Enhancing Cognitive Algorithms for Optimal Performance of Adaptive Networks

2018

Hector Lugo-Cordero

University of Central Florida

Find similar works at: http://stars.library.ucf.edu/etd

University of Central Florida Libraries http://library.ucf.edu

Part of the Computer Engineering Commons

STARS Citation

http://stars.library.ucf.edu/etd/5814

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
ENHANCING COGNITIVE ALGORITHMS FOR
OPTIMAL PERFORMANCE OF ADAPTIVE NETWORKS

by

HECTOR M LUGO-CORDERO
B.S. University of Puerto Rico, Mayagüez Campus, 2006
M.S. University of Puerto Rico, Mayagüez Campus, 2009
M.S. University of Central Florida, 2017

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2018

Major Professor: Ratan K Guha
This research proposes to enhance some Evolutionary Algorithms in order to obtain optimal and adaptive network configurations. Due to the richness in technologies, low cost, and application usages, we consider Heterogeneous Wireless Mesh Networks. In particular, we evaluate the domains of Network Deployment, Smart Grids/Homes, and Intrusion Detection Systems.

Having an adaptive network as one of the goals, we consider a robust noise tolerant methodology that can quickly react to changes in the environment. Furthermore, the diversity of the performance objectives considered (e.g., power, coverage, anonymity, etc.) makes the objective function non-continuous and therefore not have a derivative. For these reasons, we enhance Particle Swarm Optimization (PSO) algorithm with elements that aid in exploring for better configurations to obtain optimal and sub-optimal configurations. According to results, the enhanced PSO promotes population diversity, leading to more unique optimal configurations for adapting to dynamic environments. The gradual complexification process demonstrated simpler optimal solutions than those obtained via trial and error without the enhancements.

Configurations obtained by the modified PSO are further tuned in real-time upon environment changes. Such tuning occurs with a Fuzzy Logic Controller (FLC) which models human decision making by monitoring certain events in the algorithm. Example of such events include diversity and quality of solution in the environment. The FLC is able to adapt the enhanced PSO to changes in the environment, causing more exploration or exploitation as needed.

By adding a Probabilistic Neural Network (PNN) classifier, the enhanced PSO is again used as a filter to aid in intrusion detection classification. This approach reduces miss classifications by consulting neighbors for classification in case of ambiguous samples. The performance of ambigu-
ous votes via PSO filtering shows an improvement in classification, causing the simple classifier perform better the commonly used classifiers.
This dissertation is dedicated to God first of all; thanks to Him I am capable of learning and applying research to solve problems. Secondly, my family for always been there and believing in me, thus, giving me inspiration during difficult times. Also, I would like to dedicate my dissertation to my advisor, who helped me grow as a professional and had patience on me during stressful moments. I would not have been able to finish this dissertation without any of them.
ACKNOWLEDGMENTS

I would like to first of all thank my graduate committee; their teachings and guidance aided me greatly in coming up with this research and getting the most out of my PhD experience. Specially my advisor, who was with me at all times and shared his expertise on how to make this document better.

Committee Members:

Dr Ratan K Guha
Dr Annie Wu
Dr Kenneth Stanley
Dr Mainak Chatterjee
Dr Kejie Lu
# TABLE OF CONTENTS

## LIST OF FIGURES

-  xiii

## LIST OF TABLES

-  xv

## CHAPTER 1: INTRODUCTION AND GOALS

-  1

## CHAPTER 2: BACKGROUND INFORMATION

-  5

-  Heterogeneous Wireless Mesh Networks  
-  6

-  Swarm Intelligence  
-  7

-  Particle Swarm Optimization (PSO)  
-  7

-  Evolutionary Computation  
-  9

-  Novelty Search  
-  10

-  Genetic Diversity  
-  11

-  Complexification  
-  12

-  Fuzzy Logic  
-  12

-  Probabilistic Neural Network  
-  13

## CHAPTER 3: OPTIMIZING NETWORKS VIA SWARM INTELLIGENCE

-  15
Introduction .......................... 15
Optimal Network Evolver ..................... 16
Gene Representation ..................... 17
Comprehensive Learning by Individual Knowledge ..................... 18
Historical Marking ..................... 19
Complexification ..................... 20
Genetic Diversity ..................... 20
Multi-objective Optimization .............. 23
Coverage ..................... 23
Power ..................... 24
Confidentiality ..................... 24
Anonymity ..................... 25
Cost ..................... 26
Path Loss ..................... 26
Experimental Setup ..................... 27
ONE’s Feature Comparison .............. 27
ONE’s Scalability Performance ............ 27
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>One Step Variation (OSV)</td>
<td>49</td>
</tr>
<tr>
<td>Fuzzy Controller</td>
<td>50</td>
</tr>
<tr>
<td>Experimental Settings</td>
<td>52</td>
</tr>
<tr>
<td>Results and Discussions</td>
<td>55</td>
</tr>
<tr>
<td>Conclusions</td>
<td>58</td>
</tr>
<tr>
<td>CHAPTER 5: ADAPTIVE SMART GRIDS</td>
<td>59</td>
</tr>
<tr>
<td>Introduction</td>
<td>59</td>
</tr>
<tr>
<td>Smart Homes As Smart Grid Nodes</td>
<td>62</td>
</tr>
<tr>
<td>Adaptive Environment</td>
<td>63</td>
</tr>
<tr>
<td>Context Awareness</td>
<td>64</td>
</tr>
<tr>
<td>Energy Adviser</td>
<td>64</td>
</tr>
<tr>
<td>Central Control</td>
<td>65</td>
</tr>
<tr>
<td>Load Balancing</td>
<td>66</td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>67</td>
</tr>
<tr>
<td>The 2D-BPSO</td>
<td>68</td>
</tr>
<tr>
<td>Emergency Rechargeable Batteries</td>
<td>73</td>
</tr>
<tr>
<td>Cost Analysis</td>
<td>74</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1.1: Proposed Adaptive Optimization System .......................... 3

Figure 2.1: Example of HWMN in Smart Home ................................. 6

Figure 3.1: Example run of ONE .................................................. 17

Figure 3.2: PSO: Evolution of Total Fitness ..................................... 30

Figure 3.3: ONE: Evolution of Total Fitness (first 1000 Generations) .......... 32

Figure 3.4: ONE: Evolution of Total Fitness ...................................... 33

Figure 3.5: ONE: Evolution of Innovation ........................................ 34

Figure 3.6: Evolution of Coverage in 150 Generations ......................... 35

Figure 3.7: Evolution of Power Consumption in 150 Generations ............... 35

Figure 3.8: Evolution of Anonymity in 150 Generations ....................... 36

Figure 3.9: Examples of Optimal Heterogeneous Network Configuration .......... 37

Figure 4.1: Architectural View of a Feedback System .......................... 48

Figure 4.2: FLC Flow ................................................................. 51

Figure 4.3: Variation From Three Parameter Sets for 10 Runs .................. 56
Figure 5.1: The SHEAP System .................................................. 62
Figure 5.2: Status Information Header ................................. 63
Figure 5.3: Example of Sequence of Events .......................... 76
Figure 5.4: Consumption for Different House Types .............. 78
Figure 5.5: Valid Optimal Configurations ............................. 79
Figure 5.6: Effect of Connected Loads on Battery Configurations .... 79
Figure 5.7: Average Charging Time for 12V AGM Batteries ....... 81
Figure 5.8: Return of Investment for Projects ....................... 82
Figure 6.1: Overlapping of User Behaviors ......................... 86
Figure 6.2: Adaptive IDPS Infrastructure ............................. 89
Figure 6.3: IDPS Disambiguation ...................................... 93
Figure 6.4: IDPS Detection Rate ...................................... 97
Figure 6.5: IDPS Comparison ........................................ 98
LIST OF TABLES

Table 2.1: Novelty Search VS. Genetic Diversity ............................. 11

Table 3.1: ONE Configuration Parameters ........................................ 28

Table 3.2: ONE Performance per Scenario ........................................... 34

Table 4.1: Parameters Tuned ................................................................. 54

Table 4.2: Run Time Comparison .......................................................... 55

Table 4.3: ANOVA Analysis ................................................................. 56

Table 4.4: Final Parameter Values (OSV, FLC) ..................................... 58

Table 5.1: Average Consumption by a Grid Node in kW. ............................. 78

Table 6.1: Confusion Matrix Without Majority Votes ................................. 96

Table 6.2: Confusion Matrix With Majority Votes ..................................... 97
CHAPTER 1: INTRODUCTION AND GOALS

With the advances of technologies, communication networks have become indispensable to our daily life. We interact with networks when we turn on the TV or any other appliance (power grid network), browse the Internet (computer network), or even when talking with our relatives and friends (phone/cellular) network. Although these network types are different in their respective domains, they share a similar structure.

Non-optimized networks may lead to lack of resources or unbalanced nodes, leading to poor service quality or in worse cases service interruption. In this work we plan to develop optimization algorithm for dynamic environments. Therefore, we selected tools that can help optimize the complex structure of networks. These tools need to adjust to dynamic environments to maximize performance upon changes encountered in the environment or faults.

Literature shows success around Wireless Sensor Network optimization with power consumption [1]. Smart Grids optimization based on cost of electricity is shown in [2]. These research concentrate in offline optimization on static environments. Another common approach is to use some Genetic Algorithm with memory of the best solution to avoid loosing it as the GA progresses [3] [4]. Furthermore, [5] shows control of a Particle Swarm Optimization population, preventing it from converging and improving the quality of search. However, it does this by temporarily disabling particles, which may lead to slowdowns in searches.

This dissertation focuses on how to optimize dynamic networks, specifically those under the domains of communications networks and smart grids. We have considered the problem of optimizing Heterogeneous Wireless Mesh Networks which fits well in two domains we plan to apply our developed optimization algorithm.
In order to optimize the network configurations, we employ a swarm intelligence based algorithm in the form of the Particle Swarm Optimization (PSO). The PSO was chosen due to its ability to handle domains where the objective function does not have a derivative; in addition, the PSO is a robust algorithm against noise in data gathered from the environment. Our work enhances the PSO algorithm with diversity metrics inspired on Complexification, Exploration, Selection, and Novelty Search. This enhancement determines how much spread the search of the algorithm is, and in turn provides ways to find sub-optimal patterns.

In our study, we develop an enhanced Particle Swarm Optimization which we denote as Optimal Network Evolver (ONE). ONE is inspired by evolutionary computation elements to promote diversity in a population of configurations. The goal of such diversity is two folds, first we aim at optimizing the network domain with respect to some application dependent criteria. Second, we obtain a set of sub-optimal configurations with equal or similar performance which can be used to quickly adapt to changes in the domain environment.

For the network deployment problem, we optimize using coverage, power consumption, confidentiality, and network anonymity as goals. We apply this optimization result to a smart grid domain. In this application, the problem is to deploy solar panels and wind turbines which can reduce optimally the power grid dependency, by choosing the amount of power sources and appliances which connect to each type.

We then again apply our developed algorithm in the domain of Intrusion Detection Systems, on which we employ the PSO as filter to ease intruder classification. Furthermore, using the distributed nature of a network, we cast majority votes with neighbors to help clearing up when ambiguous classifications are found.

The full proposed approach is presented in Figure 1.1. Specifically, we consider the enhanced PSO algorithm and how it can be modified to adapt faster the studied networks. In addition, we include
a Fuzzy Logic Controller that can assist in modifying the optimized network configuration upon environment changes, and a Probabilistic Neural Network (PNN) for clustering and classifying data in a dynamic way, when needed.

Figure 1.1: Proposed Adaptive Optimization System

The adaptive network optimizer system works in four steps. First, the current configuration (initial or previously optimized) physical model of the domain is fed to the algorithm, which then starts to improve its overall performance as it runs. Following, the adaptive module periodically monitors the performance of the physical world, via the optimization algorithm, and tunes the algorithm for better improvements. Later, the physical world optimization is finalized and broadcasted to nearby neighbors if needed.

The rest of this dissertation is organized as stated here. In chapter 2 we provide a brief background information needed for the proposed systems. Some of these include Heterogeneous Wireless Mesh Networks (HWMN) architecture, the Particle Swarm Optimization (PSO) Algorithm, Fuzzy Logic, and the Probabilistic Neural Network (PNN). In chapter 3 we develop enhanced PSO algo-
rithm and compare our result with PSO algorithm in the Computer Networks deployment domain. We present three proposed strategies to tune and optimize the HWMNs. Later in chapter 4, we present how the behavior of the selected strategies can be used to adapt to dynamic environments using meta-level logic. Moreover, in chapters 5 and 6 we present the application of our algorithm in the domains of Smart Grids and Intrusion Detection Systems respectively. Finally in chapter 7, we provide conclusions of our work and possible future works that would help further tune and improve dynamically the HWMNs.
CHAPTER 2: BACKGROUND INFORMATION

In this chapter, we provide background information that serves as motivation and basis of the domain we want to enhance. This dissertation primarily deals with optimization of wireless mesh networks, specifically those composed of different technologies denoted as Heterogeneous Wireless Mesh Networks. Optimization can occur offline with tolerable delays to find optimal solutions, or real-time adapting to changes in the environment.

In order to optimize the network configurations, we employ swarm intelligence based algorithms in the form of the Particle Swarm Optimization (PSO). The PSO was chosen due to its ability to handle domains where the objective function does not have a derivative; in addition the PSO is a robust algorithm against noise in data gathered from the environment. The PSO is enhanced with diversity metrics inspired on Complexification, Exploration, Selection, and Novelty Search, which enhance how much spread the search of the algorithm is, and in turn provides ways to find sub-optimal patterns.

Once the PSO completes, we obtain optimal and sub-optimal solutions, which can be adapted real-time in the event a change in the environment is perceived. Fuzzy Logic is employed as the decision making mechanism, to mimic human intelligence in making partial judgment, rather than an extreme boolean (true/false) decision.

The last part of the dissertation covers applications of the proposed algorithms in domains of Smart Grids and Intrusion Detection and Prevention Systems. In particular, the later requires a noise tolerant classifier for which once more the PSO is employed for feature selection to be used by the classifier, in addition a Probabilistic Neural Network approach to do the actual classification.

Following, the explanation of each topic which constructs the methodologies of this dissertation.
Heterogeneous Wireless Mesh Networks

Heterogeneous Wireless Mesh Networks (HWMN) are composed of different wireless technologies interacting with each other. Such interaction allows to exploit the advantages of each technology, in order to employ a better service. For instance, low coverage technologies such as Zigbee and Bluetooth tend to have better battery consumption and due to their lack of coverage, more privacy. In the mean time, larger coverage technologies tend to be available on practically all end user devices, making them suitable for gateways and user interfaces.

Figure 2.1: Example of HWMN in Smart Home

In the case of a Smart Home, an HWMN can allow for social networking via WiFi, as well as controlling smart appliances or even traditional home circuits, using technologies such as Zigbee. The Gateway node acts as a technology connector and allows the interaction to occur from any end user device.
HWMNs deployment and confirmation are also highly affected by their location and changes in the environment. Such changes can occur because mobile devices, channel failures, obstacle additions, signal collisions, etc. Due to this dynamic nature, HWMN can benefit from adaptive optimization strategies that can react quickly to any of the mentioned changes. HWMNs can be interconnected with Wireless Mesh Networks (WiFi or WiMAX) technologies for inter-technology connection [6], [7].

Swarm Intelligence

Swarm algorithms mimic the behavior of animals which travel in swarms (e.g., ants and bees). They are characterized by being decentralized and self-organized systems. Independently of the type of animal, the different members (i.e., agents) in the environment are attracted to areas where high profit may be found (typically food), via chemical reactions. In terms of computation, these chemical reactions are typically represented as the addition of some scaled factor to the current experience.

Particle Swarm Optimization (PSO)

The Particle Swarm Optimization (PSO) algorithm [8] is an example of a swarm intelligence algorithm. In the PSO, the population is composed of particles, which can be decentralized. Each particle gains knowledge about its environment and records the its best known experience ($P$). The values of $P$ can be compared against each other to find a global optimum $P_{gbest}$. Particles share their knowledge through some constants $C_I$ and $C_G$, typically both equal to 2. The value of $C_I$ tells the particle how much importance they should give to their individual best knowledge (exploitation). Similarly, the value of $C_G$ tells the particle how much importance to give to the global
optimum solution (exploitation). Increasing the value of either constant too much would cause particles to stay within their zones (for high $C_I$), or pursue the global best ($C_G$). Dynamically changing, these values can help control population diversity as shown in [9]. Another parameter that can help introduce more diversity (exploration) is the value of inertia $\omega (0 < \omega < 1)$ in the velocity equation. Large $\omega$ values make the particles move fast, since it becomes hard to change the previous velocity values, thus, accumulating speed and increasing exploration. Meanwhile, small values help mitigate the current velocity, making particles move slow in the search space, thus reducing exploration. It can be customary then to start with a high value of $\omega$ and gradually decrease it as generations progress as done by [10] and [11]. The complete PSO algorithm is shown in Algorithm 1.
Algorithm 1 Original PSO (Minimization)

Init population $S$ of $N$ random particles $X_i$, in search space dimension $D$
Init particles velocities $V_i$ to 0
Init individual best $P_i$ to current population
Init $P_{gbest} \leftarrow \min_{X_i \in S} \{F(X_i)\}$

while Stopping Criteria Not Met do
    for Each $X_i$ in $S$ do
        Access each dimension of $X_i$
        for Each $d = 1$ to $D$ do
            Obtain random inertia weight $w_{id}$ in the interval $(0,1)$
            Adapt particle velocity and position in dimension $d$
            $v_{id} \leftarrow \omega \cdot v_{id} + C_I \cdot rand_1 \cdot (p_{id} - x_{id}) + C_G \cdot rand_2 \cdot (P_{gbest_{1d}} - x_{id})$
            Check Velocity Limit $[V_{min}, V_{max}]$
            $x_{id} \leftarrow x_{id} + v_{id}$
            Check Particle Limit $[X_{min}, X_{max}]$
        end for
        Update best local position vector $P_i$
        if $f(X_i) \leq f(P_i)$ then
            $P_i \leftarrow X_i$
        end if
        Update local best position vector $P_i$
        if $f(X_i) \leq f(P_i)$ then
            $P_i \leftarrow X_i$
            Update best global position vector $X_{gbest}$
            if $f(P_i) \leq f(X_{gbest})$ then
                $P_{gbest} \leftarrow P_i$
            end if
        end if
    end for
end while

Evolutionary Computation

Another important area is Evolutionary Computation (EC). EC is inspired by nature, realizing that nature did not become what it is today over night, but rather endured years of improvement. Examples of EC algorithms are Genetic Algorithms (GA), which mimic not only the life and death of individuals, but their mating and changes across time via crossover and mutation operations. GAs can introduce dramatic effects on the search space, since every generation new individuals
arrive and old ones disappear. The arrival and departure of individuals can be somehow controlled via selection schemes (e.g., elitism, rank selection, and fitness scaling) [5]. The more selection pressure there is, the more exploitation can occur. Mutation on the other hand, can serve as a variation and innovation mechanism.

**Novelty Search**

Although traditional Evolutionary Algorithms have a fitness function which is optimized, nature itself does not have a clear fitness function [12], [13]. For example, if nature’s goal was to create a human being from start, then none of the initial organisms would have exhibit high objective function; thus, extinction would be unavoidable. Hence, it is clear that a purely fitness driven mechanism can lead to either extinction or local optimum.

Therefore, nature must have some kind of mechanism to protect innovation and not ignore the stepping stones towards the goal. Such mechanism can be Novelty Search, where individuals get rewarded not for performing better, but for trying novelty. In such cases, novelty becomes the fitness of the algorithm, and our previous fitness can be used as stopping criteria. Novelty search is completely opposite to random search, where random moves may lead to re-visiting previous areas of the search space.

In contrast, Novelty Search avoids revisiting the same area, exploring more space. Novelty Search can be used as primary objective, or as a restart method for trapped individuals. Novelty can be measured in the fitness value, or better as the distance between configuration values. For example, consider using the Euclidean Distance $ED(P_x, P_y)$ on a 2-D space, values of $P_1 (3, 2)$, $P_2 (3, 4)$, and $P_3 (9, 4)$. If $P_1$ arrives first, followed by $P_2$ and $P_3$, $ED(P_1, P_2) = 2$ and $ED(P_1, P_3) = \sqrt{40}$. Therefore, $P_3$ is more novel than $P_2$, and can be recorded into the population of individuals.
Novelty Search is characterized by not converging fast (if ever), thus is not well suited for optimization algorithms like the ones described in this work. However, we can use the Novelty Search concepts and approach to sample the algorithm’s population at a certain generation frequency and ensure that diversity is promoted or particles get replaced upon finding low diversity values.

*Genetic Diversity*

To address the slow convergence issue of using a pure Novelty Search algorithm, we can apply a controlled objective driven search. The control aspect of it may be simply ensuring that particles are searching different areas of the sample space. Doing so, indirectly promotes Novelty Search, as searching different areas should provide different performance [14]. This type of search is different from Novelty Search though because reward is given to individuals with higher objective function. Furthermore, by detecting particles that have gotten stuck in the search space we can take them out of a potential local optima and reset to new areas. This type of search we denote as Genetic Diversity, rewarding higher objective function particles but ensuring they search different areas of the search space. Table 2.1 displays a summary comparison of Novelty Search vs. Genetic Diversity.

<table>
<thead>
<tr>
<th>Search Type</th>
<th>Primary Reward</th>
<th>Secondary Reward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Novelty Search</td>
<td>Different Phenotype</td>
<td>Objective Function</td>
</tr>
<tr>
<td>Genetic Diversity</td>
<td>Objective Function</td>
<td>Different Genotype</td>
</tr>
</tbody>
</table>
Complexification

Complexification is a way of achieving Genetic Diversity since it expands the sample space and uncovers new characteristics for the particles. Complexification is derived from the idea that everything evolved from simple organisms. Therefore, gradual complexification of organisms must have occurred. This procedure takes place in the problem representation, and can be easily implemented by randomly adding (or removing) parts of the representation. In the case of Networks, this complexification can be seen as starting with one node, then gradually adding nodes as generations progress. Although organisms with more complexity should be able to do more, this is not always the case, so the algorithms can choose to penalize for over complexification on individuals that perform the same as simpler ones. Using complexification to implement Genetic Diversity allows to obtain equivalent solutions which exhibit the same or similar performance on a domain. Such set of similar solutions awards with adaptive behavior on the algorithm, leading to better response to changes in the environment [15].

Fuzzy Logic

The algorithm proposed in this dissertation employs an iterative (generation and population based approach) to optimize the domain model on which it is operated. Like other similar algorithms, its behavior depends on input parameters which will alter the amount of exploration and exploitation that its search will have. Computationally speaking these solutions tend to be \( O(generation \cdot population \cdot features) \). Thus, we would like a non-iterative solution that can react based on the algorithm’s performance and adjust the input parameters to alter the algorithm’s behavior.

Traditionally in computing, decisions are taken based on the outcome of some boolean logic. Boolean logic accepts only two values (True or False). In contrast, Fuzzy Logic is a type of
logic that tries to model how humans think. Therefore it answers as True, False, Partially True, etc. For instance, Fuzzy variables $x$ and $y$ can be such that the degree of truth $\mu(x)$ and $\mu(y)$ are between 0 (False) and 1 (True). For such variables we can define the 3 main logic operators AND, OR, and NOT as follows:

\[
AND = \text{MIN}\{\mu(x), \mu(y)\} \tag{2.1}
\]

\[
OR = \text{MAX}\{\mu(x), \mu(y)\} \tag{2.2}
\]

\[
NOT = 1 - \mu(x) \tag{2.3}
\]

It is important to note that such definitions also hold true when applied to boolean logic. Fuzzy Logic can be employed to construct change rules which can help an algorithm into any environment. As seen in [16], the PSO can respond to rules inside of its definition. The work presented in this dissertation attempts to adjust the PSO performance at a meta-level (outside of the PSO equations), making the approach more generic.

Probabilistic Neural Network

The Probabilistic Neural Network (PNN) is a neural network which can be used for classification. PNNs are easy to train, which is why they are suitable for adaptive training and environment response. PNNs are divided into 4 layers:
1. Input layer: inputs are passed through a normalization process, this prepares the values to be compared against several normal distributions at the pattern layer.

2. Pattern layer: normalized inputs are compared against training samples using a euclidean distance, which gets then applied an exponential kernel. This transformation allows for comparing values, and not be highly affected by noisy data.

3. Summation layer aggregates all patterns comparison that belong to a specific class to compute a final score.

4. Classification layer is the final step, which consists on picking the category in the summation layer with the highest score.

In [17] the PNN is used for Intrusion Detection Systems, one of the reasons being the ability to enhance its training speed. The PNN is also a Neural Network which is not affected by small variations in the environment (i.e., noise) [18]. For such reasons, we employ the PNN on this dissertation for an enhance Intrusion Detection System which is both adaptive and cooperative with a node’s neighbors.
CHAPTER 3: OPTIMIZING NETWORKS VIA SWARM INTELLIGENCE

With a variety of existent technologies and the networks usage rising, the network space has become highly crowded. Such crowded environment introduces more coverage, at the cost of negative impacts such as: more power dissipation and noise, more security threats, more load on servers, etc. In this chapter we develop an approach to optimize the deployment of networks, with an algorithm which extends the Particle Swarm Optimization (PSO) algorithm. The solution is entitled Optimal Network Evolver (ONE). ONE utilizes evolutionary computation inspired approaches, to optimize networks in terms of power consumption, coverage, data confidentiality, and anonymity. The domains of computer communication networks and smart grids have been chosen as optimization case studies. ONE’s features are analyzed and compared against the base PSO to understand their contribution.

Introduction

Networks are complex structures, typically modeled by a graph, with a high performance potential. Networks are applied into many fields: computers, power distribution, telephones, just to name a few. Each of these fields help us to solve our daily needs in a different manner. However, the tools available to optimize such networks may sometimes not be effective because of the nature of the problem. Graph theory can have a collection of hard to solve problems. This work aims at solving such a complex challenge via a different type of model.

Published on [19] with some extensions made for this chapter
Therefore, an evolutionary inspired algorithm entitled “Optimal Network Evolver” (ONE) is presented as an multi-objective optimizer. ONE promises to expose an easy to build and cost-effective optimization solution, such that it can be run without the need of a sophisticated environment. The rest of this chapter contains the following. First, we describe the specifications, features, and advantages of ONE; followed by the presentation of the ONE algorithm. We then present the experimental setup along with results and a discussion comparing ONE against the base PSO as well as ONE’s performance in larger environments. Finally the chapter is closed with ONE’s remarks and future work.

Optimal Network Evolver

ONE is an algorithm for searching optimal network configurations across a given map. ONE builds primarily upon the concept of complexification and innovation. Complexification is the act of start a simple search, and only complicate the search gradually when required. The innovation is an attempting for spreading the search to new areas, such that the number of tried possibilities can be maximized.

In this section we discuss the principles upon which the Optimal Network Evolver (ONE) builds up. To avoid unnecessary search space complexity, evolutionary computation is employed as basis for the algorithm. Hence the fundamental idea, as proposed in NEAT [20], is to start with low-complexity systems and evolve them gradually (i.e. add nodes), to avoid unnecessary searching in higher order spaces. As opposed to NEAT, ONE does not use cross-over operations, to avoid the scenario of pairing two nodes of different types of technology. Rather, a PSO approach is taken [8], where the configuration is gradually moved across the sample space, increasing in dimensions (adding nodes) as time progresses.
The general algorithm for ONE is presented in algorithm 2. The algorithm process consists of: creating a population (lines 1–3) of possible network configurations, updating each configuration on every generation, and recording the best values over the evolution (lines 15–18). Figure 3.1 denotes an example run, on which each generation nodes reconfigure themselves, moving across the map; nodes are added gradually to enhance performance.

Figure 3.1: Example run of ONE

The following subsections discuss in detail the different components which form ONE.

**Gene Representation**

Gene representation refers to creating a structure, which can represent all particles (i.e network configurations) to be evolved. Gene representation can help ONE keep track of which parameters require more exploration than others. In ONE, every network configuration has a set of nodes $N$, where all nodes are tuples of the form $(x, y, tx, rx, n, type)$. With such a structure, the algorithm can find the best location (i.e. $x, y$) to place all nodes, and specify how much power should it
be used to transmit (i.e. \(tx\)) and sense (i.e. \(rx\)). The algorithm also configures each node with the most appropriate number of keys \(n\) to store in the node, following the idea presented on [21], where a Heterogeneous Wireless Sensor network key distribution algorithm was proposed. For the work in [21], the heterogeneity was considered as the processing and storage capabilities of a node. In ONE, the heterogeneity in addition comes from the technology types (i.e. \(type\)) in the Heterogeneous Wireless Mesh Network.

**Comprehensive Learning by Individual Knowledge**

The Particle Swarm Optimization (PSO) [8] is used as base for ONE. The PSO moves particles (i.e. network configurations) across the search space, according to their previous knowledge (for exploitation), and global knowledge (for exploration). Each generation, the configuration parameters contained within the gene, represented by \(I_{ACTUAL}\), are moved at a velocity \(V_{PARAM}\), and tested on the different fitness functions. If the newly updated parameters performed better than the particle’s best known solution, defined by the \(I_{BEST}\) parameters, then the current configuration \(C\), composed by all the \(I_{ACTUAL}\) parameters, replaces the current individual best \(ibest\) (line 15). Moreover, if such a configuration \(C\) is better than the individual best known by the other particles, \(C\) will replace the global best configuration \(gbest\) as well. ONE, however does not utilize the global knowledge, on the same manner as that of traditional PSO, because of the topology matching problem, described as follows:

- **Topology matching problem:** two nodes of different technology are not compatible, because they might operate at different frequency, and they do not share the same communication protocol and physical limitations. These limitations may include storage, transmission range, technology type, etc.
For example, consider the worst case scenario, where two possible network configurations $C_1$ and $C_2$ contain all their nodes of one technology $T_1$ and $T_2$ respectively. If $T_1 \neq T_2$, ONE cannot physically compare the nodes in $C_1$ with those in $C_2$, hence, the only way to create a cross-over would be to include all nodes in $C_1 \cup C_2$, yielding higher network costs.

Thus, only the individual knowledge gained by each particle is used as seen in line 10. The individual knowledge is represented by the ancestor $A$. Moreover, such ancestor $A$ of a node $N$ represents a node that shares the same historical marking id of $N$. $A$ only appears if a solution has proven to surpass its best experience, as nodes are tagged when their individual best configuration changes. If such a node has been added recently, or it has not proved to cause a significant improvement in the solution, then the node is updated randomly, according to line 11.

Furthermore, exploration may be fomented at early generations with the operation in line 9. The value of $\omega$, which is defined as the inertia weight, changes linearly from a large value (initially), to a smaller value (final). Such a change causes particles at the beginning of ONE’s execution to explore different areas of the search space; then, after a defined number of generations $G_{MAX}$, the value of $\omega$ reaches and stays on its final value, such that the particles can refine their search on their respective areas.

**Historical Marking**

The historical marking is a unique id, which identifies nodes in the order on which they appear on a network [20] at line 22 of the algorithm. The historical marking allows ONE to identify nodes, when these are compared for updating a particle (i.e. lines 7 and 10). Thus, nodes may use their previous experience on the search space to improve the results (see line 10). Moreover, nodes that are new on the solution may be updated randomly (see line 11), and use any acquired experience once its contribution helps surpass the current individual best configuration (lines 15–18).
Complexification

Complexification is fundamental for the functionality of ONE, and it is addressed at line 22 of the algorithm. The key idea is to start the search for the optimal configuration at a low dimension space (i.e. 1 node), to avoid unnecessary and slower search. For example, in an area of 50 meters by 50 meters, it is undesired to start searching an optimal configuration with 1000 nodes, if the solution may be acquired with 30 nodes.

Hence, every possible solution starts with 1 node, where the technology might be different for each solution. ONE gradually adds or removes nodes following the probabilities specified by the network preferences. If the probability for adding a node is very high, ONE will add nodes faster, without having the time to refine them on the search. Hence, the probability of adding a node should not be very high, nor very low.

In general, the addition of nodes causes the dimensionality of the search space to change, providing ONE with new configurations to explore. Furthermore, complexification in ONE can give network designers the correct number of nodes required to provide services in a given area, under the specified configuration.

Genetic Diversity

ONE is defined to take advantage of population control mechanisms such as Genetic Diversity. Genetic Diversity tries to spread the traits of population particles. The purpose of these spread particles, from ONE’s perspective, is to explore as much as of the sample space. Such exploration awards with a better view of the environment, trying out different configurations that lead to what we call a Pareto optimal surface. In a highly deceiving environment, such as network optimization, we can explore similar performance with different configurations. ONE finds such
similar performance with the Pareto surface and proceeds to store the surface diverse components in an innovation archive shown in line 20. In order for this innovation archive to be populated, the diversity of the configuration needs to be high enough (different) when compared to other configurations in the innovation archive. Note that such archive initially is empty; configurations are only evaluated for diversity when they get stalled for longer than $G_{NOVEL}$ generations. A configuration is classified as stalled when the difference between its current performance vs. its individual best is smaller than a specified threshold.

If the threshold is small (e.g. 0), then ONE will avoid diversity, unless a solution obtains the exact same result in $G_{NOVEL}$ generations; only under such condition a configuration would be considered to be trapped in a local optima. However, if the threshold is set high (e.g. half the maximum fitness), then ONE might see particles as trapped each $G_{NOVEL}$ generations, causing ONE to act as an only Genetic Diversity Search algorithm.

Once the counter exceeds $G_{NOVEL}$, the algorithm proceeds to make the proper replacement, and resets the counter. Diversity is measured with a K-Nearest neighbor approach as follows

$$Diversity_i = \frac{1}{k} \sum_{j} dist(gene_i, gene_j)$$

where $k$ is the number of particles that should be considered as neighbor to particle $i$. Thus, the expression represents the average Euclidean distance between genes $i$, and its $k$ nearest neighbor. Following particle replacement, the new particle gets a fresh counter and starts exploring an area where diversity is high.

ONE also records any removed configuration, such that future novel configurations can avoid the search region where the removed configuration was previously located. Such record of removed configurations provides the designer with sub-optimal configurations, which may be used as an
alternative to the optimal one, giving the network fault tolerance.

Algorithm 2 Optimal Network Evolver (ONE)
1: Start with a random population of network configurations \( P \)
2: Initialize individual bests as the initial network configurations
3: Choose a global best from the particles
4: while Generations Remain \( \text{AND} \) Value not reached do
5: for Each Configuration \( C \) in \( P \) do
6: for Each Node \( N \) in \( C \) do
7: Get ancestor \( A \) in individual best with historical marking equal to that of \( N \)
8: Update the velocity of each parameter \( I_{\text{ACTUAL}} \) according to:
9: \( \omega \leftarrow (\omega_{\text{INIT}} - \omega_{\text{FINAL}}) \cdot \frac{G_{\text{MAX}} - G}{G_{\text{MAX}}} + \omega_{\text{FINAL}} \)
10: if \( A \) not \( \text{NULL} \) then
11: \( V_{\text{PARAM}} \leftarrow \omega \cdot V_{\text{PARAM}} + c \cdot \text{rand()} \cdot (I_{\text{BEST}} - I_{\text{ACTUAL}}) \)
12: else
13: \( V_{\text{PARAM}} \leftarrow \omega \cdot V_{\text{PARAM}} + c \cdot \text{rand()} \)
14: end if
15: \( C \leftarrow C + V_{\text{PARAMS}} \)
16: Evaluate new configuration to obtain fitness \( F_C \)
17: if \( F_C > F_{\text{ibest}} \) then
18: \( \text{ibest} \leftarrow C \)
19: if \( F_{\text{ibest}} > F_{\text{gbest}} \) then
20: \( \text{gbest} \leftarrow \text{ibest} \)
21: end if
22: end if
23: if \( |F_C - F_{\text{ibest}}| < \text{Threshold} \) then
24: Increase stagnation counter for \( C \)
25: if \( \text{counter} = G_{\text{NOVEL}} \) then
26: Record/Replace \( C \) with a novel network configuration
27: end if
28: else
29: Add/Delete a node with a probability of \( P_{\text{ADD}}/P_{\text{DEL}} \)
30: end if
31: end for
32: end while
Multi-objective Optimization

This subsection discusses the different fitness functions (i.e. coverage, power, confidentiality, anonymity, and cost), to compare what solutions are best, and how they may affect each other. Each fitness function is linearly added, into a total fitness function, as follows:

$$F_{TOTAL} = \sum c_i \cdot F_i$$

where the coefficient $c_i$ is a real number that specifies the degree of importance of each fitness. Fitness values that should be maximized have a $c > 0$. Minimization occurs when $c < 0$, since increasing such fitness will cause a penalty on the total fitness.

**Coverage**

Coverage consists of the total area on a specified map, which is covered by a network. That is

$$F_{COVER} = \frac{A_{COVER}}{A_{TOTAL}}$$

where ONE defines an area as covered $A_{COVER}$, if there is at least one node in the network, whose sensing radius reaches $A_{COVER}$. Network coverage is greatly affected by path loss; obstacles along the way may deteriorate a signal before reaching a specific point or region. In order to remedy the effect of path loss, nodes may be set at another location, their power consumption may be increased, or the nodes may be replaced by another technology, which is less affected by path loss. All counter measures may be used by ONE to overcome the obstacles; however, this implies that the nodes genes (i.e. location, power, type) need to be changed, affecting other fitness criteria, such as: power, confidentiality, and anonymity.
Power

Nodes consume energy upon transmission and reception of packets, and also when sensing the environment [22]. Furthermore, some technologies may spend more time sensing the environment (e.g. sensor networks), others transmitting packets (e.g. Wi-Fi). Nonetheless, both cases are handled by ONE. Hence, the power consumption of a network is given as the sum of the power consumption in all of its nodes, or more formally

\[
F_{POWER} = \sum_{i} P_{txi} + P_{rxi}
\]

where \(F_{POWER}\) is normalized, such that it can be added to the coverage fitness. A negative coefficient \(c\) is used to penalize solutions which use more power than others. As seen in the definition of coverage, the power function may also be affected by path loss. Thus, increasing the power to improve coverage yields a penalty in the power fitness as well. Under these circumstances ONE may find more appropriate to add a new node, which could affect the anonymity and confidentiality of a region.

Confidentiality

Confidentiality, one of the key aspects of security, deals with the integrity of a packet. That is, the information contained inside a packet should be known only by its sender and recipient. Hence, it is required for ONE to find optimal confidentiality on a network, such that users in the heterogeneous wireless mesh will feel comfortable using the services provided. Confidentiality can then be measured as the probability of an attacker not finding the encryption key used by a link:

\[
F_{CONF} = 1 - p_{DISCOVER}
\]
Such probability may be given by the key distribution approach, encryption algorithm, number of nodes in the region, etc. For simplicity, ONE uses a brute force approach, which may be affected by the number of nodes in the region, and the number of keys in the key pool, and how much of these keys are assigned to a node [21].

**Anonymity**

Anonymity is of great interest in a heterogeneous wireless mesh network, especially on service-oriented ones. For example, in the domain of urban computing, users may want to share information regarding gas prices seen on the road, without disclosing their exact location to the mesh. Thus, anonymity can be seen, from the server’s side, as the uncertainty of receiving a packet from a specific client. Moreover, ONE uses the entropy as a mechanism to measure anonymity as proposed in [23]. The goal of ONE is to maximize the entropy (i.e. uncertainty) on all the nodes. Such a goal can be achieved by maximizing

$$F_{ANOM} = \min \{- \sum_i p_i \log_2(p_i)\}$$

where $p_i$ is the probability of a server receiving a packet from client $i$. ONE’s implementation assumes that such probability is $\frac{1}{N}$, where $N$ is the number of neighbors of the same technology, or bridge nodes, in range. Hence, by maximizing the minimal entropy among all the nodes in a network configuration, ONE achieves that all nodes will at least have that much uncertainty about their neighbors. Nodes may receive packets from other nodes of the same technology, or the bridging technology node. In the bridging nodes, packets may come from any technology; hence, the anonymity in the technology bridges will depend upon the number of neighbors surrounding it, which depends on the transmission range of each node, and affects the power consumption (i.e. longer range consumes more power).
Cost

Cost is driven by the number of nodes in the network, and their power consumption. Since ONE does not replace the global best until the new solution yields a higher fitness function, the algorithm favors solutions which possess less number of nodes, found at earlier generations. ONE ensures that solutions with more nodes are not discarded as soon as they appear, since solutions are only removed if they get trapped in local optima for more than $G_{NOVEL}$ generations. Hence, it might be cheaper to operate in an area, on a long term basis, with more nodes, but using less power consumption depending on the technology type and resources.

Path Loss

Although path loss is not a fitness function, it affects directly the performance of the different configurations, hence the multi-objective optimization. ONE has embedded the path loss of free space, trees, and light and heavy walls. Depending on the nodes frequency of operation, the path loss may be greater or weaker as defined literature by

$$L = \left( \frac{4\pi df}{c} \right)^2$$

where $L$ is the free space path loss, $d$ is the distance, $c$ is the speed of light in vacuum (i.e. 3$e10^8 m/s$), and $f$ is the frequency of operation in hertz (Hz). Hence, higher frequencies (e.g. IEEE802.11a) have a larger path loss than lower frequencies (e.g. IEEE802.15.4). ONE takes into account all obstacles found in a path of the node signal, such that the final path loss is equal to the addition of the path loss in free space, plus the path loss caused by the obstacles at the grid map.
Experimental Setup

For conducting the experiments, 10 runs were done on the selected maps, using a Java implementation of ONE. Following the details of each experiment’s purpose and implementation.

**ONE’s Feature Comparison**

The purpose of our experiments were two folded. First, we wanted to see the feasibility of ONE’s features, i.e., how changing its tuning parameters affected the performance of the algorithm when compared to the base PSO. This comparison was achieved by feature toggling each of the individual parameters, focusing on the effects of Complexity, Genetic Diversity, and Population Size. At the end of a run, we compared primarily the total fitness function value, as well as the innovation archive size to see if the features helped with performance or the ability of adapting to dynamic environments.

**ONE’s Scalability Performance**

The second part of the experiments were to test ONE’s performance on larger environments, for scalability. To test ONE’s performance, we have selected four types of maps: free space, a park with little interference caused by randomly set trees, an area with a building at the center, and a map whose dimensions are equal to that of the island of Puerto Rico (i.e. 180km by 65km). In the map of Puerto Rico, the main forest areas where considered as tree interference, metropolitan cities had randomly set buildings, as well as houses for the rest of the island.

The system was designed to evolve a heterogeneous wireless mesh network, composed of WiFi, Bluetooth, and Zigbee. Nodes could only communicate with the same technology, or a bridge
node. The system parameters were changed until satisfactory performance was obtained, with the configuration as shown in table 3.1. Hence, nodes were added and deleted, following the probabilities of $P_{ADD}$ and $P_{DEL}$, respectively. When adding nodes, ONE needed to decide on the type of technology to be used (i.e. $P_{802.11}$, $P_{802.15.1}$, or $P_{802.15.4}$), and the subtype (e.g. WiFi N), using their respective probability of occurrence. Fitness functions with positive coefficients specify the parameters to be maximized, while negative coefficients indicate that a minimization would occur, according to the importance given by the absolute value of each coefficient. The shown configuration favors complexification, and a network with WiFi technology as it is the bridging technology for the HWMN.

Table 3.1: ONE Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{ADD}$</td>
<td>Probability of adding a node</td>
<td>0.2</td>
</tr>
<tr>
<td>$P_{DEL}$</td>
<td>Probability of deleting a node</td>
<td>0.03</td>
</tr>
<tr>
<td>$P_{802.11}$</td>
<td>Probability of selecting WiFi</td>
<td>0.33</td>
</tr>
<tr>
<td>$P_N$</td>
<td>Probability of selecting WiFi N</td>
<td>0.1</td>
</tr>
<tr>
<td>$P_G$</td>
<td>Probability of selecting WiFi G</td>
<td>0.3</td>
</tr>
<tr>
<td>$P_B$</td>
<td>Probability of selecting WiFi B</td>
<td>0.3</td>
</tr>
<tr>
<td>$P_A$</td>
<td>Probability of selecting WiFi A</td>
<td>0.3</td>
</tr>
<tr>
<td>$P_{802.15.4}$</td>
<td>Probability of selecting Zigbee</td>
<td>0.33</td>
</tr>
<tr>
<td>$P_{STD}$</td>
<td>Probability of selecting Zigbee STD</td>
<td>0.65</td>
</tr>
<tr>
<td>$P_{PRO}$</td>
<td>Probability of selecting Zigbee PRO</td>
<td>0.35</td>
</tr>
<tr>
<td>$P_{802.15.1}$</td>
<td>Probability of selecting Bluetooth</td>
<td>0.33</td>
</tr>
<tr>
<td>$P_{BT1}$</td>
<td>Probability of selecting Bluetooth C1</td>
<td>0.2</td>
</tr>
<tr>
<td>$P_{BT2}$</td>
<td>Probability of selecting Bluetooth C2</td>
<td>0.3</td>
</tr>
<tr>
<td>$P_{BT3}$</td>
<td>Probability of selecting Bluetooth C3</td>
<td>0.5</td>
</tr>
<tr>
<td>$S$</td>
<td>Keypool Size</td>
<td>10000</td>
</tr>
<tr>
<td>$C_P$</td>
<td>Power Fitness Coefficient</td>
<td>-0.6</td>
</tr>
<tr>
<td>$C_C$</td>
<td>Coverage Fitness Coefficient</td>
<td>1</td>
</tr>
<tr>
<td>$C_S$</td>
<td>Confidentiality Fitness Coefficient</td>
<td>0.5</td>
</tr>
<tr>
<td>$C_A$</td>
<td>Anonymity Fitness Coefficient</td>
<td>0.5</td>
</tr>
<tr>
<td>$G_{NOVEL}$</td>
<td># generations for trap classification</td>
<td>10</td>
</tr>
</tbody>
</table>
The next section provides in depth discussion on the results.

Results and Discussions

This section of the chapter discusses on the two experiments described before: feature comparisons and contributions, and the test for performance scalability as an environment grows.

ONE’s Feature Comparison

We begin by discussing the characteristics exhibited by toggling each of the features on and off separately.

Observations with basic PSO

Problem appears to be highly deceptive, i.e., the particles were converging to the same point. This could be seen when analyzing the innovation archive which at the end had only one particle on it. Because of this phenomenon the global best was turned off (by making its influence constant equal to 0). This caused the innovation archive size to create to around 4 particles on average, but the performance was still with low. Figure 3.2 displays the evolution of total fitness found the PSO. In this graph, we see a traditional PSO trapped in a local optimum with a standard deviation a little high considering the range of the fitness (i.e., 0 – 3.2).
Effect of Complexity

Complexity, or dynamically changing the structure of the network configurations, contributed to find insights of what the best fitted network size should be. Without such insight, we could be performing many trial and error runs to find a best solution. Since the domain appears to be highly deceptive, we could spend endless combinations of runs, without a clear insight, hitting many local optima.

Since our approach added and removed nodes into the configuration, it is to note that the innovation archive ends up with many configurations, some of them with less nodes and providing still close to equal performance. This is what we defined as the sub-optimal plateau, or more formally a set
of configurations which provides the network the ability to react to environmental changes.

**Effect of Genetic Diversity**

Genetic diversity helped with two factors. The first and most notable was an increase in total fitness function, attributed to both higher coverage and lower power consumption.

The second contribution was an increase in the number of members on the innovation archive. This second contribution not only aided to finding higher total fitness, but in the process find alternatives for sub-optimal configurations which can be used upon to react or retrain upon changes in the environment.

**Effect of Population Size**

Changing the population size with genetic diversity turned on had a surprising effect. It was expected that due to genetic diversity resetting particles that got stuck in local optima, the population size would not have any effect. However, we found an increase in size of the resulting innovation archive. Moreover, later we turned off the genetic diversity to arrive into a proper conclusion; the result was that the innovation archive size dropped. Thus, proving that increasing the population size does not help. The increase of the innovation archive size was due to genetic diversity having more particles to tune and restart. Given this result, it is recommended that the population size is only increase if it does not yield a negative resource impact.
Elements Combined

The combination of all factors allowed to create fitted solutions with about half of the nodes than the algorithm by itself. Figure 3.3 shows a comparison for the first 1000 generations of the algorithm. There are several points to make here. The performance of the ONE algorithm at the beginning looks slightly lower than the PSO. This effect is due to the complexification still performing its effect. However, we can note that the algorithm does not appear to be trapped yet and the standard deviation (red dashed lines) are smaller than the PSO. This smaller standard deviation implies that the ONE algorithm has better quality solutions (or at least closer to each other).

Figure 3.3: ONE: Evolution of Total Fitness (first 1000 Generations)
We decided then to let the ONE algorithm run until 10000 generations to check when the algorithm finished. Figure 3.4 shows such evolution displaying the performance in total fitness reaching its maximum value close to 10000 generations. However, acceptable solutions can be seen earlier around 6000 generations.

![Figure 3.4: ONE: Evolution of Total Fitness](image)

Figure 3.4: ONE: Evolution of Total Fitness

Figure 3.5 presents the evolution of the innovation archive. In this figure we can appreciate how a little after starting the evolution the genetic diversity begins spreading the particles. Later, but still early in the search time, we see a maximum spread taking place. After this point the ONE algorithm focuses on improving all the configurations in the population, whose best individuals are highly likely to end up inside the innovation archive at the end of the ONE execution.
The results on average, per each type of map are presented in table 3.2.

Table 3.2: ONE Performance per Scenario

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Cover</th>
<th>Power</th>
<th>Conf</th>
<th>Anonym</th>
<th>#Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free</td>
<td>0.9956</td>
<td>0.2280</td>
<td>0.9821</td>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td>Trees</td>
<td>0.9736</td>
<td>0.2305</td>
<td>0.9665</td>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>Building</td>
<td>0.9576</td>
<td>0.3621</td>
<td>0.9121</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>Puerto Rico</td>
<td>0.9135</td>
<td>0.6165</td>
<td>0.9005</td>
<td>1</td>
<td>210</td>
</tr>
</tbody>
</table>
Figures 3.6–3.8 depict the average evolution of the connectivity, power, and anonymity fitness functions. Although there are fitness functions which may be maximized at earlier generations, others require more exploration and hence, the Pareto point is found at subsequent generations, since adding all the individual fitness with their proper coefficient of importance, achieves the highest value.

![Evolution of Coverage](image)

**Figure 3.6: Evolution of Coverage in 150 Generations**

![Evolution of Power Consumption](image)

**Figure 3.7: Evolution of Power Consumption in 150 Generations**

When the power fitness penalty was not included, nodes had the tendency to maximize their transmission and reception power, such that the map coverage could achieve 100%. However, with a
small reduction in coverage (i.e. less than 1%), the power could be reduced to 20% of the total allowed power, by adding nodes to the network.

Studying in more detail the observed power vs. coverage trade-off, it was observed that as the path loss becomes greater (i.e. map type contains less free space), nodes need to increase their power in order to cover more area. An increase in number of nodes can be appreciated as well.

![Evolution of Anonymity](image)

Figure 3.8: Evolution of Anonymity in 150 Generations

The increase in number of nodes can help with the anonymity of the network, as shown by the results. However, saturation point may be reached, which could represent the highest uncertainty, that is, the point where a server might think a packet arrived from any of its connected clients, without any information that can help improve the odds of guessing the client correctly. The discrete steps in the evolution of anonymity are caused by the manner on which the anonymity is being maximized. That is, ONE’s fitness function returns the minimum anonymity among all the nodes in the network, hence, when such anonymity is maximized, ONE guarantees that all nodes in the network will contain at least that amount of anonymity.

Increasing anonymity can cause also a drop in confidentiality, which may be due to the increase in neighboring nodes, as results have shown in the section 3. Furthermore, scaling ONE is affected
by a small drop in performance as shown in table 3.2, and its execution speed becomes slower, which we believe it may be due to the map and fitness representation. Thus, for future work, we plan on modifying the representation for obtaining better scalability.

Figure 3.9 illustrates how ONE, considering the number of defined fitness functions, can produce optimal networks for any type of map.

![Figure 3.9: Examples of Optimal Heterogeneous Network Configuration](image)

The maps used were 50 meters by 50 meters, with isotropic radiation assumed from all nodes. The red circle, centered at a node X, represents the sensitivity of that node. Black, Blue, and Green nodes denote Wi-Fi, Bluetooth, and Zigbee nodes respectively. The filled rectangle 3.9(b) are areas containing trees; similarly the empty rectangle 3.9(c) is the building defined for the building map.

The next section closes the paper with some remarks on ONE.

Conclusions

In this paper, we have introduced ONE, an evolutionary algorithm for optimal heterogeneous wireless mesh networks deployment. ONE was able to find optimal Pareto points, by maximizing coverage, anonymity, and confidentiality; the cost and power consumption were minimized as well, in this process of finding the optimal surface. ONE was shown to provide the advantage of indicating the network designer, and how many nodes of each type of technology are needed to
provide the best service, considering the fitness functions specified to the algorithm.

With ONE, it becomes possible to deploy complex networks that can solve emerging problems, including: Ubiquitous Computing, Smart Homes, Multimedia Provision, etc. As further study, we shall employ ONE to find an optimal network for a Multimedia Provision network, such that its performance may be studied on real hardware.

As future work, we also plan to use ONE with more fitness, considering multiple QoS design issues such as speed, data storage, throughput, and collisions in the medium (i.e. channel assignment). We want to enhance the algorithm with additional improvements, such that the resource consumption and speed of the algorithm are improved, and obtain a better cost fitness function. ONE could also provide adaptive configurations, considering dynamic data as the network traffic and mobility.
CHAPTER 4: ADAPTING TO DYNAMIC ENVIRONMENTS

Parameter tuning in Evolutionary Algorithms (EA), is a great obstacle that can become the key to success. Good parameter settings can yield optimal solutions, while bad settings may trap the EA, thus removing the chances of finding the optimal solutions. Therefore, it is vital that an optimal set of parameters configuration is chosen. It is a common practice to have a human expert that analyzes such parameters and modifies them accordingly. Such process is inefficient and expensive, since it requires time and is subject to human fatigue; it even becomes impractical if the environment is dynamic.

In this chapter, we develop 2 adaptive strategies to tune such parameters: One Step Variation (OSV) and a Fuzzy Logic Controller (FLC). The OSV is used as building block to construct what we consider optimal fuzzy sets that can be applied into the domains considered in this dissertation. For such reason, we chose to test the strategies with fitness functions known to be highly deceiving (i.e., many sub-optimal solutions).

In particular, the 0-1 Knapsack problem was chose as one of the test functions. The 0-1 Knapsack is chosen because in this problem, we either chose an item (configuration) to gain some value at a given cost (known as weight). This situation is seen throughout this dissertation across the considered domains. In the computer network deployment domain values can be seen as the benefit (e.g., higher coverage) at the cost of number of nodes and subsequently power consumption. Similarly, in the domain of Smart Grids, the value added is the reduction in power grid at the cost of having more solar panels or wind turbines. Moreover, the in the Intrusion Detection System we

Published on [24] with some extensions made for this chapter
choose a feature and consulting a neighbor to reduce chances of anonymity and classify correctly a malicious user, at the cost of having a more complicated model with more nodes on the environment. Thus, at the meta-algorithm level, we can say that the considered domains in this dissertation follow the 0-1 Knapsack problem.

Given the highly deceiving nature of the domains, our enhanced PSO is able to find a population of sub-optimal and one optimum. Thus, in this chapter we introduce a ranking scheme such that the adaptive strategies can determine which solution is best based on diversity and quality of the final population. This scheme ensures that if a final configuration has slightly less performance but much better adaptiveness when compared to another final configuration, the first is favored over the latter. This combination of selecting based on quality and diversity ensures good performance in the domain as well as ability to react to dynamic environments (e.g., nodes failing, power source failing, ambient light or environmental noise changes, etc.) Results show that it may be possible to tune the parameters in an EA for achieving better results, without the need of an expert.

Introduction

Evolutionary algorithms such as genetic algorithms have been subject to deep analysis to determine their effectiveness on a wide range of applications. Many practitioners have come to the understanding that in order to determine the performance of an algorithm, it is required to first study how it behaves on the domain, when its configuration parameters are changed. In [25], Schaffer states that “It has long been acknowledged that the parameters that control a genetic algorithm can have a significant impact on its performance and that the theory behind this technology gives little guidance for their proper selection”.

Such procedure is inefficient and expensive, since an expert on the field has to analyze how each
parameter affects the algorithm on the given domain. To overcome such an obstacle, there have been several attempts in the past to model the parameters effect on the algorithm [25–28]. Unfortunately none have succeeded in devising a framework to study the algorithms fairly in a way that can be generalized. Thus, the community has accepted that the only way to tune up parameters is to have a human analyze the parameters’ behavior per domain.

In this chapter, it is successfully attempted to remove the human from the parameter tuning chain. First, a One Step Variation (OSV) strategy was used. The OSV works by running multiple runs of an EA, each time varying one new parameter (e.g., mutation rate, crossover rate, crossover type, etc.). The OSV searches for possible convergence on the parameter values. Convergence is defined as parameters that arrive at the same region (i.e., low values, medium values, or high values), and succeed in solving the problem domain. Parameters without convergence are not important for the specific algorithm and domain; therefore any value is suited. Moreover, the convergence value of the parameters suggest an acceptable configuration for the parameters of an EA on a given domain. The convergence of parameters can be visually examined by a human expert, or with simple average range by a software program.

Furthermore, a feedback system is proposed to mimic the humans’ cognitive abilities. Such a feedback system employs a Fuzzy Logic Controller (FLC). The FLC tunes parameters real-time, adjusting the algorithm behavior after a desired of generations has passed. The usage of fuzzy logic can help mimic the humans decisions, since it is able to distinguish between good, bad, and acceptable performance.

Moreover, using an analysis of variance (ANOVA), we present a way of distinguishing among different algorithms or similar ones with the same behavior. Along with the ANOVA, a ranking scheme is proposed, to help find which algorithm performs the best. Results show a fast convergence in the adaptive algorithm onto acceptable solutions, better than those obtained via brute
force search and human parameter tuning. Furthermore, the proposed strategies, particularly the OSV, allows an easy understanding of dependencies that may exist among parameters.

The rest of the chapter is organized as follows: section 4 presents a list of phenomena that provide feedback to our adaptive system. Section 4 presents a list of proposed strategies. In section 6 we show the experimental setup, followed by results and discussions at section 6. We then close the chapter with some remarks on section 4.

Feedback Definition

This section describes the different parameters that our strategies consider, events that we wish to measure, and the output variables that allow us to measure such events. We also present a ranking scheme to rank the evolutionary algorithms (EA) performance in a given problem.

Parameters Tuned

The parameters described in this subsection are essential for our strategies to work, as they allow for the EA to behave differently, depending on their values. Different behaviors lead to different performance results. Following the list of such parameters:

- **Gene Length**: The Gene Length limits the range of values that a gene can assume; thus, limits the search space size, allowing faster and more effective search if it happens to fall on the same region as the global solution. The process of increasing the structure to search further regions is known as complexification [29].

- **Mutation Rate**: Mutation Rate affects how EAs explore the sample space. Larger mutation rates would have the same effect as random search, since individuals in the EA change fast,
preventing fine tuning on any potentially good solution. However, low mutation rates prevent any new information from appearing in the population, thus, risking getting trap under local optima. According to [25], the expected number of mutations should remain a constant, function of the population size, mutation rate, and length of the genes.

- **Crossover Rate:** Crossover Rate defines how often individuals combine to pass on to future generations a mixture of genes among themselves and the individuals with whom they mate. High crossover rates may extinct non-fitted individuals, unless the selection pressure is not tight. Low crossovers removes the possibility of knowledge sharing, making individuals learn only with their own experience.

- **Crossover Types:** The Crossover Types define information exchange schemes. There are two common types of crossovers: n-point and uniform. Crossovers, depending on their type, may have positional or distribution bias. Adapting the system with different crossovers may provide mechanisms to overcome such biases identified in [30].

- **Introns Length:** Introns can help reduce the crossover bias by inserting non-coding gene regions into the genome [31]. Such an action reduces the possibility of crossover destroying building blocks, since it gives more crossing points. In nature, it is believed that introns can sometimes, on some generation, become active codes and introduce spontaneous innovation.

- **Population Size:** The Population Size affects the sampling rate of the EA on the search space. Higher population makes the algorithm computationally more expensive, but introduces more precision to the EA. There comes a time as seen in [6], where increasing the population does not improve the EA performance; the proposed adaptive strategies try to find that point where increasing the population stops improving sampling, and keeps increasing computational time of the EA. Such a point is the optimal population size.

- **Selection Type:** The Selection Type defines the selection pressure, and helps the EA de-
termine which solution is more suited to expand. Strong selection pressure creates a highly competitive scenario were individuals are forced to become better or disappear from the evolution chain. By contrast, weak selection pressure gives the opportunity for innovation in apparently bad solutions, which could improve and become the optimal solution.

- **Generational Gap:** This parameter tells the EA to preserve part of the current population for future generations. Preserving the population can make an algorithm from being generational to serial EA, and control better the diversity.

- **Cull Rate:** The Cull Rate works similar to Generational Gap, except that it tells the EA to dispose of the worst individuals, stopping innovation if it is set too high, allowing faster convergence to apparent feasible solutions.

- **Elitism:** Elitism is a parameter that tells the EA to keep the first best individuals alive from generation to generation, theoretically the fitness function should increase by generation when such action is taken.

**Events**

This subsection elaborates on the different events which can help describe the current status of the EA. Based on this events we can decide which parameters to tune and evaluate on subsequent generations of the EA.

- **Trapped in Local Optima:** This event is possibly the most common problem with EAs. Local optima are solutions in high deceiving problems, where individuals tend to attract since it provides apparently good solutions. There might be many local optima, while fewer global solutions should exist. To avoid local optima enough sampling, exploration and innovation should exist in the EA.
• **Exploration:** The Exploration is defined as the percent of the sample space that has been covered by the individuals in the generation. Higher exploration provides better understanding of where potentially good solutions might be.

• **Innovation:** Innovation refers to novelty among individuals [12], [13], it is best to have higher innovation along with exploration, rather than merely exploration since all individuals might be concentrated in the same regions. It is desired that the individuals search as spread as possible, for a faster convergence to the correct value.

• **Speed:** Faster EAs require less number of function evaluations (NFEs). Such criterion gives fair treat to EAs to compare among different population sizes and number of generations. A larger population EA takes more time on evaluating a generation, while a smaller population EA takes more generations to refine its search. Hence, the NFEs can be seen as a function of the population size $S$ and the number of generations $G$ (i.e. $\text{NFE}(S, G)$).

• **Quality:** Ideally better performing EAs are those with fast speed and high fitness on the objective function. The quality of the EA can be seen on how much improvement has been on the last sampling of generations, specifically, the slope of change in fitness over the change in number of generations.

• **Diversity:** Unlike exploration which studies the genotype variety, diversity can express the phenotype variation. If such is a small variation it might be possible that the population is trapped on a local large landscape.

The next subsection defines the parameters which provide feedback to the algorithm, via our adaptive module.
The outputs of the EA covered on this subsection helps our strategies to react to the events earlier described. They are key specially for the Fuzzy Logic controller, as the events are defined in terms of fuzzy memberships of such outputs.

- **Delta Fitness:** The Delta Fitness represents the change in best fitness since last sampling of the algorithm. It is formally defined as

\[
\frac{F_{actual} - F_{previous}}{G_{actual} - G_{previous}}
\]

(4.1)

where \( F \) is the fitness function evaluated at two timestamps \( \text{actual} \) and \( \text{previous} \). Similarly, \( G \) is the number of elapsed generations at \( \text{actual} \) and \( \text{previous} \) time. This definition of delta fitness not only checks that the fitness is improving, but also penalizes for solutions that get trapped for several generations.

- **Standard Deviation:** The current standard deviation of the population can help give a value to the phenotype diversity described in the Events subsection. In ideal solutions we should aim to have small standard deviation, but with high innovation, such that we have a plateau of unique solutions, yielding similar performance.

- **Innovation Archive:** The Innovation Archive keeps track of the most novel individuals across history of the population. Ideally such archive should be highly populated. An individual is added to this archive whenever it exceeds a novelty measure defined as the average distance to its \( k \) nearest neighbors.

Next we proceed to discuss ranking among different EAs, using a novel ranking scheme, which considers its overall performance against others.
Algorithm Ranking

Upon studying different EA performance, it becomes difficult to evaluate them to find out which EA performs better on a given domain. Moreover, which problems are better suited for what type of algorithm. Preliminary experience can help us define an initial empirical function, in terms of benefit-cost ratio, to express such ranking as

\[ \text{performance} = e^{(F_{\text{Best}} \cdot NFE(S,G)^{a} b)} \]  

(4.2)

where \( F_{\text{Best}} \) is the best fitness found by the algorithm, \( a \) and \( b \) are integers greater than 0 for maximizing problems, less than 0 for minimizing problems, and 0 to neglect the benefit-cost utilities with \( a \) and \( b \) respectively. Higher values of \( a \) should maximize the objective function; higher values of \( b \) should maximize the resource utilization. The opposite can be said for lower values of \( a \) and \( b \). Furthermore, a cost function can be defined in terms of the Number of Function Evaluations \( NFE(S,G) \), defined in terms of the population size \( S \) and number of generations \( G \) as

\[ NFE(S,G) = S \cdot G \]  

(4.3)

Another powerful tool is the Analysis of Variance (ANOVA). This tool can filter out similar algorithms which provide the same performance. These similar algorithms can be used as start points to test whenever adaptation is required on a domain. Following, we present the proposed adaptive strategies.
Adaptive Strategies

With the model defined in section 4, we can establish a feedback system to modify efficiently the parameters as seen in figure 4.1. First, an initial input parameters are specified to the algorithm. Then the strategies modify them according to the output of the algorithm (i.e., performance) based on the domain model. The optimization process is repeated while the algorithm is monitored, allowing real-time adaptive behavior. The proposed model can serve also as stopping criterion for the algorithm; the strategy might decide to stop the algorithm whenever an acceptable solution has been found.

![Figure 4.1: Architectural View of a Feedback System](image)

Any evolutionary algorithm may be monitored for obtaining better results. The EA receives a domain model, which attempts to optimize, while the strategy samples the EA periodically. By utilizing such strategies, change of the original parameters is possible at run-time.

Following we describe the proposed strategies to be implemented.
**One Step Variation (OSV)**

The first strategy is known as the One Step Variation (OSV). Although the OSV is not fully automated or applicable to real-time environments, the strategy provides a baseline framework for comparison. The OSV receives three initial sets of control parameters, one with generally low parameter values, one with mixed parameter values, one with high parameter values. Such parameters come from interaction with an expert; thus, the lack of full automation. The OSV works with each initial set in sequence, modifying each parameter one step at a time as determined by a variations set \( V = \{ v \in \mathbb{R}^2 \} \).

Each parameter is then chosen in an iterative manner as:

\[
p_i = \arg\max_{p_j} \{ \text{perf}(p_j) \} \tag{4.4}
\]

where \( p_i \) represent the elements of a parameter vector \( P \), and \( \text{perf}(p_j) \) is an output function that measures the performance of the EA with the given parameters, the value of \( p_j \) is formed as a vector of parameters, where the parameter under study is drawn from the variation vector \( V \). Some examples of such functions (e.g., Delta Fitness) were described earlier at section 4.

The OSV graphs the movement of parameters, which managed to solve the problem domain. Parameters which converge will arrive to a value range appropriate for the problem. Similarly, the parameters that do not have much effect will exhibit no convergence; therefore, any value can be assigned to them. Although using such an approach in some complicated domains might not be feasible, the OSV provides the human expert an idea of what values have to be set in low, medium, or high. Furthermore, using the computation of a standard deviation, it becomes possible to find programmatically which parameters have convergence and which do not. The human interaction...
Fuzzy Controller

The second strategy considered is a Fuzzy Logic Controller (FLC). Fuzzy logic can serve as a mechanism to model human intelligence. The goal of the FLC is to fully automate the process of parameter tuning, by analyzing the EAs and deciding whether or not a parameter should be changed. Such decisions can be accomplished with a logical structure defined as follows:

\[
\text{If Event}(\text{inputs, outputs}) \rightarrow \text{Modify}(\text{inputs})
\]

where the events are a function of the input parameters, which return the outputs states, as defined in the earlier section. Complex fuzzy sets (i.e., sets joined by logical functions, such as AND/OR/NOT) may be defined for either the Event or the Modify to allow for more elaborated analysis. A graphical view of the FLC is shown in Figure 4.2
Once a defined number of generations has passed, the FLC is executed. Given the current performance of the algorithm, as defined in Section 4, these values are transformed into their low, medium/normal, and high grade sets. Then, the Algorithm 3 decides which rule fires. Finally, the decoded sets tells the FLC what new parameter settings to try. In the next generations passed event, the process is repeated until convergence. Resumming such a process can take place whenever a change in the domain occurs.

The algorithm for selecting the correct state is presented in Algorithm 3. Such procedure activates the fuzzy set which has more certainty of being true (i.e., the highest decoded value). There are mainly two input sets on the system: quality \((q)\), defined as the slope of change in fitness per change in generations, and diversity \((d)\) measured by the standard deviation of the evaluated fitness. Both inputs have sets defined for low grade of values (Z shape sets, min, max), medium/normal grade of values (triangle shape sets, min, center, max), and high grade of values (S shape sets, min, max) values. In such cases, the min, center, and max are defined thresholds for which the extremes of the fuzzy sets are established. For decoding the rules, a center of mass approach is employed as follows

\[
value = \frac{\sum \mu(x_i) \cdot x_i}{\sum x_i} \tag{4.5}
\]

where \(\mu(x)\) is the activation level (degree of truth) of \(x\) in the fuzzy set.

Following we describe the experimental setup used to test our methodologies.
Algorithm 3 Fuzzy Adaptive Controller

Select Maximum Activated Rule Set, for the following:

- \( \neg high(q) \cap low(d) \rightarrow high(d) \)
- \( increasing(q) \cap \neg low(d) \rightarrow med(d) \)
- \( high(q) \cap high(d) \rightarrow low(d) \)

Decode the results to find which state to activate

Experimental Settings

For testing the proposed strategies, experiments were conducted, using a C# implementation of a GA. The fuzzy sets were initialized via trial and error as: quality (low(Z, 1/sampling, 5/sampling), medium(Triangle, 4/sampling, 8/sampling, 12/sampling) and high(S, 10/sampling, 20/sampling)), where sampling is how often to monitor the algorithm (in this experiment 10 generations), and diversity (low(Z, 0.25, 5), medium(Triangle, 4.8, 8, 10) and high(S, 9.8, 20)).

The following problems were set as objective functions, due to their sensitivity to parameter tuning, and in some cases applicable to model common problems:

1. **Least Max Best Fit**: Find the \( a, b, c, d, \) and \( e \) such that the maximum distance between \( g(i) \) and \( y(i) \) is minimized, where \( y(i) \) is a given target vector, and \( g(i) \) has the form:

\[
g(x) = ax^2 + bx + c + d\sin(x) + ec\cos(x)
\]

\[
F_{chromo} = \sum_{i \in |y|} |g(chromo_i) - y(chromo_i)|
\]

\[
BestChromo = \arg\min_{chromo}\{F_{chromo}\}
\]

2. **One Sum**: For a binary number \( chromo \) of length 100 (i.e. \( |chromo| = 100 \)), with bits
\(chromo_i\), maximize

\[F_{chromo} = \sum_{i \in |chromo|} chromo_i\]

\[BestChromo = \arg\max_{chromo}\{F_{chromo}\}\]

3. **Rastrigin**: The Rastrigin function was used since it is highly deceiving. It contains a large amount of local optima and one global optima when all dimensions are equal to 0. The function is formerly defined as:

\[F_{chromo} = 10n + \sum_{i \in |chromo|} (chromo_i^2 - 10 \times \cos(2\pi chromo_i))\]

\[chromo_i \in [-5, 5]\]

\[BestChromo = \arg\min_{chromo}\{F_{chromo}\}\]

where \(n\) is the number of dimensions in the domain. For our experiments we chose \(n = 3\).

4. **Rosenbrock**: As the Rastrigin, the Rosenbrock is hard to solve function because it has some local optima. Additionally, it introduces wide and almost flat landscape, which at a narrow point of it contains the global optima (i.e. where all inputs are 1). Moreover,

\[F_{chromo} = \sum_{i \in |chromo|} (1 - chromo_i)^2 + 100 \times (chromo_{i+1} - chromo_i^2)^2\]

\[BestChromo = \arg\min_{chromo}\{F_{chromo}\}\]

5. **0-1 Knapsack**: For a given set of items \(chromo_i\), of weight \(w_i\), maximize the amount of selected items (\(chromo_i = 1\)) such that the total weight does not exceed a maximum \(W\).
Formally,

\[ F_{\text{chromo}} = \sum_{i|\text{chromo}|} \text{chromo}_i \cdot w_i \leq W \]

\[ \text{chromo}_i \in \{0, 1\} \]

\[ \text{BestChromo} = \arg\max_{\text{chromo}} \{F_{\text{chromo}}\} \]

For this problem, 6 items where used, with weights (100, 50, 45, 20, 10, 5) and values (40, 35, 18, 4, 10, 2) respectively.

Although these functions are mostly experimental, they provide the opportunity to test the adaptive capabilities of the proposed strategies. Moreover, the Knapsack problem gives the opportunity to test the strategies on real problems, since it has been recently used on load balancing problems [32] and Intrusion Detection and Prevention Systems [33]. Table 4.1 presents different available choices for each parameter. The mutation and crossover probabilities are not shown in the table because they vary numerically, and from a set selection.

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Choices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crossover Type</td>
<td>1pt, 2pt, uniform</td>
</tr>
<tr>
<td>Crossover Steps</td>
<td>±0.1, ±0.2, ±0.3</td>
</tr>
<tr>
<td>Mutation Steps</td>
<td>±0.001, ±0.01, ±0.1</td>
</tr>
<tr>
<td>Population Size Steps</td>
<td>±10</td>
</tr>
<tr>
<td>Introns</td>
<td>Yes or No</td>
</tr>
<tr>
<td>Ranked Selection</td>
<td>Yes or No</td>
</tr>
</tbody>
</table>
Results and Discussions

During the first experiment, we wanted to test how feasible each strategy is, versus a simple but expensive brute force search. Table 4.2 presents the difference in time requirement per strategy. From such table, it can be noticed that FLC is faster than the OSV, and the OSV is faster than a Full Product. The Full Product is defined as a brute force search over a very large space of possible values. However, the value of the fitness function of the OSV were slightly better than those of the FLC. Such values are not presented here because the interest is to show that both strategies can adapt to the different algorithms.

Table 4.2: Run Time Comparison

<table>
<thead>
<tr>
<th>Problem</th>
<th>Fuzzy</th>
<th>OSV</th>
<th>Full Prod</th>
</tr>
</thead>
<tbody>
<tr>
<td>BestFit</td>
<td>3 mins</td>
<td>13 mins</td>
<td>24 hrs, 11 mins</td>
</tr>
<tr>
<td>OneSum</td>
<td>2.15 mins</td>
<td>8 mins</td>
<td>2 hrs, 46 mins</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>1 mins</td>
<td>4 mins</td>
<td>4 hrs, 46 mins</td>
</tr>
<tr>
<td>Rosenbrock</td>
<td>1.6 mins</td>
<td>5 mins</td>
<td>16 hrs, 32 mins</td>
</tr>
<tr>
<td>Knapsack</td>
<td>3.7 mins</td>
<td>20 mins</td>
<td>31 hrs, 22 mins</td>
</tr>
</tbody>
</table>

Figure 4.3 shows the graphical results for the 3 Meta Tests on the Knapsack problem. The graphical presentation of the results suggest many hypotheses about the importance and nature of the control parameters in determining performance. For instance, the premature convergence towards zero of the mutation rate during the first steps for all 3 tests, and the contemporaneous improvement of the objective function, suggests that the mutation rate is a critical value.
Figure 4.3: The GA is run with 3 sets of parameters: “start small”, “start medium”, and “start high”. Convergence in successful attempts indicate importance and suggested value for given parameters. No convergence represents a parameter with little or no importance. Figures show convergence for mutation rate (a), crossover rate (b), and population size (c).

Similarly, we can see a convergence in the values of crossover rate and population size. In the case of the population size, values around 30 seem to be appropriate; therefore, it is not required to have a larger population size. Uniform crossover with values around 0.8 were selected among the best configurations for many of the test runs.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Variation Step</th>
<th>Probability Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutation Rate</td>
<td>0.0001</td>
<td>(0.55*, 0.02, 0, 0, 0)</td>
</tr>
<tr>
<td>Population Size</td>
<td>±10</td>
<td>(0, 0.58, 0.48*, 0.87, 0.09)</td>
</tr>
<tr>
<td>Cull Rate</td>
<td>0.1</td>
<td>(0.92, 0.54, 0.67*, 0.54, 0.41)</td>
</tr>
<tr>
<td>Introns</td>
<td>±10</td>
<td>(0.84*, 0.18, 0.82, 0.79, 0.31)</td>
</tr>
</tbody>
</table>
Following, we have created an Analysis of Variance (ANOVA) test. ANOVA is a technique for determining whether two sample distributions are truly different, or only different because of the random variation in the samples. The conclusion of an ANOVA test is summarized in a probability value. If the probability value is low then the distributions compared are likely really different. If it is high then the sample variation could come from the random nature of the samples and not reflect any real difference in the underlying distribution parameters. 5% is a standard cutoff point for this type of analysis.

The ANOVA analysis, presented in Table 4.3, supports the conclusions prompted by a visual inspection of the graphs. The star values are referring to the scores of the best configuration. With such an analysis, we can reject the assertion that the distributions determined by parameters from the OSV are the same as those generated by the FLC. This tells us that the population values obtained by each strategy could be different.

A drawback of the OSV procedure is that even if you have convergence you may have just found a local optimum. However you can use insights gained from the OSV analysis to streamline a follow-up Full Product analysis.

In Table 4.4, we compare the OSV method to the FLC for the five optimization problems reviewed. The results are similar in terms of average of all problems. The small values of standard deviation, demonstrates a significant convergence in the value of such parameters. A parameter like the population size and introns number had a larger standard deviation. This can help conclude that such values do not affect much the performance.
Table 4.4: Final Parameter Values (OSV, FLC)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Average Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutation Rate</td>
<td>(0.001 ± 0.00058, 0.01 ± 0.00013)</td>
</tr>
<tr>
<td>Crossover Rate</td>
<td>(0.9 ± 0.11, 0.95 ± 0.002)</td>
</tr>
<tr>
<td>Population Size</td>
<td>(20 ± 5, 30 ± 10)</td>
</tr>
<tr>
<td>Cull Rate</td>
<td>(0.7 ± 0.046, 0.6 ± 0.179)</td>
</tr>
<tr>
<td>Introns</td>
<td>(0 ± 0, 15 ± 3.46)</td>
</tr>
</tbody>
</table>

The following section closes this chapter with some remarks.

Conclusions

Both strategies presented in this chapter provided a way to overcome the stochastic nature of EAs in different domains. The OSV method is a fast way to find good control parameters for a genetic algorithm. The five problems examined in this chapter suggests that the control parameters for the meta-test are not problem specific and thus we avoid an infinite search for meta test on meta test parameters. The results obtained here match closely those obtained by the more complete but time consuming Full Product method. A visual examination of the graphed results provides insight into the importance and range of acceptable values for the control parameters.

The FLC controller reduces further the interaction of having a human analyze the end results to select a best parameter. The Fuzzy approach gives the fastest convergence, and can work real-time with little performance degradation compared to OSV. For future work we would like to have better cognitive decisions on the data, including metrics such as novelty of the individuals, and test such an approach with dynamic objectives. We expect to see the better benefits from using the Fuzzy approach in such scenarios.
CHAPTER 5: ADAPTIVE SMART GRIDS

In recent years, Smart Grids have been a common interest for many consumers, because of their comfort, safety, robustness, and economic characteristics. This chapter presents the development of a computational tool, as an adaptive cognition system for smart grids, having smart homes as their composing nodes. Such a tool has been named Smart Home Energy Aware-Preserver (SHEAP). SHEAP incorporates evolutionary computation algorithms, and communication protocols, to provide users with context awareness and fault tolerance. Moreover, SHEAP considers a smart home powered by solar and wind energy, as a small version of the smart grid. SHEAP demonstrates the benefits of having a smart home that can control the amount of power needed, according to the context of usage. Furthermore, SHEAP includes fault tolerant mechanisms to monitor and react on fault occurrences. Simulation shows that with a smart control of the load, the requirements for a Green Energy System are reduced. An economic analysis of the approach demonstrates the viability of the project reducing the usage of grid energy by utilizing green energy.

Introduction

Recently in September 2011, a power outage occurred in the San Diego area, which according to the National University Institute for Policy Research, had a cost of around $100 million dollars, due to the duration of the outage and the number of users affected by it [35].

Our homes are a common instance of electric loads. In recent years, the integration of renewable energy technologies to our homes has been considered as a possible solution to our increasing

Published on [34] with some extensions made for this chapter
dependence of fossil fuels. Unfortunately, some of the main limitations of the use of renewable energy technologies for residential purposes are economic constraints, limitation in power produced by the green energy sources and fast dynamic balancing of electric loads, among many others.

A residential electric system with renewable energy sources could be considered as a small scale version of a large electric grid. These similarities include generation (e.g., solar panels or wind turbines), transmission (e.g., wiring), distribution (e.g., outlets), and loads (e.g., appliances, pumps, lighting). At the present, the electric grid system has many smart technologies built into it (e.g., advanced metering infrastructure), but the grid is not taking advantage of its full potential. In the case of the residential electric systems, smart technologies are not part of the equation except for some elements (e.g., Maximum Power Point Tracking). In order to obtain the full potential of a renewable residential electric system, a house should have a smart home system. Information, efficient energy use, and load control should be critical elements to develop a smart home system. With such a smart home system, homes can use sensors and actuators to adapt to the environment; thus, optimizing energy usage with the given context accordingly. In this chapter, a smart home system will be referred to as a smart home.

Similar to the smart grid, a smart home should benefit from the present available technologies. Today sensors could be interconnected with the power lines to transport not only energy but also information. The use of computer monitoring and appropriate software tools will integrate and monitor renewable energy sources to the residential electric system, for efficient energy management, using the sensors data.

To develop a complete smart grid, researchers from the energy [36, 37], artificial intelligence [38, 39], and communications communities [40] are joining efforts. The DoE defines several characteristics for a smart grid, among them: load adjustment, resilience, consumer participation, and power sources [41]. Similarly, [42] enumerates self-healing, high reliability, energy management,
and real-time pricing. Based on these results, this work provides a smart grid design and a computational tool called SHEAP (Smart Home Energy-Aware Preserver), integrating the concepts of load balancing, energy advising, and disaster monitoring.

SHEAP is an adaptive cognition system for smart grids, having smart homes as their internal nodes. The system incorporates renewable energy sources using an optimal strategy to achieve higher comfort to the smart grid network (e.g., smart residential electric grid) utilizing an optimal amount of power sources by taking into consideration the economic viability and the proper conservation of energy. SHEAP involves users in the optimization model without changing their life patterns. It accomplishes such goals with two main modules: context awareness and fault tolerance. Context awareness is achieved using sensors attached to smart nodes (i.e., smart homes). Such sensors include microphones, cameras, photo-sensors among others. The sensors are used to reduce the power consumption of loads, depending on the activities performed by users. SHEAP uses an extension of the 2D-BPSO (2 Dimensional Binary Particle Swarm Optimization) algorithm [32, 43], for dynamically balancing the load and automatically adjusting the energy resources in case of any fault occurrence in the energy resources. The results of a Pareto ranking scheme is utilized to automatically find alternative load configurations. SHEAP uses any unused, generated energy, to charge an emergency battery system. Such a system can be activated, to power a certain important factor of the total load, and keep on functioning for an extended period.

The rest of the chapter is organized as follows. We describe the smart home system at a high level in section 5, with details of implementing adaptive environment and central control in sections 5 and 5 respectively. The project cost analysis is presented in section 5. Then, we present the experimental setup for each experiment under consideration in section 5. We present the results and discussion in section 5. Finally in section 5, we close the chapter with some remarks and conclusions.
Smart Homes As Smart Grid Nodes

In this work a smart home interacts with the users and the smart grid (shown as part of sensors). The same idea has been previously considered by others [44]. This concept overcomes the main challenges of smart grids as identified by [38]: correct automation, and the lack of knowledge of users on how to react to the benefits that a smart grid provides (e.g., dynamic pricing).

Figure 5.1: The SHEAP System

Central control obtains information from the context awareness, thus, interacting with both users and the environment (given by the sensors). Such information allows for an effective load balance, which provides fault tolerant configurations, using a Pareto ranking scheme.

To target such challenges SHEAP is proposed as a computational tool (see Figure 5.1). SHEAP creates context awareness from the input of users and sensors in a smart home. Using such information, SHEAP keeps an optimal load balancing configuration by employing a 2D-BPSO algorithm. The load balancing algorithm employed by SHEAP uses a proactive approach to determine optimal loads for each energy source, without needing previous data. Moreover, a Pareto surface of optimal solutions is given, yielding fault tolerance and thus, reducing the probability of failures in the grid. Such fault tolerant approach has been considered in [39], using the probability of static
power outage. In this work, we consider dynamic data, adapting to changes in the environment.

SHEAP also provides load requirement/usage information to the smart grid (i.e., neighboring nodes of the grid). Nodes periodically send a status report message to their neighbors as seen in Figure 5.2, letting know how much energy is been consumed at their location. During normal operations, the system provides the grid with information on how much energy is required at a node and how much of this energy must be provided by the grid. When a power failure occurs, such message is not received. This causes a policy violation that informs neighbors that there was a fault. At this point, neighbors can forward the message to higher levels in the hierarchy (i.e., distribution station, generation plant).

![Figure 5.2: Status Information Header](image)

A small header is sent periodically to each neighbor. The goal is to let know that everything is in order, and how much energy is being used by the node. If a fault occurs, such a message is not delivered, and neighbors can learn about the fault, so that authorities can take the proper action.

In the following sections we describe in further details the components of SHEAP.

Adaptive Environment

In this section we describe how SHEAP adapts to the environment, using environmental information. Such information is given directly by sensors (e.g., photo-resistors, motion, temperature,
etc.) and indirectly from the users activity patterns. Using this information, SHEAP can establish context awareness (subsection 5) to optimize appliance utilization. SHEAP can also advise users on energy efficient alternatives to currently used appliances. We now elaborate on these two components.

**Context Awareness**

Context awareness provides users a better interaction with the grid, which helps them to improve their activity patterns in a natural way (i.e., automatically only use the energy required). Thus, higher comfort and economy is achieved while conserving energy. For example, the photo-sensors of a home could detect that the sunlight is bright enough to reduce the lights intensity, thus reducing the power consumption of users. Similarly, the lights could also be reduced if the home realizes that its users are watching a movie, yielding a comfortable near theater experience, without any action required by the user.

Furthermore, the smart homes can advise users with the most appropriate technology for them, employing cost vs benefit analysis, according to life-time of technology, its value and usage. In the next sub-section, such concept of energy advising is explored further.

**Energy Adviser**

Energy advising consists on having a smart home that can make suggestions to users on how to reduce energy consumption, depending on the habits of each user. The smart home can detect the usage of non-energy efficient technologies. This detection can be done by combining the readings from sensors in the smart home, and the power consumption of loads. For example, the home is aware that an efficient fluorescent light bulb might be consuming $15W$ to produce the same lumen
that an incandescent bulb of 60W would produce, but it realizes that the bulb which is at home consumes 60W, hence, the system can infer that a better bulb can be used. An alarm can then be triggered to notify users of the possible enhancement. Information about such upgrade may come from server technologies such as server subscription, information loaded manually by the user or the smart grid using the status information protocol.

However, a straightforward approach might not work, since the new light bulb might not be feasible for the usage that the users are giving to the bulb, rather SHEAP considers Cost vs Benefit Ratio (CBR), as follows

\[
 CBR(x, y) = \frac{C_x}{C_y} \cdot \frac{ROI_x}{ROI_y} \cdot \frac{L_y}{L_x}
\]

where \( C \) is the initial cost of the alternative in dollars, \( ROI \) is the return of investment or how many months will the user need to wait, before a profit using the alternative can be seen. Finally \( L \) is the estimated lifespan of the alternative; hence, an alternative is acceptable if \( CBR(x, y) < 1 \), which can be achieved by having a small cost, small (i.e., faster) return of investment, and/or long lifespan.

Central Control

The central control is the heart of the smart home system; it receives its input from the adaptive environment. Using such input data, central control balances the usage of energy sources, via the execution of the 2D-BPSO algorithm. After the 2D-BPSO finds optimal and sub-optimal configurations, the central control uses Pareto ranking to rank all the configurations from best to worst. The best configuration is used as the running configuration of the smart home system. Alternate configurations are kept in memory in case of a future fault. Any surplus on generated energy by the green energy sources is stored into rechargeable batteries, to be used in case of an emergency.
Therefore, the central control is logically divided into the following modules: Load Balancing, Fault Tolerance, the 2D-BPSO algorithm, and Emergency Rechargeable Batteries.

Load Balancing

SHEAP employs load balancing to increase the lifetime of energy sources. SHEAP accomplishes this goal by decreasing the chances of a fault due to overload in an energy source. Thus, this chapter defines load balancing as the act of distributing the different loads in the home across the different energy sources. A load may not be connected to more than one type of energy source. SHEAP’s goal is to use the most green energy as it can; therefore, removing grid dependency. Load balancing can be accomplished with context information (to find out the current energy demands), and the 2D-BPSO algorithm. After the 2D-BPSO finds an optimal configuration, the actuators in the home (e.g., relays and dimmers) may tune up the required appliances.

The goal of SHEAP is to minimize the traditional grid dependency, by maximizing the usage of green energy sources (e.g., solar panels and wind turbines) in an adaptive manner, as the load demand changes. For SHEAP, load balancing will be considered as the power needed by the power loads to satisfy the necessary conditions to keep the power network working, without sacrificing the performance of the smart grid. Moreover, SHEAP ensures that there is no power interruption on the loads, while minimizing the use of the power utility, and the expected annual average load energy consumption. Meanwhile, the use of the available Distributed Energy Resources (DER) can be maximized. The use of SHEAP can facilitate the management of the power dispatch by using and optimizing the available DER, and by minimizing the loads’ dependence on grid energy (if it is economically feasible).

In this work, SHEAP is using solar energy (photo-voltaic power system) and wind energy (wind turbine power system) as DER. SHEAP has the capacity to incorporate other type of DER, in-
including thermoelectric energy (TEG generators), wave energy (for systems with the infrastructure and close to the coast), electrochemical energy (fuel cells), etc. However, it is critical to provide to SHEAP the appropriate energy patterns and load consumption to obtain realistic and useful results.

SHEAP’s priority is to help the customer minimize the average energy consumption, along with the initial investment needed (e.g., reduction of sensors and distributed energy resources materials), such that the return of investment can be obtained in the shortest amount of time. As a side effect, SHEAP can cause a reduction in the generation of energy in the grid, thus, lowering the probability of a power outage, which can be very expensive (e.g., around $100 million [35]).

**Fault Tolerance**

Fault tolerance is vital for future smart grids, to provide self healing, and/or disaster detection and preparations. With fault tolerance, a smart grid can react in the event of some disaster. Its reaction could include a switch to use an emergency power source, change its current configuration to adapt on the fault, or notify its neighbors to let know of its current status (Figure 5.2).

To develop fault tolerance configuration in central control, a Pareto ranking scheme, shown in Algorithm 4, is used to determine an optimal solution surface. This optimal solution surface is developed with the help of the 2D-BPSO algorithm.

```
Algorithm 4 Configuration Ranking Scheme
1: Start with a set C as a copy of a population containing the individual bests of each solution
2: rank ← 0
3: Remove invalid solutions from C
4: while There are c unranked do
5:    for Each Individual Best c in C do
6:        Assign rank to those c which are not surpassed by any of the solutions with no ranking assigned
7:        rank ← rank + 1
8:    end for
9: end while
```
A configuration $c_1$ is Pareto optimum to $c_2$, if its fitness function is for at least one of the components in $c_1$ better than $c_2$, and is still within the acceptable parameter range. At the end of the ranking scheme, the solutions with rank 0 are the best solutions, followed by those with rank 1, and so forth. When a fault in the running configuration occurs, the SHEAP goes to the set of ranked scheme, to search for a new usable solution with the lowest possible rank (i.e., the best solution).

The 2D-BPSO

This sub-section discusses the 2D-BPSO, an algorithm based on the Particle Swarm Optimization (PSO) [45], designed for balancing the load requirement on the different sources in the smart home. There are several advantages on using the PSO as core for SHEAP’s load balancing:

1. implicit parallelism shared by all evolutionary computation algorithms, which help to find solutions faster
2. by moving particles across the sample space, instead of replacing them in each generation, it is guaranteed that the number of valid solutions will be higher
3. the particles in the population do not require to be reset, making the algorithm well suited for adaptive processing, since the search can resume when needed and from the point it left out (i.e., last best known configuration)
4. the algorithm may provide us with Pareto surfaces without much modification by using the individual best values in its structure, thus, yielding fault tolerance

Following the idea of software algorithms, it is possible to formulate an evolutionary approach to solve the smart grid load balancing problem, by solving it first in its individual nodes (i.e.,
the smart homes). Evolutionary approaches are also characterized by their ability to handle non-differentiable and non-linear multi-modal functions. All evolutionary approaches consist of the following main operations:

- **Reproduction or crossover**: aims to construct new offspring from current particles in the parent population

- **Mutation**: adds diversity to the population and affects global search capability
  
  - represented by random variation

- **Selection**: updates/removes particles from population
  
  - affects exploitation search capability
  
  - strong selection enhances exploitation search, but may arrive to sub-optimum solutions

Moreover, the different loads in the smart homes can be defined as

\[
A = \{a_1, a_2, a_3, ..., a_i\},
\]

where each load consumes a total of \(w_i\) watts. Such a load may refer to an appliance [46] or at larger scale a home [40], and it may be connected to a green source and/or the traditional grid energy. Each source has \(W_k\) of total supplied watts, such a number varies with respect to the location of the load, on the planet. It is considered that the grid can satisfy any load requirement, but at high cost. Using this definition, the load balancing problem can be seen as a combinatorial problem, with fitness function of

\[
F_{SOURCE} = \sum_{i \in |A|} a_i \cdot w_i < W_k, \quad a_i \in \{0, 1\} \quad (5.2)
\]

where \(F_{SOURCE}\) is the total power demanded from the source under consideration (i.e., grid, solar, wind). The value of \(a_i\) is 1 when the load is selected for the source under consideration, and 0 otherwise. An optimal configuration is found by distributing the load across sources, in such a
way that the demand in green sources is maximized, therefore the traditional grid dependency is minimized.
Algorithm 5 2D-BPSO Algorithm

1: Init population of size P randomly
2: Init particles velocities to 0
3: Init individual best to current population
4: Init global best to \( \min_{\text{particle}} \{ F(\text{particle}) \} \)
5: Set individual influence constant \( c_1 \) (e.g., 2)
6: Set global influence constant \( c_2 \) (e.g., 2)
7: Set \( \alpha \) inertia factor to a desired step size (e.g., 0.9)
8: while Generations Remain AND Value not reached do
9:     for Each Particle \( P \) do
10:        \( v \leftarrow \alpha \cdot v + c_1 \cdot \text{rand}_1 \cdot (\text{individual}_\text{best} - \text{particle}) + c_2 \cdot \text{rand}_2 \cdot (\text{global}_\text{best} - \text{particle}) \)
11:          if \( v > 4 \) then
12:              \( v \leftarrow 4 \)
13:          end if
14:          if \( v < -4 \) then
15:              \( v \leftarrow -4 \)
16:          end if
17:     for Each bit do
18:         if \( \text{rand} < \frac{1}{1 + e^{-v}} \) then
19:             \( \text{bit} \leftarrow 1 \)
20:         else
21:             \( \text{bit} \leftarrow 0 \)
22:         end if
23:     end for
24:     if \( P_1 \& P_2 \ NOT \ 0 \) then
25:         Generate random integer \( \text{index} \in \mathbb{Z}_3 \)
26:         if \( \text{index} = 0 \) then
27:             \( P_2 \leftarrow P_2 \land 1 \)
28:         else if \( \text{index} = 1 \) then
29:             \( P_1 \leftarrow P_1 \land 1 \)
30:         else
31:             \( P_1 \leftarrow P_1 \land 1 \)
32:             \( P_2 \leftarrow P_2 \land 1 \)
33:         end if
34:     end if
35:     Evaluate particle
36:     if \( F(\text{particle}) < F(\text{individual}_\text{best}) \) then
37:         \( \text{individual}_\text{best} \leftarrow \text{particle} \)
38:     end if
39:     if \( F(\text{particle}) < F(\text{global}_\text{best}) \) then
40:         \( \text{global}_\text{best} \leftarrow \text{particle} \)
41:     end if
42: end for
43: end while
Global optimization problem is defined as:

- Given $f : \mathbb{R}^n \rightarrow \mathbb{R}$
- Find $x^* \in \mathbb{R}^n$ for which $f(x^*) \geq f(x)$

The 2D-BPSO algorithm presented in Algorithm 5, is an extension of the original 2D-BPSO algorithm defined in [32, 43]. For completeness, we explain the key aspects of the algorithm. The algorithm starts with a population of possible solutions all randomly generated. During each generation, the algorithm computes the fitness of each solution in the population. Then as presented in line 7, the velocity of each particle is updated properly, according to an inertia weight $\alpha$, the individual knowledge of each particle, and the global knowledge (i.e., best solution known by all particles). The value of $\alpha$ can help to converge; the algorithm starts with a maximum value and linearly decreases the value to a minimum on each generation. Such decrease causes the search for solution to explore more at the beginning, and refine the search to exploit the local discoveries, at later stages of the evolution.

The algorithm contains $N$ populations of solutions, one for each green energy source (e.g., solar panels, wind energy, natural gas, etc.). With these populations, it becomes possible to achieve mutual exclusion of loads, that is, no load should be connected to more than one energy source at any particular time. Such mutual exclusion, can be performed using simple logical operations, like $AND$ and $XOR$ in lines 17-22. The performance (fitness) of solutions are evaluated on each energy source, and then added together such that the total benefit of the solution can be compared among the possible solutions. The smart home system can then switch the energy source use per appliance given the 2D-BPSO solution.
Emergency Rechargeable Batteries

Emergency batteries can help the smart home continue working locally with small, but important loads such as the refrigerator, smoke detectors, lights, or the air conditioner during heat waves. By keeping such loads operational the smart home can prevent the occurrence of events like loosing all the food in the refrigerator, not detecting a fire during a disaster, prevent a safe passage out of a house that is on fire or users passing out during heat waves because of the high temperatures.

In SHEAP, batteries are only used for emergencies, to avoid high maintenance costs when the batteries reach the end of their lifetime. Batteries are charged using the excess energy (i.e., generated energy that is unused by loads) generated by the green sources. Depending on the amount of load demand by the system, these emergency batteries may take too much time to charge, hence, if the users want to guarantee the batteries charge on time, then the total load requirement should be increased by some factor (e.g., 20%).

Several types of batteries exist, performing better under different circumstances, as follows:

- **Regular (Car):** regular batteries are not suited for smart homes with high load, since regular batteries loose their lifetime when subject to many charge/discharge cycles. Regular batteries can perform well only with low loads.

- **Flooded:** flood batteries are only recommended for outside, because they emit gas when charging. If an enclose is used with this type of batteries, it should have a vent to release the gas.

- **Gel:** gel batteries are suited for indoor use, since they do not release any gas when charging. Hence, venting them on an enclosure is not needed, and this causes gel batteries to work under cooler temperatures. Thus, they can perform better.
Absorbed Glass Mat (AGM): although AGM batteries are the most expensive type; they possess longer lifetime and slower discharge time, than other types. Like gel batteries, AGM batteries do not emit gas when charging, nor they leak chemicals.

Once selected, batteries can then be connected together to obtain higher capacities, depending on the configuration. That is, batteries in series can help increase the voltage of the system; similarly, battery blocks in parallel can help increase the total current supplied by the system. The capacity of the emergency system can then be derived from [47] as

\[
\lambda_{EM} = \frac{P_{SYS}}{Ld} = \frac{(V_{battery} \cdot NB_{series})(I_{battery} \cdot NB_{parallel})}{Ld}
\]

where \( P_{SYS} \) is the total power supplied by the battery bank, during emergencies, to a load of specific Watts per hour given by \( Ld \). \( V_{battery} \) and \( I_{battery} \) are the voltage and current capacities of a single battery, in Volts and Amperes-per-Hour respectively, which creates a complete emergency system, with identical \( (NB_{series} + NB_{parallel}) \) batteries connected in series and parallel respectively. Hence, \( \lambda_{EM} \) is the amount of time in hours, that the emergency system can keep \( Ld \) watts per hour running, while the primary sources are restored.

Cost Analysis

This section considers the cost of the proposed system. Rather than defining the cost in terms of the initial cost, it is more convenient to consider the Return of Investment (ROI) of the project. The ROI can be formally defined as

\[
Cost_{traditional}(t_m) - Cost_{project}(t_m) > 0
\]
where \( t_m \) represents the time in years, at which continuing using the proposed system would be more cost efficient, than keeping the traditional system. Hence, the goal is to have a \( t_m \) as small as possible.

The cost of the proposed project can be mathematically expressed as

\[
Cost_{project}(t) = \int_0^t c(t)dt = \int_0^t (\eta(t) \cdot c_{traditional}(t) + C_\eta(t))dt + C_i
\]

where \( \eta \) is a cognition factor between 0 and 1, which models the reduction of the amount of energy needed from the traditional grid. Therefore, the cost can be minimized by reducing the grid dependency with green energy, and/or using the cognition \( \eta \) of smart homes. Moreover, whenever \( \eta \) is 1 means that no cognition can be used, which occurs with fix costs such as the minimal monthly cost for staying connected to the grid. \( C_\eta(t) \) represents the cost for running the smart home; in case of a traditional and green home, this cost is 0. In general \( C_\eta(t) \), should be considerably small, because the devices used for such cognition consume less than 1W/hr. Furthermore, \( C_i \) represents the installation cost of the project. Such a cost is only 0 (or neglected) for the traditional power grid, but may be lower in the case on which smart homes are used.

Experimental Setup

For our experiments, we have assumed virtual houses around the globe. We have utilized the data given in [48] for the length of the day or time of exposure to sun light, and average wind speed for the selected virtual house based on the location of the house (i.e., latitude and longitude). Based on the information given in [49], we have also assumed a 15% average loss of rated energy obtained...
from the solar panels. We have assumed the cost of energy utilized as charged by the Progress Energy Company in Orlando, Florida.

For validating SHEAP’s performance, in terms of the load balancing algorithms ability to minimize grid dependency, and charge the emergency batteries, we have generated a stochastic process, in a Markov Decision Process manner, to simulate the activities that users may interact with the smart home. Such a process builds a sequence of events over a year, as seen in Figure 5.3. The stochastic process’s events, in general, are tuples of the form \((A, D, U, \text{timestamp})\), or more formally

\[
A = \{\text{WORK, COOK, EAT, READ, WATCH, SLEEP, TRAVEL, VACATION}\}
\]

\[D \in R\]
\[U = \{\text{DAYS, HRS, MINS}\}\]

where \(A\) is an action with duration \(D\), whose units are \(U\) and occurs at \(\text{timestamp}\) time.

\[
\text{WORK} \quad 12 \quad \text{HRS} \quad 08/26/2010 \quad 21:00 \\
\text{COOK} \quad 32 \quad \text{MINS} \quad 08/26/2010 \quad 21:32 \\
\text{EAT} \quad 25 \quad \text{MINS} \quad 08/26/2010 \quad 21:57 \\
\text{READ} \quad 107 \quad \text{MINS} \quad 08/26/2010 \quad 23:44 \\
\text{WATCH} \quad 201 \quad \text{MINS} \quad 08/27/2010 \quad 03:05 \\
\text{SLEEP} \quad 5.92 \quad \text{HRS} \quad 08/27/2010 \quad 09:00
\]

\text{Figure 5.3: Example of Sequence of Events}

Events flow from \(E_i\) to \(E_j\) following a transition probability \(p_{ij}\), where \(i\) represents the current event index and \(j\) represents the next event index. The events are not memoryless, since the patterns do not repeat an action on the same day, and the probabilities lower as time passes, to enforce an action change.
Using a Java implementation of SHEAP, the generated scenarios are then executed 10 times each to find an average performance of energy usage for each selected virtual house. We discuss the results of our experiments in the following section.

Results and Discussion

During the first set of data in the experiment, we analyzed the performance of a traditional home, powered only by the energy from the traditional grid. In a second scenario, the home was equipped with green energy; a third scenario introduced cognition as well. The comparison is shown in Figure 5.4, where the green house can effectively reduce the grid dependency. In addition, SHEAP allows further reduction using its cognition. There is not much difference between the solar energy in the green home and SHEAP. Such a similarity was due to the lack of reduction in the devices connected to the solar system. In general, the solar system was able to produce higher energy values than the wind system. Hence it is appropriate for loads with high energy demand. The general case of SHEAP, for different cities across the planet can be seen in Table 5.1, where the location of the house affects the DER units capacity. Thus, SHEAP is able to adjust accordingly.
Figure 5.4: Consumption for Different House Types

Each home project contains different traditional grid dependence. However, the smart node approach presents a reduction in overall power consumption. This reduction depends on the location of the house.

Table 5.1: Average Consumption by a Grid Node in kW.

<table>
<thead>
<tr>
<th>Source</th>
<th>Mayaguez, PR</th>
<th>Orlando, FL</th>
<th>Tokyo, JP</th>
<th>Sacramento, CA</th>
<th>New York, NY</th>
<th>Toronto, ON</th>
<th>Vancouver, BC</th>
<th>Montreal, QC</th>
<th>Fairbanks, AK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid</td>
<td>3817</td>
<td>4103</td>
<td>4068</td>
<td>3988</td>
<td>3763</td>
<td>4652</td>
<td>3908</td>
<td>3915</td>
<td>4154</td>
</tr>
<tr>
<td>Solar</td>
<td>3860</td>
<td>3896</td>
<td>3923</td>
<td>3931</td>
<td>3947</td>
<td>3987</td>
<td>3975</td>
<td>4152</td>
<td></td>
</tr>
<tr>
<td>Wind</td>
<td>1748</td>
<td>1427</td>
<td>1435</td>
<td>1506</td>
<td>1716</td>
<td>788</td>
<td>1543</td>
<td>1120</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source</th>
<th>Beijing, CN</th>
<th>Madrid, ES</th>
<th>Paris, FR</th>
<th>London, UK</th>
<th>Sydney, AU</th>
<th>Kuala, MY</th>
<th>Rome, IT</th>
<th>Brasilia, BR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid</td>
<td>3973</td>
<td>4518</td>
<td>4438</td>
<td>3831</td>
<td>4795</td>
<td>4751</td>
<td>3925</td>
<td>3205</td>
</tr>
<tr>
<td>Solar</td>
<td>3945</td>
<td>3940</td>
<td>3977</td>
<td>3988</td>
<td>3755</td>
<td>3829</td>
<td>3934</td>
<td>3955</td>
</tr>
<tr>
<td>Wind</td>
<td>1302</td>
<td>968</td>
<td>1011</td>
<td>1607</td>
<td>876</td>
<td>846</td>
<td>1547</td>
<td>1465</td>
</tr>
</tbody>
</table>

The second set of data in the experiment dealt with calculation of fault tolerant configurations. The results of the Pareto ranking scheme are shown in Figure 5.5. The solutions in the highest peaks (highest fitness), are all Pareto optimum with rank 0, and provide fault tolerance to the smart grid. Only few peaks provided the same rank 0 solution. Such effect is mainly because of the energy sources limits. When such peaks fail, SHEAP has to select a rank 1 solution, which are given by the same region but with little changes. Similarly, rank 2 solutions belong to a different region of the search space.
Figure 5.5: Valid Optimal Configurations

Figure shows the effect of limiting the green sources, because of batteries, or lack of energy. Pareto surface got smaller, but the presence of peaks at the same level provides the smart grid, with fault tolerant alternative configurations. In figure, there are 4 peaks that provide around the same grid dependency reduction.

Figure 5.6: Effect of Connected Loads on Battery Configurations

Some loads are more energy demanding than others; Others, such as the AC, have high energy demand, but may be optional.
Continuing with the fault tolerance enhancements, we tested SHEAP’s ability to charge the emergency battery systems. The charging time in hours was determined for different battery types (see Figure 5.7). The term $nSmp$ (e.g., $4S4P$) denotes the system configuration represented as series blocks with $n$ batteries, connected in parallel with $m$ identical blocks. Using such charging time, SHEAP can help select the proper configuration and battery, depending on what type of emergency configuration is desired. As seen in Figure 5.6, the amount of time that a battery configuration will provide energy to essential appliances, varies with the appliance types. With the refrigerator being the most essential load, it can be seen that the lights demand a relatively small amount of energy, when compared to the AC. Moreover, the AC load suggests that a separate emergency system would be more appropriate for powering the AC, if desired. Furthermore, different types of battery configurations suggest that more batteries in parallel provide better performance. Such suggestion can be seen from the increase of lifetime, which tends to be higher as the number of parallel blocks increase. There are cases where a configuration of batteries in parallel is only dominated by the same capacity batteries, with more batteries in series (e.g., $2S4P$ to $4S4P$ in Figure 5.6). Hence, if the charging time is longer than the discharge time, the battery configuration is not appropriate for the desired load. The battery type or load should then be changed, in order to meet the requirements (charging time smaller than load duration).
Figure 5.7: Average Charging Time for 12V AGM Batteries

SHEAP can charge most batteries configurations within less than a day. At most, 36 hours were required to charge a 4S4P emergency system.

The final set of data can help with cost analysis. Given the initial costs of each project (i.e., traditional, green, SHEAP), a projection in time is calculated, to find the return of investment (ROI) point. In Figure 5.8, it can be seen that due to SHEAP’s cognition, the slope of the smart nodes is smaller. This causes a quicker ROI (before 3 years), in contrast with the green home project (7 years). Hence, the SHEAP project is viable for any type of user, since in few years profit can be obtained.
In this chapter we have presented SHEAP, an adaptive cognition system for smart grids, using smart homes as micro-units of the grid, with context awareness and fault tolerance. SHEAP has been realized using approaches from power and energy, artificial intelligence, and communication fields. SHEAP’s design is simple to implement, and results show that good performance can be obtained, without the need of complex approximations or long training periods.

Load balancing and the introduction of the factor $\eta(t)$ (i.e., cognition), allowed to reduce the required amount of materials for powering the loads. Because SHEAP did not employ the batteries for daily use, total grid independence was not achieved. However, the batteries’ life is preserved, reducing maintenance costs for replacement in long periods. By enhancing the 2D-BPSO with Pareto ranking, it was possible to find fault tolerant configurations, rather than an optimal single
solution. Furthermore, context awareness can help on providing energy reduction.

The next step in our research is to extend the deployment of SHEAP in virtual random communities for a grid network around the globe.
CHAPTER 6: ADAPTIVE AND DISTRIBUTED SECURE ANOMALY DETECTION

Over the last decade, cyber security has become of great importance, since attacks are getting more sophisticated and easy to execute. Through time, it has alarmingly become easier for attackers to steal information at the comfort of their homes, without putting their lives at risk. Hence, intruders in a network must be identified and counter actions must be taken. Failing to do so may result in negative consequences, including identity theft, stealing of credit card numbers, and unavailability of services, among others.

Therefore as a counter measure, intrusion detection systems (IDS) have been proposed to identify malicious users. Moreover, intrusion prevention systems (IPS) take such detection one step further, by taking counter measures, which may block the malicious users from performing their attack. However, intrusion detection/prevention systems (IDPS) face two main challenges: 1) requirement of large amount of data and processing in order to classify behaviors correctly, and 2) the overlapping of actions on masked intruders' behavior. It is because of such challenges, that an IDPS system may incorrectly classify users (or intruders), thus obtaining an undesired outcome.

The problem of Intrusion Detection has been widely studied. Probably the most popular approaches involve data mining tools such as Support Vector Machines (SVMs) [50]. Another recently studied approach is the use of evolutionary computation to evolve optimal classifiers; [51] works with evolving a Fuzzy Logic classifier that can increase the accuracy on difficult classifications. Although some of the previous approaches manage to obtain good results and feature space reduction [52]. Our main interest is to identify intruders correctly by cooperation among neighboring network nodes, and reducing the required dataset by each network node.

Published on [33] with some extensions made for this chapter
This paper presents a distributed cooperative algorithm to detect intruders using a small random set of local data for training purposes. Using a Binary Particle Swarm Optimization (BPSO) algorithm, the best features for classification are selected by each network node locally. Based on such features, a user is identified as a normal user or an intruder with a certain probability. Whenever such probabilistic measures are not very definitive, the result is classified as ambiguous. This ambiguity is resolved by consulting neighboring nodes through a voting mechanism.

Classification in the system is handled by each node using a Probabilistic Neural Network (PNN). The PNN compares entries against the distribution of what is believed to be an intruder and what is not. The class which provides a higher probability confidence becomes the classification for the specific node. The implementation of such network has been enhanced for improved adaptation from one node into the other. The mixture of BPSO with PNN make the system more tolerable to noise in the data, and results show how the system is able to correctly classify in most cases, with almost no false alarms.

The rest of the paper is organized as follows. Section 6 describes the characteristics and challenges of intrusion detection. Following, section 6 presents alternatives to selecting the best set of features for classification, as well as examples of common features in intrusion detection. Later in section 6, the proposed system is described and its advantages are also presented. Section 6 presents the experimental setup, along with the results and discussion in section 6. Finally, previous work that motivated this research is presented in section 6, and section 6 concludes the paper.

Intrusion Prevention Systems

Intrusion prevention systems have capabilities to detect an intruder, and block such intruder from accessing the system where it was detected. Detection of the intruders may be done using two pri-
mary techniques: signature and statistical based approaches. Signature based intrusion detection rely on common attack signatures, which are stored inside a database, and are consulted to be compared against incoming packets. Such intrusion detection requires noiseless data and continuous signature updates to achieve improved detection accuracy.

Meanwhile, statistical based detection employs analysis of typical user behavior and attempts to find anomalies in traffic; such anomalies may correspond to intruders in the network. Problems with such approach arise from the fact that typically there is a lot of data of valid users, while at the same time, little is known about intruders’ data. Furthermore, as Figure 6.1 shows, there might be areas where legitimate users’ behavior overlaps with intruders’ behavior.

![Figure 6.1: Overlapping of User Behaviors](image)

An IDPS biggest challenge is dealing with false positive (a legitimate user classified as malicious) and false negatives (a malicious user classified as legitimate). Such problem occurs because of outliers which create an overlapping region between the behavior of both types of users.

The next section describes common characteristics that may serve to classify between an intruder and non-intruder.
Feature Extraction in IDPS

A good feature extraction mechanism is essential in all classification problems. Extracting features reduces the dimensionality of data, thus, providing an easier classification. The better the quality of the features extracted, the better the accuracy of the classification system is obtained; this is because a clearer separation between classes is achieved. Different strategies exist, for gathering the best features: empirical [53], Principal Component Analysis [54], and Evolutionary Computation algorithms [55, 56]. Empirical analysis is easy to perform, but requires multiple tests, which have to be analyzed by an expert that can decide which are the best features to be used, based on current data. The problem with employing current data is that the data might change in the future. Principal Component Analysis (PCA) automates such process by using a set of equations that can help discriminate features, and select the ones that provide the most information; however, PCA is highly affected by noise in the data [56].

The last strategy uses an evolutionary algorithm (EA), such as the Binary Particle Swarm Optimization, to search for the best feature set. EAs have the characteristic of being less affected by noise, as well as being implicitly parallel, allowing them to search multiple possible solutions at the same time. This work benefits from this nature, in order to run a distributed BPSO algorithm across all intrusion detection nodes in the network. Typical features for intrusion detection include:

- # of IP Addresses in the network
- Avg. interval between packets
- # of protocols in the network
- # of protocols used by a single host
- Type of protocol requested
• # of packets with 0 length size

• Avg. data length

• # of non-ASCII characters in data

• Time interval of packets

• Avg. error rate

The next section discusses the proposed methodology and its advantages.

Methodology

This section describes the methodology proposed for an improved classification of intruders and non-intruders, taking advantage of the network capabilities to communicate among neighbors. The system is composed of three components (Figure 6.2): features discrimination, behavior classification based on selected features, and taking action whenever an intruder is detected.
The service network scans the environment, where there may be users and/or intruders. To correctly detect who is an intruder, the system consults with the adaptive IDPS, which replies with an action to take. Such action may be to do nothing, if the user is legitimate.

**Feature Selection**

Choosing the best features, given a large amount of data, can be a challenging task. Instead of obtaining empirical results, the proposed system employs a Binary Particle Swarm Optimization (BPSO) approach. The BPSO has been utilized in the past for determining the best features to be used for classification problems [55]. In previous work, the BPSO has been proven to be less affected by noise; therefore, the BPSO can find features which better represent the data. The goal
of the algorithm is to maximize the inter-class variance given by:

\[ F = \sqrt{\sum_{i=1}^{L} (M_i - M_o)^T (M_i - M_o)} \]

where \( M_i \) represents the mean for each class of features, and \( M_o \) the overall mean, considering a number of \( L \) distinct classes (in this work 2, intruder or not).

Moreover, the BPSO finds a binary mask \( T \), which can be applied as a point-by-point product between the bits in the mask and the elements of the feature vector. To find such mask, the BPSO starts with a collection of randomly initialized masks, known as particles. These particles have a velocity associated with each one, which determine how fast they explore the sample space on each generation. During each generation and for each particle, the particle velocity is updated according to some inertia factor \( \alpha \), which prevents drastic changes in the velocity. Particles also consider their past individual knowledge and the knowledge gained by other particles. The importance that each particle gives to such knowledge can be set with some influence constants \( c_1 \) and \( c_2 \) respectively. Once the velocities are updated, each particle will set a bit as 1 (select the corresponding feature) or 0 (neglect the corresponding feature), following a sigmoid function of the particle velocity. To ensure that the solutions become stable, the velocities should be kept between a finite interval \([-d, d]\). After some empirical trials we have found that \( d = 4 \) is an adequate value for such interval. Otherwise, values smaller than \(-4\) would prevent a bit from being set to 1, while values larger than \( 4 \) would prevent bits from being set to 0. Finally, the updated particles are compared in fitness (inter-class variance) and replace old individual/global best if their fitness is higher. The complete detailed procedure can be seen in algorithm 6.

In the proposed system, each node in the network runs a local BPSO, with its own local data. Thus, the amount of data required is less, and nodes can communicate with neighbors whenever
Algorithm 6 The Binary Particle Swarm Optimization

1: Init population of size P randomly
2: Init particles velocities \( v \) to 0
3: Init individual best to current population
4: Init global best to \( \arg\max_{\text{particle}} \{F(\text{particle})\} \)
5: Set individual influence constant \( c_1 \) (e.g., 2)
6: Set global influence constant \( c_2 \) (e.g., 2)
7: Set \( \alpha \) inertia factor to a desired step size (e.g., 0.9)
8: while Generations Remain AND Value not reached do
9:     for Each Particle \( P \) do
10:         \( v \leftarrow \alpha \cdot v + c_1 \cdot \text{rand}_1 \cdot (\text{individual}_\text{best} - \text{particle}) + c_2 \cdot \text{rand}_2 \cdot (\text{global}_\text{best} - \text{particle}) \)
11:         if \( v > 4 \) then
12:             \( v \leftarrow 4 \)
13:         end if
14:         if \( v < -4 \) then
15:             \( v \leftarrow -4 \)
16:         end if
17:         for Each bit do
18:             if \( \text{rand} < \frac{1}{1 + e^{-v}} \) then
19:                 bit \( \leftarrow 1 \)
20:             else
21:                 bit \( \leftarrow 0 \)
22:             end if
23:         end for
24:         Evaluate particle
25:         if \( F(\text{particle}) > F(\text{individual}_\text{best}) \) then
26:             \( \text{individual}_\text{best} \leftarrow \text{particle} \)
27:         end if
28:         if \( F(\text{particle}) > F(\text{global}_\text{best}) \) then
29:             \( \text{global}_\text{best} \leftarrow \text{particle} \)
30:         end if
31:     end for
32: end while

High uncertainty about a decision is present. Such a process will be better described in the next sub-section, when the Probabilistic Neural Network classifier is discussed.
Behavior Classification

Classifying the behavior of users (or intruders) requires some confidence or probability, that tells the node how much is known about a packet. The Probabilistic Neural Network (PNN) becomes a good classifier, using the best features selected by the BPSO, which filters out some or most of the possible noise in the data.

PNNs create statistical data, mean $\mu$ and standard deviation $\sigma$, for each sample features. Such statistical data is used to compute the euclidean distance between input feature vector and the sample feature vector. The distance value is used as argument for a radial basis function that describes how related is the input vector to the samples (i.e., intruder/non-intruder). The total probability for each class (i.e., intruder/non-intruder) is computed at the end and the larger of these two values determines the classification.

PNNs can prune unneeded comparison samples, making the classification process faster. Since the PNN uses a statistical approach to classify its subjects, it is easy to update the mean $\mu$ and standard deviation $\sigma$ of each feature, such that the amount of times revisiting the data is reduced, as follows:

$$
\mu_{new} = \frac{N_{old} \cdot \mu_{old} + newSample}{N_{new}}
$$

$$
\sigma_{new} = \sqrt{\frac{\sum_i (x_i - \mu_{new})^2}{N_{new}}}
$$

where $N_{old}$ is the number of samples before the new observed feature vector value $newSample$ is considered, $N_{new}$ is the amount of samples in the new dataset (i.e., $N_{new} = N_{old} + 1$).

Upon encountering high ambiguity in the classification (probabilities close to 0.5), nodes follow
the procedure shown in figure 6.3, to find a better classification, based on the results from neighbors classifications. Once the neighbors cast their votes, the classifying node counts the number of votes per class. The class that gets the most number of votes is the resulting classification. Such decision involves a learning phase in the analyzing node, to adapt to the previously ambiguous packet.

![Figure 6.3: IDPS Disambiguation](image)

Upon encountering a classification with high ambiguity, the system proceeds to consult its neighbors, which reply back with their vote; such vote is represented by the probability of the entity being an intruder. Once their reply is received, the system adapts to the situation by creating a new sample, based on the features of the ambiguity case, and the result of the majority vote as target classification.

**Types of Actions**

Once an intrusion is detected, IDPS can take various actions to protect the network and its users’ from malicious attacks. Such actions are:
• Log: record the intrusion event, along with all data that can identify the intruder

• Alert: alert others (administrator and neighbors) by sending email and/or packet to other nodes in the network, such that they may limit or block access of the intruder

• Limit: limit the bandwidth of the intruder, in case there is the possibility that is a legitimate user, who is incorrectly utilizing the network

• Block: block all connections from the intruder

It is important to notice that more than one of such actions can be applied to the same event, depending on the severity of the event. The next section discusses the experimental setup to test the performance of the proposed IDPS.

Experimental Setup

This section describes the experimental setup for testing the performance of the proposed IDPS. The BPSO and PNN algorithms were implemented in Java. For training and testing data, the dataset from the third International Knowledge Discovery and Data Mining Tools Competition [57] was used.

Such data has been used for IDS benchmarks, dealing with data mining approaches [50], feature extraction [52], and evolutionary computation [51]. The data contains both normal users and common network attacks in a simulated military environment. Among such attacks are: rootkits, smurf, buffer overflow, and others. The features selected were limited to those with continuous values, for a total of 34 features. Features with symbolic values, such as the service and protocol type, were discarded. Nodes were trained by dividing at random the amount of the total data among them (i.e., simulated nodes). Around one fifth (20%) of the assigned was used for testing purposes.
Two experiments were used for comparison. In the first experiment, the data was processed using by only node using the BPSO feature selection and PNN classifier. In the second experiment, the training was done locally by 5 IDPS, using distributed local data. In this experiment, during testing, the same node from the first experiment consulted the additional IDPS nodes when ambiguous cases were found. During such cases, the additional IDPS nodes issued their classification vote, and the classification that earned the most votes won.

The results obtained and presented in the next section, were studied at the central classifier (i.e., central node), using average performance of 10 trained classifiers (i.e., 10 runs). Neighbors (the other 4 nodes) were only consulted for classification upon finding ambiguity. The experiment is done with 5 running processes, so no network communication overhead is studied at this point. All nodes were trained with 1000 random entries from the dataset, labeled both as normal or some type of attack (e.g., rootkit, spy, nmap, smurf, buffer overflow, ipsweep, etc.). Different types attacks, were considered as an intruder regardless of the attack type. For testing, a random number of testing samples was generated, centered each run at 200 samples. The features contained in the samples included: source bytes, destination bytes, number of failed logins, number of created and access files, error rates, among others.

Next section discusses the results obtained by the system.

Results and Discussion

In this section, we first describe the results without the majority vote.

Table 6.1 presents the confusion matrix for the no majority casting vote. The Ambiguous column states the percent of ambiguous cases, where a user acted as an intruder (2%), or an intruder as acted as user (40%). Out of such ambiguous cases, the system sometimes made a correct classification,
sometimes not. Such impact may be seen in the User and Intruder columns, which give the total rate per scenario. Although the accuracy was not too high, it was necessary to try out different set of features from the beginning. Hence, it was vital to observe what features the BPSO filter selected, which allowed correct classification of the ambiguous cases. Furthermore, the system had no problem identifying the normal users, where almost a perfect score was obtained. The challenge arose with the intrusion detection scores, which included a large amount of ambiguous classifications that resulted in a relatively low accuracy.

<table>
<thead>
<tr>
<th>Actual/Classification</th>
<th>User</th>
<th>Intruder</th>
<th>Ambiguous</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>0.97</td>
<td>0.03</td>
<td>0.02</td>
</tr>
<tr>
<td>Intruder</td>
<td>0.18</td>
<td>0.82</td>
<td>0.4</td>
</tr>
</tbody>
</table>

**Detection Rate** 0.82  **False Alarm Rate** 0.03

In the second experiment, majority vote was added for improving the ambiguous classifications. The results of adding the majority vote to the same IDPS are presented in Table 6.2. The ambiguous cases are not shown because the two runs were seeded equally. Such configuration makes the ambiguous cases be the same as the run without majority vote. However, with majority vote, the accuracy of the IDPS was increased. It is to be noted that an odd number of neighboring nodes avoids a tied result.

We have also traced the correct classification rate in both experiments. This trace value is shown in Figure 6.4. The blue and red traces are for the first and second experiment respectively. A point to noted that first traces were all normal users, having correct classification. The trace also shows the improvement in the classification rate in the second experiment.
Table 6.2: Confusion Matrix With Majority Votes

<table>
<thead>
<tr>
<th>Actual/Classification</th>
<th>User</th>
<th>Intruder</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>0.99</td>
<td>0.01</td>
</tr>
<tr>
<td>Intruder</td>
<td>0.02</td>
<td>0.98</td>
</tr>
</tbody>
</table>

Detection Rate 0.98  False Alarm Rate 0.01

Majority votes helped to correct cases where the difference between an intruder and a normal user were ambiguous. Initially both systems had the same level of accuracy because no attack was present at the beginning of data.

In order to validate our experiment, we have examined the results presented in [50]. For comparison, we have presented the results of [50] with our results in Figure 6.5. In [50], Support Vector Machine (SVM) provided poor performance. On the other hand, Naïve Bayes, Random Tree, Random Forest, and Multilayer Perceptron showed a 82%, 93%, 93%, and 92% respectively. Our first experiment’s results compares evenly with Naïve Bayes result, but the second experiment shows
significant improvement in the classification compared to the other methods. However, more detailed work needs to be studied with overhead for network neighbors cooperation.

Figure 6.5: IDPS Comparison

Good accuracy is possible using common machine learning tools. Without majority vote, the system may not perform as well as it can; majority vote helps boost ambiguous classification results.

The next section presents some related work that help guide this research.

Related Work

Intrusion detection has been a topic of interest for many researchers of different areas. The feature extraction to classify transmissions can be found among the most common interests. Techniques like Fuzzy Logic [53, 58], Principal Component Analysis [59], Generalized Discriminant Anal-
ysis [54], and Genetic Algorithms [60, 61], have been satisfactorily used. The BPSO surpasses traditional GAs because its convergence speed is faster, and generates more feasible solutions, by having multiple mechanisms to avoid local optima [56], [55].

Different characteristics of IDPS were studied in [62], among them the differences between host-based [63] vs network-based [64] IDPS. The approach used in this work is both host-based because it runs locally (although scanning the network traffic), and network-based when the IDPS requires the majority votes from its neighbors.

Other approaches involve analyzing behavior through signature detection [65–67]. Our research will consider as future work what benefits majority vote can acquire via such approaches.

The next section concludes the paper.

Conclusion

Intrusion prevention can help preserve the integrity, availability, and confidentiality of data in a network. Computer networks can benefit from their distributed nature, by distributing the data samples, and only querying for more data when needed; hence, solving the intrusion detection with less amount of data. The proposed cooperative system employed a BSPO algorithm as feature selector, along with a PNN classifier. Both being robust against noise data, the classification of behavior became more consistent in accuracy. The cooperation among neighboring nodes in the system was able to reduce the number of false alarms, without sacrificing accuracy in intrusion detection.
Conclusions

In this dissertation the domains of Computer Networks, Smart Grids, and Intrusion Detection and Prevention Systems were examined. These domains were optimized with optimal and sub-optimal solutions for adaptive optimal performance. These domains were chosen for being deceiving domains. A deceiving domain is defined as one with many sub-optimal solutions without a clear global optimum. Such characteristic is exploited by the enhanced Particle Swarm Optimization algorithm which we denoted ONE. ONE employs Genetic Diversity and Complexification in order to promote exploration and exploitation, in turn finding a set of optimal and sub-optimal which we denoted Pareto Plateau.

ONE was able to find such Pareto Plateau of configurations using multiple fitness functions name for coverage, power consumption, confidentiality, and anonymity. It is not advised to use ONE in a domain with only one optimum since the algorithm is computationally expensive and it is meant to be used for finding optimal and sub-optimal configurations. On a non-deceiving domain, sub-optimal configurations simply require minor modifications from the optimal one. Moreover, we know that every modification which goes farther from the optimal will just have worse performance.

In order to adapt to changing environments, ONE can be monitored by a meta-level algorithm. This work proposed the usage of Fuzzy Logic to humanly represent algorithm behavior and specify the conditions under which ONE needs to have more diversity and exploration, and when it needs to have more exploitation. Furthermore, the domains on which operates, can be seen as a 0-1 Knapsack problem, where ONE needs to decide if a configuration can be selected as long as
its cost does not exceed some value. Examples of value added include coverage or power grid reduction, while costs may be power consumption or number of power sources to install.

On the Smart Grid domain, ONE can find optimal number of sources to have installed and which appliances connect to each power source. To reduce maintenance costs, the ONE test did not consider the usage of batteries; thus, total disconnection from the power grid is not possible. However, due to the load balancing capabilities of ONE, faster Return of Investment and benefits can be seen on a Smart Home (a home with ONE load balancing and green energy sources) as Smart Grid node.

Finally, the Intrusion Detection/Prevention System can utilize ONE as a feature filter which can help reduce the complexity of finding the difference between legitimate and intruder users. Upon finding ambiguous cases, such as those where maximum entropy is found (probability of intruder equal to 0.5), ONE can be used by adjacent neighbors in order to obtain classification from such neighbor nodes and arrive to a final classification via majority votes. The final classification score was obtained by a Probabilistic Neural Network since it easy to train and adapt to new samples and it is not affected by slight variations in the environment data.

**Future Work**

This dissertation went over optimizing computer network configurations by relying on machine learning methods which could become adaptive via a meta-layer reading environment conditions and tuning the running optimization to adapt to changes.

Next steps will envelop enhancing the algorithms presented in this work with the usage of parallel computing frameworks such as MPI and CUDA, in order to fully exploit the network’s distributed nature. Such distributed nature can be used to accomplish both speed, accuracy, and resiliency.
Parallelism of the PSO can occur at two levels. One of such levels is the main algorithm, where every node would run an independent algorithm, communicating the results at the end. A pseudo-algorithm of such an approach is presented in Algorithm 7. Notice how the algorithm does not differ much from its original, since it is simple with almost no parallel communication (only at start-up and end). The challenging task with this approach is to define an aggregation function, which determines how the results will merged from the different processing sources.

Algorithm 7 PSO - Main Level Version
1: Scatter Start-up
2: Do sequential Enhanced PSO
3: Gather Global Solutions
4: Aggregate Global Solutions

The second level is the population level. In this kind of scenario the parallelism’s goal would be to speed up the search process. Thus, each node would be equivalent to 1 (ideal) or $p$ (all for sequential) particles, where $p$ is the number of particles assigned to each node. Therefore, on each generation, the particles would run in parallel; thus, ending the generations computation faster. This second version of the pseudo-algorithm is shown in Algorithm 8, and is analyzed more in this chapter, as it can help with performance. In the particle level parallelism, nodes broadcast global results upon finding an update. The individual experience however, stays local across all corresponding nodes. Although at this level we can obtain a faster algorithm, it can only be applied when global experience is used.
Algorithm 8 PSO - Particle Level Version

1: Start with a random population of network configurations $P$
2: Initialize individual bests as the initial network configurations
3: Choose a global best from the particles
4: while Generations Remain AND Value not reached do
5:    Scatter the configurations
6:    for Each Configuration $C$ in $P$ do
7:        for Each Node $N$ in $C$ do
8:            Do individual particles task for as in sequential version
9:            if $F_C > F_{ibest}$ then
10:                $ibest \leftarrow C$
11:                if $F_{ibest} > F_{gbest}$ then
12:                    $gbest \leftarrow ibest$
13:                    Broadcast $gbest$
14:            end if
15:        end for
16:    end for
17: end while

Thus, we can evaluate our solution either by speedup or resilience as follows:

- **Speedup** ($\sum$): defined as how much faster the parallel version of the algorithm is when compared to its sequential counterpart. Such value is commonly of great interest in parallel computing and can be found with Amdahl’s law.

$$\sum = \frac{1}{(1 - f) + \frac{f}{S}}$$  (7.1)

where $f$ is the fraction of the algorithm subject to improvement, therefore $1 - f$ is the fraction that is not improved. $S$ is the measured speedup ratio of the time it takes the parallel algorithm to run, divided by the time it takes the sequential algorithm to run. The maximum
value of $\sum$ will be bounded by

$$\max \left\{ \sum \right\} \leq \frac{S}{(1 + t_\Delta \cdot (S - 1))} \quad (7.2)$$

where $t_\Delta$ is the percent of time spent by the sequential algorithm on the parts that are running in parallel now. This metric can help decide whether or not an improvement can be considered good under the $\sum$ criteria.

- **Resilience ($\rho$):** defined as the chances of a network configuration to be affected by either changing environment or faults in the network. As shown in [68], the PSO is able to get an optimal platoon using only the individual knowledge of the particles and using Pareto ranking on the individual best solutions at the end to obtain the global best along with the platoon. To find then a resilience score we can use the following expression:

$$\rho = \frac{\binom{|\eta|}{1} \cdot \binom{n_{particles} - |\eta|}{|\beta| - 1}}{\binom{n_{particles}}{|\beta|}} \quad (7.3)$$

where $|\eta|$ is the number of successful configurations, with fitness equal or better to a target value of $E$, found in the population of size $n_{particles}$ and $|\beta|$ is the number of unique configurations, that is, configurations that are not duplicates of another (particles in the same search) area which in turn provide no fault tolerance. For a better set $\beta$ we can limit the count to only the particles which fall under the $\eta$ set. The $\eta$ and $\beta$ sets can be defined as follows:

$$\eta = \{ c : c \in P \land \Delta_{fitness}(c, globalBest) \leq E \} \quad (7.4)$$

$$\beta = \{ c : c \in \eta \land Novelty(c, \eta) \geq \kappa \} \quad (7.5)$$
Novelty(c, s) = \frac{1}{|s|} \sum_{j}^{s} \text{dist}(c, s_j) \tag{7.6}

where \( c \) are configurations and \( P \) is the population set containing all particles (i.e., possible configurations). The successful configurations set \( \eta \) can be filtered out to obtain the novel configurations set \( \beta \), which can provide fault tolerance. This is done by finding a Novelty score for each \( c \) in \( \eta \) and keeping those which exceed a novelty threshold \( \kappa \). If \( \kappa \) is not exceeded between two configurations then the two configurations would be so similar that would not provide any fault tolerance in the event that one of the two configurations fails.
APPENDIX : BIOGRAPHICAL SKETCH
Hector M Lugo-Cordero was born in July 15 of 1983 at the city of San Juan, Puerto Rico. Hector is the son of Hector M Lugo-Santiago and Diana Cordero-Arias. He has a younger brother named Luis A. Lugo-Cordero and a younger sister named Liz Y. Lugo-Cordero. Hector studied the elementary and middle school at the Colegio San Rafael at Quebradillas and his high school at the Colegio San Antonio at Isabela.

In June of 2006 he received the title Bachelor in Science in Computer Engineering with Magna Cum Laude (high honors) from the University of Puerto Rico Mayaguéz Campus. He then continued his studies on August 2006 at the University of Puerto Rico Mayaguéz Campus in the area of Computer Engineering with a specialty in Computer Networks under the supervision of Dr. Kejie Lu. On April 2006 Hector passed the Fundamentals of Engineering exam and then on October 2006 he passed the Principles and Practice of Engineering Exam.

Hector completed his Master’s degree in Computer Engineering from the University of Puerto Rico on June 2009. His dedication to research and academia then took him to the University of Central Florida at Orlando Florida. At UCF he enrolled as a PhD student in Computer Engineering on August 2009 and then later a double major as a Master student for the Computer Science program; both degrees were guided by Dr Ratan K Guha. Hector completed his Master’s in Computer Science on May 2017 with specialization in Computer Networks Security. One year later in May 2018 Hector completes the PhD in Computer Engineering with concentration on Evolutionary Computation Inspired Optimization Algorithms.

His research interests include software architecture and development, artificial intelligence, computer networks, wired and wireless communications, network management and security, embedded systems, digital signal processing, and control systems.
LIST OF REFERENCES


and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), volume 1, pages 639–642, July 2017.


