieee.org/consumer-electronics/standards/radios-get-smart [Accessed Dec. 13, 2014]

[8] M. Maskery, V. Krishnamurthy, Q. Zhao, "Decentralized dynamic spectrum access for cognitive radios: cooperative design of a non-cooperative game", IEEE Trans. Commun., vol. 57, no. 2, pp. 459-469, 2009.

[9] Hossain, Ekram, and Vijay K. Bhargava, eds. Cognitive wireless communication networks. Springer Science & Business Media, 2007.

[10] FCC, “Spectrum policy task force report, ET Docket No. 02-155,” Technical Report Series, November 2002.

[11] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, S. Mohanty, "Next generation/dynamic spectrum access/cognitive radio wireless networks: A survey", Comput. Netw., vol. 50, no. 13, pp. 2127-2159, 2006.

[12] K.J. Ray Liu, “Cognitive Radio and Game Theory” Mar. 22, 2011 [Online]

Available: https://spectrum.ieee.org/telecom/wireless/cognitive-radio-and-game-theory [Accessed Dec. 12, 2014]

[13] Tyler Renelle, "Hyperparameters 1" *Podcast minutes 36-44,* Jan. 1,2018 [Online] Available: http://ocdevel.com/mlg/27 [Accessed: June 4 2018]

[14] W. Duch, & N. Jankowski. "Transfer functions: hidden possibilities for better neural networks." In ESANN (pp. 81-94) April, 2001.

[15] Xin Yao, "Evolving artificial neural networks," in *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423-1447, Sep 1999.

[16] D. Bushey, "RAYTHEON AWARDED $4 MILLION CONTRACT TO CONTINUE DEVELOPMENT OF NEXT GENERATION WIRELESS COMMUNICATIONS SYSTEM FOR DARPA" *Raytheon: Investors: News Release*. [Online] Available at: http://investor.raytheon.com/phoenix.zhtml?c=84193&p=irol-newsArticle&ID=449383 [Accessed 17 Jul. 2018].

[17] Share Spectrum Company, "Shared Spectrum Company Awarded Contract by NOAA/NESDIS" *SSC News.* May 22, 2018. [Online] Available at: http://www.sharedspectrum.com/wp-content/uploads/SSC-SPRES-Press-Release-Task-6-2018.pdf [Accessed 17 Jul. 2018].

[18] Shared Spectrum Company, "Reference Design Specifications – DSA Radio" August 12, 2010. [Online] Available: http://www.sharedspectrum.com/wp-content/uploads/SSC-DSA-2100-Reference-Spec-Sheet.pdf [Accessed: July 16, 2018]

[19] F. Perich, E. Morgan, O. Ritterbush, M. McHenry, & S. D'Itri. "Efficient dynamic spectrum access implementation." MILITARY COMMUNICATIONS CONFERENCE, 2010-MILCOM 2010. IEEE, 2010.

[10] F. Granelli, P. Pawelczak, V. Prasad, K. Subbalakshmi, R. Chandramouli, J. Hoffmeyer, & H.S. Berger. "Standardization and Research in Cognitive and Dynamic Spectrum Access Networks: IEEE SCC41 Efforts and Other Activities," IEEE Communications Magazine, vol. 48, no. 1, pp. 71-79, Jan. 2010 (Copyright IEEE): An update on the activities of SCC41 working groups [posted: 22 January 2010]

[21] P. F. Ribeiro, & W. K. Schlansker, "A hybrid particle swarm and neural network approach for reactive power control." IEEE 2006

[22] Beatriz A. Garro and Roberto A. Vázquez, "Designing Artificial Neural Networks Using Particle Swarm Optimization Algorithms," Computational Intelligence and Neuroscience, vol. 2015, Article ID 369298, 20 pages, 2015. [Online]

Available: https://doi.org/10.1155/2015/369298. [Accessed: April 1, 2018]

[23] C. Wang, Z. Shi, & F, Wu, "An Improved Particle Swarm Optimization-Based Feed-Forward Neural Network Combined with RFID Sensors to Indoor Localization." Information 2017, 8, 9.

[24] K. Stanley, J. Clune, "Welcoming the Era of Deep Neuroevolution" Dec. 18 2017 [Online] Available: https://eng.uber.com/deep-neuroevolution/ [Accessed: June 5, 2018]

[25] I. Khan, A. Waqar, S. Wasi, & S. Khadim. "Comparative Analysis of ANN Techniques for Predicting Channel Frequencies in Cognitive Radio." INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE & APPLICATIONS 8.12 (2017): 296-303.

APPENDICES

The programming suite used for this research is MATLAB, created by MathWorks. The code was developed and can run on MATLAB2012 but about 30% improvement in computational speeds was obtained after upgrading to a trial of MATLAB2018. Moreover, to run the code provided, a parallel computing add-on is recommended. This add-on allows MATLAB to run on all the cores of the computer, significantly speeding up run-time even further. If the parallel computing add-on is not available, replace the 'parfor' loop in the code with a regular 'for' loop.

Each subsection of the appendices corresponds to a MATLAB script, known as '.m file' given the '.m' extension. Within a script, many functions may reside but usually at least one function of the same name as the script. Sometimes supporting functions may be located in a script as well, as long as these functions are not being called outside of that file.

Script 'Main' was used to setup the graphs of section 4.4 while script 'Optimize_PSO_ANN' was used for the experiments in 4.2-4.3. 'PSO_ANN_FullSimulation' is used to create PSO-ANN graphs used by both 'Main' and 'Optimize_PSO_ANN'.

'MaskerySystem' and 'Fitness_Func' are scripts that follow the outline of the static and dynamic systems presented in the methodology chapter, sections 3.1-3.2. 'MaskeryDecision' is the where the regret tracking algorithms reside that correspond to the theory presented in section 3.3.

'PSO_ANN' and 'ANN_CalcOutput' are the functions that contain the PSO-ANN algorithm. The former follows the PSO explanation of section 3.5 and the latter is a feed-forward computation of the neural network explained in 3.4.

Main.m

```
clc;
clear;
close all;
global SecUser PriUser NumChannels m K
global Tmax delta beta alpha1 alpha2
global NumScenarios DecisionPeriodIndex ro w
global delta_exploration

%Initialize constants
SecUser = 6; %Total # of Cog Radio Users
PriUser = 2;
NumChannels = 10;
m = 2; %num of transceivers on radio.
K = 20;  %CSMA attempts per decision period
Tmax = 0.00001; %10us max back-off time for CSMA
delta = Tmax/10;
beta = 0; %Grace for exceeding demand
alpha1 = 0.2;
alpha2 = 1.8;
ro = 0; %Zero Means Do Nothing - Static Environment
w = 8;

%Global parameters for PSO-ANN
global Num_Elements NumParticles
global ANN_TransferFunctions % 1=HardLim; 2=Log-Sigmoid; 3:Tanh-Sigmoid; 4:ReLU;
global ANN_Bias % 1= Use Bias; 2= Do not use bias;
global Init_Position_Range Init_Velocity_Range
global PSO_W MutResetInterval% PSO_W = [a1 a2 b1 b2 c1 c2];

NumParticles = 5;
Num_Elements = NumChannels*(NumChannels+1)+(NumChannels^2)+2*NumChannels; %w1
+ w2 + 2*bias
```

```
MutResetInterval = 50;
Init_Velocity_Range = [0.6 0.8];
PSO_W = [0.75 0.2 0.6];
ANN_TransferFunctions = 2;
ANN_Bias = 1;
Init_Position_Range = [0 1];

Maskery_Static = 1;
Maskery_Dynamic = 0;
ANNPSO_Static = 0;
ANNPSO_Dynamic = 0;
% DecisionPeriodIndex = 1000; % Full
% NumScenarios = 100; % Full
DecisionPeriodIndex = 50; %Short For Testing
NumScenarios = 3; %Short For Testing
delta_exploration = 0.01;%Mod regret tracking

ANN_TransferFunctions = 2;
Init_Position_Range = [0 1];
Init_Velocity_Range = [0.6 0.8];
DecisionPeriodIndex = 3000;
NumScenarios = 100;
NumChannels = 15;
MutResetInterval=-50;
Num_Elements = NumChannels*(NumChannels+1)+(NumChannels^2)+2*NumChannels;
m = 2; %num of transceivers on radio.

tic
PSO_ANN_FullSimulation(1)
PSO_ANN_FullSimulation(2)
PSO_ANN_FullSimulation(3)
toc
```

```
tic
Maskery_FullSimulation(1)
Maskery_FullSimulation(2)
Maskery_FullSimulation(3)
toc

SecUser = 6; %Total # of Cog Radio Users
PriUser = 2;
NumChannels = 15;
Num_Elements = NumChannels*(NumChannels+1)+(NumChannels^2)+2*NumChannels;
m = 2; %num of transceivers on radio.
DecisionPeriodIndex = 10; %Short For Testing
NumScenarios = 3; %Short For Testing
tic
PSO_ANN_FullSimulation(1)
PSO_ANN_FullSimulation(2)
PSO_ANN_FullSimulation(3)
toc
tic
Maskery_FullSimulation(1)
Maskery_FullSimulation(2)
Maskery_FullSimulation(3)
toc

fprintf('\n \n')
```

```
function Maskery_FullSimulation(SimulationType)

global NumScenarios DecisionPeriodIndex ro w
global LocalU_Avg LocalU_min
if SimulationType == 1
    fprintf('\n Running Maskery Static ...\n')
    [Global_U_Regret, MaxRegVal_Regret] = MaskerySystem(1);
    [GlobalU_BestResponse, MaxRegret_BestResponse] = MaskerySystem(2);
    [GlobalU_FictPlay, MaxRegret_FictPlay] = MaskerySystem(3);
    [Global_U_ModRegret, ~] = MaskerySystem(4);

    figure('rend','painters','pos',[1000 100 440 640])
    subplot(2,1,1);
    hold on
    plot(Global_U_Regret, 'b');
    plot(GlobalU_BestResponse,'g');
    plot(GlobalU_FictPlay,'r');
    plot(Global_U_ModRegret, 'k');
    hold off

    title({'Maskery Performance Comparison', ['Static Env; Number of Scenarios = '
num2str(NumScenarios)]}, 'FontSize',14)
    xlabel('Decision Period Index', 'FontSize',10);
    ylabel('Proportion of Demand Satisfied (Worst Player)', 'FontSize',10)
    ylim([0 1]);
    legend('Regret Tracking', 'Best Response', 'Fictitious Play', 'Modified Regret', 'Location', 'Best')
    legend boxoff

    subplot(2,1,2);
    hold on
    plot(MaxRegVal_Regret, 'b');
    plot(MaxRegret_FictPlay,'r');
    plot(MaxRegret_BestResponse,'g');
    % plot(MaxRegVal_ModRegret, 'k');
```

```
    hold off

    title({'Maskery Equilibrium Comparison', ['Static Env; Number of Scenarios = '
num2str(NumScenarios)]}, 'FontSize',14)
    xlabel('Decision Period Index', 'FontSize',10);
    ylabel('Maximum Regret Value (Worst Player)', 'FontSize',10)
    ylim([0 1]);
    legend('Regret Tracking','Fictitious Play', 'Best Response', 'Location', 'Best')
    legend boxoff
end

if SimulationType == 2
    % Call Maskery a few times and plot demand satisfied vs mean innovation time
    T_ro = [1000 500 250 125 62.5 31.25 15.625 7.8125 3.90625 1.953125];
    ro_w8 = [0.000125055 0.000250219 0.000500877 0.001003518 0.002014142 ...
        0.004057146 0.008233394 0.016975005 0.036289442 0.085776295];

    D8_Global_U_Regret = zeros(10,DecisionPeriodIndex);
    D8_GlobalU_BestResponse= zeros(10,DecisionPeriodIndex);
    D8_GlobalU_FictPlay= zeros(10,DecisionPeriodIndex);
    D8_Global_U_ModRegret= zeros(10,DecisionPeriodIndex);

    fprintf('\n Running Maskery Dynamic w=8 ...\n')
    for i = 1:10
        ro = ro_w8(i);
        [D8_Global_U_Regret(i, :), ~] = MaskerySystem(1);
        [D8_GlobalU_BestResponse(i, :), ~] = MaskerySystem(2);
        [D8_GlobalU_FictPlay(i, :), ~] = MaskerySystem(3);
        [D8_Global_U_ModRegret(i, :), ~] = MaskerySystem(4);
    end
```

```
w = 16;
% Call Maskery a few times and plot demand satisfied vs mean innovation time

ro_w16 = [6.25293e-5 0.000125117 0.00025047 0.000501885 0.001007578 ...
    0.002030635 0.004125206 0.00852383 0.018312393 0.043849539];

D16_Global_U_Regret = zeros(10,DecisionPeriodIndex);
D16_GlobalU_BestResponse= zeros(10,DecisionPeriodIndex);
D16_GlobalU_FictPlay= zeros(10,DecisionPeriodIndex);
D16_Global_U_ModRegret= zeros(10,DecisionPeriodIndex);

fprintf('\n Running Maskery Dynamic w=16 ...\n')
for i = 1:10
    ro = ro_w16(i);
    [D16_Global_U_Regret(i, :), ~] = MaskerySystem(1);
    [D16_GlobalU_BestResponse(i, :), ~] = MaskerySystem(2);
    [D16_GlobalU_FictPlay(i, :), ~] = MaskerySystem(3);
    [D16_Global_U_ModRegret(i, :), ~] = MaskerySystem(4);
end
figure('rend','painters','pos',[1500 100 475 425])
hold on
plot(T_ro, mean(D8_Global_U_Regret, 2), 'b');
plot(T_ro, mean(D8_GlobalU_BestResponse, 2),'r');
plot(T_ro, mean(D8_GlobalU_FictPlay, 2),'g');
plot(T_ro, mean(D8_Global_U_ModRegret, 2), 'k');
plot(T_ro, mean(D16_Global_U_Regret, 2), 'b:');
plot(T_ro, mean(D16_GlobalU_BestResponse, 2),'r:');
plot(T_ro, mean(D16_GlobalU_FictPlay, 2),'g:');
plot(T_ro, mean(D16_Global_U_ModRegret, 2), 'k:');
hold off

title({'Maskery Performance Comparison',  ['Dynamic Env; Number of Scenarios = '
num2str(NumScenarios)]}, 'FontSize',14)
xlabel('Mean innovation time T(ro)', 'FontSize',12);
```

```
    ylabel('Proportion of Demand Satisfied (Worst Player)', 'FontSize',12)
    ylim([0 1]);
    legend('Regret Tracking','Best Response', 'Fictitious Play', 'Modified Regret', 'Location', 'Best')
    legend boxoff
end

if SimulationType == 3
    w = 16;
    ro = 0.05;
    fprintf(['\n Running Maskery Dynamic; ro = ' num2str(ro) ' ...\n'])
    [Global_U_Regret, MaxRegVal_Regret] = MaskerySystem(1);
    [GlobalU_BestResponse, MaxRegret_BestResponse] = MaskerySystem(2);
    [GlobalU_FictPlay, MaxRegret_FictPlay] = MaskerySystem(3);
    [Global_U_ModRegret, ~] = MaskerySystem(4);

    figure('rend','painters','pos',[1000 100 440 640])
    subplot(2,1,1);
    hold on
    plot(Global_U_Regret, 'b');
    plot(GlobalU_BestResponse,'g');
    plot(GlobalU_FictPlay,'r');
    plot(Global_U_ModRegret, 'k');
    hold off

    title({'Maskery Performance Comparison', ['Dynamic Env; ro = ' num2str(ro) ' w = '
num2str(w)]}, 'FontSize',14)
    xlabel('Decision Period Index', 'FontSize',10);
    ylabel('Proportion of Demand Satisfied (Worst Player)', 'FontSize',10)
    ylim([0 1]);
    legend('Regret Tracking', 'Best Response', 'Fictitious Play', 'Modified Regret', 'Location', 'Best')
    legend boxoff

    subplot(2,1,2);
    hold on
```

```
    plot(MaxRegVal_Regret, 'b');
    plot(MaxRegret_FictPlay,'r');
    plot(MaxRegret_BestResponse,'g');
    % plot(MaxRegVal_ModRegret, 'k');
    hold off

    title({'Maskery Equilibrium Comparison', ['Dynamic Env; ro = ' num2str(ro) ' w = '
num2str(w)]}, 'FontSize',14)
    xlabel('Decision Period Index', 'FontSize',10);
    ylabel('Maximum Regret Value (Worst Player)', 'FontSize',10)
    ylim([0 1]);
    legend('Regret Tracking','Fictitious Play', 'Best Response', 'Location', 'Best')
    legend boxoff
end

if SimulationType == 4
    fprintf('\n Running Maskery Static "2" ...\n')
    [Global_U_Regret, ~] = MaskerySystem(1);

    hold on
    plot(Global_U_Regret,'r');
    plot(LocalU_Avg, 'b')
    plot(LocalU_min, 'g')
    hold off

    title({'Performance Comparison of Maskery Techniques', '(Static Environment)', ['Number of
Scenarios = ' num2str(NumScenarios)]}, 'FontSize',14)
    xlabel('Decision Period Index', 'FontSize',12);
    ylabel('Proportion of Demand Satisfied (Worst Player)', 'FontSize',12)
    ylim([0 1]);
    legend('Global U', 'Avg Local U', 'Min Local U', 'Location', 'Best')
    legend boxoff
end
end
```

Optimize_PSO_ANN.m

```
function Optimize_PSO_ANN

global SecUser PriUser NumChannels delta Tmax
global NumScenarios DecisionPeriodIndex ro w m
global beta alpha1 alpha2;
%Initialize constants
SecUser = 6; %Total # of Cog Radio Users
PriUser = 2;
NumChannels = 10;
m = 2; %num of transceivers on radio.
K = 20;  %CSMA attempts per decision period
%K = 2;  %CSMA attempts per decision period
Tmax = 0.00001; %10us max back-off time for CSMA
delta = Tmax/10;
beta = 0; %Grace for exceeding demand
alpha1 = 0.2;
alpha2 = 1.8;
ro = 0; %Zero Means Do Nothing - Static Environment
w = 8;

%Global parameters for PSO-ANN
global Num_Elements NumParticles
global ANN_TransferFunctions % 1=HardLim; 2=Log-Sigmoid; 3:Tanh-Sigmoid; 4:ReLU;
global ANN_Bias % 1= Use Bias; 2= Do not use bias;
global Init_Position_Range Init_Velocity_Range
global PSO_W % PSO_W = [a1 a2 b1 b2 c1 c2];
global MutResetInterval % 0= off; # = how many intervals of GB=GB_old before mutation
global PSO_TurnOff

NumParticles = 3;
Num_Elements = NumChannels*(NumChannels+2)+(NumChannels^2)+2*NumChannels; %w1
+ w2 + 2*bias
```

```
MutResetInterval = 0; %(NumScenarios/6);
%Setting these in the event I comment things out. They should default to best findings
ANN_TransferFunctions = 2;
ANN_Bias = 1;
Init_Position_Range = [0 1];
PSO_W = [0.75 0.2 0.4];
Init_Velocity_Range = [0.6 0.8];


   DecisionPeriodIndex = 2%00; %Short For Testing
   NumScenarios = 1%0%0; %Short For Testing
t = clock;
PSO_W = [0.1 0.1 0.1]; %Original for testing
Init_Velocity_Range = [-0.2 0.2];
   DecisionPeriodIndex = 1000; %Short For Testing
   NumScenarios = 100; %Short For Testing
ANN_Init(1); % To determine best XFER-func, weights and bias on/off.
%what I learned:
ANN_TransferFunctions = 2;
   DecisionPeriodIndex = 200;
   NumScenarios = 100;
ANN_Init(2); %For verification, static case
ANN_Init(3); %For verification, dynamic case
%More of what I learned
ANN_Bias = 1;
Init_Position_Range = [0 1];
   DecisionPeriodIndex = 10; %More scenarios, shorter decision period
   NumScenarios = 1000;
Velocity_Init(1); %To determine starter velocity in PSO
Init_Velocity_Range = [0.6 0.8]; %What I learned
   DecisionPeriodIndex = 100;
   NumScenarios = 100;
PSO_W_temp = PSO_Init(1, 0); %To determine a/b/c in PSO
   DecisionPeriodIndex = 200;
   NumScenarios = 100;
```

```
PSO_Init(2, PSO_W_temp); %For verification, static case
PSO_Init(3, PSO_W_temp); %For verification, dynamic case
%what I learned
PSO_W = [0.75 0.2 0.6];
   DecisionPeriodIndex = 200;
   NumScenarios = 200;
Mut_Init(1); %To determine mutations percentage and interval
   DecisionPeriodIndex = 200;
   NumScenarios = 100;
%what I learned
MutResetInterval = 50; %Regular mutations as determined by Mut_Init(1);
   DecisionPeriodIndex = 100;
   NumScenarios = 100;
PSO_W_temp = PSO_Init(1, 0); %To determine a/b/c in PSO
   DecisionPeriodIndex = 200;
   NumScenarios = 100;
%     PSO_W_temp = [1 0.6 0.4; 1 0.4 0.6; 0.75 0.2 0.4] %For use w/Mutations
   PSO_W_temp = [0.75 0.2 0.4; 0.75 0.4 0.4; 0.75 0 0.6] %For use w/o Mutations
PSO_Init(2, PSO_W_temp); %For verification, static case
PSO_Init(3, PSO_W_temp); %For verification, dynamic case


   ANN_TransferFunctions = 2;
   Init_Position_Range = [0 1];
   Init_Velocity_Range = [0.6 0.8];
   DecisionPeriodIndex = 30%00;
   NumScenarios = 100;
   PSO_ANN_FullSimulation(1)
   PSO_ANN_FullSimulation(2)


fprintf(['\n Total time elapsed ' num2str(etime(clock, t)) 'seconds ...\n'])
 end
```

```
function ANN_Init(ANN_Init_Check)

datetime('now')
global ro NumScenarios ANN_TransferFunctions Init_Position_Range
global w DecisionPeriodIndex ANN_Bias
ANN_Bias_old = ANN_Bias;
w_old = w;
ro_old = ro;
ANN_TransferFunctions_old = ANN_TransferFunctions;
Init_Position_Range_old = Init_Position_Range;


ro = 0;
Init_Position_Range_temp = [0 1; -1 1; -0.2 1; -0.5 0.5];
max_i = size(Init_Position_Range_temp,1);
plotStyle = {'b','r','g','k','b:','r:','g:','k:'};
plotTitle = {'Hard-Limiter', 'Log-Sigmoid', 'TanH-Sigmoid', 'ReLU'};
if ANN_Init_Check == 1
   figure('rend','painters','pos',[0 0 680 600])
   for j=1:4
      tic
      fprintf(['\n Running ANN Initialization ' plotTitle{j} '...\n'])
      ANN_TransferFunctions = j;
      subplot(2,2,j)
      hold on
      ANN_Bias = 1;
      for i=1:max_i
         Init_Position_Range = Init_Position_Range_temp(i, :);
         [GlobalU, ~] = MaskerySystem(12);
         plot(GlobalU, plotStyle{i});
         legendInfo{i} =  ['[' num2str(Init_Position_Range_temp(i, 1)) ' '
num2str(Init_Position_Range_temp(i, 2)) ']'];
      end
      ANN_Bias = 0;
      for i=1:max_i
```

```
            Init_Position_Range = Init_Position_Range_temp(i, :);
            [GlobalU, ~] = MaskerySystem(12);
            plot(GlobalU, plotStyle{i+max_i});
        end
        title(plotTitle{j}, 'FontSize',14)
        xlabel('Decision Period Index', 'FontSize',10);
        ylabel('% Demand Satisfied (Worst Player)', 'FontSize',10)
        ylim([0 1])
        legend(legendInfo, 'Location', 'Best')
        hold off
        toc
    end
end

if ANN_Init_Check == 2
    tic
    ANN_TransferFunctions = 2;
    fprintf(['\n Running ANN Extensive Initialization ' plotTitle{ANN_TransferFunctions} '...\n'])
    figure('rend','painters','pos',[10 10 440 640])
    subplot1 = subplot(2,1,1);
    subplot2 = subplot(2,1,2);
    ANN_Bias = 1;
    for i=1:max_i
        Init_Position_Range = Init_Position_Range_temp(i, :);
        [GlobalU, Converge] = MaskerySystem(12);
        hold(subplot1, 'on')
        plot(subplot1, GlobalU, plotStyle{i});
        hold(subplot1, 'off')
        hold(subplot2, 'on')
        plot(subplot2, Converge, plotStyle{i});
        hold off
        legendInfo{i} =  ['[' num2str(Init_Position_Range_temp(i, 1)) ' '
num2str(Init_Position_Range_temp(i, 2)) ']'];
    end
```

```
    ANN_Bias = 0;
    for i=1:max_i
        Init_Position_Range = Init_Position_Range_temp(i, :);
        [GlobalU, Converge] = MaskerySystem(12);
        hold(subplot1, 'on')
        plot(subplot1, GlobalU, plotStyle{i+max_i});
        hold(subplot1, 'off')
        hold(subplot2, 'on')
        plot(subplot2, Converge, plotStyle{i+max_i});
        hold off
    end
    title(subplot1, {['Performance Comparison of ' plotTitle{ANN_TransferFunctions}], ['Static
Env; Number of Scenarios = ' num2str(NumScenarios)]}, 'FontSize',14);
    xlabel(subplot1, 'Decision Period Index', 'FontSize',10);
    ylabel(subplot1, '% Demand Satisfied (Worst Player)', 'FontSize',10);
    ylim(subplot1, [0 1]);
    legend(subplot1, legendInfo, 'Location', 'Best');

    title(subplot2, {['Equilibrium Comparison of ' plotTitle{ANN_TransferFunctions}], ['Static
Env; Number of Scenarios = ' num2str(NumScenarios)]}, 'FontSize',14);
    xlabel(subplot2, 'Decision Period Index', 'FontSize',10);
    ylabel(subplot2, 'Std Deviation Value', 'FontSize',10);
    legend(subplot2, legendInfo, 'Location', 'Best');
    toc
end

if ANN_Init_Check == 3
    datetime('now')
    tic
    w = 16;
    % Call ANN PSO a few times and plot demand satisfied vs mean innovation time
    T_ro = [1000 500 250 125 62.5 31.25 15.625 7.8125 3.90625 1.953125];
    ro_w16 = [6.25293e-5 0.000125117 0.00025047 0.000501885 0.001007578 ...
        0.002030635 0.004125206 0.00852383 0.018312393 0.043849539];
```

```
    ANN_TransferFunctions = 2;
    GlobalU = zeros(10,DecisionPeriodIndex);
    GlobalU_noBias = zeros(10,DecisionPeriodIndex);
    fprintf('\n Running ANN PSO Dynamic w=16 ...\n')

    figure('rend','painters','pos',[10 10 475 425])
    hold on
    ANN_Bias = 1;
    for i=1:max_i
        Init_Position_Range = Init_Position_Range_temp(i, :);
        fprintf(['On InitPositions=[' num2str(Init_Position_Range_temp(i, 1)) ...
            ' ' num2str(Init_Position_Range_temp(i, 2)) '] & with Bias\n']);
        for j = 1:10 %Cutting down on computation
            ro = ro_w16(j);
            [GlobalU(j, :), ~] = MaskerySystem(12);
        end
        plot(T_ro, mean(GlobalU, 2), plotStyle{i});
        legendInfo{i} =  ['[' num2str(Init_Position_Range_temp(i, 1)) ' '
num2str(Init_Position_Range_temp(i, 2)) ']'];
    end
    ANN_Bias = 0;
    for i=1:max_i
        Init_Position_Range = Init_Position_Range_temp(i, :);
        fprintf(['On InitPositions=[' num2str(Init_Position_Range_temp(i, 1)) ...
            ' ' num2str(Init_Position_Range_temp(i, 2)) '] & no Bias\n']);
        for j = 1:10 %Cutting down on computation
            ro = ro_w16(j);
            [GlobalU(j, :), ~] = MaskerySystem(12);
        end
        plot(T_ro, mean(GlobalU, 2), plotStyle{i+max_i});
    end
    title({'Performance Comparison of Log-Sigmoids', ['Dynamic Env; Number of Scenarios = '
num2str(NumScenarios)]}, 'FontSize',14)
    xlabel('Mean innovation time T(ro)', 'FontSize',12);
```

```
    ylabel('% Demand Satisfied (Worst Player)', 'FontSize',12)
    legend(legendInfo, 'Location', 'Best')
    hold off
    toc
end
ANN_Bias = ANN_Bias_old;
w = w_old;
ro = ro_old;
ANN_TransferFunctions = ANN_TransferFunctions_old;
Init_Position_Range = Init_Position_Range_old;
end


function PSO_W_temp = PSO_Init(PSO_Init_Check, PSO_W_temp)

global NumScenarios DecisionPeriodIndex ro w
global ANN_Bias % 1= Use Bias; 2= Do not use bias;
global PSO_W MutResetInterval% PSO_W = [a1 a2 b1 b2 c1 c2];
% PSO_W_temp = [0.75 0.2 0.6; 0.25 2 0.6; 0.75 0.6 0.4];
% PSO_W_temp = [1 0.4   0.8; 1  1.2 0.2; 1  0.6 0.4];
max_i = size(PSO_W_temp,1);
plotStyle = {'b','r','g','b:','r:','g:'};
ANN_Bias_old = ANN_Bias;
MutResetInterval_old = MutResetInterval;
PSO_W_old = PSO_W;
w_old = w;
ro_old = ro;

if PSO_Init_Check == 1
    datetime('now')
    tic
%     ANN_TransferFunctions = 2;
%     ANN_Bias = 1;
%     Init_Position_Range = [0 1];
```

```
    fprintf('\n Running PSO Extensive Initialization Log Sigmoid...\n')
    [X,Y,Z] = meshgrid((0:10)/5, (0:10)/5,(1:4)/4);
    C = zeros(size(X));
    for i=1:numel(X)
        PSO_W = [Z(i) X(i) Y(i)];
        [GlobalU_PSOInit, ~] = MaskerySystem(12);
        C(i) = mean(GlobalU_PSOInit, 2);
    end
    [value, idx] = max(C(1:numel(X)));

    figure('rend','painters','pos',[10 50 600 550])
    hold on
    for i=1:numel((1:4)/4)
        surf(X(:, :, i), Y(:, :, i), Z(1,1,i)*ones(size(X(:, :, 1))), C(:, :, i))
    end
    caxis([min(C(1:numel(X))) max(C(1:numel(X)))]);
    colorbar
    title({'Average Demand Satisfied', ...
        [num2str(DecisionPeriodIndex) ' Decisions, ' num2str(NumScenarios) ' Scenarios'], ...
        ['Max of ' num2str(value) ' at a=' num2str(Z(idx)) ', b=' num2str(X(idx)) ', c='
num2str(Y(idx))]}, 'FontSize',14)
    xlabel('Personal Best (b) ', 'FontSize',12);
    ylabel('Global Best (c) ', 'FontSize',12);
    zlabel('Inertia Coefficient (a)', 'FontSize',12)
    view([-35 15])
    hold off

    figure('rend','painters','pos',[1000 50 600 550])
    hold on
    for i=1:numel((1:4)/2)
        subplot(2, 2, i)
        surf(X(:, :, i), Y(:, :, i), C(:, :, i))
        colorbar
```

```
        title({'Average Demand Satisfied', [num2str(DecisionPeriodIndex) ' Decisions, '
num2str(NumScenarios) ' Scenarios'], ['a = ' num2str(Z(1,1,i))]}, 'FontSize',14)
        xlabel('Personal Best (b) ', 'FontSize',12);
        ylabel('Global Best (c) ', 'FontSize',12);
        caxis([min(C(1:numel(X))) max(C(1:numel(X)))]);
        view([0 90])
    end
    hold off

    figure('rend','painters','pos',[1000 50 200 1000])
    hold on
    for i=1:numel((1:4)/2)
        subplot(4, 1, i)
        surf(X(:, :, i), Y(:, :, i), C(:, :, i))
        title({['a = ' num2str(Z(1,1,i))]}, 'FontSize',14)
        xlabel('Personal Best (b) ', 'FontSize',12);
        ylabel('Global Best (c) ', 'FontSize',12);
        caxis([min(C(1:numel(X))) max(C(1:numel(X)))]);
        view([0 90])
    end
    hold off

    C_temp = C;
    [value, idx] = maxk(C_temp(1:numel(X)),10);
    BestValues = [Z(idx); X(idx); Y(idx); value;]'
    filename = 'testdata.xlsx';
    xlswrite(filename,BestValues,1,'B2')

    [~, idx] = maxk(C_temp(1:numel(X)),3);
    PSO_W_temp = [Z(idx); X(idx); Y(idx);]';
    toc
end

if PSO_Init_Check == 2
```

```
    datetime('now')
    tic
    fprintf('\n Running PSO Extensive Init with Mutations; Log Sigmoid; Static...\n')

    figure('rend','painters','pos',[10 10 480 640])
    subplot1 = subplot(2,1,1);
    subplot2 = subplot(2,1,2);
    ANN_Bias = 1;
    for i=1:max_i
        PSO_W = PSO_W_temp(i, :);
        [GlobalU, Converge] = MaskerySystem(12);
        hold(subplot1, 'on')
        plot(subplot1, GlobalU, plotStyle{i});
        hold(subplot1, 'off')
        hold(subplot2, 'on')
        plot(subplot2, Converge, plotStyle{i});
        hold off
        legendInfo{i} =  [num2str(PSO_W_temp(i, 1)) '/' num2str(PSO_W_temp(i, 2)) '/'
num2str(PSO_W_temp(i, 3))];
    end
    ANN_Bias = 0;
    for i=1:max_i
        PSO_W = PSO_W_temp(i, :);
        [GlobalU, Converge] = MaskerySystem(12);
        hold(subplot1, 'on')
        plot(subplot1, GlobalU, plotStyle{i+max_i});
        hold(subplot1, 'off')
        hold(subplot2, 'on')
        plot(subplot2, Converge, plotStyle{i+max_i});
        hold off
    end
    title(subplot1, {'PSO Hyper-Parameters Performance Comparison', ['Static Env; Number of
Scenarios = ' num2str(NumScenarios)]}, 'FontSize',14)
    xlabel(subplot1, 'Decision Period Index', 'FontSize',10);
```

```
    ylabel(subplot1, '% Demand Satisfied (Worst Player)', 'FontSize',10)
    ylim(subplot1, [0 1]);
    legend(subplot1, legendInfo, 'Location', 'Best');

    title(subplot2, {'PSO Hyper-Parameters Equilibrium Comparison', ['Static Env; Number of
Scenarios = ' num2str(NumScenarios)]}, 'FontSize',14)
    xlabel(subplot2, 'Decision Period Index', 'FontSize',10);
    ylabel(subplot2, 'Std Deviation Value', 'FontSize',10)
    legend(subplot2, legendInfo, 'Location', 'Best');
    toc
end

if PSO_Init_Check == 3
    datetime('now')
    tic
    w = 16;    % Call ANN PSO a few times and plot demand satisfied vs mean innovation time
    T_ro = [1000 500 250 125 62.5 31.25 15.625 7.8125 3.90625 1.953125];
    ro_w16 = [6.25293e-5 0.000125117 0.00025047 0.000501885 0.001007578 ...
        0.002030635 0.004125206 0.00852383 0.018312393 0.043849539];
    fprintf('\n Running PSO Extensive Init with Mutations; Log Sigmoid; Dynamic...\n')

    GlobalU = zeros(10,DecisionPeriodIndex);
    GlobalU_Mut = zeros(10,DecisionPeriodIndex);

    fprintf('\n Running ANN PSO Dynamic w=16 ...\n')
    figure('rend','painters','pos',[10 10 475 425])
    hold on
    ANN_Bias = 1;
    for i=1:max_i
        PSO_W = PSO_W_temp(i, :);
        fprintf(['On Hyper-params= ' num2str(PSO_W_temp(i, 1)) '/' ...
            num2str(PSO_W_temp(i, 2)) '/' num2str(PSO_W_temp(i, 3)) ' & with Bias\n']);
        for j = 1:10 %Cutting down on computation
            ro = ro_w16(j);
```

```
            [GlobalU(j, :), ~] = MaskerySystem(12);
        end
        plot(T_ro, mean(GlobalU, 2), plotStyle{i});
        legendInfo{i} =  [num2str(PSO_W_temp(i, 1)) '/' num2str(PSO_W_temp(i, 2)) '/'
num2str(PSO_W_temp(i, 3))];
    end
    ANN_Bias = 0;
    for i=1:max_i
        PSO_W = PSO_W_temp(i, :);
        fprintf(['On Hyper-params= ' num2str(PSO_W_temp(i, 1)) '/' ...
            num2str(PSO_W_temp(i, 2)) '/' num2str(PSO_W_temp(i, 3)) ' & without Bias\n']);
        for j = 1:10 %Cutting down on computation
            ro = ro_w16(j);
            [GlobalU(j, :), ~] = MaskerySystem(12);
        end
        plot(T_ro, mean(GlobalU, 2), plotStyle{i+max_i});
    end
    title({'PSO Hyper-Parameters Performance Comparison', ['Dynamic Env; Number of Scenarios
= ' num2str(NumScenarios)]}, 'FontSize',14)
    xlabel('Mean innovation time T(ro)', 'FontSize',12);
    ylabel('% Demand Satisfied (Worst Player)', 'FontSize',12)
    legend(legendInfo, 'Location', 'Best')
    hold off
    toc
end

ANN_Bias = ANN_Bias_old;
MutResetInterval = MutResetInterval_old;
PSO_W = PSO_W_old;
w = w_old;
ro = ro_old;
end
```

```
function Velocity_Init(Velocity_Init_Check)

global NumScenarios DecisionPeriodIndex ro w
global ANN_Bias Init_Velocity_Range % 1= Use Bias; 2= Do not use bias;
global PSO_W MutResetInterval% PSO_W = [a1 a2 b1 b2 c1 c2];
Velocity_Start_Range = -1:.2:1;
Velocity_End_Range = -1:.2:1;
max_j = size(Velocity_End_Range,2);
ANN_Bias_old = ANN_Bias;
MutResetInterval_old = MutResetInterval;
PSO_W_old = PSO_W;
w_old = w;
ro_old = ro;
Init_Velocity_Range_old = Init_Velocity_Range;

Init_Velocity_Range = [-0.2 0.2];
if Velocity_Init_Check == 1
   datetime('now')
   tic
   fprintf('\n Running PSO Velocity Initialization Log Sigmoid...\n')
   [X,Y] = meshgrid(Velocity_Start_Range, Velocity_End_Range);
   Z = nan*zeros(size(X));
   for j=1:max_j
      for i=1:j
         Init_Velocity_Range = single([Velocity_Start_Range(i) Velocity_End_Range(j)]);
         [GlobalU, ~] = MaskerySystem(12);
         Z(j, i) = mean(GlobalU, 2);
      end
   end

   [value, idx] = max(Z(1:numel(X)));

   figure('rend','painters','pos',[10 50 600 550])
   hold on
```

```
    h1 = surf(X, Y, Z);
    h1.FaceColor = 'interp';
    colorbar
    title({'Average Demand Satisfied', ...
        [num2str(DecisionPeriodIndex) ' Decisions, ' num2str(NumScenarios) ' Scenarios'], ...
        ['Max of ' num2str(value) ' at Vstart=' num2str(X(idx)) ', Vend=' num2str(Y(idx)) ]},
'FontSize',14)
    xlabel('V start', 'FontSize',10);
    ylabel('V end', 'FontSize',10);
    zlabel('Avg Demand Satisfied', 'FontSize',10)
    view([-90 90])
    hold off

    figure('rend','painters','pos',[10 50 600 550]);
    hold on
    h=bar3(Z);
    title({'Average Demand Satisfied', ...
        [num2str(DecisionPeriodIndex) ' Decisions, ' num2str(NumScenarios) ' Scenarios'], ...
        ['Max of ' num2str(value) ' at Vstart=' num2str(X(idx)) ', Vend=' num2str(Y(idx)) ]},
'FontSize',14)
    xlabel('V start', 'FontSize',10);
    ylabel('V end', 'FontSize',10);
    zlabel('Avg Demand Satisfied', 'FontSize',10)
    z_lim= [min(Z(Z>0)) max(Z(Z>0))];
    zlim(z_lim);
    view([150 15])
    set(gca,'XTickLabel',[-1 -0.66 -0.33 0 0.33 0.66 1])
    set(gca,'YTickLabel',[-1 -0.66 -0.33 0 0.33 0.66 1])
    hold off

    for i=1:numel(h)
        %# get the ZData matrix of the current group
        Z = get(h(i), 'ZData');
        rowsInd = reshape(1:size(Z,1), 6,[]);
```

```
        barsIdx = all([Z(2:6:end,2:3) Z(3:6:end,2:3)]==0, 2);
        Z(rowsInd(:,barsIdx),:) = NaN;
        set(h(i), 'ZData',Z)
    end

    for k = 1:length(h)
        zdata = h(k).ZData;
        h(k).CData = zdata;
        h(k).FaceColor = 'interp';
    end
    caxis(z_lim)
    view([-130 50])
    toc
    %     Z_temp = Z;
    %     [value, idx] = maxk(Z_temp(1:numel(Z)),10);
    %     BestValues = [Z(idx); X(idx); Y(idx);]'
    %     filename = 'testdata.xlsx';
    %     xlswrite(filename,BestValues,3,'B2')
end
Init_Velocity_Range = Init_Velocity_Range_old;
end


function Mut_Init(Mut_Init_Check)

global NumScenarios DecisionPeriodIndex ro w
global MutResetInterval % 0 = off; # = intervals before mutation
MutResetInterval_temp = [0; 20; 40];
max_i = size(MutResetInterval_temp,1);
plotStyle = {'b','r','g','b:','r:','g:'};
MutResetInterval_old = MutResetInterval;
w_old = w;
ro_old = ro;
if Mut_Init_Check == 1
```

```
    datetime('now')
    tic
    fprintf('\n Running PSO Mutations Extensive Initialization Log Sigmoid...\n')
    MutResetInterval_temp = 0:10:100;

    C = zeros(size(MutResetInterval_temp));
    for i=1:numel(MutResetInterval_temp)
        MutResetInterval = MutResetInterval_temp(i);
        [GlobalU_MutInit, ~] = MaskerySystem(12);
        C(i) = mean(GlobalU_MutInit, 2);
    end
    [value, idx] = max(C);

    figure('rend','painters','pos',[0 0 400 350])
    hold on
    bar(MutResetInterval_temp, C);
    title({'Average Demand Satisfied', ...
        [num2str(DecisionPeriodIndex) ' Decisions, ' num2str(NumScenarios) ' Scenarios'], ...
        ['Max of ' num2str(value) ' at Interval=' num2str(MutResetInterval_temp(idx))]},
'FontSize',14)
    xlabel('Mutation Interval', 'FontSize',12);
    ylabel('Average Demand Satisfied', 'FontSize',12);
    hold off
    Values = [MutResetInterval_temp; C;]'
    filename = 'testdata.xlsx';
    xlswrite(filename,Values,2,'B2')
    toc
end
MutResetInterval = MutResetInterval_old;
w = w_old;
ro = ro_old;
end
```

PSO_ANN_FullSimulation.m

```
function PSO_ANN_FullSimulation(SimulationType)

global ro NumScenarios w DecisionPeriodIndex
global MutResetInterval PSO_W ANN_Bias
w_old = w;
ro_old = ro;
MutResetInterval_old = MutResetInterval;
PSO_W_old = PSO_W;
ANN_Bias_old = ANN_Bias;

max_i = 4;
MutResetInterval_temp = [0; 0; -50; -50];
PSO_W_temp = [0.75 0.2 0.4; 0.75 0.2 0.4; 1 0.4 0.6; 1 0.6 0.4] %For use w/o Mutations
ANN_Bias_temp = [1; 0; 0; 1];
plotStyle = {'b','r','g','k'};
legendInfo = {'No Mut#1', 'No Mut#2', 'Mut#1', 'Mut#2'};
if SimulationType == 1
   fprintf(['\n Running PSO-ANN Verification Static...\n'])
   figure('rend','painters','pos',[10 100 440 640])
   subplot1 = subplot(2,1,1);
   subplot2 = subplot(2,1,2);
   for i=1:max_i
      MutResetInterval = MutResetInterval_temp(i, :);
      PSO_W = PSO_W_temp(i, :);
      ANN_Bias = ANN_Bias_temp(i, :);
      [GlobalU, Converge] = MaskerySystem(12);
      hold(subplot1, 'on')
      plot(subplot1, GlobalU, plotStyle{i});
      hold(subplot1, 'off')
      hold(subplot2, 'on')
      plot(subplot2, Converge, plotStyle{i});
      hold off
```

```
    end
    title(subplot1, {'PSO-ANN Performance Comparison', ['Static Env; Number of Scenarios = '
num2str(NumScenarios)]}, 'FontSize',14);
    xlabel(subplot1, 'Decision Period Index', 'FontSize',10);
    ylabel(subplot1, '% Demand Satisfied (Worst Player)', 'FontSize',10);
    ylim(subplot1, [0 1]);
    legend(subplot1, legendInfo, 'Location', 'Best');
    legend(subplot1, 'boxoff')
    title(subplot2, {'PSO-ANN Equilibrium Comparison', ['Static Env; Number of Scenarios = '
num2str(NumScenarios)]}, 'FontSize',14);
    xlabel(subplot2, 'Decision Period Index', 'FontSize',10);
    ylabel(subplot2, 'Std Deviation Value', 'FontSize',10);
    ylim([0 1]);
    legend(subplot2, legendInfo, 'Location', 'Best');
    legend(subplot2, 'boxoff')
end

if SimulationType == 2
    w = 16;
    % Call ANN PSO a few times and plot demand satisfied vs mean innovation time
    T_ro = [1000 500 250 125 62.5 31.25 15.625 7.8125 3.90625 1.953125];
    ro_w16 = [6.25293e-5 0.000125117 0.00025047 0.000501885 0.001007578 ...
        0.002030635 0.004125206 0.00852383 0.018312393 0.043849539];
    GlobalU = zeros(10,DecisionPeriodIndex);
    fprintf(['\n Running PSO-ANN Verification Dynamic...\n'])

    figure('rend','painters','pos',[500 100 475 425])
    hold on
    for i=1:max_i
        MutResetInterval = MutResetInterval_temp(i, :);
        PSO_W = PSO_W_temp(i, :);
        ANN_Bias = ANN_Bias_temp(i, :);
        fprintf(['On PSO-ANN=' legendInfo{i} '\n']);
        for j = 1:10 %Cutting down on computation
```

```
            ro = ro_w16(j);
            [GlobalU(j, :), ~] = MaskerySystem(12);
        end
        plot(T_ro, mean(GlobalU, 2), plotStyle{i});
    end
    title({'PSO-ANN Performance Comparison', ['Dynamic Env; Number of Scenarios = '
num2str(NumScenarios)]}, 'FontSize',14)
    xlabel('Mean innovation time T(ro)', 'FontSize',12);
    ylabel('% Demand Satisfied (Worst Player)', 'FontSize',12)
    ylim([0 1]);
    legend(legendInfo, 'Location', 'Best')
    legend boxoff
    hold off
end

if SimulationType == 3
    w = 16;
    ro = 0.05;
    fprintf(['\n Running PSO-ANN Verification 1 RO...\n'])
    figure('rend','painters','pos',[10 100 440 640])
    subplot1 = subplot(2,1,1);
    subplot2 = subplot(2,1,2);
    for i=1:max_i
        MutResetInterval = MutResetInterval_temp(i, :);
        PSO_W = PSO_W_temp(i, :);
        ANN_Bias = ANN_Bias_temp(i, :);
        [GlobalU, Converge] = MaskerySystem(12);
        hold(subplot1, 'on')
        plot(subplot1, GlobalU, plotStyle{i});
        hold(subplot1, 'off')
        hold(subplot2, 'on')
        plot(subplot2, Converge, plotStyle{i});
        hold off
    end
```

```
    title(subplot1, {'PSO-ANN Performance Comparison', ['Dynamic Env; ro = ' num2str(ro) ' w =
' num2str(w)]}, 'FontSize',14)
    xlabel(subplot1, 'Decision Period Index', 'FontSize',10);
    ylabel(subplot1, '% Demand Satisfied (Worst Player)', 'FontSize',10);
    ylim(subplot1, [0 1]);
    legend(subplot1, legendInfo, 'Location', 'Best');
    legend(subplot1, 'boxoff')
    title(subplot2, {'PSO-ANN Equilibrium Comparison', ['Dynamic Env; ro = ' num2str(ro) ' w = '
num2str(w)]}, 'FontSize',14)
    xlabel(subplot2, 'Decision Period Index', 'FontSize',10);
    ylabel(subplot2, 'Std Deviation Value', 'FontSize',10);
    ylim([0 1]);
    legend(subplot2, legendInfo, 'Location', 'Best');
    legend(subplot2, 'boxoff')
end
w = w_old;
ro = ro_old;
MutResetInterval = MutResetInterval_old;
PSO_W = PSO_W_old;
ANN_Bias = ANN_Bias_old;
end
```

MaskerySystem.m

```
function [Global_U, ConvergeVal, t] = MaskerySystem(AlgorithmOption)

%AlgorithmOption:
%1=Regret Tracking, 2=Best Response, 3=Fictitious Play, 4=Modified Reg Tracking
global SecUser PriUser NumChannels delta Tmax
global NumScenarios DecisionPeriodIndex ro w m delta_exploration
global Num_Elements NumParticles
global LocalU_Avg LocalU_min
global Init_Position_Range Init_Velocity_Range
global beta alpha1 alpha2;
global ANN_TransferFunctions ANN_Bias
global PSO_W MutResetInterval %NumScenarios, Not needed since no annealing
Global_U = zeros(NumScenarios, DecisionPeriodIndex);
t = zeros(NumScenarios, DecisionPeriodIndex);
ConvergeVal = zeros(NumScenarios, DecisionPeriodIndex, SecUser);
Local_U_All = zeros(SecUser, DecisionPeriodIndex, NumScenarios);
Epsilon = GetEpsilon(AlgorithmOption);

 %Init space of possible choices. (Will not change if ro=0, static env)
Master_S = (de2bi(1:((2^NumChannels)-1), NumChannels)); %All combinations of channels
BadRows = (sum(Master_S, 2) > m); % Bad if more channels chosen than transceivers
Master_Options = find(BadRows == 0); %Get indexes of rows of good combinations
Master_S = Master_S(Master_Options, :);
NumMasterOptions = size(Master_Options, 1);

parfor p= 1:NumScenarios
   Demand = randi([1,4], SecUser, 1);%User Demand Vector
   QualityBits = randi([1,4], 1, NumChannels); %Channel Quality Vector.  Bits/time on each
channel
   PriUsage = randperm(NumChannels);
   PriUsage = PriUsage<=PriUser;
   [S, ValidOptions] = CalculateSS(PriUsage, Master_S);
```

```
    %Init Regret & Probabilities variables
    Regret =zeros(NumMasterOptions, NumMasterOptions, SecUser);
    Prob_Xn= zeros(SecUser, NumMasterOptions);
    Prob_Xn(:, ValidOptions) = (1/size(ValidOptions, 1)); %All Options = likely
    OptionChosen = ValidOptions(randi([1 size(ValidOptions,1)],SecUser, 1)); %Column matrix
w/index for the State Space for each Sec user

    X = Master_S(OptionChosen, :); %Actually pulls the entire row from S for each Sec User

    if AlgorithmOption > 10 %PSO_ANN
        %Going to create matrix that stores all particle data.
        ANN_Array = randi(Init_Position_Range*1000, Num_Elements, NumParticles,
SecUser)/1000;
        Velocity = randi(Init_Velocity_Range*1000, Num_Elements, NumParticles, SecUser)/1000;

        Personal_Bests = zeros(Num_Elements, NumParticles, SecUser);
        Personal_Bests = ANN_Array;
        GlobalBest = zeros(Num_Elements, 1, SecUser);
        %Init mutations for option 11
        ResetCounter = zeros(SecUser);
    end
    for n= 1:DecisionPeriodIndex
        %Model starts here.
        N = repmat(sum(X, 1), SecUser, 1) - X;

        %Calculating Throughput
        %Equation (9) page 462
        R = (1 ./ (1+N)) .* ((1-(delta/Tmax)).^(1+N));
        R(N == 0) = 1;   %If the channel is not attempted, throughput will stay as 1.
        %R still needs .* X or each X in StateSpace
        R = R - repmat(PriUsage, SecUser, 1); %To take out the ones from primary users.
        %Calculating the Probability of Collision
        %Equation (11) page 462, Modified by C.A. to represent true probab
```

```
Q = 1-((1-(delta/Tmax)).^(1+N));
Q(N == 0) = 0;

%Global System Utility
Global_U_temp = (X.*R) * (QualityBits.* ~PriUsage)' ./Demand;
[Global_U(p, n), ~] = min([Global_U_temp; 1]);

%It's calculating the throughput from each channel times the bits per
%channel to determine how much got transmitted then divides by the demand
%to get a ratio. Then it finds out which radio did the worst (min funct).

for L = 1:SecUser
   if AlgorithmOption < 10 %Maskery
      %For comparing Local U to Global U
      Local_U = zeros(NumMasterOptions,1);
      Local_U(ValidOptions) = Fitness_Func(S, N(L, :), R(L, :), Q(L, :), QualityBits,
Demand(L), PriUsage, beta, alpha1, alpha2); %Calculates the fitness functions
      Local_U_All(L, n, p)  = Local_U(OptionChosen(L));
      %Makes the decision below
      [Prob_Xn(L, :), OptionChosen(L), Regret(:, :, L)] = ...
         MaskeryDecision(OptionChosen(L), Regret(:, :, L), NumMasterOptions, Local_U,
ValidOptions, n, AlgorithmOption, Prob_Xn(L, :), Epsilon(n), delta_exploration);
      X(L, :) = Master_S(OptionChosen(L), :); %Actually pulls the entire row from S for
each Sec User
       ConvergeVal(p, n, L) = max(max(Regret(:, :, L), [], 1), [], 2);
   else %PSO_ANN
      ANN_Inputs = [(R(L, :).*QualityBits/Demand(L)) m]';
      Channels_Chosen = ANN_CalcOutput(ANN_Inputs,[ANN_Array(:, :, L)
Personal_Bests(:, :, L) GlobalBest(:, :, L)], NumChannels, m, ANN_TransferFunctions,
ANN_Bias);
      Local_U= Fitness_Func(Channels_Chosen', N(L, :), R(L, :), Q(L, :), QualityBits,
Demand(L), PriUsage, beta, alpha1, alpha2);
      [ConvergeVal(p,n,L), ResetCounter(L), GlobalBest(:, :, L), Personal_Bests(:, :, L),
Velocity(:, :, L), ANN_Array(:, :, L)] = ...
```

```
                PSO_ANN(Local_U', n, ResetCounter(L), GlobalBest(:, :, L), Personal_Bests(:, :, L),
Velocity(:, :, L), ANN_Array(:, :, L), ...
                Num_Elements, NumParticles, PSO_W, MutResetInterval, Init_Position_Range,
Init_Velocity_Range);
            X(L, :) = ANN_CalcOutput(ANN_Inputs, GlobalBest(:, :, L), NumChannels, m,
ANN_TransferFunctions, ANN_Bias);
        end
    end

    if ro ~= 0
        %Slow varying: Demand or PriUsage can change
        ChangeDemand = rand(SecUser, 1) < ro;
        Demand = randi([1,4], SecUser, 1) .* ChangeDemand + Demand .* (~ChangeDemand);
%User Demand Vector

        %Updating PriUser positions if needed
        NewPriUsage = find(PriUsage == 0);
        CurrentPriUsage = find(PriUsage);
        NewPriUsage = NewPriUsage(randperm(NumChannels-PriUser));
        Change = 0;
        for pri_change_test = 1:PriUser
            if rand < ro
                PriUsage(CurrentPriUsage(pri_change_test)) = 0;
                PriUsage(NewPriUsage(pri_change_test)) = 1;
                Change = 1;
            end
        end

        if w == 16 %Fast varying, quality may also change by max +/- 10%. Pg467-Section C
            QualityBits = QualityBits.* (1 + (rand(1, NumChannels) < ro).*(0.2*rand(1,
NumChannels) - 0.1));
            QualityBits= max([QualityBits; zeros(1, NumChannels)], [], 1); %Zero out unlikely
negs
        end
```

```
            if Change == 1
                [S, ValidOptions] = CalculateSS(PriUsage, Master_S);
            end
            %else Do Nothing - Static Environment
        end
    end
    %p
end
Global_U = mean(Global_U, 1);

if AlgorithmOption < 10
    LocalU_Avg = mean(Local_U_All, 1);
    LocalU_Avg = mean(LocalU_Avg, 3);
    LocalU_min = min(Local_U_All, [], 1);
    LocalU_min = min(LocalU_min, [], 3);
    ConvergeVal = max(ConvergeVal,[], 3); %Max across each user.
    ConvergeVal = mean(ConvergeVal, 1); %Average across each scenario
else
    ConvergeVal = mean(ConvergeVal, 3); %Average across each user
    ConvergeVal = mean(ConvergeVal, 1); %Average across each scenario
end
end
```

```
function [Epsilon] = GetEpsilon(AlgorithmOption)

%For Maskery's average regret matrix
global DecisionPeriodIndex ro
if AlgorithmOption ~= 2
   if (ro == 0) % Per discussion of 5.1 on page 464:
      Epsilon = 1./((1:DecisionPeriodIndex)+1);%Decreasing stepsize option, works best for
unchanging parameters
   else
      Epsilon = 0.1*ones(1, DecisionPeriodIndex); %Constant stepsize, works best for evolving
parameters.
   end
else
   Epsilon = ones(1, DecisionPeriodIndex); %Makes decision based purely on H and not avg
Regret. For Best Response
end
end


function [S, ValidOptions] = CalculateSS(PriUsage, Master_S)

%Calculate State Space
   BadRows = (Master_S * PriUsage'); %bad if channel is used by primary.
   ValidOptions = find(BadRows == 0); %Get indexes of rows of good combinations
   S = Master_S(ValidOptions, :);  %Retrieve only the good row combinations.
end
```

Fitness_Func.m

```
function [U] = Fitness_Func(StateSpace, N, R, Q, QualityBits, DemandL, PriUsage, beta, alpha1,
alpha2)

QualityBits = QualityBits .* ~PriUsage; %Effective quality is zero for off-limits channels
NumOptions=size(StateSpace, 1);
 %Local System Utility 0
Local_U0 = (QualityBits./DemandL).*R;
%This part is the ratio of how much demand could've been fulfilled per channel
Local_U0 = min([StateSpace * Local_U0' ones(NumOptions, 1)], [], 2);
%Sums across the channels to get each radio utility, then takes minimum between that and 1, to
not reward using more than what was needed.

%Local System Utility 1
Local_U1 = StateSpace *(QualityBits .* R)' - (DemandL + beta);
%Takes care of inside parenthesis in Eq.14, pg 463.
Local_U1 = - max([Local_U1 zeros(NumOptions, 1)], [], 2) ./DemandL;
%This part zeros out negative values (underachievement) because you don't want to penalize a
radio that didn't go over the demand. Then div by demand to get a ratio not a big number.

%Local System Utility 2
D = Q ./ N; %Right above Eq. 15, page 463.
D(N == 0) = 0; %Clean up the NaNs due to N==0, aka X=0;
 %D is collisions per user on a channel.
Local_U2 = (-1./(StateSpace *QualityBits')).* (StateSpace *((QualityBits.*D)'));
%D*Q is how many bits are lost due to collisions. Then you divide by the total to get a ratio

%Total Utility
U = Local_U0 + alpha1 * Local_U1 + alpha2 * Local_U2; %This is a weighted average.
U = max([U zeros(NumOptions, 1)], [], 2); %This zeros out negative values. No neg fit funct.
U(StateSpace* ~PriUsage' == 0) = 0;  %To penalize and make U = 0 if no channel used aside pri
usage.
end
```

MaskeryDecision.m

```
function [Prob_Xn, OptionChosenL, RegretL] = MaskeryDecision(OptionChosenL, RegretL,
NumMasterOptions, Local_UL, ValidOptions, n, DecisionType, Prob_Xn, Epsilon,
delta_exploration)

%DecisionType:
%1=Regret Tracking, 2=Best Response, 3=Fictitious Play, 4=Modified Reg Tracking
mu=size(ValidOptions, 1);
H=zeros(NumMasterOptions); % H(j,k)
 if DecisionType ~= 4
    %Eq. 19, pg 464.
    H(OptionChosenL, :) = Local_UL - Local_UL(OptionChosenL);
else
    %Modified H of Eq. 27, pg 465.
    H(ValidOptions, OptionChosenL) =
(Prob_Xn(ValidOptions)'/Prob_Xn(OptionChosenL)).*Local_UL(OptionChosenL);
    H(OptionChosenL, ValidOptions) = H(OptionChosenL, ValidOptions) -
Local_UL(OptionChosenL)';
end
if n==1 %Page 464 Step #1
    [~, OptionChosenL] = max(H(OptionChosenL, :));
    RegretL = H; %initialization
    %Page 464, Eq. 20
    Prob_Xn= zeros(1, NumMasterOptions); %For Mod Regret Tracking
    Prob_Xn(ValidOptions)=RegretL(OptionChosenL,
ValidOptions)/(mu*max(RegretL(OptionChosenL, ValidOptions)));
    Prob_Xn= max([Prob_Xn; zeros(1, NumMasterOptions)]); %Removes negative values
    Prob_Xn(OptionChosenL) = 0;
    Prob_Xn(OptionChosenL) = 1- sum(Prob_Xn(ValidOptions));%Makes last option the
remainder that adds to 1.
else
    if n==2
        RegretL = H; %initialization
```

```
    else
        RegretL = RegretL +(Epsilon)*(H - RegretL); %Avg Regret Update
    end
    if ((DecisionType == 1) || (DecisionType == 4))
        %Page 464, Eq. 20
        Prob_Xn= zeros(1, NumMasterOptions);
        if max(RegretL(OptionChosenL, ValidOptions)) > 1
            Prob_Xn(ValidOptions)=RegretL(OptionChosenL,
ValidOptions)/(mu*max(RegretL(OptionChosenL, ValidOptions)+0.00001));
        else
            Prob_Xn(ValidOptions)=RegretL(OptionChosenL, ValidOptions)/mu;
        end
        Prob_Xn= max([Prob_Xn; zeros(1, NumMasterOptions)]); %Removes negative values
        Prob_Xn(OptionChosenL) = 0;
        Prob_Xn(OptionChosenL) = 1- sum(Prob_Xn(ValidOptions));%Makes last option the
remainder that adds to 1.
        OptionChosenL = find(rand<=cumsum(Prob_Xn),1,'first');
        if (DecisionType == 4  && (rand<delta_exploration))%Page 465, just under Eq. 27
            OptionChosenL = ValidOptions(randi(size(ValidOptions,1)));
        end
    else %FictitiousPlay & Best Response: Page465-Section C -> c) ->step 2a
        RegretL_temp = min(RegretL(OptionChosenL,ValidOptions))*ones(1,NumMasterOptions)-
1; %So it's never chosen.
        RegretL_temp(1,ValidOptions) = RegretL(OptionChosenL,ValidOptions);
        idx= find(max(RegretL_temp)==RegretL_temp); %In the event there are several of same
size
        OptionChosenL = idx(randi(size(idx,2))); %Just choose one at random
    end
end
 if (isempty(Prob_Xn) || isempty(OptionChosenL) || isempty(RegretL))
    msgbox('ERROR');
end
end
```

PSO_ANN.m

```
function [StdDev, ResetCounter, GB_old, Personal_Bests, Velocity, ANN_Array] =
PSO_ANN(Fitness, curr_scenario, ResetCounter, GB_old, Personal_Bests, Velocity,
ANN_Array, Num_Elements, NumParticles, PSO_W, MutResetInterval, Init_Position_Range,
Init_Velocity_Range)

ANN_Fitness = Fitness(1:NumParticles);
PB_Fitness = Fitness((1:NumParticles)+NumParticles);
if curr_scenario == 1
   Personal_Bests = ANN_Array;
   [~, I] = max(Fitness);
   Global_Best = Personal_Bests(:, I);
else
   [PB_Fitness, I] = max([ANN_Fitness; PB_Fitness]);
   I = repmat(I, Num_Elements, 1);
   Personal_Bests = ANN_Array.*(2-I) + Personal_Bests.*(I-1);
   [~, I] = max([PB_Fitness Fitness(1+2*NumParticles)]);
   Global_Best = [Personal_Bests GB_old]; %GB_Fitness
   Global_Best = Global_Best(:, I);
end
 %Update the ANNs here with velocity from prior calc
ANN_Array = ANN_Array + Velocity; %Equivalent to x = x + v;
 %Update the velocity for the next iteration here.
Velocity = PSO_W(1)*Velocity + PSO_W(2)*(Personal_Bests -
ANN_Array)*(randi(1000)/1000) + ...
         (repmat(PSO_W(3)*Global_Best, 1, NumParticles) - ANN_Array)*(randi(1000)/1000);
 if ((MutResetInterval > 0) )
   %if each element is the same then
   if isequal(Global_Best, GB_old)
      ResetCounter = ResetCounter + 1;
   else
      ResetCounter = 0;
```

```
    end
    if ResetCounter > MutResetInterval
        ResetCounter = 0;
        Y = sort(ANN_Fitness);
        Num_Mutations = ceil(0.2*NumParticles); % 20percent
        [~, I] = find(ANN_Fitness <= Y(Num_Mutations));
        ANN_Array(:, I) = randi(Init_Position_Range*1000, Num_Elements, length(I))/1000;
        Velocity(:, I) = randi(Init_Velocity_Range*1000, Num_Elements, length(I))/1000;
        Personal_Bests(:, I) = ANN_Array(:, I); %Update the personal best to be current ANN
    end
end

 if ((MutResetInterval > 0))% && (rand<0.05))
    %if each element is the same then
    if isequal(Global_Best, GB_old)
        ResetCounter = ResetCounter + 1;
    else
        ResetCounter = 0;
    end
    if ResetCounter > MutResetInterval
        ResetCounter = 0;
        Y = sort(ANN_Fitness);
        Num_Mutations = ceil(0.6*NumParticles); % 20percent
        [~, I] = find(ANN_Fitness <= Y(Num_Mutations));
        ANN_Array(:, I) = randi(Init_Position_Range*1000, Num_Elements, length(I))/1000;
        Velocity(:, I) = randi(Init_Velocity_Range*1000, Num_Elements, length(I))/1000;
        Personal_Bests(:, I) = ANN_Array(:, I); %Update the personal best to be current ANN
    end
end
GB_old = Global_Best;
StdDev = std(PB_Fitness);
end
```

ANN_CalcOutput.m

```
function Output = ANN_CalcOutput(Input_Array, ANNs_2_Eval, NumChannels, m,
ANN_TransferFunctions, ANN_Bias)

    Num_ANN = size(ANNs_2_Eval, 2);

    h = zeros(NumChannels, Num_ANN);
    Output = zeros(NumChannels, Num_ANN);

    start = 1;
    finish = start -1 + NumChannels*(NumChannels+1);
    w1 = ANNs_2_Eval(start:finish, :);

    start = 1+finish;
    finish = start -1 + NumChannels;
    b1 = ANNs_2_Eval(start:finish, :);

    start = 1+finish;
    finish = start -1 + (NumChannels^2);
    w2 = ANNs_2_Eval(start:finish, :);

    start = 1+finish;
    finish = start -1 + NumChannels;
    b2 = ANNs_2_Eval(start:finish, :);

    %Calculate Input to Hidden Layer
    x = repmat(Input_Array, NumChannels, Num_ANN);
    x = w1 .* x;
    %Calculate Hidden Layer
    for i = 0:(NumChannels-1)
        h(i+1, :) = sum(x(i*NumChannels + (1:NumChannels), :));
    end
```

```
    if ANN_Bias
        h = h - b1;
    end
        switch ANN_TransferFunctions
        case 1    %Calculate Hard Limiter on Hidden Layer Neurons
            s = h >0;
        case 2    %Calculate Sigmoid of Hidden Layer Neurons
            s = 1./(1+exp(-h));
        case 3    %Calculate Hyperbolic Tangent Sigmoid on Hidden Layer Neurons
            s = (exp(h) - exp(-h)) ./ (exp(h) + exp(-h));
        otherwise    %Calculate ReLu on Hidden Layer Neurons
            s = max(h, zeros(size(h)));
    end
     %Calculate Input to Output Layer
    x2 = repmat(s, NumChannels, 1);
    x2 = w2 .* x2;

    %Calculate Output Layer (Output)
    for i = 0:(NumChannels-1)
        Output(i+1, :) = sum(x2(i*NumChannels + (1:NumChannels), :));
    end
    if ANN_Bias
        Output = Output - b2;
    end
    Output = max(Output, zeros(size(Output))); %Anything less than 0% is zeroed out.
    temp = sort(Output, 1);
    Output = not(Output <= repmat(temp(end-m, :), NumChannels, 1));
    %This was done to only take the two highest. But may take more if identical values so I need
to zero out bad ANNs. No reward!
    IsValid = (sum(Output, 1) <= m);
    Output = Output .* repmat(IsValid, NumChannels, 1);
end
```

VITA

Carolina Arbona

Candidate for the Degree of

Master of Science

Thesis: DYNAMIC SPECTRUM ACCESS UTILIZING NEURAL NETWORKS AND PARTICLE SWARM OPTIMIZATION IN COGNITIVE RADIOS

Major Field: Electrical Engineering

Biographical:

Education:

Completed the requirements for the Master of Science in Electrical Engineering at Oklahoma State University, Stillwater, Oklahoma, USA in July, 2018.

Completed the requirements for the Bachelor of Science in Electrical Engineering at Oklahoma State University, Stillwater, Oklahoma, USA in December, 2012.

Completed the requirements for the Bachelor of Science in Computer Engineering at Oklahoma State University, Stillwater, Oklahoma, USA in December, 2012.

Experience:

Co-op Engineer with American Electric Power in Tulsa, Oklahoma, USA from May 2011 to April 2012.

Software Engineering Intern with Raytheon in Indianapolis, Indiana, USA from May 2012 to December 2012.

Senior Engineer with American Electric Power in Tulsa, Oklahoma, USA from January 2013 to Present.

Professional Memberships:

Institute of Electrical and Electronics Engineers, IEEE. Total Years – 5.
State of Oklahoma Professional Engineer. Total Years – 1.