Applications of Evolutionary Algorithms in Ultra-High Energy Neutrino Astrophysics

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Julie Rolla, M.Sc.

Graduate Program in Physics

The Ohio State University

2021

Dissertation Committee:

Professor Amy Connolly, Advisor

Professor Antonio Boveia

Professor Richard Furnstahl

Professor Christopher Hirata

Abstract

The discovery of ultra-high energy (UHE) neutrinos provides a promising tool to probe the highly energetic hadronic accelerators at the far reaches of our universe with little deflection or interaction in their travels through the cosmos. However, detecting these neutrinos is rare and requires expansive detectors and medium, optimized hardware, and extensive analysis. This thesis explores potential applications for evolutionary algorithms to improve the experimental design and analysis of UHE neutrino experiments.

The first project describes the use of genetic algorithms for designing antennas with better sensitivities to ultra-high energy neutrino-induced radio pulses than current designs. This thesis describes the history and work of the Genetically Evolving NEuTrIno TeleScopes (GENETIS) project. First, initial projects and exploratory GAs are described. Second, a discussion is given on the evolution of increasingly complex antenna designs optimized to improve the detection of UHE neutrinos. Finally, a project to optimize antenna response patterns are evolved for a given array geometry is presented.

The second project described is an investigation using genetic programming to distinguish signals from instrumental or environmental noise triggers. Using Karoo GP, a genetic programming software suite, we present a study using genetic programming

to classify signal and background in ARA data. In addition to variable exploration and feature extraction, coherently summed waveforms (CSWs) are also analyzed.

Evolutionary algorithms are powerful techniques that are underutilized in astrophysics. With the potential to improve experimental design and analysis techniques, these algorithms could help lead the search in UHE neutrino detection and the field of multi-messenger astronomy.

This is dedicated to my astounding husband, Sean Chilelli, and my big sister who never let me quit.

Acknowledgments

There are far too many people to thank and, with hopes of not making you go through an even lengthier task of reading this, I will focus on those that were monumental to this PhD adventure. This journey has been long and arduous, yet an absolutely extraordinary endeavor thanks to the company and support of those around me.

Above all, I want to extend my deepest thanks to my advisor Amy Connolly, whose guidance, poise, and ambition has forever impacted me. Working with you has given me the joy of loving what I learn, and deeply adoring those I work for. You have been there to support me through each up and down, and have guided me past my greatest challenges as a student, scientist, and person. Without you this journey may not have been so remarkable. I hope for the opportunity to work along side you again and to support your brilliance. Thank you again, Amy; you are truly exceptional!

To the GENETIS undergrads, old and new, I want to give my deepest thanks for making us a team. You are of the most brilliant and driven people I have had the pleasure of being around. I have not met a more incredible and creative group to conquer science (and donut bets) with. Each of you will forever be a friend and I'll miss the days of Starburst binges and Buffalo Wild Wings. A huge thanks to Ethan Fahimi, Alex Patton, Ryan Debolt, Ben Sipe, and Cade Sbrocco for their immense contributions to this project and my experience with GENETIS. Finally, a special

thanks to Alex Machtay. You are an astounding leader, scientist, student, and friend. Your contributions have been monumental to GENETIS, and my experience as a graduate student.

To my colleagues and mentors, such as Brian Clark, and Jorge Torres, thank you for always being there to support me as I follow in your tracks. You have been guidance on the days where the road was dark. To Kai Staats, you have been such a deep inspiration. Your passion for team work and mentorship have brought so much perspective and has shaped my work with the GENETIS team. I am eternally grateful to have the opportunity to work along side you. To Carl Pfender, your warmth and passion have been a joy to work along side. Thank you for all of your support through the last four years!

I want to ask extend my gratitude to my committee members Richard Furnstahl, Christopher Hirata, and Antonio Boveia for their time. Your support and insight has been extraordinary!

My deepest thanks to The Ohio State University for providing an extraordinary place to be a graduate student. Thanks to program coordinator Kristina Dunlap for all her work in supporting me in the graduate program and beyond. Thank you to all the organizations who have supported my graduate career financially, such as the Department of Physics for first-year fellowship support, the National Science Foundation for support under Award 1806923 and the Ohio State University Alumni Grants for Graduate Research and Scholarship.

To my best friends, Allie Snyder and Bryan Reynolds, I couldn't be more grateful to know you. To Allie, thank you for filling my life with so much happiness and fulfillment over the many years. Your positive attitude is contagious and I can't

express how much I appreciate your willingness to always listen. You will always be a sister to me and for that (and our cats), I am grateful. You have always been glad to provide assistance and support (until all hours of the night), even when we barely knew each other. Your loyalty and commitment to friendship is unparalleled.

To Bryan Reynolds, a huge thanks for your undying support and friendship, and for trying to fix the last remaining light in our life (office) as it started to die. You were the first to eagerly support me through candidacy. You spent hours into the night answering my questions and watching my presentation. As a scientist with a passion for teaching, I aspire to be like you and I hope a plethora of others get the opportunity to be impacted by you. As a very dear friend, you have brought me so much joy and support over the years; I can't imagine future endeavors without you. I hope we can make more top hits like "rock beach" and "Flat Bottom Egg," and raise our cat families near each other for many years to come.

Finally a huge thanks to my beautiful husband, Sean Chilelli, and my family. Sean, you have believed in me and supported me with endless energy every single day. You are my inspiration, my joy, my motivation, and my world. It is to you that I dedicate this work; without you none of this would have been possible. With you, I've achieve my biggest dream and, for that (and our cats), I thank you. Thanks to my big sister Ginny for making sure I never resigned myself to a life of less than I was capable of and for raising my nephews to love science enough that they might actually read this someday (and correct any mistakes). Ginny, without you I would have never even started this journey and accomplished my dreams. To my sisters, brothers, moms and dad, thank you for always believing in me!

Vita

April 21, 1986 ..... Born - Dickinson, ND USA

May 2010 ..... B.A., University of California, Berkeley
Berkeley, CA USA

June 2018 ..... M.Sc., The Ohio State University
Columbus, OH USA

Publications

Research Publications

J. Rolla et. al. for the GENETIS Collaboration. "Evolving Antennas for Ultra-High Energy Neutrino Detection." *36th International Cosmic Ray Conference*, 2019

J. A. Aguilar, et. al. "The Next-Generation Radio Neutrino Observatory – Multi-Messenger Neutrino Astrophysics at Extreme Energies." *arXiv:1907.12526*, 2019

N. Mirabolfathi, et. al. "Neganov-Luke Phonon Amplification in P-type Point Contact Detectors." Journal of Low Temperature Physics, 176:209-215, 2014.

A. Kenany, et. al. "SuperCDMS Cold Hardware Design." Journal of Low Temperature Physics, 167:1167-1172, 2012.

Fields of Study

Major Field: Physics

Table of Contents

List of Tables

Table **Page**

List of Figures

Figure **Page**

Chapter 1: Neutrino Astronomy

1.1 Introduction

Over the last few decades, scientists have used ultra-high energy (UHE) particles to explore the most extreme events in the distant universe. These events accelerate hadronic matter to relativistic speeds and lead to the production of three main UHE particles that can be detected on Earth: (1) cosmic rays (CRs), (2) gamma rays, and (3) neutrinos. Each of these has both advantages and disadvantages for probing the distant corners of the universe. As charge-carrying particles, cosmic rays are bent by galactic magnetic fields making it difficult to retrace their origins. UHE cosmic rays have been detected, but no sources have been found. Further, at energies above $\sim 10^{19.5}$ eV, cosmic rays are attenuated by the GZK process [3]. Gamma rays are uncharged and can carry information apropos their origin; however, gamma rays attenuate above approximately 100 TeV and the gamma-ray signal leaves an ambiguity between signal creation from leptonic processes and hadronic processes that has yet to be solved [29]. Only gamma-ray signals produced from hadronic processes originate from these extra-galactic phenomena of interest that would produce UHE cosmic rays. Furthermore, high energy gamma rays above 10^{14} eV are absorbed by cosmic radiation [40], [73]. This leaves the highest energy gamma rays emitted from

sources hidden from telescopes. The third particle, neutrinos, do not carry an electric charge, and are only subject to the weak interaction and the force of gravity. This makes them distance-resilient communicators since their direction is not influenced by galactic magnetic fields and they rarely will interact with matter. Unlike cosmic rays and gamma rays, neutrinos that reach earth will point directly back at their source.

Neutrinos are exceptional extra-galactic messengers. Likely born from violent events such as blazars, active galactic nuclei (AGN), gamma ray bursts (GRBs), and starburst galaxies, UHE neutrinos are expected to be present in the universe and, as elementary particles, play a significant role in understanding the fundamental questions of astrophysics. With neutrino astronomy, we can expand our understanding of the most powerful particle sources in the universe, which are currently not well understood. UHE neutrino astronomy (at and above 10^{17} eV) seeks to provide new information in the highest energy regimes where other particles are inadequate.

1.2 Production of UHE Neutrinos

There are two main predictions for the creation of UHE neutrinos. First, extra-galactic sources act as nature's most powerful particle accelerators to generate UHE neutrinos. Called astrophysical neutrinos, these come directly from point sources. Second, the GZK process creates cosmogenic neutrinos through indirect production. An illustration of the neutrino flux from various sources is given in Fig. 1.1. Due to the GZK process, UHE cosmic rays can only travel within tens of Mpc before interacting with CMB and producing UHE cosmogenic neutrinos [40].

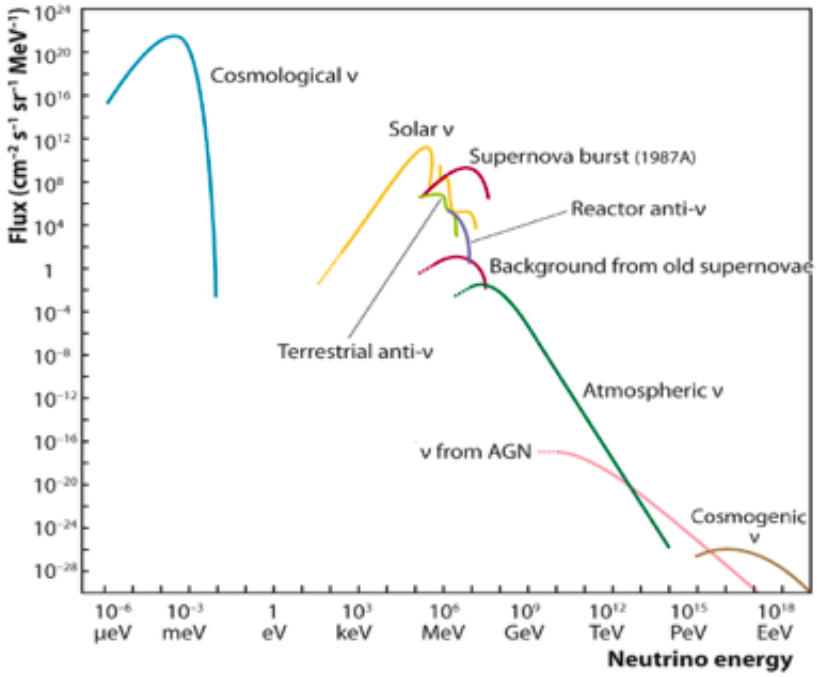


Figure 1.1: Measured and expected fluxes of neutrinos from different sources. Energies above approximately 10 GeV are addressed by Cherenkov and Askaryan detectors underwater and in ice. The highest energies are only accessible with detectors one to three orders of magnitude larger than IceCube [105].

1.2.1 Astrophysical Neutrinos

Many theories that exist regarding the origin of astrophysical neutrino flux at and above the PeV scale are based on extra-galactic sources; these extra-galactic sources are capable of accelerating hadrons (likely protons or possibly heavier nuclei such as iron) to speeds much higher than man-made accelerators on earth [16]. These UHE hadrons interact with gas near the source or with ambient radiation, producing kaons and charged pions which subsequently decay into neutrinos and anti-neutrinos [38]. An example process can be seen below:

$$
\begin{aligned}
p_{uhe} + \gamma_{bg} &\rightarrow n + \pi^{+} \\
n &\rightarrow p + e^{+} + \nu_{e} \\
\pi^{+} &\rightarrow \mu^{+} + \nu_{\mu} \\
\mu^{+} &\rightarrow e^{+} + \bar{\nu}_{\mu} + \nu_{e}
\end{aligned}
\tag{1.1}
$$

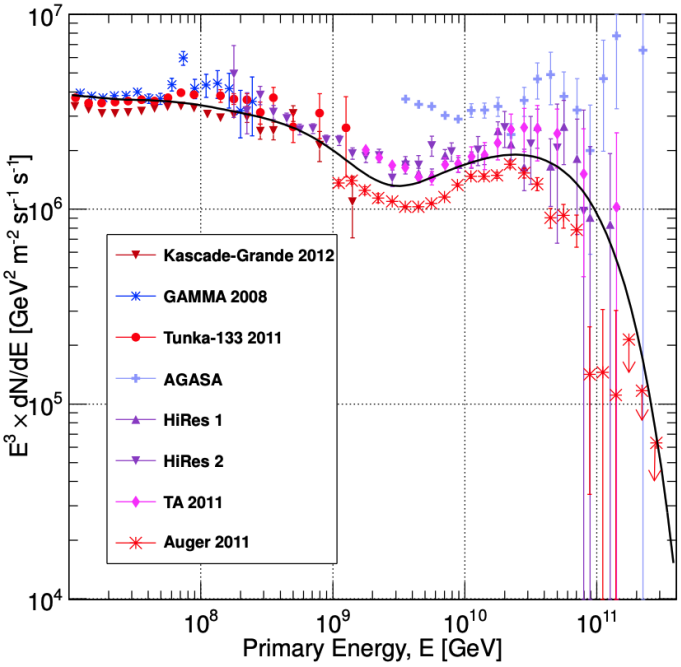


Figure 1.2: Cosmic ray spectrum from a variety of experiments. As evidence for the GZK limit there is a sharp cut off above 10^{10} GeV. Figure from [110], adapted from [54].

1.2.2 Cosmogenic/GZK Neutrinos

Above 10^{17} eV, ultra-high energy cosmic rays (UHECR) are theorized to interact with the CMB through the Δ resonance in a process first theorized by Greisen, Zatsepin, and Kuzmin [59, 114]; the decay of the charged pions results in UHE neutrinos

called cosmogenic or BZ neutrinos (after Berezinksy and Zatsepin) [28]. The GZK process promises that as long as UHECRs above the threshold energy of 5×10^{19} eV exist, there will be the production of UHE neutrinos. As evidence for the existence of the GZK process, the UHECR flux displays a drop-off at this energy as shown in Fig. 1.2. This is called the Greisen-Zatsepin-Kuzman (GZK) process and is fundamentally the same as Eq. 1.1. The GZK process is given below.

$$\begin{aligned} \gamma_{\text{CMB}} + p &\rightarrow \Delta^{+} \rightarrow p + \pi^{0} \\ \gamma_{\text{CMB}} + p &\rightarrow \Delta^{+} \rightarrow n + \pi^{+} \end{aligned} \tag{1.2}$$

1.3 Challenges of Detecting UHE Neutrinos

Though the interest in multi-messenger astronomy is growing, the discovery of UHE neutrinos remains extraordinarily difficult. The low flux of neutrinos, in conjunction with their low cross-section, presents significant challenges. Experiments must implement massive detector volumes for a small chance at detecting a UHE neutrino signature. For example, cosmogenic neutrinos are expected to arrive at Earth with an incident flux of approximately $\sim$10 ν/km^2/yr/str and the low cross-section of $\sigma\sim$ 10 $^{-32}$ cm^2 gives an interaction rate in ice (ρ = 0.9 g/cm^3) of $< 3 \times 10^{-3}$ ν/km^3/yr/str [49, 37]. This means detector volumes on the order of 100 cubic kilometers would detect approximately 1 ν/yr.

1.4 Cherenkov Radiation

Physicist Gurgen Askaryan proposed that UHE neutrinos could be observed through their interactions within a dielectric medium via coherent electromagnetic radiation [24]. He theorized that when a UHE, relativistic neutrino collides with the nucleus in

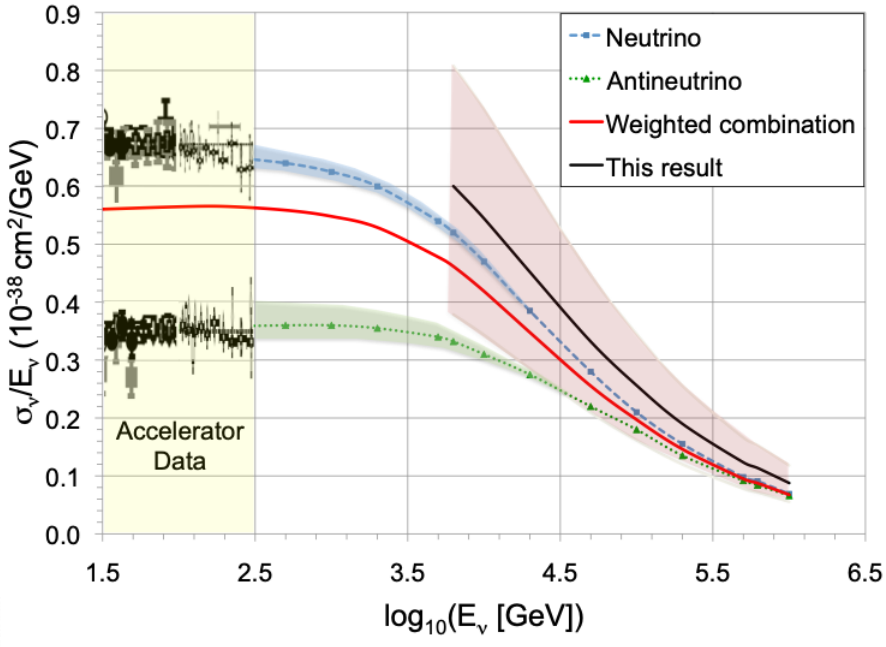


Figure 1.3: Figure from the IceCube Collaboration. Depicted is the neutrino charged current cross section measurements, divided by neutrino energy, obtained by accelerator experiments. The blue and green lines are the Standard Model predictions for the ν_μ and $\bar{\nu_\mu}$. The uncertainties on the deep inelastic cross sections are shown by the corresponding shaded regions. The red line is for the predicted mixture of ν_μ and $\bar{\nu_\mu}$ in the IceCube sample. The black line shows the 2017 published measurement of the multi-TeV neutrino cross section with IceCube. The pink band shows the total 1σ (statistical plus systematic) uncertainty. The cross section rises linearly with energy up to about 3 TeV, but then the increase moderates, to roughly as $E_\nu^{0.3}$, due to the finite $W^\pm$ and Z^0 masses [49].

a dielectric, a particle shower consisting of photons, electrons, and positrons would be produced. If this particle shower traveled in the medium at a speed greater than the speed of light in the medium, Cherenkov radiation will be produced [40]. These high-energy neutrinos interact with a nucleus primarily through deep inelastic scattering through the weak interaction. There are two types of interactions that neutrinos experience: (1) neutral current (NC) which is mediated by a Z boson, and (2) a charged

current (CC) which is mediated by a W boson [38]. These interactions can be seen below [27], [77]:

$$\begin{aligned} \overset{(-)}{\nu}_l + N &\rightarrow \overset{(-)}{\nu}_l + X \qquad (NC) \\ \overset{(-)}{\nu}_l + N &\rightarrow l^{\pm} + X \qquad (CC) \end{aligned} \tag{1.3}$$

where N is the nucleus, l is the lepton flavor and X is a hadronic shower. Most current neutrino detection techniques are focused on an indirect method of searching for the remnants of these two types of weak interactions. Fig. 1.4 shows Feynman diagrams of each process.

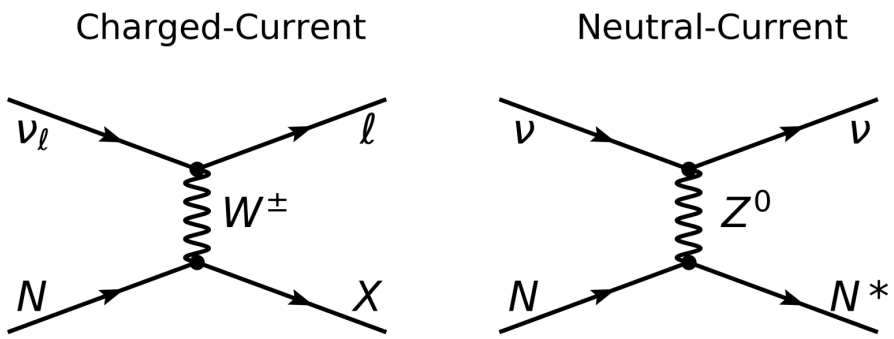


Figure 1.4: Illustration of the Feynman diagrams for the Charged-Current (left) and Neutral-Current (right) deep inelastic neutrino-nucleon scattering [37].

As mentioned, if particles are moving at relativistic speeds and are in a dense dielectric medium with an index of refraction greater than 1, then the particles will propagate faster than the phase velocity of light in the medium, thus emitting optical-Cherenkov radiation (dominated by blue light). Similarly, an electromagnetic shower (produced by an interaction) traveling at relativistic speeds in a similar media will emit Askaryan radiation. Askaryan radiation occurs because as the particle shower travels through the dielectric, it develops a 20% negative charge due to Compton

scattering of electrons and the annihilation of positrons in the shower with electrons in the medium [40]. The change in the net current of the cascade over time causes the radio pulse [24]. For wavelengths comparable to the width of the shower, the radiation will add coherently, thus creating measurable broadband (10 - 1200 MHz), fast radio pulses.

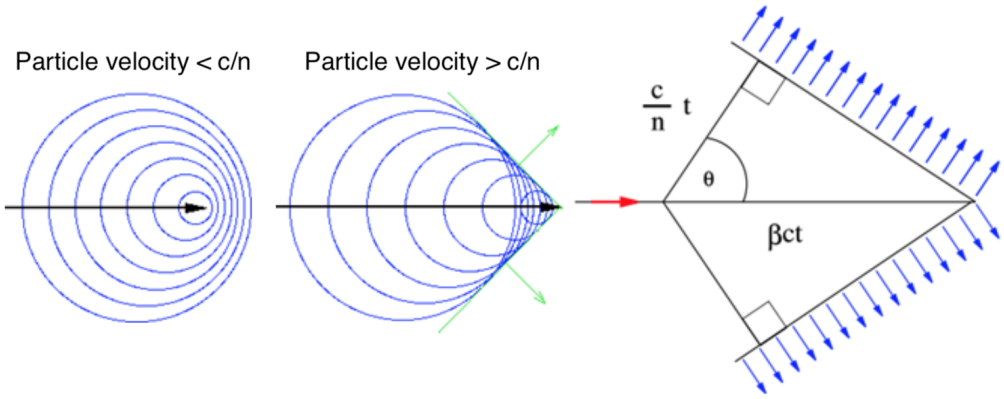


Figure 1.5: Illustration of the Cherenkov Effect. If the particle velocity is less than c/n, the resulting electromagnetic radiation does not constructively interfere (left). When the particle velocity is greater than c/n, the radiation constructively interferes (center). This produces a cone of light (right). Adapted from [91].

The discovery of Askaryan emissions has been revolutionary in the search for UHE neutrinos. Unlike optical-Cherenkov, attenuation lengths in the Antarctic ice have been measured to be on the order of 1 km, whereas optical light attenuates on the order of tens of meters; thus, the use of Askaryan emissions grants a detection volume that is orders of magnitude larger than that of optical-Cherenkov-based experiments. Large detector volumes are essential in the search for UHE neutrinos as potential events are so rare.

1.5 Experiments Detecting Cherenkov or Askaryan Radiation

A number of experiments exist that seek to improve limits on the diffuse flux of UHE neutrinos and to detect a UHE neutrino; this section will provide a brief overview of some leading experiments. Fig. 1.6 shows the current leading limits for the experiments ARA, ANITA, and IceCube, which are mentioned in the following sections, as well as Auger and ARIANNA which will not be discussed in detail.

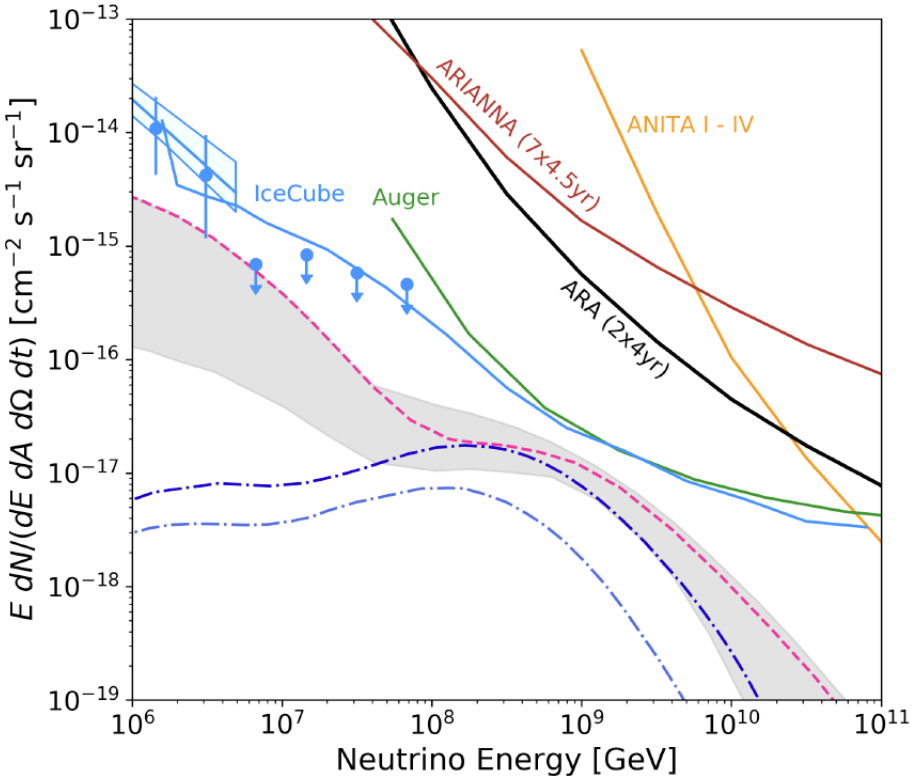


Figure 1.6: Theoretical predictions for the cosmogenic flux of neutrinos and recent limits from existing UHE neutrino experiments. The measured astrophysical neutrino flux (data points) is from IceCube [9] and the limits (solid lines) are from IceCube [7], Auger [1], ANITA [56], ARA [17, 20], and ARIANNA [93]. The dashed lines and shaded bands are theoretical models from Olinto et. al. [92], Kotera et. al. [75], and Ahlers and Halzen [15]. Figure by Amy Connolly.

1.5.1 IceCube

Optical-Cherenkov experiments have been widely successful at measuring neutrinos up to 10 PeV. These experiments rely on photomultiplier (PMT) tube detection technologies, where PMTs are arranged in an array in an optically transparent material. As discussed in the previous sections, the interaction of a neutrino with the medium – usually ice or water – causes a particle shower. When the shower, or a daughter lepton, disperses in the ice, optical-Cherenkov is emitted. As this shower or lepton passes a PMT, signals from the optical (blue) spectrum are recorded and details such as signal strength, pattern, distance traveled can be inferred. An analysis of the data allows for reconstruction of neutrino direction, lepton flavor, and energy.

One of the most prominent optical-Cherenkov experiments utilizing these methods is IceCube. The IceCube Neutrino Observatory is a cubic kilometer, optical Cherenkov detector located in Antarctica. IceCube aims to measure signals created from particles – such as neutrinos and cosmic rays – emitted from astrophysical sources with hopes of gathering information about their origin [106]. IceCube focuses on three main measurements: (1) cosmic rays, (2) neutrino oscillations, and (3) astrophysical neutrinos.

In September 2017, IceCube detected a 0.3 PeV neutrino within 0.1 degree of a flaring blazar, TXS 0506+056 [16]. Further analysis of more than 9 years of IceCube data found an excess flux of neutrinos above the expected atmospheric background in the direction of the blazar [8]. This result provides the first significant (3.5σ) evidence for an astrophysical source of neutrinos. This result prompted a multi-messenger investigation with additional experiments finding significant results in the direction of TXS 0506+056 [14]. Prior to the 2017 discover, IceCube had detected

high energy neutrinos in the direction of astrophysical sources, but the results were not significant [71].

IceCube consists of a lattice of photo-multiplier tubes (PMTs) buried over 1 km deep in Antarctic ice to detect Cherenkov radiation (Fig. 1.7). IceCube's instrumentation allows for the detection of both tracks and cascades, and the estimation of the parent neutrino's energy and direction [5]. The number of photons produced is related to the energy of the parent neutrino [4]. For both tracks and cascades, the energy of the parent neutrino can be calculated from the deposited energy within 15%. Since muons of high energy can travel distances larger than that of the detector's dimensions, the energy has to be calculated indirectly [61]. The energy loss of the muon throughout the detected track is used to estimate the parent neutrino's energy [2]. In addition, tracks allow for a better directional reconstruction, within $<1^\circ$ of the parent neutrino compared to $\sim 15^\circ$ for cascades.

The main volume of IceCube is an array of 5160 digital optical modules, or DOMs (each containing a PMT), installed between 1450 m and 2450 m below the surface on 86 strings [5]. Each string holds 60 DOMs, which reside along a single cable. For the in-ice array, each vertical string has a separation of 17 m per DOM. The strings are arranged over a volume of one cubic kilometer of ice in a hexagonal pattern on a triangular grid with a 125 m horizontal spacing [5]. Note that depths between 2000 m and 2100 m are not instrumented due to a "dust layer" where the ice contains impurities resulting in optical scattering and absorption. DeepCore is a denser central region of DOMs beneath 1750 m that provides insight into lower energy neutrinos. Finally, IceTop is an array of DOMs on the surface of the ice with the primary goal of detecting secondary particle showers resulting from interactions of high-energy cosmic

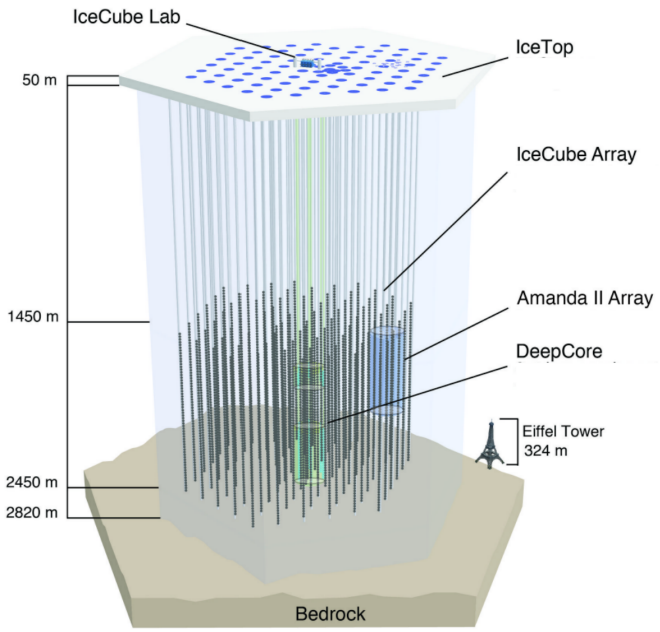


Figure 1.7: Schematic of IceCube Neutrino Observatory, showing the location of the detectors in the ice [16].

rays in the atmosphere. IceCube's identification of both tracks and cascades caused by neutrino interactions can be seen in Fig. 1.8.

1.5.2 ARA

Differing from optical-Cherenkov experiments such as IceCube, Askaryan-focused experiments seek to explore higher energies at limits above $10^{19.5}$ eV. In this thesis, two main Askaryan experiments will be reviewed: (1) the Askaryan Radio Array (ARA) and (2) the Antarctic Impulsive Transient Antenna (ANITA).

ARA is located at the South Pole, a few kilometers from the IceCube observatory. It consists of five stations buried in the Antarctic, as seen in Fig 1.9. These stations consist of a prototype station, the "Testbed", one station at 100m depth (A1), and

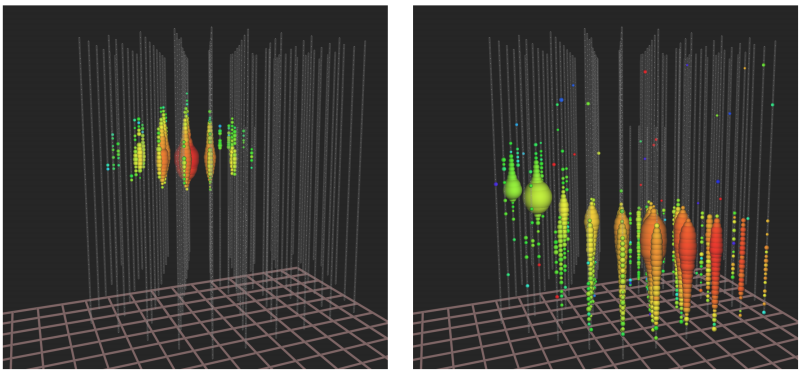


Figure 1.8: Visualizations of two high energy neutrino events detected by IceCube. Each faint vertical white line represents a string of detectors with white dots representing DOMs that did not detect any photons. The color illustrates the arrival time of the signal, with red being the earliest and blue being the latest. The larger the sphere, the more photons detected. On the left, a spherical cascade from an electron or tau neutrino is shown with a deposited energy of 1.16 PeV. On the right, an up-going muon track from a neutrino is shown with a deposited energy of 2.6 PeV [16]. Every high energy event can be seen in 3D at icecube.wisc.edu/viewer/he_neutrinos.

four stations buried 200m deep (A2-5). Each station, or set of antennas, operates separately. The full five-station array is referred to as "ARA5," with individual stations identified as A1, A2, A3, A4, and A5 [21].

Each station consists of 4 measurement strings, which consist of two vertically-polarized antennas (VPol) and two horizontally-polarized antennas (HPol), for a total of 16 antennas per station, as seen in Fig. 1.10. The measurement strings are located 30 m apart, forming a square. The ARA antennas are broadband using azimuthally-symmetric bicone antennas for VPol and quad-slot antennas for HPol. In addition to the measurement strings, each station has two calibration strings located at 40 m, with one VPol and one HPol antenna. These provide pulses that are used to verify the lifetime and, as the name implies, calibrate the measurement antennas. The ARA

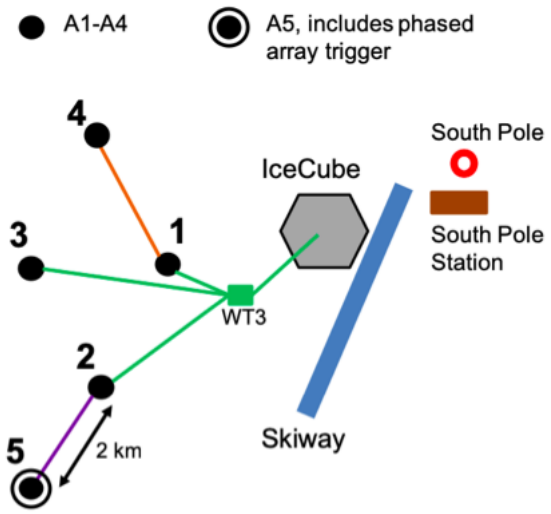


Figure 1.9: A map of the current five ARA stations instrumented at the South Pole, and nearby landmarks and buildings [21].

antennas must fit into narrow holes in the ice, with the VPol antennas as birdcage bicones (13.9 cm diameter) and the HPol antennas as ferrite-loaded quad-slot antennas (12.7 cm diameter) [23, 19, 21].

When a signal reaches the ARA antennas, it is band-pass filtered to 150-850 MHz, as well as notch-filtered at 450 MHz to remove South Pole radio communications [21]. After this filtering, the signal is amplified by 34 dB, converted to an optical signal, and passed on the Data Acquisition Electronics (DAQ) box on the surface of the ice. All data is then transmitted to the IceCube Counting Laboratory (ICL). More details on the ARA data are provided in Chapter 4.1.3.

The ARA stations A2 and A3 have accumulated fractional livetime since deployment that can be seen in Fig. 1.11; these stations have accumulated approximately 1100 days of livetime as of 2019.

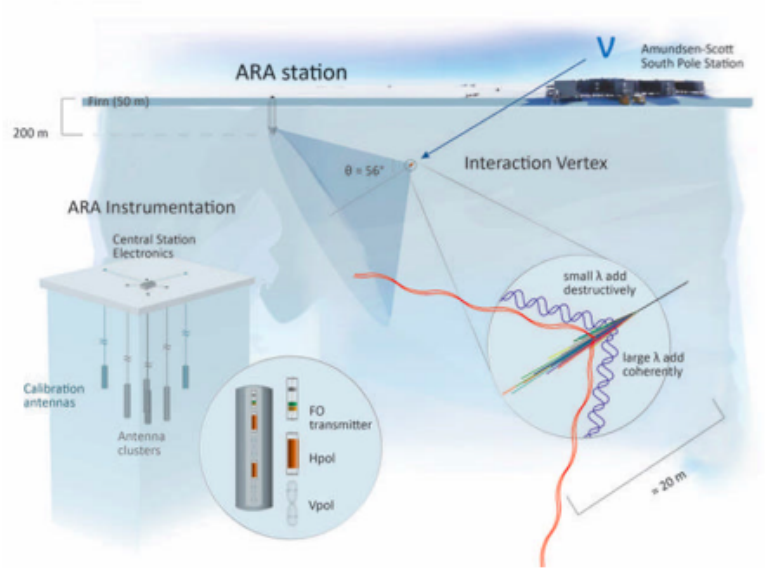


Figure 1.10: An illustration of an ARA station and the neutrino detection methods, showing a neutrino interacting in the ice [21].

1.5.3 ANITA

ANITA's mechanism for observing these UHE neutrino signals is through an array of dual-polarized horn antennas suspended on a NASA Long-Duration Balloon, flown about 37,000 meters above the Antarctic ice. The ANITA experiment is illustrated in Fig. 1.12. ANITA seeks to identify particle cascades refracting out of the ice surface from neutrinos interacting in the Antarctic ice. ANITA is launched from a starting point near McMurdo Station, Antarctica. After ANITA launches, it flies in circular orbits over the continent of Antarctica as seen in Fig. 1.13. ANITA scans approximately a million cubic kilometers of ice, making it the neutrino detector with the largest detection volume. ANITA, sitting approximately 8 m tall, flies about 40 km above the ice. It is sensitive to radio signals between 200 and 1200 MHz [50]. The

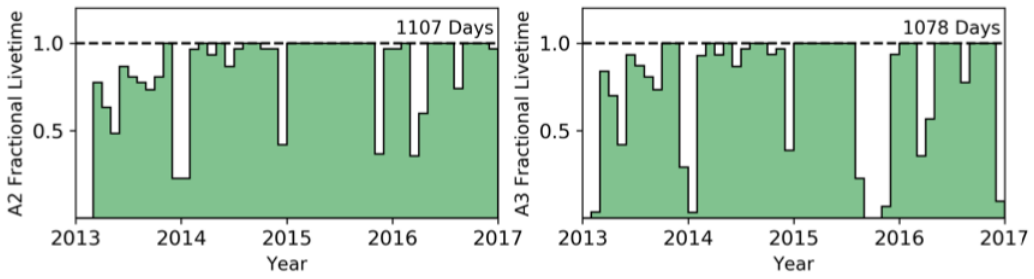


Figure 1.11: Fractional livetime of ARA A2 and A3 stations through 2017 [21].

ANITA experiment has launched four flights and currently sets limits above about $10^{19.5}$ eV on the diffuse flux of neutrinos [56].

ANITA-IV is made up of 48 highly directional horn antennas. The upper two layers of antennas are evenly spaced 45 degrees apart and the layers offset by 22.5 degrees for uniform coverage. Antennas in the third layer are spaced 22.5 degrees apart, and the antennas on the lowest layer are spaced 45 degrees apart. The entire array is approximately 8 m tall. All of the antennas are pointed 10 degrees below the horizon in order to observe distant ice and are painted white to reflect sunlight and regulate temperature. ANITA's antennas are capable of observing both horizontal and vertical-polarized signals between 200 and 1200 MHz [55].

ANITA flies with an added experiment box where data processing and storage occurs; thus, recovering the balloon is crucial. This box contains ANITA's triggering logic computer systems and data storage, which all rest within a large Faraday housing. High-priority digitized data is transmitted back via a bandwidth-limited satellite network to ensure that ANITA is properly functioning.

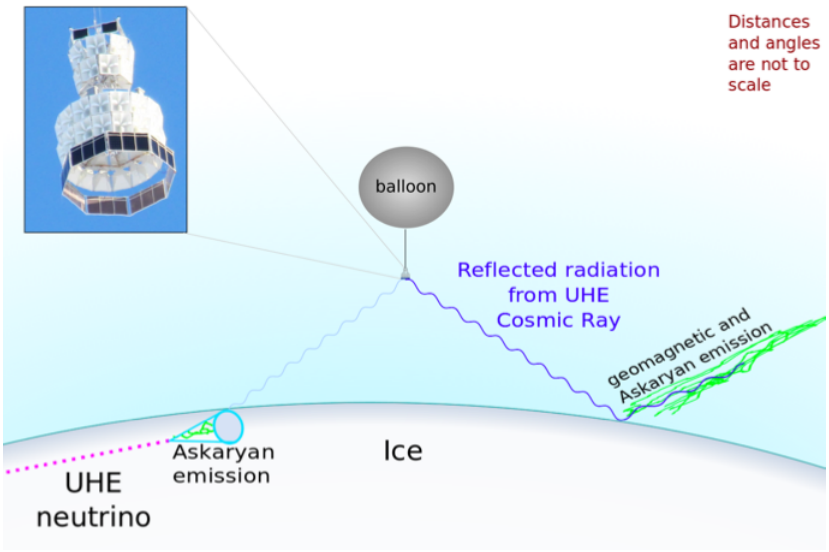


Figure 1.12: The ANITA experiment concept with a photo of the ANITA-IV payload in the upper left and potential detection methods. Ultra-high energy neutrinos interact with the Antarctic ice, thus producing a radio pulse (Askaryan radiation). UHE cosmic ray interactions in the atmosphere produce a shower of secondary particles that interact with the geomagnetic field and can also produce a radio signal [50].

ANITA-IV was launched and successfully deployed for 28 days of flight time starting December 2, 2019. It had a livetime of 94 percent of the time of the flight, leaving ANITA with a livetime of 27.3 days [55].

1.5.4 Importance of optimized detectors

As of this publication, UHE neutrinos have not been detected. The first ARA station became active in 2011 and the first flight of ANITA was in 2006; however, despite searching for UHE neutrinos for over a decade, no definitive signals have been found. The lack of UHE neutrino events highlights the need for improved detectors and experiments. To that effect, the next generation experiments are undergoing large improvements to increase their detection volume compared to current experiments

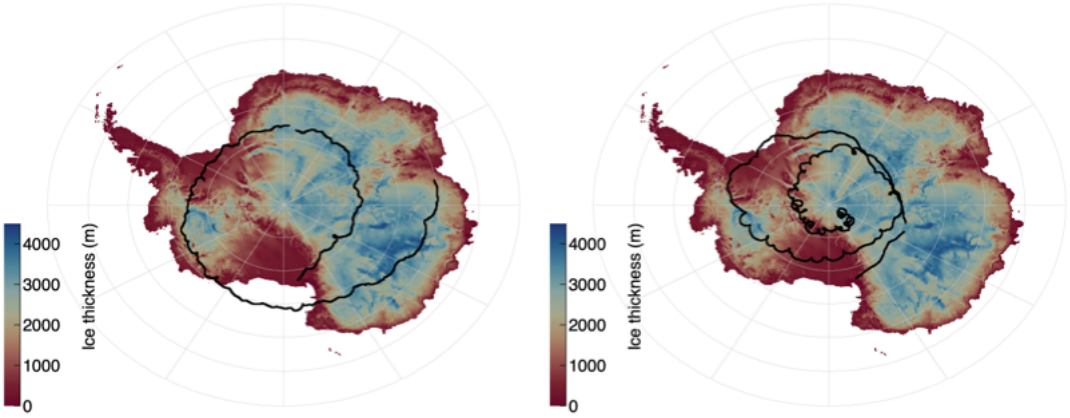


Figure 1.13: Flight path simulated in icemc, ANITA collaboration simulation software, for the ANITA-III (left) and ANITA-IV (right) flights [50].

and thus increase the probability of a UHE neutrino signal [12]. Unfortunately, these improvements are costly and require significant time to implement. Improving our sensitivity will provide a larger sample size which can give us a better understanding on the density of sources producing UHE neutrinos, and cosmic ray composition [48]. Furthermore, a larger sample provides greater resolution in understanding UHE neutrino cross sections [39].

The value of detector optimizations can be quantitatively explored by estimating the expected number of neutrinos detected by an experiment. This requires the effective volume of the experiment and the neutrino flux. The flux is a measurement of the number of neutrinos per energy per effective area per solid angle per second. This can be mathematically expressed as:

$$F(E) = \frac{dN}{dE_\nu \; dA_{\text{eff}} \; d\Omega \; dt} \tag{1.4}$$

where N is the number of particles, A_{eff} is the effective area, t is the time and Ω is the fractional sky coverage or solid angle [51]. The effective area is related to the effective volume by the equation [57]:

$$[A\Omega]_{\text{eff}} = \frac{[V\Omega]_{\text{eff}}}{\ell_{\text{int}}} \tag{1.5}$$

where ℓ_{int} is the interaction length.

The differential of Eq. 1.4 can then be separated and integrated to find the number of particles. This is done under the assumption that the effective area is constant over each energy bin and that each variable is independent of their relationships known.

$$dN = F(E)\ dE_\nu\ dA_{\text{eff}}\ d\Omega\ dt$$

$$\int dN = \int F(E)\ dE_\nu\ dA_{\text{eff}}\ d\Omega\ dt$$

$$N = F(E) \cdot \Delta E \cdot T \cdot [A\Omega]_{\text{eff}} \tag{1.6}$$

where ΔE is the energy bin, T is the operational time of the experiment, and $[A\Omega]_{\text{eff}}$ is called the acceptance. Substituting Eq. 1.5 into Eq. 1.6, the number of neutrinos detected by an experiment is given by:

$$N = F(E) \cdot \Delta E \cdot T \cdot \frac{[V\Omega]_{\text{eff}}}{\ell_{\text{int}}} \tag{1.7}$$

One goal of the detector optimization given in Ch. 3 is to improve the effective volume by a factor of two. Eq. 1.7 demonstrates that doubling the effective volume would double the expected number of neutrinos for a given flux.

The completed ARA37 experiment (whose development is currently paused at 5 stations) is expected to have a acceptance of $2.33 \times 10^9\,\mathrm{cm^2 str}$ at an energy of $10^{18}\,\mathrm{eV}$ [19]. Using the models shown in Fig. 1.6 the cosmogenic flux of neutrinos at $10^{18}\,\mathrm{eV}$ can be estimated to be $10^{-35}\,\mathrm{cm^{-2}s^{-1}str^{-1}eV^{-1}}$. Substituting the ARA37 acceptance and cosmogenic flux at $10^{18}\,\mathrm{eV}$ into Eq. 1.6 gives

$$N = [10^{-35}\,\mathrm{cm^{-2}s^{-1}str^{-1}eV^{-1}}] \cdot [10^{18.5}\mathrm{eV} - 10^{17.5}\mathrm{eV}] \cdot [0.75\,\mathrm{year}] \cdot [2.33 \times 10^9 \mathrm{cm^2 str}]$$

which gives a total number of detected $10^{18}\,\mathrm{eV}$ neutrinos per year of $N = 1.57$. This was found using an energy bin of $10^{18.5} - 10^{17.5}\,\mathrm{eV}$ and an operational livetime of 75%.

Assuming a Poisson distribution with a mean of 1.57, there is more than a 20% probability of no events being detected in a given year. Doubling the effective volume through optimizations, would result in an expected number of detected $10^{18}\,\mathrm{eV}$ neutrinos per year of $N = 3.14$. This would reduce the probability of no detected neutrinos in that energy range to under 5% (95% probability of detecting at least one UHE neutrino signal).

This result demonstrates the potential impact of improving the sensitivity of UHE neutrino detectors. By optimizing the antennas themselves, and improving analysis techniques, we can improve the sensitivity of each new generation of experiment at minimal costs. The use of evolutionary algorithms to optimize the detectors and analysis will accelerate the discovery of UHE neutrinos and other astrophysics outcomes.

Chapter 2: Evolutionary Computation and Machine Learning

Evolutionary algorithms and machine learning are powerful approaches to optimize experiments and perform data analysis. These techniques provide means to discover solutions to complex problems that traditional methods would be unlikely to find. This chapter initially provides an introduction to the critical concepts of overfitting and underfitting. The following section gives a brief discussion of different machine learning methods and their application. Genetic algorithms and genetic programming are discussed in detail in the final sections. This chapter provides the necessary background to understand the main projects in this thesis.

2.1 Machine Learning

Machine learning (ML) is a type of data analysis that automates analytical model building and is closely related to the evolutionary algorithms investigated in this dissertation. It exists as a branch of artificial intelligence, as it is based on the premise of systems learning from data and, thus, identifying patterns with minimal human intervention. There are two types of ML algorithms, supervised and unsupervised. Though we will not be utilizing unsupervised ML in this thesis, we will discuss both briefly.

2.1.1 Unsupervised Learning

ML algorithms that are unsupervised cluster and analyze unlabeled data sets with no preassigned scores or labels. In this case, the goal is to learn the data's inherent structure without the use of explicitly provided labels, thus discovering all of the unknown patterns in the data. Examples of unsupervised learning algorithms include clustering and principal component analysis. In clustering analysis, the algorithm will automatically group the training sample into categories with correlated features. In principle component analysis, the algorithm compresses the training data set by identifying useful features and discarding the useless ones. Unsupervised ML has a significant advantage in that it requires a minimal workload to prepare a training sample.

2.1.2 Supervised Learning

Supervised ML algorithms use sample data, known as the "tr aining sample," to build a model used to make independent predictions or decisions without the further assistance of the user. These data sets used for training are labeled or have preassigned scores. Supervised learning is usually done in the context of classification or regression. Classification categorizes a set of data into classes, such as facial recognition technology. Regression predicts continuous values, such as finding the best fit line. A depiction of these methods can be seen in Fig 2.1. Note that evolutionary algorithms can also solve both of these types of problems (as well as many others). Some of the most widely used learning algorithms we will discuss are K-nearest neighbor, neural networks, and random forest.

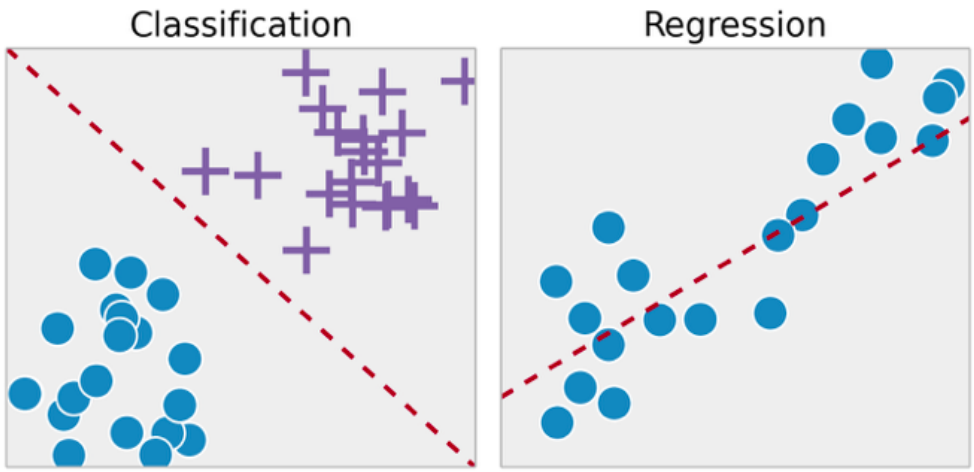


Figure 2.1: A depiction of classification (left) and regression (right) used in supervised machine learning. The red dashed line for classification indicates the models threshold between the two groups. The red dashed line for regression indicates the models best fit of the data [104].

K-nearest neighbor (k-NN) is a pattern recognition algorithm used for both classification and regression. With k-NN, the algorithm is given a training sample, which classifies coordinates into groups identified by some feature. When the algorithm is given a testing sample to test the classification accuracy, some of these new data points may hold features that easily classify them into one group or the other; however, for any unclassified point, we can assign it to a group by observing what group its nearest neighbors belongs to using the Euclidean distance. It is important to note that the k is a user-defined constant and determines the number of neighbors to consider when assigning a classification. A depiction of this classification method can be see in Fig. 2.2.

Random forest is a supervised ML algorithm that also solves regression and classification problems. Random forest is an ensemble learning method that utilizes a

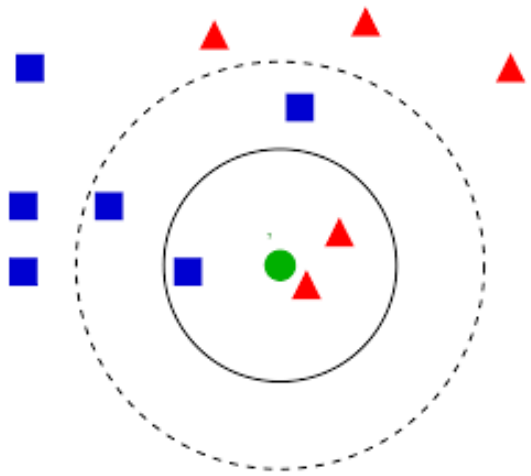


Figure 2.2: Illustration of the k-NN classification, where the green circle is being classified to red triangles or blue squares. The resulting classification is dependent on the value of k. If k=3, the green dot would be classified with the red triangles because there are more red triangles in the three nearest neighbors (solid circle). If k=5, the green dot would be classified with the blue squares because there are more blue squares in the five nearest neighbors (dotted circle). Illustration by Antti Ajanki is licensed under CC BY-SA 3.0.

multitude of decision trees and merges them to get a more accurate and stable prediction. Decision trees are used as a predictive model that maps observations about data to make conclusions about the target value. Branches represent the data attributes that are found in observation and the conclusions about the target value are the leaves. When learning the data, the training set is divided into subsets based on an attribute value test, which is repeated on each of the derived subsets recursively. Once the subset at a node has a value equal to the target, the recursion stops. In the case of classification, the output of random forest is the class selected by the most trees. A depiction of this method can be seen in Fig. 2.3.

Artificial neural networks (NN) are another subset of machine learning, inspired by the human brain by mimicking how biological neurons work. The basis of a NN

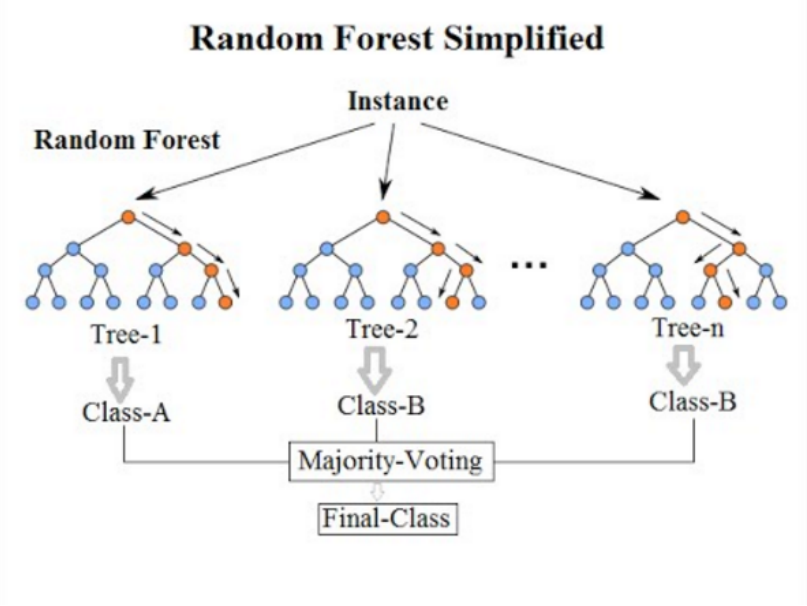


Figure 2.3: Illustration of a random forest showing multiple decision trees. Illustration by Venkata Jagannath is licensed under CC BY-SA 4.0.

are nodes, called artificial neurons, which are mathematical functions that collect and classify information. Each connection acts like the synapses in the brain and transmits a signal to other neurons. Once the artificial neuron receives and processes a signal, it communicates to the neurons connected to it. NNs contain layers of interconnected nodes with an input layer, one or more hidden layers, and an output layer. Each node has a weight (representing the significance of the node) and a threshold associated with it. All weighted inputs in a layer are summed and then passed through an activation function to determine the output. If the output exceeds the threshold, it triggers the node and sends data to the next layer. Data is not passed on to the next layer if the output does not exceed the threshold. For supervised learning, we evaluate the algorithm's accuracy as it trains using a cost (or loss) function, typically the mean squared error [36].

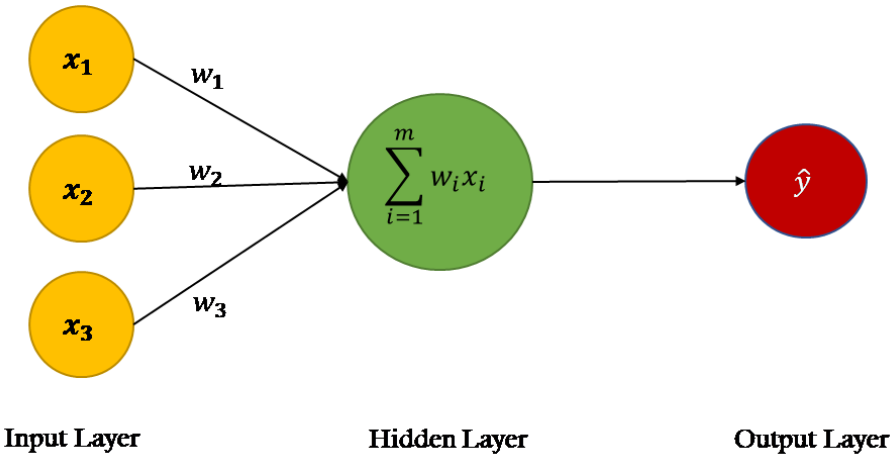


Figure 2.4: General model of an artificial neural network, where the input values x_n are multiplied by a weight w_n to reflect their relative importance to the determination of an output target. The neuron adds up the weighted inputs, and the activation function determines whether the summed input meets the set threshold value. Figure from [36].

2.2 Underfitting and Overfitting

Many computational methods have similar concepts that are important to understand. In this section, I will introduce some of those ideas before describing specific techniques.

An important technique in machine learning and model building is how well the model is fit to the data. Typically, data is split into training and testing samples, where the model is built using the training sample, but the evaluation of the success of the model is performed on the testing sample [96]. This division helps prevent the model from only being able to describe the given data well but failing to extrapolate to new data.

A more advanced technique of model evaluation is k-fold cross-validation. In this case, the initial sample is partitioned into k equally sized samples at random. Of the k samples, one sample is reserved to be used as the testing sample, which is used to validate the training the algorithm has undergone [42]. The rest of the k-1 samples are then used as the training sample. This process is repeated k times, where each of the k samples is used once as the testing sample. The k results are then averaged to produce an estimate.

Overfitting occurs when a model fits the particular set of data too closely and fails to model additional data accurately [96, 108]. An example of overfitting is shown in Fig. 2.5. Typically, overfit models are too complex with more variables than necessary. Instead of gaining a universal understanding of the population data, an overfit model will have a complete understanding of the sample training data (including noise).

Conversely, underfitting occurs when the model is too simplistic and fails to describe the data accurately. Underfit models typically have too few variables or parameters and thus inaccurately predict both the training and testing samples.

In practice, underfitting is generally avoided, because the model fails to perform well on all data sets. It can be much more challenging to identify and prevent overfitting. Using training and testing samples can help reduce the possibility of overfitting since the evaluation of the model is done on a data set different than the training sample. However, this does not completely eliminate the possibility because the model could still be overfitted on the training data and perform well enough on the testing sample. One solution is to attempt to produce a model with the fewest number of parameters. For example, if two models both describe a testing sample the same, but

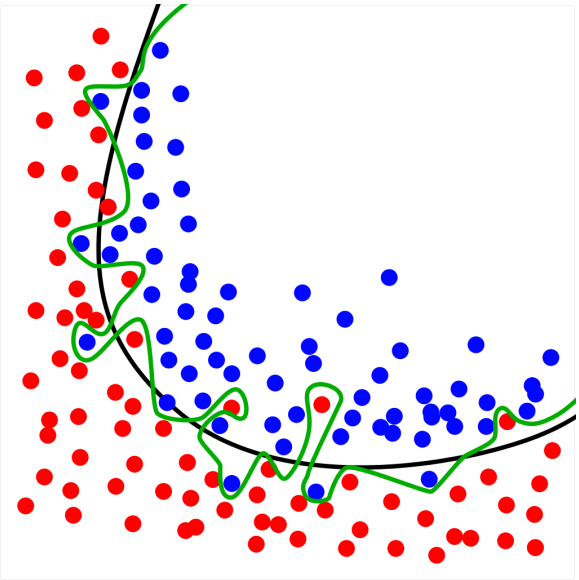


Figure 2.5: An illustrated example of overfitting a data set. While the green line perfectly separates both colors, it is overly complicated and unlikely to perform well on an unseen data set. The black line depicts a model that is properly fitted, with slightly more error on this data set, but will likely perform just as well on new data. Illustration by Ignacio Icke is licensed under CC BY-SA 4.0.

one uses four variables and the other uses ten, the model with for variables would be preferred. Some techniques will consequently penalize more complex models.

2.3 Genetic Algorithms

GAs are a type of evolutionary computation used to discover possible solutions to complex problems with a large parameter space [42]. The resulting multiple generation convergence can evolve solutions for problems that would have otherwise been difficult or not possible through more traditional techniques [42, 115].

Each GA consists of multiple individuals, which are potential solutions characterized by a set of genes. Genes define components of the object being optimized and are used to define a model or individual. Genes can be represented by using an

integer value, Boolean, or multi-unit array. Borrowing from biology, individuals are sometimes called chromosomes, as they are a collection of genes. A population describes a group of individuals all belonging to the same iteration/generation. As with many computation-based optimization algorithms, GAs are a process that optimizes solutions through an iterative approach. We define these iterations as our generation.

Each generation is tested against a predefined set of goals. To quantify the performance of an individual, a fitness function is created that generates a fitness score by using the genes of an individual to assess its performance based on an optimal or desired goal. This value informs selection methods which pick individuals to create offspring. Each generation of individuals has the potential, but is not guaranteed, to improve upon the prior [102]. After the individuals are assessed for their fitness to achieve the desired goal, we select parents to breed and populate the next generation using selection methods and operators.

The GA workflow is presented in Figure 2.6. First, a population of individuals is generated, each with randomly generated genes. Each individual is tested through the fitness function to generate a fitness score. A selection method is implemented to decide which individuals, called parents, will create the next generation. An operator then uses the parents' genetic code to create offspring individuals for the next generation [68]. The fitness test is applied to each new individual, generating their fitness scores. This process is iterated until one or more individuals surpass a specified fitness score threshold, or until a specified number of generations have passed.

There are many different techniques to create the new generation. Assuming the GA has proper structures, such as a variety of selection methods and operators, to

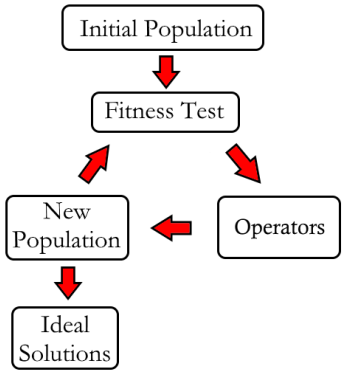


Figure 2.6: General GA evolution procedure.

maintain genetic diversity (and therefore prevent a local convergence), any combination of techniques will find the same optimized solutions. The methods of generating new populations will affect the speed of convergence, and the size of the parameter space searched. The various methods discussed in the following sections are only a small subset of the possible methods. New or unique methods are regularly created to fit the needs of individual problems.

2.3.1 GA Fitness Evaluation

Fitness functions are essential in GAs, as this is what guides the evolution toward optimal design solutions. Fitness functions are a techniques that provides a single figure of merit, which tells us how good the individual is with respect to the problem we are attempting to solve. The goal is to create future populations with higher scoring individuals and tend away from the worst-performing individuals, while still fully exploring the parameter space.

Each individual generated must be evaluated by the fitness function to obtain a fitness score. The evaluation of a fitness function is run repeatedly when using GAs;

thus, we need to ensure that it is fast. Often the easiest way to ensure speed is to have a fitness function that maximizes or minimizes the given objective measurement.

Fitness scores must be carefully designed to evaluate the solution properly, and they can come in many forms. Often an objective function provides a score that is scaled to produce the final fitness function [80]. Fitness scores can be designed where the ideal score is either maximized or minimized.

2.3.2 GA Selection Methods

Selection methods are methods used to choose individuals from the current generation to act as parents, producing the children for the next generation [62]. Each type of selection method will promote convergence or diversity to varying degrees [30]. GAs often combine different selection methods and operators in order to ensure genetic diversity. There is a multitude of different standard selection methods, although it is common for researchers to design unique selection methods [81, 52]. The following section will discuss the most common methods: (1) roulette, (2) tournament, (3) rank, and (4) elite.

Roulette selection, or proportional selection, is a popular selection method used in GAs, where the probability of selection is based on the ratio of individual fitness to the total fitness. For this reason, it is also called fitness-proportionate selection. The probability of selection of individual i from a population of N individuals is represented by

$$P(i) = \frac{F(i)}{\sum_{n=1}^{N} F(j)} \tag{2.1}$$

Roulette selection is often visualized as a pie chart where the area occupied by each individual on the roulette wheel, as shown in Fig. 2.7, is proportional to that ratio of individual fitness to total fitness [81, 52]. This area is set up such that the sum of all individual's areas is 1. A random number is chosen between 0 and 1, and whichever individual's area it lands within is declared the winner and is passed onto the operator. Individuals with better fitness values will occupy a bigger area in the pie chart and, thus, will have a higher probability of being selected.

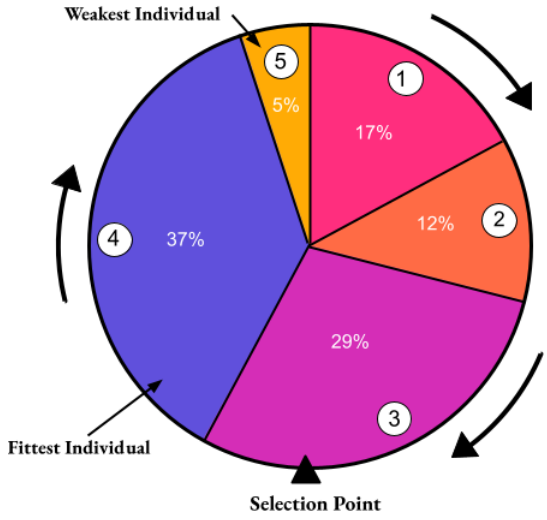


Figure 2.7: Illustration of roulette selection.

Roulette functions can be linear or non-linear. For linear functions, as used in our applications, it will give a larger area of the pie chart to larger fitness scores and distribute the pie slices proportionally. Using non-linear functions, however, can allow customization of the selection. For example, the use of functions like $P(i) \propto \sqrt{F}$, or $P(i) \propto \log(F+1)$ allows for lower fitness scored individuals to have a higher

probability of selection, as compared to their probabilities in a linear function. Alternatively, suppose we are looking to prioritize small differences in the fitness score or are nearing the end of evolution and wish to be more elitist, we can use functions like $P(i) \propto F^2$ or $P(i) \propto e^F$, which will give extra weight to higher scores [58].

Tournament selection is another method of selecting an individual from a population of individuals. In tournament selection, individuals are randomly placed into groups (tournaments) and compared by fitness scores. The individual(s) with the highest fitness score in each group, or tournament, are then selected as parents [117]. The probability of a participant's selection can be adjusted by changing the tournament size; if the tournament size is larger, weaker performing individuals have less of a chance of being selected, as there is a higher probability of a stronger individual existing in that group [90].

In rank selection, the N individuals in a population are sorted based on their fitness score, and they are each assigned a rank, where the best individual is assigned a rank of N, and the worst individual is assigned a rank of 1 [30, 62]. The rank of an individual i is represented by $\text{rank}(i)$. Then the probability of being selected is

$$P(i) = \frac{2 \text{ rank}(i)}{N(N+1)}$$

The selection of parents is then done in the same method as the roulette method. However, rank selection performs better than roulette when fitness scores are close together, which effectively causes the roulette method to be a random selection [62].

In order to ensure high fitness individuals survive, elite selection (or elitism) can be utilized. In elitism, one or more individuals with the highest fitness score from the last generation are carried over, without any changes in genes, to the next generation.

This allows us to improve the GA's performance and ensure that the best individuals are continuing to participate.

2.3.3 GA Operators

Genetic operators are the primary methods of producing new gene variations after the parents have been selected using selection methods. Genetic operators are techniques used to modify the parent(s) genes to produce a child (new individual) in the next generation. This section focuses on the following types of genetic operators utilized in the GENETIS studies: (1) crossover, (2) mutation, and (3) reproduction. Note that these operators could be run in parallel by with different groups of the parents using only one genetic operator each or with groups of parents going through multiple operators in a row to produce offspring.

Crossover is one method to stochastically generate new individuals from a previous population. Crossover typically uses two parents to produce two new offspring. One-point crossover chooses a location, designated as the crossover point, on a vector containing both parents' genes at random. As seen in Fig 2.8, genes to the right of the crossover point are swapped between the two parents. The result gives two children that contain genetic information passed on from both parents. Two-point crossover chooses two points on both parents' sets of genes at random. Genes in between the two crossover points are swapped between the parents, as seen in Fig 2.9.

Similarly, k-point crossover is a generalization of this strategy where k crossover points are selected. Finally, there is uniform crossover. For uniform crossover, the genes are no longer broken up into segments by choosing crossover points; instead,

each gene is treated separately and, using equal probability, we choose which gene to include in the child. An illustration of this can be seen in Fig. 2.10. This method can also be repeated using a higher weighted probability on genes for one parent to bias those genetics. This method can produce one or two children and is not bound to the same number of offspring as parents, whereas k-point crossover always produces the same number of offspring as parents.

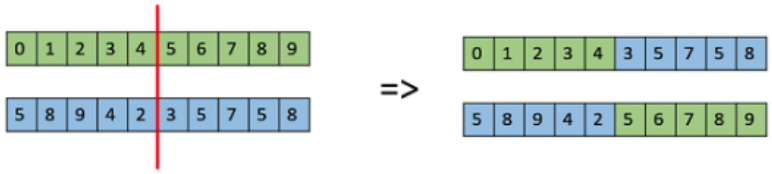


Figure 2.8: One-point crossover, where one point is randomly selected and everything to the right of that point is swapped between the two parents. Illustration from [94].

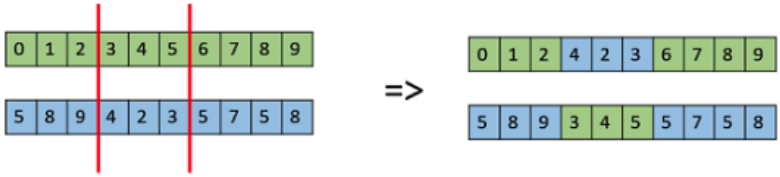


Figure 2.9: Two-point crossover, where two points are randomly selected and alternating segments are swapped between the two parents. Illustration from [94].

Mutation is a process that utilizes only one parent and produces a child by altering the value of one or more genes. The mutation operator can also be used on a child recently created by another genetic operator. Mutation is a type of genetic operator that is attractive for its ability to maintain genetic diversity in a population, which is

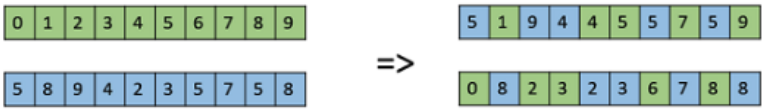


Figure 2.10: Uniform crossover, where, with equal probability, we select one of the genes from either of the two parents for each gene passed to the offspring; in this case each gene is treated separately. Illustration from [94].

important for avoiding convergence at a locally optimal solution instead of a globally optimal one.

Similar to crossover, there are many approaches to mutation, such as swap mutation, scramble mutation, uniform mutation, and Gaussian mutation. With swap mutation, two of the gene positions are selected at random, and the gene values are swapped with each other, as seen in Fig. 2.11. This is only possible when the gene values can be reasonably exchanged. Scramble mutation has a similar approach, where instead of choosing two gene values and swapping them, a subset of genes are chosen and their values are randomly reorganized. An example of this method is shown in Fig. 2.12. Finally, the last methods are uniform and Gaussian mutation. These methods are relatively similar, as they both require selecting one or more genes on a single individual and reassigning each gene a new value. For uniform mutation, the new value is selected from a range of allowed values, with each value having an equal probability of selection. For Gaussian mutation, our new gene value is selected based on a Gaussian distribution centered around the original parent's value for that gene (or a user selected value) and with a predetermined width. It is also possible to mutate every gene of an individual; this is sometimes known as immigration since it is effectively introducing a new individual.

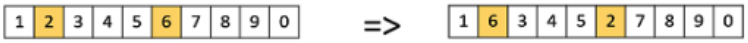


Figure 2.11: Swap mutation operator, where two genes on one individual are selected at random and the values are swapped. Illustration from [95].

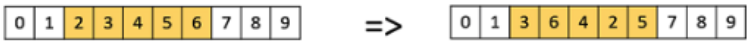


Figure 2.12: Scramble mutation operator, where a subset of genes on one individual are selected at random and the values are shuffled. Illustration from [95].

Finally, our last operator to be discussed in this review is reproduction (also known as cloning or copying). Reproduction works by taking a single selected parent and passing its genes directly to the child without modification. This is a form of elitism; however, in this case, it is not necessarily taking an individual with the highest fitness score over the entire population, as the best overall may not have survived the selection process. Thus, it is taking one of the best performing individuals that (1) survived the selection process and (2) exists within the group of parents making their offspring via the reproduction operator.

2.4 Genetic Programming

Genetic Programming (GP) is a type of evolutionary algorithm extremely similar to GAs, with the distinct difference that GAs evolve chromosomes that contain values of genes. In contrast, GPs evolves individual programs [96, 108]. The simplest

example of a program that can be evolved is an equation or a model. GPs operate with the same steps as GAs shown in Fig. 2.6, but the individuals are programs.

While a GA explores the solution space, a GP explores the program space, where the program could explain the solution [96]. For example, a GA may attempt to evolve genes that describe a specific geometry. A GP would attempt the same problem by evolving a mathematical function (program) that is maximized when the geometry is correct. Each individual in the GP would be an equation comprised of the variables combined using various mathematical operators.

The simplest example of GP, and still one of the most common techniques, is a tree-based equation. In these GPs, the program is represented with a syntax tree structure, where each node is a mathematical operator or an operand [96]. The operand nodes are always at the end of a branch and are also called leaves. An example of this can be seen in Fig. 2.13, which shows an example tree structure.

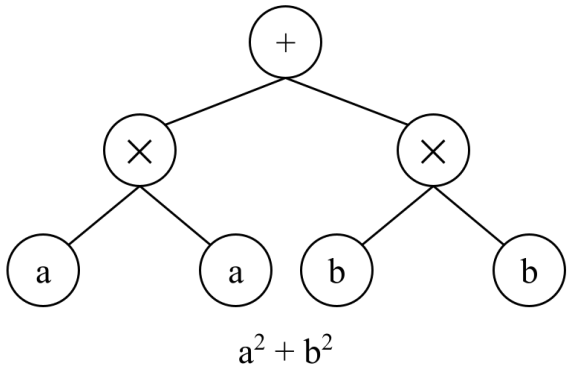


Figure 2.13: Representation of the $a^2 + b^2$ with a tree structure.

Many possible operators can be used in GP syntax trees. Besides the elementary mathematical operators (+, −, ×, and ÷), more complex mathematics operators (exponential terms, log, ln, $\sqrt{}$, exp, $\sum$, etc), trigonometric functions (cos, sin, tan, etc), and Boolean operators (AND, OR,). It is also possible to design unique or specific function terms. Every operator has an *arity* value, which describes the number of nodes that it acts on [96]. The operands are the user-defined variables and coefficients that the operators act on.

Trees can be described by the number of nodes they contain and the tree depth. The tree depth is the number of layers, starting with 0 at the topmost node. For example, the tree shown in Fig. 2.13 has a depth of 3. Tree depth is a vital user parameter because restricting it can help reduce overfitting and bloat. As previously mentioned, more complex solutions are more likely to be overfit, with the additional complexity leading only to capturing additional noise in the data. The number of nodes is related to the depth of a Full tree by the following equation, n is the number of nodes and d is depth [107].

$$n = 2^{d+1} - 1 \tag{2.2}$$

Tab. 2.1 shows the number of nodes for some possible depths. Notably, the hypothesis equation can become highly complicated at relatively low depths.

There are two common structures in syntax trees, called Full and Grow [96]. The Full structure is symmetrical, with each branch reaching the same final depth. The Grow structure allows for each branch to have a different depth. Grow trees are generally preferred because they can evolve to more straightforward solutions. While a Full tree can often achieve the same simplified equation as a Grow tree, it would

Table 2.1: Relationship between the depth of a Full tree and the number of nodes

Depth	Nodes
0	1
1	3
2	7
3	15
4	31
5	63
6	127
7	255
8	511
9	1023
10	2047

require operands canceling out and a more complex structure to achieve the same result. The difference between Full and Grow trees can be seen in Fig. 2.14. Grow trees can suffer from being too simplified and failing to produce the desired result if a more complex structure is required to describe the data.

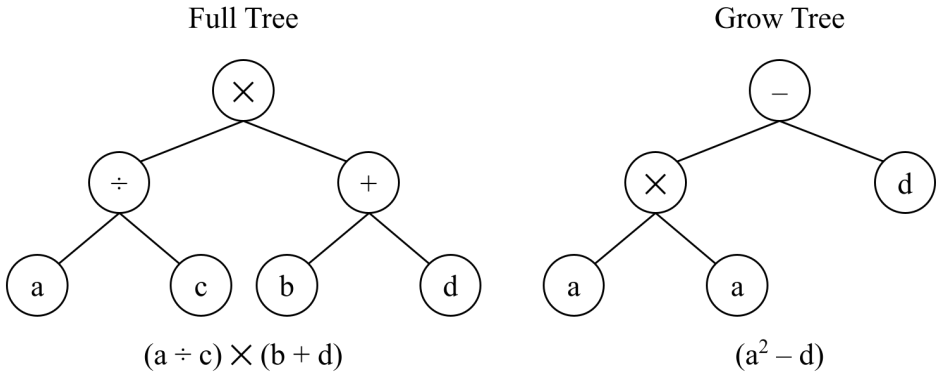


Figure 2.14: Illustration of a Full tree structure (left) and a Grow tree structure (right).

The most common technique for GP tree structures is called the Ramped Half/Half method [96]. In the ramped method, the initial population of trees is built with half as Full trees and half as Grow trees. All subsequent generations allow for the individual's structure to be "grow". The mixed initialization allows for a high level of complexity to be introduced initially, with future generations having improved genetic diversity.

2.4.1 GP Operators

GP fitness functions and selection methods operate similarly to GAs, so they will not be discussed in detail here. Genetic operators are conceptually the same for each type of algorithm but behave somewhat differently for tree structures. GPs use four main genetic operators to search the programmatic space, although unique or customized operators are possible. Unlike the operators used in GAs, these modify the structure of the tree itself.

The first operator is reproduction and acts in the same fashion as described for GAs, where the individual is copied directly into the next generation.

The next type of genetic operator is point mutation, which takes a single parent and modifies one node to produce a child, as illustrated in 2.15 [96]. Note that any of the nodes, including mathematics operators, could be modified.

Another genetic operator used by GPs is branch mutation, where each node of a branch of a single parent tree is modified. First, a node is selected from the tree. Next, that node and each of the nodes lower on the tree are modified. If the individual is either a ramped or grow tree, the size of the branch can also be modified, allowing it to grow or shrink [96]. Note that if a terminal node is selected, branch mutation behaves in the same fashion as point mutation. An example is presented in Fig. 2.16.

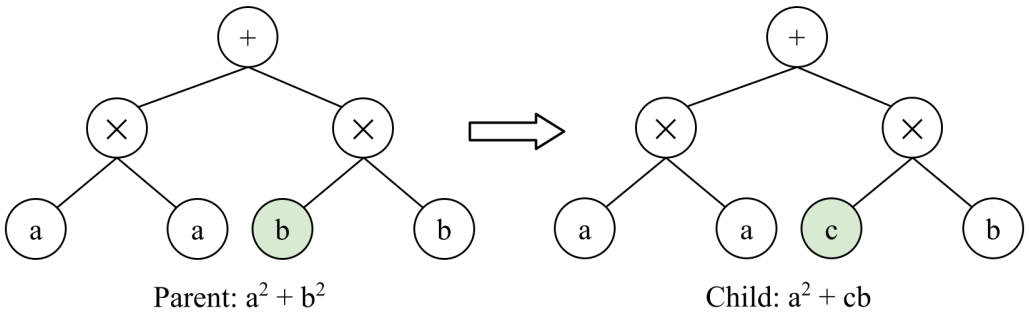


Figure 2.15: Representation of the point mutation genetic algorithm. One node from the parent is chosen and altered.

The last genetic operator discussed is crossover. Crossover takes two parents and swaps a branch from each of them to produce two children [96]. As with branch mutation, the size and shape of the trees could change (if permitted by the settings). Also, only a single node could be altered if terminal nodes were selected from each parent. An example of crossover is given in Fig. 2.17.

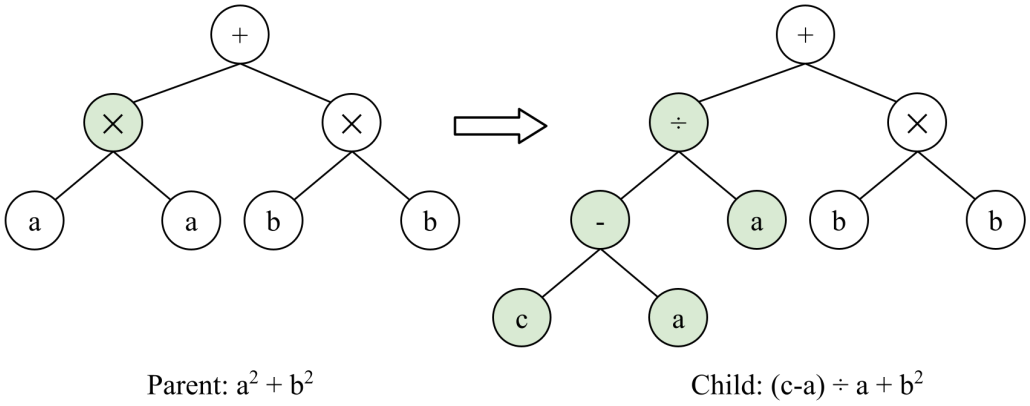


Figure 2.16: Representation of the branch mutation genetic algorithm. One branch from the parent is chosen and altered.

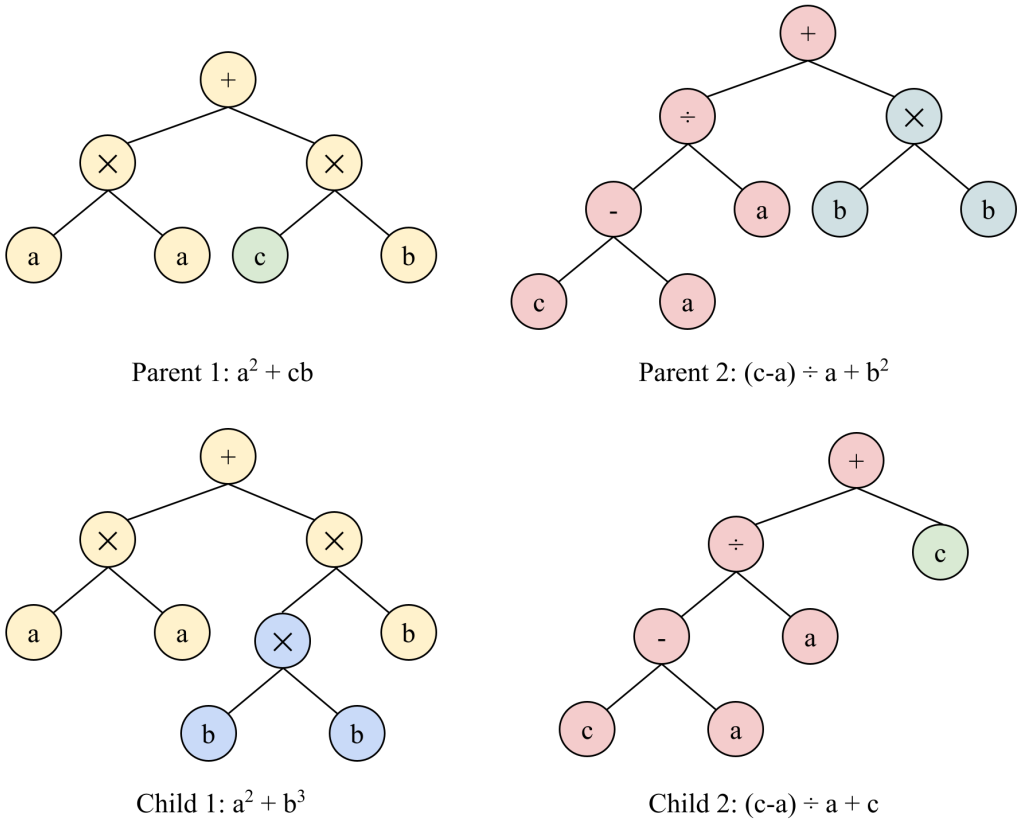


Figure 2.17: Representation of the crossover genetic algorithm. One branch from each parent is selected and swapped to produce two children.

Chapter 3: Antenna Optimization with Genetic Algorithms

3.1 Introduction to GENETIS

The goal of the Genetically Evolving NEuTrIno TeleScopes (GENETIS) project is to optimize the science outcome of detector designs in high-dimensional parameter spaces to advance the field of astroparticle physics. GENETIS was started in 2017 at The Ohio State University. As a first application, GENETIS has produced a Genetic Algorithm (GA) that evolves antenna geometries intended for ultra-high energy neutrino detection with the ARA experiment. GENETIS is rare in its application of machine learning for designing an antenna using a science outcome as the sole measure of fitness in a GA. The GENETIS group consists of very unique group of individuals, with much of the work being undergraduate-driven. This chapter begins with an introduction to the GENETIS project, then describes the early projects and the current GENETIS GA. Finally, I will discuss the two main applications of the GA: evolving physical antennas and evolving gain patterns.

GENETIS is seeking to help the search for one of the important missing piece of particle astrophysics, the detection of ultra-high energy (UHE) neutrinos at energies above about 10^{18} eV [6]. A number of experiments employ antenna arrays to detect

Askaryan radiation produced from a neutrino-ice interaction (such as those in Antarctica or Greenland) [24, 66]. These experiments include ANITA, ARA, ARIANNA, and RNO-G, which utilize a variety of different antenna types [40, 10, 20, 56, 22, 12]. The initial GENETIS goal is to explore optimizing current detector designs residing with some of these primary experiments via the use of GAs.

The high-dimensional parameter spaces of detector design problems motivate computational methods to improve upon designs made using traditional techniques. In particular, the design of antennas for UHE neutrino detection has explicit constraints and a large parameter space, which makes it well suited for machine learning. Given the immense scale of these experiments and the difficulty in detecting UHE neutrinos, each detector element must be designed to return the most science for its cost.

GAs were chosen, among other algorithms, because of their effectiveness at complex optimization problems, especially when many optima could exist [98]. The use of GAs was initially motivated by the NASA ST-5 antenna in which a GA designed a simple, segmented, wire antenna for satellite communications [69]. Many other examples exist of antenna design optimization using GAs including Yagi-Uda antennas [70], electrically loaded wire antennas [31], broadband cage antennas [45], planar antennas [60], pyramid horn antennas [44], ultra-wideband slot antennas [113], helical antennas [85], patch antennas [46], adaptive antennas [65, 79] and others [64]. GAs are also more transparent than other optimization techniques, which allows for a better understanding of how the algorithm arrived at a final result instead of a black-box model. Searching a 6-dimensional parameter space (as is the case for the asymmetric bicone discussed in Chapter 3.4.2), using increments necessary to find a peak fitness score would require evaluation of more than 10^8 designs. In comparison, the results

presented required only 1550 antenna designs to search the same parameter space with the GA.

GAs have previously been used in the design of various detectors and experiments [84, 83], although rarely used to optimize for a science outcome directly. Some examples of evolution toward a science outcome include a horn antenna designed using a GA optimized for detecting Cosmic Microwave Background radiation [87]. Another example is from both the Long Baseline Neutrino Oscillation experiment (LBNO) and the Deep Underground Neutrino Experiment (DUNE), where GAs were utilized to optimize the design of neutrino beamlines using simulations of a science outcome to determine the fitness [33, 101]. GAs have also been used to optimize the layout of detectors, sensors, shielding, and for trigger optimization [72, 53, 74, 11].

The early endeavors of GENETIS mostly involved proof of concept designs and tests that are covered in Chapter 3.2. The first project in 2017 used a GA to evolve to the known length of a quarter wavelength dipole antenna at 3 GHz. Other early work involved the evolution of a paperclip antenna toward set patterns and performance tests [100].

Chapter 3.3 describes the heart of the GENETIS project, the custom genetic algorithm. Each step of the GA is discussed in detail, covering the initialization, fitness evaluation, new generation creation, and termination. What makes the GENETIS GA impressive is the integration of various types of simulation software to generate a fitness score. This chapter and Appendix A discuss these programs in detail.

The second stage of investigation is the Physical Antenna Evolution Algorithm (PAEA) project and is discussed in Chapter 3.4. PAEA uses the GENETIS GA to investigate the optimization of the ARA collaboration's in-ice vertical polarization

bicone antennas. PAEA initially optimized a symmetric bicone antenna with linear sides, and is now exploring more complex geometry, such as (1) asymmetric bicone with linear sides and (2) asymmetric bicone with nonlinear sides. The results of each project are discussed below.

As a third investigation, GENETIS is optimizing the antenna response pattern, without any antenna designs or geometry. This investigation, called the Antenna Response Evolutionary Algorithm (AREA), is being run with minimal constraints, as the goal of this project is to explore what improvement to the neutrino sensitivity is possible due to improvements in antenna responses alone, without regard to what physical design might be needed to bring about that response. The results of this investigation are presented in Chapter 3.5.

My contributions to GENETIS involve heavy involvement in creating the loop and, more recently, as a mentor and leader for the group. Over time, the early code has grown, and our network of primary contributors, which now includes myself, 11 undergraduate students, and experts in the fields of GAs, ML, antennas, and additive manufacturing. As one of the significant architects of our current software package, I have been guiding all GENETIS projects and working alongside undergraduate students in the drive to make meaningful contributions. I wrote the user guide to the loop, which is necessary to understand all of the complex moving parts. Additionally, I built a training course that on-boards students and prepares them to contribute. This course includes information on the big picture of UHE neutrino experiments, GAs, and the GENETIS group, and instructs them in the principles of coding in the different languages necessary to improve the GA. Furthermore, I led weekly planning

and working meetings and developed task tracking and prioritization sheets to help students stay on track.

3.2 Early GENETIS Investigations

Before getting into the optimization of the antennas, GENETIS produced proof-of-concept studies to understand GAs better and build a foundation of knowledge. In this section, we will briefly discuss some of the details of our earlier studies.

3.2.1 Quarter-Wavelength Dipole Antenna

As our first proof-of-concept, the GENETIS team used a GA to recreate the length solution for a quarter-wavelength dipole at 3 GHz. Since this is a problem with a known solution, this helped us explore both the effectiveness of a GA at finding the solution and the number of generations it takes on average to arrive at the correct answer.

The quarter-wavelength dipole study was the first version of our loop software. It was similar to the current version; however, the fitness evaluation was more straightforward. This GA began the loop by randomly generating genes for the initial population (radius and length). Genes for the initial population were initialized via a uniform distribution, converted to a normal distribution using the Box Muller method. The Box Muller method can be used to convert a pairs of numbers that are uniformly distributed to numbers that are normally distributed [32]. The fitness function involved using XFdtd, an antenna simulation software, to produce gain patterns for each solution. The fitness score was simply the measure of peak gain relative to that expected for a quarter-wavelength dipole at 3 GHz. Solutions with a smaller difference in gain pattern to the expected result were scored as performing better. The GA modified the

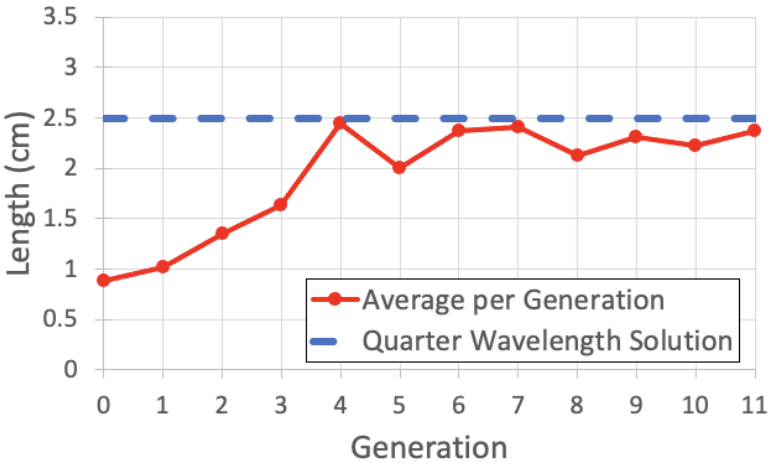


Figure 3.1: Results for the evolution of a quarter-wavelength dipole antenna solution showing the improvement of the average length toward the known solution.

generations using mutation, where each individual's genes were mutated and passed on to the next generation; thus, all individuals went through the mutation operator to construct the next generation, and no selection methods were used. Each individual's genes were mutated through a Gaussian distribution centered on the current value, with a standard deviation proportional to the fitness score. The GA converged by adjusting the magnitude of the mutations based on the fitness score. If an individual was closer to the solution than prior generations, it mutated with a standard deviation 0.9 times the last deviation. If the individual were further from the solution, it would mutate using a standard deviation of the prior deviation divided by 0.9.

As seen in Fig. 3.1, the results of this study show that with a population of 5 individuals and 13 generations, the design converged to the expected result very quickly. The best individuals matched the expected results. The average per generation doesn't quite reach the solution because the mutation operator is constantly

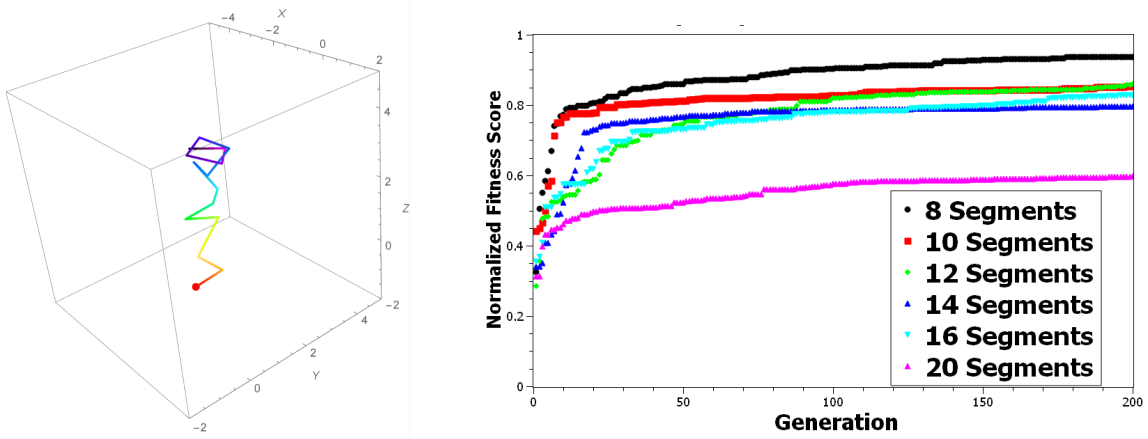
adjusting each individual. This results provided an initial proof of concept for the use of a GA for antenna design and laid an initial foundation for future GENETIS endeavours.

3.2.2 Paperclip Antenna Evolution

The subsequent proof-of-concept GENETIS investigation examined a simple, segmented wire antenna design modeled on the antenna design evolved for NASA satellite communications in 2006 [69]. The following provides the overview and results of our investigation. Suren Gourapura contributed to this project. The main goal of this investigation was to further develop the GA with a known type of antenna design and various types of fitness functions.

The antenna geometry consists of multiple, unit-length segments of wire connected sequentially. Due to the segmented and bent nature of the resulting antenna, we call this design a "paperclip antenna." Each segment can point toward any direction. Thus, the genes that define each individual are the three rotation angles between 0 and 2π about the three Cartesian axes for each segment. The final individual geometry consists of several randomly rotated unit vectors attached tip to tail. The Euler angles for each segment are initially uniformly distributed from 0 and 2π.

While several different fitness functions were explored, one, in particular, directed the evolution of the antenna to arrive at a desired counterclockwise spiral shape, as illustrated in Fig. 3.2. This "curl" function was sensitive to changes in the initial parameters and complex in that all of the evolved rotations had to work together to produce the final shape, which made it a good test of the algorithm.



(a) (b)

Figure 3.2: (a) Example of a partially evolved paperclip antenna. Note the general counterclockwise spiral. (b) The best fitness score of 100 paperclip antennas over 200 generations improved as the antennas evolved to produce a counterclockwise curl. The GA was performed for various number of segments [100].

The curl fitness function calculates the cross-product between adjacent vectors, $\vec{s_i}$ and $\vec{s}_{i+1}$. The fitness score, F, defined by the equation below:

$$F = \sum_{i=1}^{n-1} \vec{s_i} \times \vec{s}_{i+1}$$

Defined this way, the angle between neighboring antenna segments is preferred to be 90° and oriented counterclockwise in the x-y plane.

The paperclip antennas evolved over 200 generations composed of 100 individuals using a tournament selection method and a combination of mutation and crossover operators. Antennas with various numbers of segments were tested. Fig. 3.2 presents the results of this analysis, which shows the highest scoring designs per generation for different segment lengths. As shown, fewer segments result in achieving a higher fitness score over fewer generations. Also, a higher quantity of segments increases

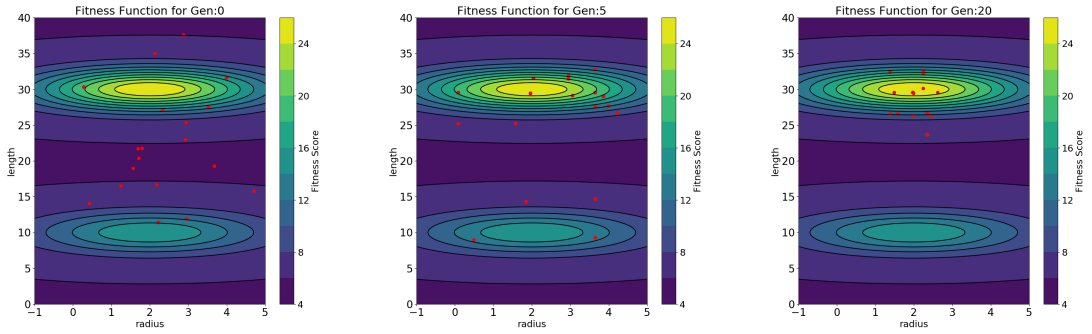
the complexity of the antennas, thereby resulting in a slower approach to the desired solution.

3.2.3 GA Performance Test

In order to test the performance of the algorithm in the presence of local and global maximums, individuals were evolved with two genes called length and radius (although in this case the names are irrelevant as they just represent two parameters), where the fitness function is the sum of two displaced Gaussian distributions of different heights. Suren Gourapura was a main contributed to this investigation. In this case, the genes In the first generation (Generation 0), the 20 individuals, shown as red dots, covered a region of parameter space that contained both Gaussian distributions. By the 20^{th} generation, 19 of the 20 individuals were within 2σ of the global maximum, despite some individuals finding the local maximum in earlier generations. Some initial results are presented in Fig. 3.3. These initial investigations demonstrated the viability of using a GA to solve complex problems and laid the foundation for the evolution of antennas for UHE neutrino experiments.

3.3 The Current GENETIS GA

GENETIS prides itself in the development of our unique software package. This package, referred to as "the loop," integrates a GA with commercial antenna simulation software, Monte Carlo neutrino simulation software, and more. This software works together to simulate the testing and performance of the antenna designs. The remainder of this section will describe the intricacies of the PAEA GA. The AREA GA utilizes similar mechanisms; differences in the software loop for the AREA GA will be provided in the AREA section, Ch. 3.5.



(a) Generation 0 (b) Generation 5 (c) Generation 20

Figure 3.3: Example of PAEA algorithm results at (a) Generation 0, (b) Generation 5, (c) Generation 20. The fitness score is shown in contour plot. Individuals, shown as red dots, began spread over a wide range, and evolved to group near the global maximum, despite some finding the local maximum initially [100].

The loop begins by creating the initial population of solutions. The genes are the parameters that describe the geometry of an antenna, where each antenna is an individual. After the GA builds the initial set of individuals, it passes through to the next stage, where we test and assign fitness scores. Our fitness evaluation consists of simulating the antenna response pattern in XFdtd and testing the sensitivity of the ARA detector when that individual is used for its VPol antennas using AraSim, a Monte Carlo simulation software that simulates the in-ice environment for ARA. Once the performance of the individuals in a generation are assessed in the ARA in-ice simulation software, we can apply our selection methods and operators to generate the next solutions to be tested. A general summary of this loop can be seen in Fig. 3.4. In the remainder of this section, I will be going over the details of each step in the loop.

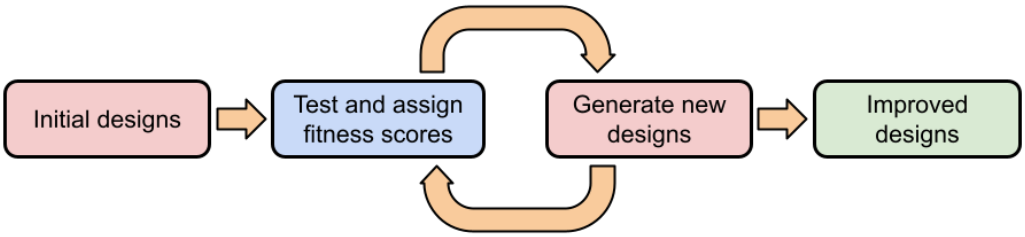


Figure 3.4: Simplified flowchart of the GENETIS GA.

3.3.1 Initialization

The initialization is the beginning of the GA. For the first generation, each gene is selected either (1) from a uniform distribution between a set maximum and minimum value or (2) at random within a maximum or minimum value; which method we utilize differs depending on the run. For early investigations, such as the symmetric bicone, method (1) is used. For all later investigations, method (2) is used.

The genes fully define the geometry of each antenna. Each individual is forced to follow certain constraints. For antenna designs, the main constraint prevents the antenna from being too large to fit in the ARA boreholes (25 cm). While no required borehole diameter clearance was specified in the GA, ARA utilized a borehole clearance of 1.1 cm for the VPol antennas and 2.3 cm for the HPol antennas [23]. Future experiments may utilize larger boreholes (over 28 cm in diameter) which would improve the design sensitivity [13].

The GA also constrains the minimum length of a design due to the limitations of the simulation software. Simulations of antenna gains become unreliable when the frequencies being simulated are outside of the antenna's bandwidth. Treating each half of the bicone as a quarter-wave dipole, the minimum length of the antenna is

found using: $L = \frac{c}{4f}$ where L is the length, c is the speed of light in a vacuum, and f is the minimum frequency. As power for Askaryan radiation is linear with frequency, low power galactic noise dominates Askaryan radiation below 100 MHz [18]. A minimum frequency of approximately 100 MHz gives a full length of 75 cm (each side 37.5 cm), which is defined as the minimum length allowed by the algorithm. The higher frequencies we are testing could have a smaller length (7.5 cm at 1000 MHz), but these would not be valid at the lower frequency ranges.

3.3.2 Fitness Evaluation

Once every individual in a generation is defined, the fitness score of each individual must be determined. The calculation of fitness scores is a multi-step process with two primary programs integrated with the GA. First, the gain pattern of each individual is simulated. Second, the effectiveness of the design at detecting neutrino radio signals is simulated and used as the fitness score.

3.3.2.1 Gain Pattern Simulation with XFdtd

XFdtd is a computational electromagnetism simulation software developed by REMCOM using the finite difference time domain method for calculations [86]. XFdtd is a crucial part of our software package, calculating the antenna response of our designs. The antenna and antenna properties are simulated in XFdtd by hitting each model with an artificial burst of radiation to calculate gain patterns.

XFdtd performs simulations by solving Maxwell's equations in the time domain [99, 78]. In this approach, the geometry of the device and the surrounding space are divided into small discrete cubic segments, called cells, that have associated field lines in each direction. The size of the cells must be small compared to the wavelength

of the EM waves. The simulation takes small steps forward in time, with the size of the step based on the time it takes for a field to travel between each cell. For each segment of time, the electric fields are calculated and then the magnetic fields. Since every cell in the simulation space is directly adjacent to others, the fields from one cell will impact the surrounding cells in each subsequent segment of time. Each cell has given material properties associated with it, with appropriate boundary conditions. Excitation conditions, like a pulse or constant EM wave, allow for the response of the provided geometries to be tested. The calculations continue until a steady-state position is reached.

In the context of the GA, the antenna geometry parameters are passed into XFdtd and modeled accordingly. However, XFdtd was built to be operated manually, with a graphic interface, so a customized back-end script had to be built to operate XFdtd without manually building each design. XFdtd then simulates the response of an individual to radiation with a particular wavelength and direction. The simulation needs to be repeated for each individual for the full range of theta (0 – 180) and phi (0 – 360), stepping by 5 degrees, until a complete gain pattern is produced for that single frequency. That process is repeated for each individual 60 times, gathering antenna response patterns for a range of frequencies between $83.33\,\text{MHz} - 1.066\,\text{GHz}$ in equal steps. The results are in the form of gain (dBi) for a range of frequencies in all directions. Note that the AREA project does not use XFdtd, as the goal is to evolve gain patterns directly (instead of antennas). Thus, this project bypasses XFdtd, with the remainder of the loop operating in the same fashion as PAEA.

3.3.2.2 Effective Volume Simulation with AraSim

For the second step in calculating the fitness score, the gain patterns output by XFdtd are used to calculate the effective volume of the ARA experiment for each individual antenna design produced in the evolution. The effective volume is the final fitness score of the individual.

Developed by the ARA collaboration, AraSim is a Monte Carlo neutrino detection simulator that is able to model neutrinos with energies between $E_\nu = 10^{17} - 10^{21}$ eV [67, 18]. AraSim simulates high-energy neutrino interactions in the Antarctic ice, which produce electromagnetic showers resulting in the production of Askaryan radiation. AraSim uniformly distributes these interactions within a cylindrical volume with a 3 km radius centered around the detector [67]. The direction of the incoming neutrino is randomly distributed over a solid angle of 4π. The radio emission propagation is modeled using ray tracing, which determines the path length from the interaction to the detector. The ray tracing models the depth-dependent index of refraction of the ice, which is n=1.3 at the surface to n=1.8 at 200 m deep [67]. Because of this variable index of refraction, the electromagnetic waves emitted from the interaction bend en route from the interaction point to the antenna. AraSim then calculates the polarization, viewing angle, travel time at the receivers, and then models the system electronics, noise waveforms, and time-domain trigger [67]. The output of AraSim is a file that contains the same data in the same format (as well as additional information) as an actual ARA event, including the event waveforms.

Since the number of expected neutrinos detected is directly proportional to effective volume, we can directly use this value as the fitness score. The effective volume, $[V\Omega]_{\text{eff}}$, quantifies the volume of ice and solid angle of the sky the detector can detect

signals within, as well as the trigger efficiencies and interaction cross-sections. This is determined in AraSim by measuring the fraction of simulated events detected by the array and multiplying by the simulated ice volume and solid angle. The effective volume is given by Eq. 3.1 [76].

$$[V\Omega]_{\text{eff}} = 4\pi \; V_{\text{ice}} \; \frac{N_{\text{detected}}}{N_{\text{simulated}}} \tag{3.1}$$

Where V_{ice} is the total volume of ice simulated in AraSim, N_{detected} is the total number of neutrinos detected (accounting for trigger efficiencies and interaction cross-sections), and $N_{\text{simulated}}$ is the total number of neutrinos simulated. In this analysis, V_{ice} is given by a cylinder around the detector with a radius of 3 km, with a total volume of approximately 85 km^3. For each individual, $N_{\text{simulated}}$ is 3×10^5 neutrinos with an energy of 10^{18} eV. Simulating this number of neutrinos gives a standard deviation of 0.11 km^3str.

At the end of the generation, each individual will have been simulated, and the corresponding effective volume from AraSim will be assigned as the fitness score. Running AraSim is a computationally heavy process and is conducted using the Ohio Supercomputing Center.

3.3.3 New Generation Creation

GAs use various selection methods to decide which parents and operators will generate the offspring to create a new generation. First, selection methods are used to choose all of the parents needed to make the next generation. Second, each individual in the new generation is created using genetic operators. The selection methods and operators used by GENETIS have grown more diverse as our software has grown.

More details on the types of selection methods and operators used in our runs will be discussed in the following section on each project.

3.3.4 Iteration and Termination

The GA continues to iterate and consequently evolve individuals toward more optimal solutions. In the same fashion as the first generation, fitness scores are found for each iteration and another generation is built from the prior. The selection methods and genetic operators work together to cause evolution toward parameters more optimal for neutrino detection. The loop is terminated when either a preset number of generations is completed, a set fitness score is found, or the fitness score has plateaued.

3.3.5 Computation Time

One potential challenge in machine learning and evolutionary algorithms is slow computation times. This challenge is one GENETIS has continuously been working toward improving. Since each generation runs many XFdtd and neutrino simulations, the GENETIS GA has historically had high computation times. In an early version of the loop used to evolve a symmetric bicone antenna, the total run time was approximately 4 hours per generation for 10 individuals, with AraSim generating 100,000 neutrinos. Computational improvements were made by splitting up AraSim jobs and running them in parallel; thus, if we were to throw 100,000 neutrinos, we could instead run 10 super-computing jobs of 10,000 each for one individual. More recent versions of the GA use 50 individuals with 300,000 neutrinos. However, the computational improvements allow a generation to complete while only taking a factor of 3 longer, despite 15 times the number of neutrinos. The additional neutrinos reduce the error

Previously

XF	AraSim	Rest of Code	Total
2-3 hours	1 hour	Negligible	3-4 hours

Presently

XF	AraSim	Rest of Code	Total
30 minutes	11-12 hours	Negligible	12 hours

Figure 3.5: Run time improvements for the current GENETIS software loop. The improved run is only three times longer, despite simulating 15 times the number of neutrinos.

on the fitness score, which is related to the number of neutrinos simulated. XFdtd run time was also sped up by using a virtual desktop interface (VDI) for computation power versus the original usage of an interactive job. The breakdown of this time can be seen in Fig. 3.5.

More efforts to further improve upon our run time are in progress. Alex Patton has contributed notably to efforts to increase the speed of AraSim calculation. One project aimed at speeding up our loop involves building a neural network to predict the fitness score from the geometry based on fitness scores already calculated in previous generations, which would allow for a decrease in computation time by circumventing the simulation steps for some individuals. The neural network and initial results are discussed in more detail in Chapter 3.4.4.

3.4 Physical Antenna Evolution Algorithm (PAEA)

One of the primary goals of GENETIS is to investigate the evolution of antenna designs. In this section, I will be discussing the different investigations apropos physical antenna evolution. These GAs all involve the integration of XFdtd and AraSim to obtain the fitness scores. Fig. 3.6 presents a schematic of the GENETIS algorithm.

The results presented in this thesis involve the evolution of a bicone-based antenna. A bicone antenna consists of two cones with the openings facing opposite directions, as illustrated in Fig. 3.11. This shape was selected for an initial evolution because it is the same geometry as antennas currently deployed in the ARA experiment.

In addition to the initial symmetric bicone evolution, there have been multiple other versions of the physical antenna evolution, each with increasing complexity. The following are the iterations that will be discussed in this section: (1) The symmetric bicone antenna, (2) the asymmetric bicone evolution, and (3) the asymmetric bicone with nonlinear sides.

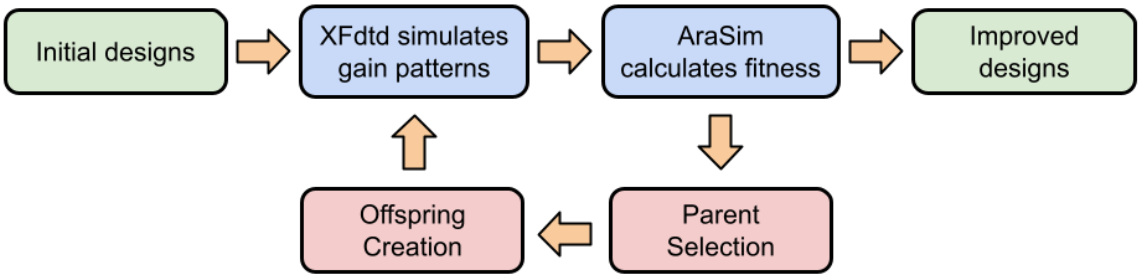


Figure 3.6: A diagram of the PAEA GA used to evolve antennas. The fitness calculation steps are shown in blue, and the generation creation steps are in red.

3.4.1 The Symmetric Bicone Evolution

The symmetric bicone antenna design was the most basic bicone evolution conducted. An notable contributor to this project was Alex Machtay. It is fully defined by three genes (parameters): the inner radius (r), the length (L), and the opening angle (θ) as seen in Fig. 3.7. Because this run is symmetric, each of these values is used to make the top and bottom cones. A single individual in the GA is an antenna design given by these three parameters.

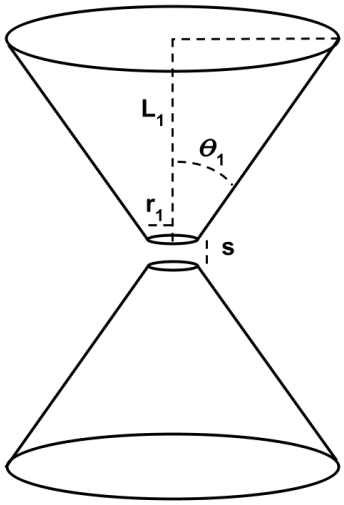


Figure 3.7: Geometry of bicone antenna showing the genes of length (l), opening angle theta (θ), and minor radius (r). The separation distance (s) is held constant.

3.4.1.1 The Symmetric Bicone Loop

The Symmetric Bicone Loop follows the procedure described in Chapter 3.3. The first generation is initialized by selecting values for the three genes for each individual

from a uniform distribution with a mean at the gene values for the current ARA designs. The parameters for the initialization can be seen in Tab. 3.1 below.

Table 3.1: Range of of uniform distributions used for each gene.

Gene	Minimum	Maximum
Length (cm)	37.5	140
Radius (cm)	0.0	7.5
Opening Angle (degrees)	0.0	11.3

Instead of a hard-coded restriction on the diameter, this GA penalizes the fitness score of individuals with an outer radius larger than the borehole size. As seen below, the penalty was implemented by making the fitness score a piecewise function, where $R_{\text{max}} = 7.5\,\text{cm}$.

$$\text{Fitness Score} = \begin{cases} V_{\text{eff}} e^{-(R_{\text{max}} - R)^2} & if\ R > R_{\text{max}} \\ V_{\text{eff}} & if\ R \leq R_{\text{max}} \end{cases}$$

If the outer radius were smaller than the radius of the borehole, the fitness score would be equal to the effective volume produced by AraSim. However, if the outer radius were larger than the radius of the borehole, then the fitness score would be penalized by multiplying the effective volume by an exponentially decreasing function, which would lower the fitness score relative to the amount the radius exceeds the borehole size. This ensured that potential solutions would become much less dense in the region of the parameter space where the value for this gene was large.

Once the fitness scores were calculated, the GA selected parents and children in a two-step process. The first step uses roulette selection, whereby pairs of individuals are selected as parents. Genes for the children are randomly selected using uniform

crossover to create two offspring. Once genes for the children were selected, they went through a second step before the evolution continued. The second step improved genetic diversity by introducing mutations to 60% of the offspring generated in the first step. This was done by modifying the values of each gene by a small perturbation drawn from a Gaussian distribution.

3.4.1.2 Symmetric Bicone Results

The results presented below utilized 50 individuals over 11 generations, simulating 30k neutrinos per individual. The results of the algorithm are presented in the violin plot in Fig. 3.8. The width of the line represents the probability density of the population. The solid red line shows the population's mean, and the dashed green line shows the population's median. This figure illustrates that although the loop was functioning correctly, the solution was not evolving over generations. There are a few potential reasons for this occurring. First, the solution could already be optimized without much room for improvement, as the geometry does not allow for more diversity than ARA's design. Second, the GA may not adequate search the parameter space, as it was not further optimized. A non-optimized GA evolves less efficiently, causing the use of excess generations prior to showing improvements.

The GENETIS team consequently decided to improve the GA further in the next iteration of the loop, the asymmetric bicone, discussed in Chapter 3.4.2 below. Fig. 3.9 shows a 3D model of the top-performing individual, and Tab. 3.2 gives the genes for that solution. It is important to note that a bug in the fitness score calculations of the symmetric bicone evolution caused the scores to be inflated. This bug was due to the polarization not being set correctly in AraSim. While this did not affect the evolutionary process (as all fitness scores were affected similarly), it does make it

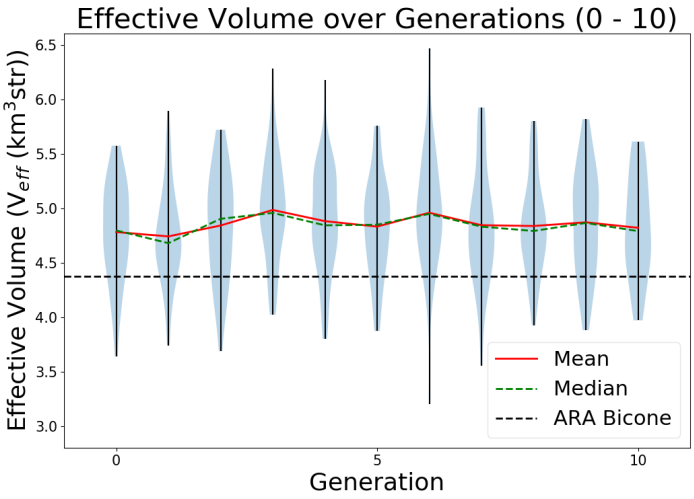


Figure 3.8: Initial results for the evolution of the symmetric bicone. The current ARA bicone fitness is shown as the horizontal dotted line. Figure by Alex Machtay.

difficult to compare the results of this evolution to later iterations. This was resolved in future GENETIS projects.

Table 3.2: Parameters of the highest scoring individual found in Generation 6.

Parameter	Value
L (cm)	38.17
r (cm)	1.56
Θ (rad)	0.036
Fitness Score	6.47

Fig. 3.10 illustrates the broad range of antenna designs and the correlation between gene value and fitness score. In particular, antennas had high scores when they had radii between 0 and 3 cm, and angles between 0 and 7 degrees. Higher scoring

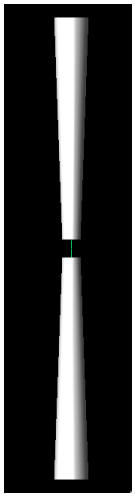


Figure 3.9: Model of the best symmetric antenna design.

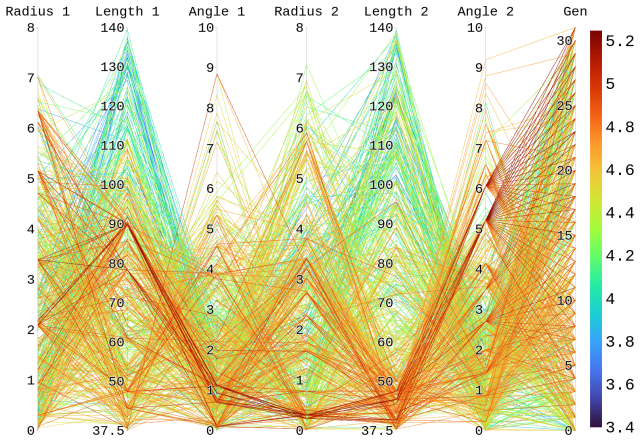


Figure 3.10: Evolution of the three antenna parameters optimized so far, showing trends toward preferred features (red being most fit). Note that in order to better differentiate the individuals, the color was set by the square of the fitness score. Figure by Alex Machtay.

antennas tended to have shorter lengths, although this could have been a remnant of the aforementioned bug.

3.4.2 The Asymmetric Bicone Evolution

In hopes of producing antenna designs with higher fitness scores, the next run allowed for more diversity in antenna shape by permitting an asymmetric evolution of the bicone antenna. A notable contributor to this project was Alex Machtay. As illustrated in Fig. 3.11, the asymmetric bicone antenna consisted of two cones with the openings facing opposite directions. The asymmetric bicone is fully defined by six genes: the inner radius(r), the length (L), and the opening angle (θ) for the top and bottom cones. Again the separation distance (s) remains constant. A single individual in the GA is an antenna design given by these six parameters.

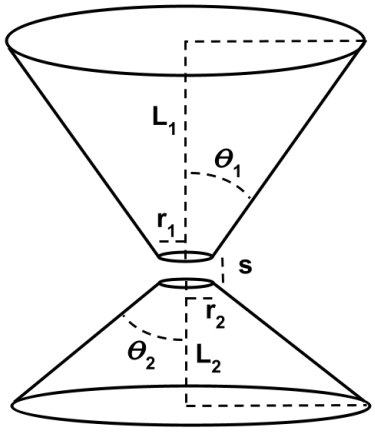


Figure 3.11: A schematic of an asymmetric bicone antenna. The lengths (L_1, L_2), inner radii (r_1, r_2), opening angles (θ_1, θ_2), and separation distance (s) fully define the geometry. In the results presented here, the separation distance was held constant, and the other six parameters were varied.

3.4.2.1 The Asymmetric Bicone Loop

The asymmetric bicone GA is similar to the symmetric bicone, except for modifications that allowed the evolution of six genes instead of three. Two other main differences will be described further in this section: (1) the constraints and (2) the selection methods and genetic operators. Additionally, it is worth noting that the GA for this run was optimized for its efficiency to determine the ratios of selection methods and operators used in the evolutionary process.

For the asymmetric run, a constraint was implemented, which prevented the outer diameter of the antenna from being larger than the ARA borehole width (both during initialization and in later generations). If the individual's genes resulted in an outer diameter larger than the borehole width, the GA would regenerate genes for that individual until the requirement was satisfied. This was a more straightforward and equally effective solution than the penalty function, which was consequently removed.

In our symmetric bicone run, all parents selected to breed were done so using the roulette method. For the asymmetric case, both roulette and tournament selection methods were utilized [82]. For each generation, the parents were split up, allowing some of the parents to be selected through roulette, while the remainder were chosen through tournament. Then, three genetic operators were used to produce the next population: reproduction, complete mutation (immigration), and uniform crossover. For each new generation, 80% of parents were selected using roulette, and 20% were selected using a tournament with 10 individuals. The new population was generated using 72% crossover, 22% complete mutation, and 6% reproduction.

The proportions for the selection methods and genetic operators were found through an optimization analysis. Different combinations of selection methods and

genetic operators were tested through an exercise where an asymmetric bicone with six parameters is evolved to a predetermined geometry. For this optimization analysis, a simplified GA was used, and the best fitness scores were determined after a set number of generations. The tests were repeated to obtain an absolute maximum fitness score (highest of all runs) and an average maximum from all runs. The results of this study can be seen in Fig. 3.12.

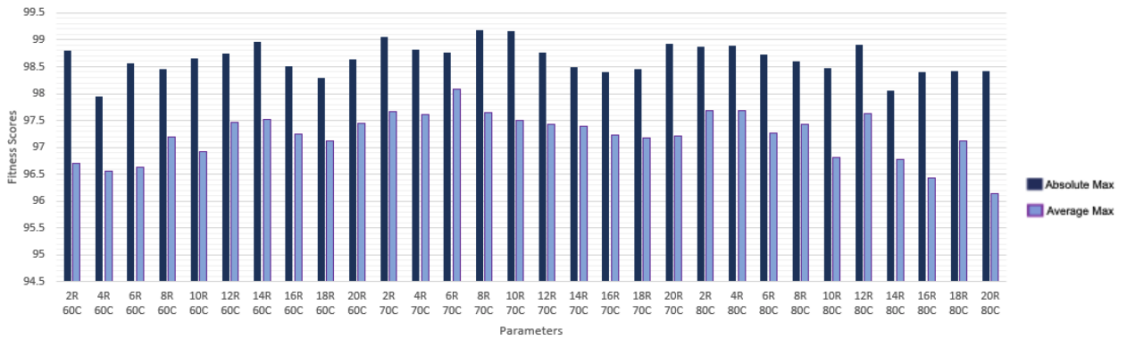


Figure 3.12: Optimization of the asymmetric bicone GA. The horizontal axis gives various combinations of GA parameters. The test was ran multiple times with each setting. The vertical axis gives the maximum fitness score obtained. The optimal ratio for selection was found to be 80% percent roulette, 20% tournament. The optimal ration of genetic operators was 72% crossover, 22% mutation, 6% reproduction. Figure by Ryan Debolt.

3.4.2.2 Asymmetric Bicone Results

The results presented below utilized 50 individuals over 31 generations, with 300,000 simulated neutrinos. The algorithm results are presented in the violin plot in Fig. 3.13, showing clear evolution toward improved solutions. The highest scoring antenna had a fitness score of $5.2 \pm 0.1\,\mathrm{km}^3\,\mathrm{sr}$, 22% higher than the current ARA detector. For each generation, the vertical lines illustrate the range from best and worst

fitness scores. The width of the line represents the probability density of the population. The mean of the population is shown by the solid red line, and the median is shown by the dashed green line. Since the complete mutation (immigration) operator continually introduces new individuals with diverse genes, lower-scoring individuals remain present in later generations, which can help to prevent early convergence to local maxima.

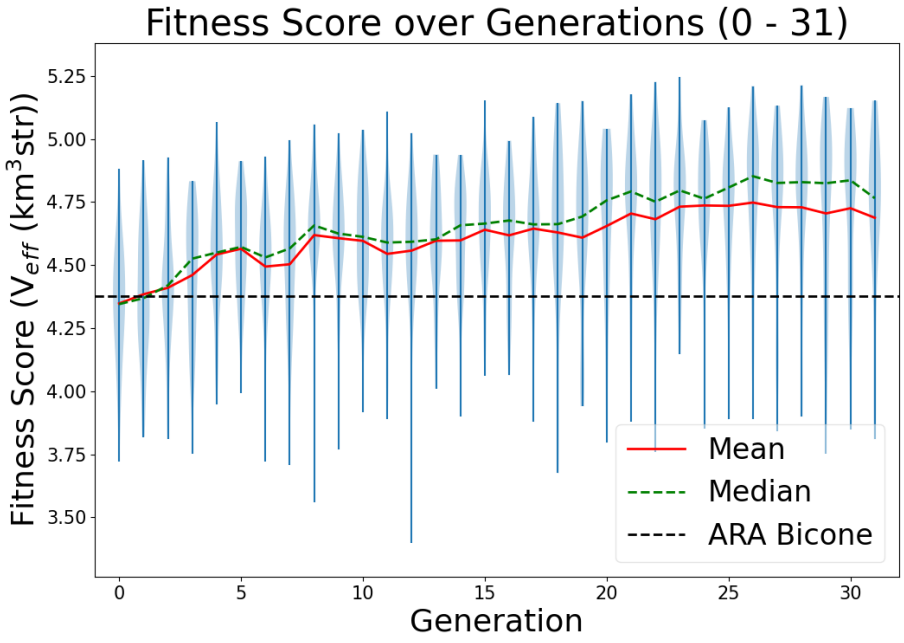


Figure 3.13: Initial results for the evolution of the asymmetric bicone. The current ARA bicone fitness is shown as the horizontal dotted line. Figure by Alex Machtay.

Fig. 3.14 shows the evolution of each parameter over the entire run. Each line represents a single individual, and the color illustrates the fitness of the individual. This demonstrates the effectiveness of the GA at producing higher scoring individuals in later generations. This figure also shows general trends in each parameter and its

impact on the fitness score. For example, most high scoring antennas share similar values with opening angles of the top cone (Angle 1) being under 1 degree, the length of the bottom cone (Length 2) being less than 50 cm, and the angle of the bottom cone (Angle 2) being between 4 and 6 degrees for high scoring antennas. However, the other parameters have a larger spread in viable values, with the radius of the top cone (Radius 1) spread across the entire parameter space for high-scoring individuals.

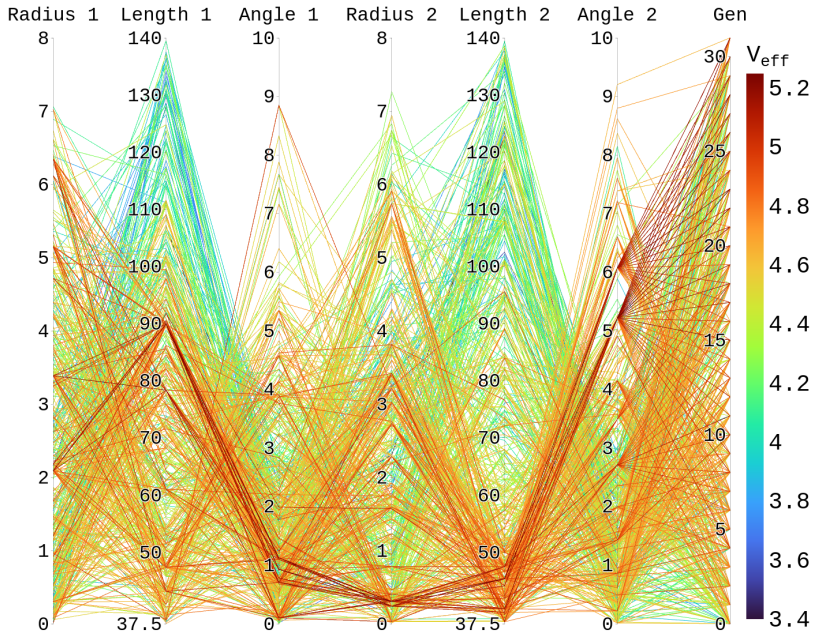


Figure 3.14: Evolution of the six antenna parameters for the asymmetric bicone, showing trends toward preferred features (red being most fit). Note the color is proportional to the fitness score. Figure by Alex Machtay.

Fig. 3.15 shows a 3D model of the top antenna. Notice that the top section of each antenna is longer than the bottom, has a larger inner radius, and has a smaller opening angle.

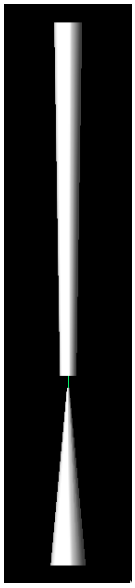


Figure 3.15: Model of the best antenna design from the asymmetric evolution. Individual 8, evolved in Generation 23.

3.4.2.3 Antenna Design Details

Antenna design involves more than just the physical shape of the antenna and includes the power source, circuit, and wiring to the antenna. These elements have the ability to inhibit the performance of the antenna and must therefore be considered in the design. Thus far, the GENETIS group has not included these elements in their analysis and will be adding these elements in the future. This section provides a brief introduction to these elements, as well as the current effect of this on the asymmetric bicone results.

There are multiple relevant power quantities in antenna design. The source power, P_{source}, is the power directly coming from the source. Other common names for the source power include the incident power, start power, or available power. The accepted power, P_{acc}, refers to the power that reaches the antenna after any reflection losses. The accepted power is also referred to as the input power. Finally, the P_{rad} is the radiated power and refers to the power that leaves the antenna via EM radiation. $U(\theta, \phi)$ is the radiation intensity as a function of direction. These powers are related through the following equations.

$$P_{\text{acc}} = P_{\text{source}} \ (1 - S_{11}^2) \tag{3.2}$$

where S_{11} is the reflection coefficient, also given by Γ.

$$P_{\text{rad}} = P_{\text{acc}} \ \eta \tag{3.3}$$

where η is the radiation efficiency.

These terms lead to the two related definitions of gain. First, gain, or absolute gain is defined as the ratio of the radiation intensity, $U(\theta, \phi)$, to the radiation intensity produced in a given direction if the power accepted by the antenna was isotropically radiated ($\frac{P_{\text{acc}}}{4\pi}$). Typically the peak gain is reported as a single value from the direction of maximum radiation but could be at other directions.

$$G = \frac{4\pi \ U(\theta, \phi)}{P_{\text{acc}}} \tag{3.4}$$

The realized gain, G_{re} is the gain reduced by any mismatched losses:

$$G_{\text{re}} = \frac{4\pi \ U(\theta, \phi) \ (1 - \Gamma^2)}{P_{\text{acc}}} \tag{3.5}$$

which is equivalent to

$$G_{\text{re}} = \frac{4\pi\ U(\theta, \phi)}{P_{\text{source}}} \tag{3.6}$$

The realized gain is smaller than the gain because it includes mismatch losses (or equivalently because P_{source} is larger than P_{acc}).

The design of the cabling and circuits between the power source and the antenna is an important element in maximizing the realized gain by minimizing the input port coefficient, S_{11} or Γ_{in}. The input port reflection coefficient refers to the fraction of the reflected voltage to the input voltage, with losses due to an impedance mismatch between the antenna and the circuit or cable. Impedance, Z, is the opposition a circuit (or circuit element) has to an AC current. It is composed of the resistance, R, which is effectively the DC current equivalent, and the reactance. Reactance, X, is caused by the changing EM fields in an AC current and is composed of an inductance and a capacitance component. Impedance is expressed as a complex value, $Z = R + jX$.

If the antenna impedance (or any load), Z_{L}, does not equal the circuit (characteristic) impedance, Z_0, some signal is reflected toward the source. Therefore it is often desirable to create a matching circuit that changes the antenna impedance to equal the characteristic impedance, which results in no reflection. If the load impedance and characteristic impedance are complex conjugates, it results in the maximum power transfer. The characteristic impedance is often given as $50\,\Omega$. This default is the standard value in coaxial cables and is a compromise between the experimentally tested maximum power transfer ($30\,\Omega$) and the maximum attenuation ($77\,\Omega$). It is also a practical value that is easy to design coaxial cables to match. For this reason, this standard of $50\,\Omega$ was used as the characteristic impedance in further analysis.

The definition of S_{11} is defined as the reflected voltage, V_{R}, divided by the input voltage, V_{I}.

$$S_{11} = \frac{V_{\mathrm{R}}}{V_{\mathrm{I}}} = \frac{Z_{\mathrm{L}} - Z_0}{Z_{\mathrm{L}} + Z_0} \tag{3.7}$$

If $\mathrm{S}_{11} = 0$, then there is no reflected wave, and there is a perfect match of impedance. This occurs if the characteristic impedance and the load impedance are equal. If the $S_{11} = 1$, all of the signal is reflected toward the source. Since V_{I} is always greater than or equal to V_{R}, S_{11} always falls between 0 and 1, although it is often reported in dB. As the impedance varies over frequency, the S_{11} is similarly affected. Consequently, the bandwidth of an antenna is often defined as the range of frequencies where the S_{11} is below a predefined value.

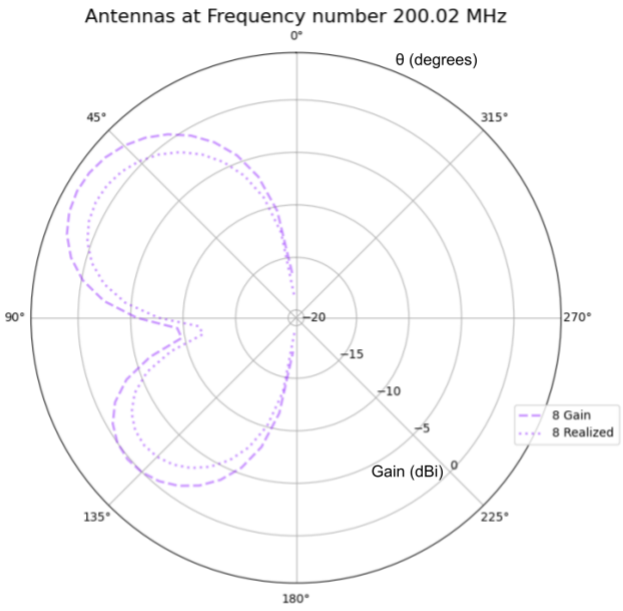


Figure 3.16: Comparison between the absolute gain and the realized gain at 200 MHz of the best evolved asymmetric antenna. Figure by Alex Machtay.

The reflected wave interferes with the input wave producing a standing wave. As the constant input and reflected waves move in opposite directions, they will interfere differently as they progress, resulting in the magnitude of the standing wave changing over time. The Voltage Standing Wave Ratio (VSWR) is the ratio of the peak amplitude of the standing wave to the minimum amplitude of the standing wave. It is related to S_{11} by the following equation.

$$\text{VSWR} = \frac{1 + |\text{S}_{11}|}{1 - |\text{S}_{11}|} \tag{3.8}$$

Since the GENETIS algorithm is built using the gain and not the realized gain, we investigated the difference between the two. As expected, the realized gain was smaller than the absolute gain at 200 MHz, as shown in Fig. 3.16. While the shape of the radiation pattern is similar, the realized gain has a smaller magnitude at all angles.

As a function of frequency, the difference between the absolute gain and the realized gain, averaged over all angles, is given in the left panel of Fig. 3.17, which illustrates the wide range of differences. The right panel shows the corresponding S_{11} plot. Fig. 3.17 shows the same pattern in both the plot showing the difference in gain and the S_{11} plot.

In order to match the impedance of the antenna with the circuit, it is necessary to utilize a Smith chart and a network analyzer (although software can complete the calculations now). Single-frequency impedance matching circuits can be constructed with either one capacitor or inductor in parallel and a capacitor or inductor in series with the load. Typically, one of each is used, although it is possible to use the same component in each position.

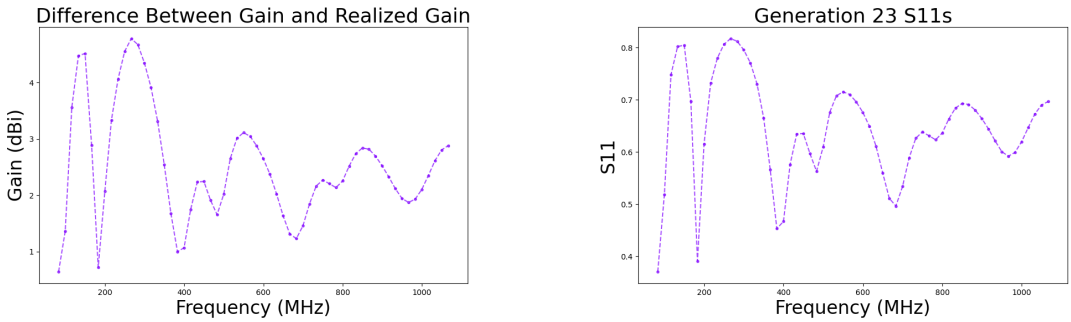


Figure 3.17: (Left) Comparison between the absolute peak gain and the peak realized gain for a range of frequencies of the best evolved asymmetric antenna. (Right) The S_{11} vs frequency for the asymmetric evolved antenna. Figure by Alex Machtay.

The Smith chart is a polar grid showing the impedance and admittance grids. A full Smith chart is provided in Appendix D [89]. The impedance is shown in red, and the admittance is shown in blue. Smith charts are often scaled to be normalized to the characteristic impedance. When plotting an impedance value, the red circles give the resistance component (real), while curved arcs from the right point are the reactance (imaginary) components. Conversely, the blue circles give the conductance (real), and the curved arcs from the left point give the susceptance.

The basic procedure is as follows. First, one connects the load (antenna in this case) to the network analyzer, which plots the location on a Smith chart and reads out the impedance (alternatively, the impedance can be found through simulation). Adding a capacitor/inductor in series moves the impedance counterclockwise/clockwise along the circles of the Smith chart. Similarly, adding a capacitor/inductor in parallel moves the impedance clockwise/counterclockwise along the circles of the Smith chart. Inductors always move the point upward on the chart, while capacitors move it downward. Combining two components makes it possible to move the

impedance to the characteristic impedance, thus matching the circuits. The distance traveled can be used to calculate the values of the capacitor/inductor. However, since impedance is related to frequency, it is typically not realistic to manually match the impedance for all frequencies. In practice, a broadband matching circuit with potentially many components could improve the gain across a wide range of frequencies. Such a circuit could be designed and optimized using various electrical engineering software.

For illustrative purposes, consider the asymmetric evolved antenna discussed above. At a frequency of 300 MHz, XFdtd gives an impedance of $Z_L = 311 - 197j$, normalized to 50 Ohms, this is $Z_L = 6.2 - 3.9j$. This can be plotted on the Smith chart below, as Z_L. To move the point to the center, we can rotate counterclockwise upward with a parallel inductor to the 1.0 reactance circle at point A. Then rotate counterclockwise downward with a series capacitor to the center point, Z_0, representing the characteristic impedance.

Starting with the inductor, the distance to 50 Ohm reactance circle is found by reading off of the blue susceptance lines to be 0.41, (0.08 below horizontal axis and 0.33 above). After multiplying the reciprocal by 50 Ohms, we can calculate the required inductor.

$$\frac{50\ \Omega}{0.41} = 122\ \Omega$$

$$\frac{122\ \Omega}{2\pi\ 300MHz} = 66.3\ \text{nH} \tag{3.9}$$

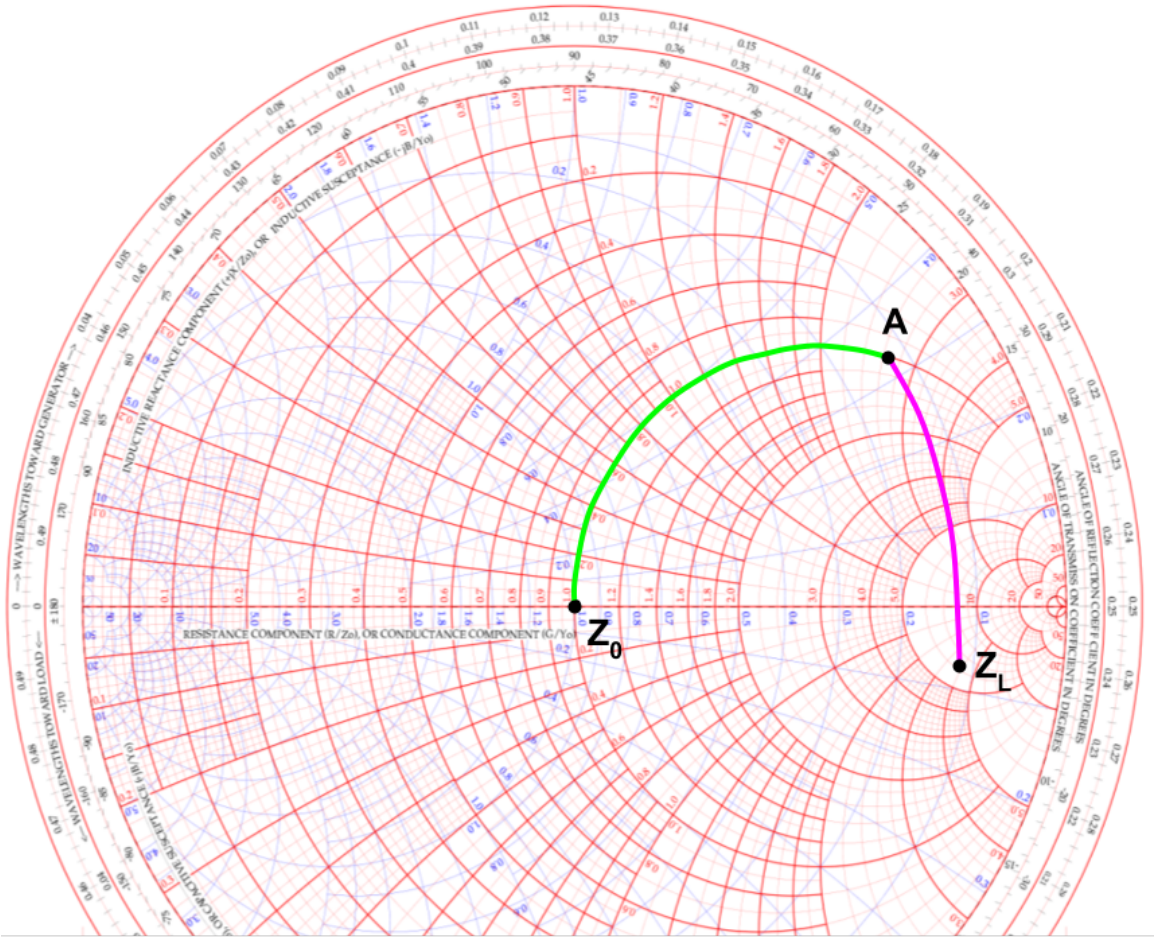


Figure 3.18: Subset of a Smith chart showing impedance matching at 300 MHz for the asymmetric bicone antenna. Z_L is the impedance of the antenna and Z_0 is the characteristics impedance of 50 Ohms. The green line shows the path taken after applying a 66.3 nH parallel inductor to point A. The magenta line shows the path from applying a series 3.9 pF capacitor to match the impedance. A full Smith chart can be found in Appendix D. Adapted from [89].

For the capacitor, the distance along the 50 Ohm reactance circle to the center point is 2.7. Again after removing the normalization, we can calculate the required capacitor.

$$50\ \Omega \times 2.7 = 135\ \Omega$$

$$\frac{1}{2\pi\ 300\ \text{MHz} \times 135\ \Omega} = 3.9\ \text{pF} \tag{3.10}$$

This manual result closely matches that of a software calculation, which finds an inductor of 67.8 nH and a capacitor of 3.8 pF. While a number of online tools are available for simple circuit matching, this site was used for this calculation. A diagram of the circuit is given in Fig. 3.19.

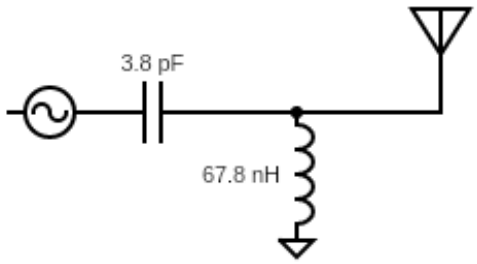


Figure 3.19: Circuit designed to match the evolved asymmetric antenna at 200 MHz. The source is connected in series to a 3.8 pF capacitor and then a 67.8 nH parallel inductor before connecting the antenna.

3.4.2.4 Verification of Results

In order to explore these results further, the GENETIS team explored two different aspects of our produced solutions. Ben Sipe contributed meaningfully to this investigation. First, we looked at how small changes an individual's genes altered the antenna response pattern and resulting fitness score. Second, we investigated

some of the physics concepts behind the results to enhance our understanding of the performance.

To explore how small changes in the genes changed the performance of a solution, we made minor perturbations to the genes of a previous antenna and then passed those new solutions through the XFdtd simulation and AraSim run, producing new gain patterns and fitness scores. This investigation was performed using the individual with the highest fitness score from the asymmetric bicone run. Permutations were applied both to the length and opening angle of each cone. The genes of the highest performing individual can be seen in Tab. 3.3.

Table 3.3: Parameters of the highest scoring individual.

Gene	Value
L_1 (cm)	89.92
r_1 (cm)	2.087
Θ_1 (rad)	0.016
L_2 (cm)	45.36
r_2 (cm)	0.30
Θ_2 (rad)	0.091

The length was altered by adding and subtracting 2.8 cm (chosen because it is one tenth of the shortest wavelength); there are 8 permutations for this modification. In order to modify the opening angle, the inner radius was changed by adding and subtracting 2.81 cm; similarly, there are 8 permutations for this modification. The new genes for each altered individual can be seen in Tab. 3.4 and Tab. 3.5. The differences in the antenna response patterns for each newly created individual at multiple frequencies can be seen in Fig. 3.20 and Fig. 3.21. This study confirms

that small alterations in the genes of an individual gives expected differences on the antenna response pattern and fitness score.

Table 3.4: Length Values Tested for 8 Individuals (cm)

ID	119	120	121	122	123	124	125	126
l_1	92.92	89.92	86.92	89.92	92.92	92.92	86.92	86.92
l_2	45.36	48.36	45.36	42.36	48.36	42.36	48.36	42.36

Table 3.5: Angle Values Tested for 8 Individuals (radians)

ID	165	166	167	168	169	170	171	172
θ_1	0.0474	0.0162	-0.0151	0.0162	0.0474	0.0474	-0.0151	-0.0151
θ_2	0.0910	0.1521	0.0910	0.0293	0.0293	0.0293	0.1521	0.0293

For our second investigation, we conducted an exploration using AraSim comparing the best asymmetric bicone and the ARA bicone. The purpose of this investigation is to identify the reason why the best individual might be performing better. In this analysis, an angle of 0 is vertical downward, $\pi/2$ (90°) is horizontal, π (180°) is vertical up. Fig. 3.22 shows a comparison of the RF arrival angles for the simulated neutrinos detected by each antenna. The RF arrival angle ranges from 0 to 2.5 radians (0° – 143°), with the majority occurring between 1 and $\pi/2$ radians (57° – 90°). At angles larger than $\pi/2$ radians, the number of detected neutrinos quickly decrease, which is expected as at the energies ARA is probing, neutrinos cannot pass through the earth. Fig. 3.22 also illustrates that the GA antenna detects more events, confirming the larger fitness score, and the detected neutrinos tend to come from slightly smaller

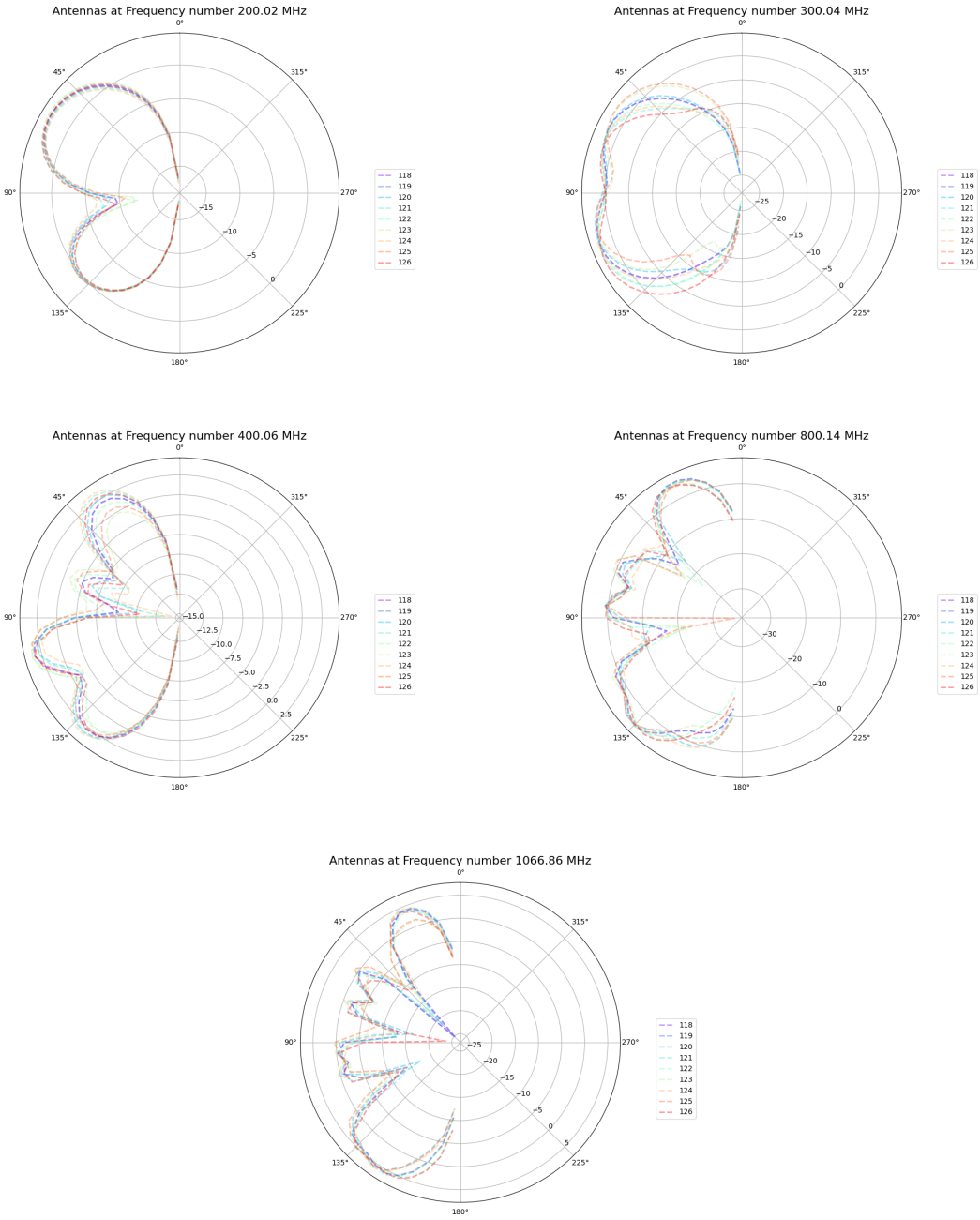


Figure 3.20: Difference in gain patterns for antenna models of similar shape and slightly differing length genes. Figure by Alex Machtay.

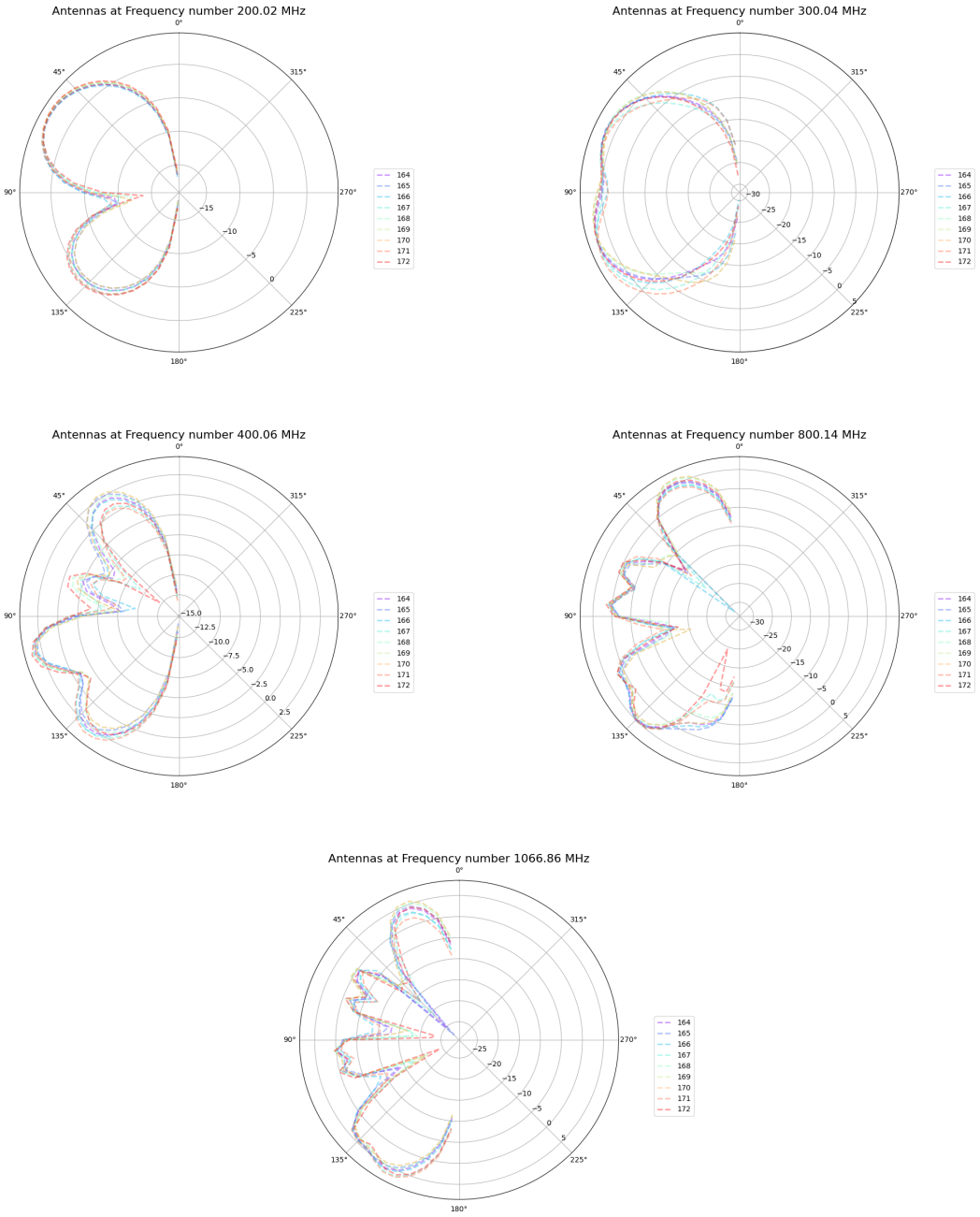


Figure 3.21: Difference in gain patterns for antenna models of similar shape and slightly differing angle genes. Figure by Alex Machtay.

angles. The most significant increase in sensitivity between the antennas is at the evolved antenna's peak angle at approximately 1.25 radians (71.6°). This peak occurs at a zenith angle that is about 0.25 radians (14.3°) smaller than the peak zenith angle for the ARA bicone. Further, at angles greater than $\pi/2$ radians (90°), the evolved bicone is equal to or slightly less sensitive. This shift in distribution could be due to the shape of the evolved bottom bicone, causing smaller arrival angles to be closer to perpendicular with the sides of the bicone.

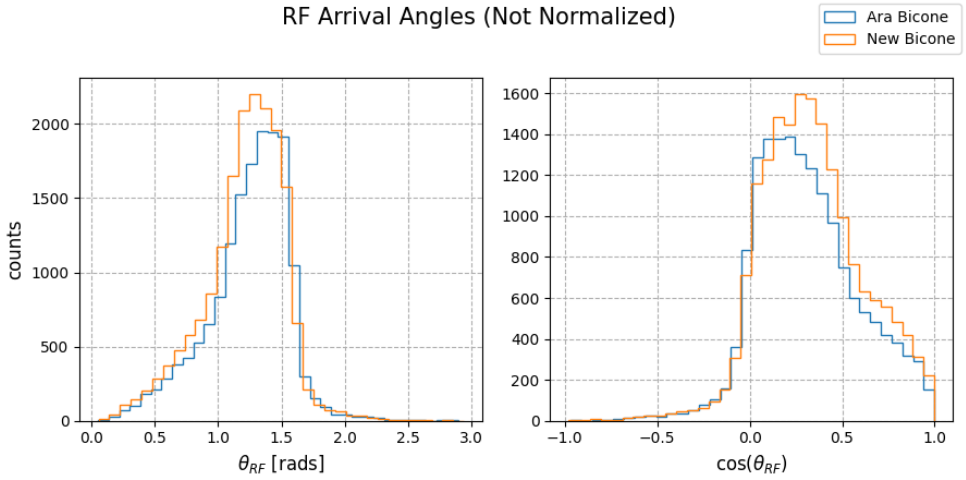


Figure 3.22: Comparison of the RF arrival angle of the best evolved asymmetric bicone antenna and the ARA bicone. Figure by Ben Sipe.

The angular reconstruction comparison shown in Fig. 3.23 shows very similar results for each antenna, besides the gradual increase in the cumulative count for the evolved antenna. This demonstrates that both antennas detect RF signals from different directions equally.

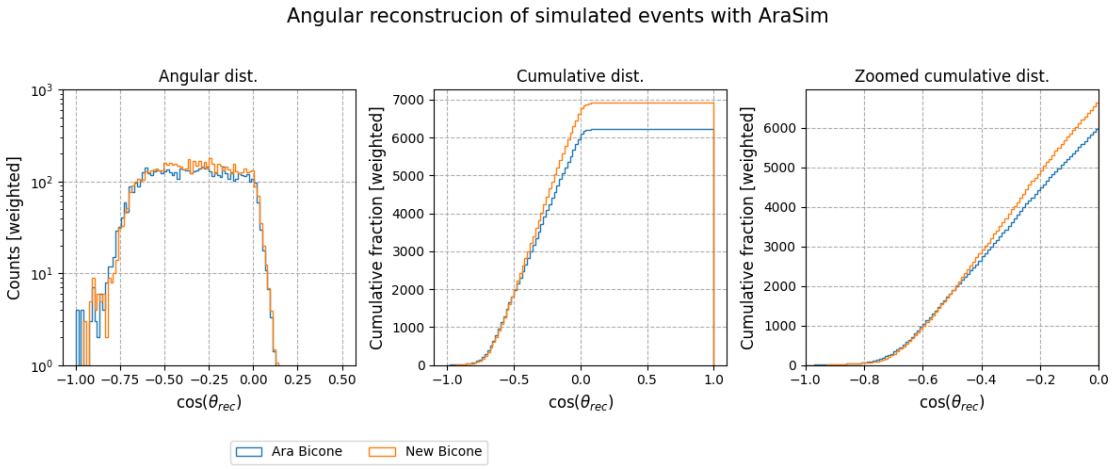


Figure 3.23: Comparison of the reconstructed neutrino angles of the best evolved asymmetric bicone antenna and the ARA bicone. Figure by Ben Sipe.

Finally, Fig. 3.24 shows the energy distributions for each antenna. The plot indicates that the results are independent of shower energy, with both antennas detecting events at a relatively flat rate before a peak at the highest energy. One point of interest is that only the ARA bicone detected neutrinos in the lowest energy bins although it is not clear if this is significant. This could be related to the results of Fig. 3.23, lower energy neutrinos are more likely to travel farther through the ice/earth, so if the ARA bicone is more sensitive at those energy ranges, it will see more neutrinos at arrival angles larger than $\pi/2$, although the distributions are not significantly different.

3.4.3 The Nonlinear Asymmetric Bicone

The GENETIS GA was expanded to evolve bicone designs with nonlinear sides to further explore more complex and diverse solutions. Eliot Ferstl, Leo Deer, and Ryan Debolt were notable contributors to this investigation. To define this geometry,

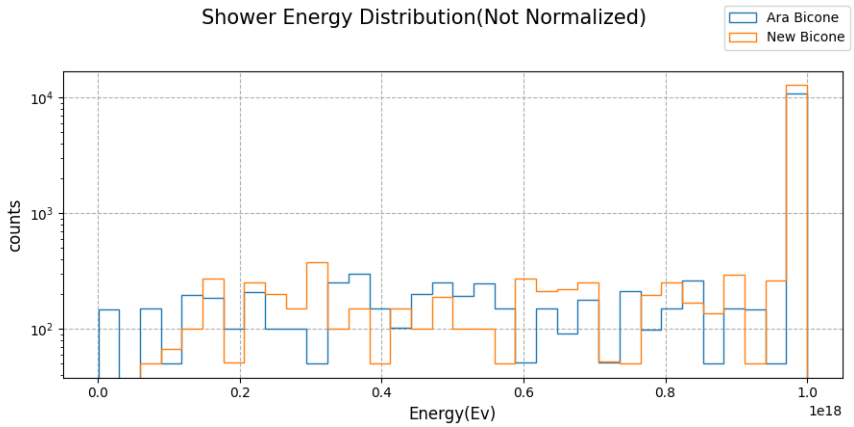


Figure 3.24: Comparison of the shower energy of the best evolved asymmetric bicone antenna and the ARA bicone. Figure by Ben Sipe.

the GA describes the shape of the surface of the bicone using a polynomial. Genes are given by the inner radius (r), the length (L), with the coefficients of a polynomial that describes the sides, as seen in Fig. 3.25.

In the symmetric and asymmetric linear bicones, each cone could be described by the equation $R(z) = r + z \tan\theta$, where R is the radius at a particular value of z (the height) that terminates at $z = L$. The tangent of θ is the constant slope in the equation. The final shape of the single cone could be described by rotating the equation about the horizontal axis.

The nonlinear bicone aims to find more diverse solutions by allowing this equation to form a polynomial $R(z) = az^2 + bz + r$. In this case, the genes that define an individual are r, L, a, and b. Note that this method is flexible for higher-order polynomials, which will be the focus of future investigations.

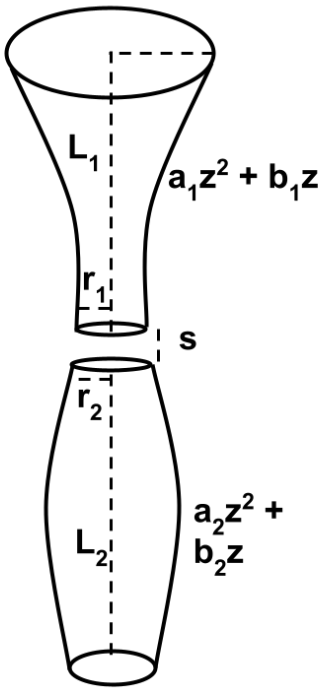


Figure 3.25: A schematic of an asymmetric, nonlinear bicone antenna. The lengths (L_1, L_2), inner radii (r_1, r_2), quadratic coefficients (a_1, a_2), linear coefficients (b_1, b_2), and separation distance (s) fully define the geometry. In the results presented here, the separation distance was held constant, and the other eight parameters were varied.

3.4.3.1 The Nonlinear Bicone Loop

We have completed two main evolutions for the nonlinear bicone: (1) an evolution evolving genes to the first order, and (2) an evolution evolving genes up to the second order. The first order run was a test to show that we could evolve using the coefficients of a polynomial as genes that describe our linear bicone instead of using the length, radius, and theta values. For these runs, we evolve using 50 individuals per generation and run simulations with 300,000 neutrinos. A new operator, called elite, was introduced for the nonlinear bicone that selects the highest-scoring individual and

copies it to the next generation. This ensures that the best high-scoring individuals are not stochastically removed during the loop. The remainder of the next generation is created using the same selection methods and operators as the asymmetric linear bicone described in Chapter 3.4.2. While the ability to use rank selection was added to the GA, it has yet to be used in a run. It is important to note that the ratio of each of selection methods and operators used has not yet been optimized for the nonlinear bicone evolution; this is intended for an upcoming investigation.

The introduction of nonlinear sides necessitates additional constraints. Previously, the antenna was restricted from having an outer diameter that exceeds the current ARA borehole. For the nonlinear sides investigation, the antenna has the potential to be larger than the borehole at any point along the length, not just at the outer diameter. The individuals are constrained by looking at the maximum width; if that width is greater than the borehole, the individual's genes are regenerated before moving on. To do so, our algorithm treats the edges as a polynomial and finds the maximum value. Furthermore, the equation could cross over the horizontal axis and self-intersect. The GA prevents this by regenerating individuals that have a width of zero at any point. The individual creation steps had to be modified to account for these constraints and prevent generating non-valid.

3.4.3.2 First Order Results

As an initial test, we took the nonlinear bicone and evolved its genes using first-order parameters. This first-order test is equivalent to the linear asymmetric bicone test and was constructed as a means to test the software's ability to evolve using the coefficients of a polynomial as genes. Results for this study can be seen in Fig. 3.26.

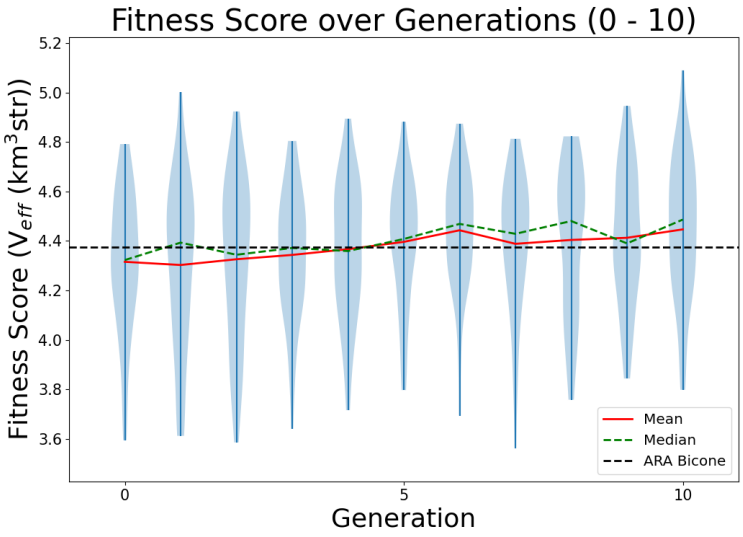


Figure 3.26: Initial results for the evolution of the first order nonlinear bicone. The current ARA bicone fitness is shown as the horizontal dotted line. Figure by Alex Machtay.

Evolution of the genes can be seen in Fig. 3.27. CAD models of the best performing individuals can be seen in Fig. 3.28.

It is worth noting that this run was conducted for 10 generations, whereas the asymmetric bicone evolved up to 30 generations. Since this was intended to be a test using the coefficients of a polynomial and because evolving takes up substantial computation time, this run was only extended into generation 10. Additionally worth noting, the fitness score plots in Fig. 3.26 show slower growth than that of the asymmetric bicone over the first 10 generations. A contribution to this may be due to non-optimized GA parameters for this run, though other sources of error are currently being explored.

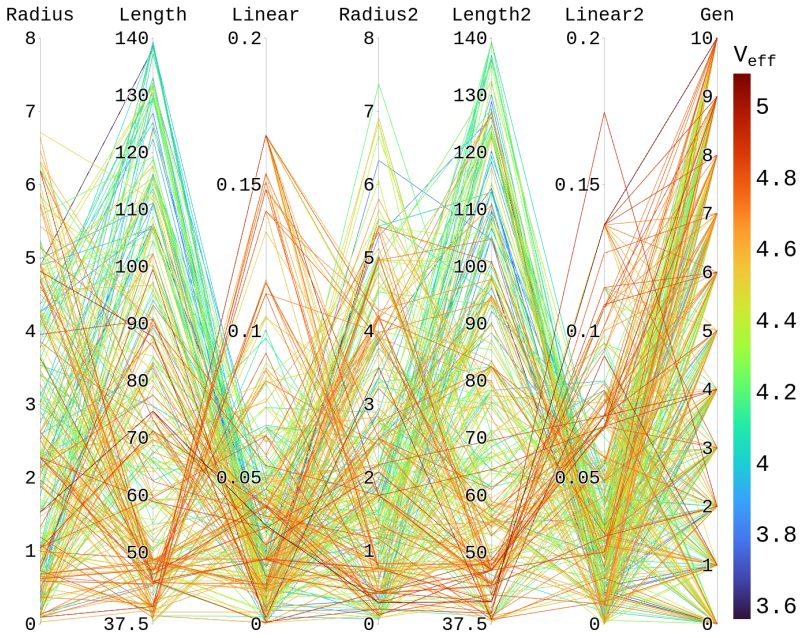


Figure 3.27: Evolution of the antenna parameters with the linear term converted to radius for easy comparison to the asymmetric bicone evolution, showing trends toward preferred features (red being most fit). Figure by Alex Machtay.

3.4.3.3 Second Order Results

The second-order run is our first attempt at adding more variability to the asymmetric bicone design. This is an early presentation of this investigation and as of this writing, the evolution is currently running and our GA operators and selection methods have not yet been optimized. Results for this study thus far can be seen in Fig. 3.29. Evolution of the genes can be seen in the top panel of Fig. 3.30. CAD models of the best-performing individuals can be seen in the bottom panel Fig. 3.30. Since this evolution is currently still in progress, the results from this section are merely

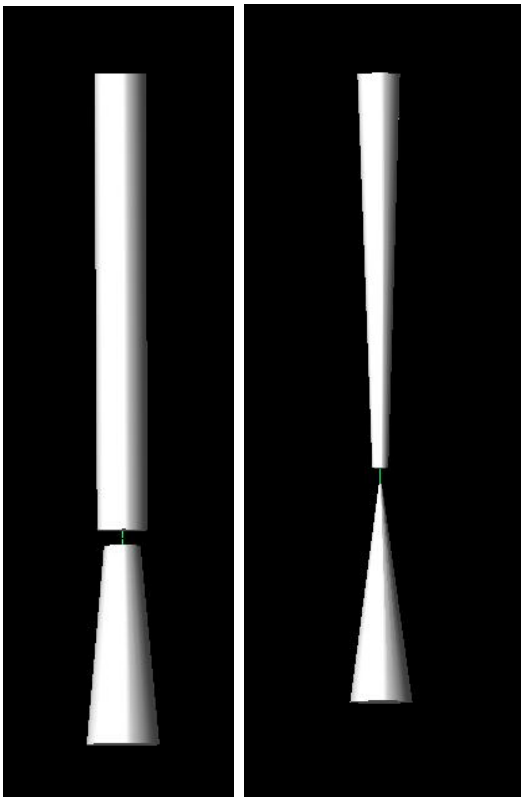


Figure 3.28: CAD models of the best individuals from the first order non-linear bicone evolution. Note the similarity in design to the asymmetric bicone, with a longer top cone and a wider bottom opening angle.

preliminary, with more soon to come. This GA includes any first-order solutions, as the algorithm allows the quadratic coefficient to go to zero.

3.4.4 Future Work

Future work for PAEA includes optimizing the GA for the second-order run, and adding more complexity (orders) onto the possible antenna geometry. The GA will have to be re-optimized for the current run parameters for each order we add on. Furthermore, we have identified that the uncertainty in fitness scores is too large, and we are working to reduce it, which will improve the evolution. Additionally, the

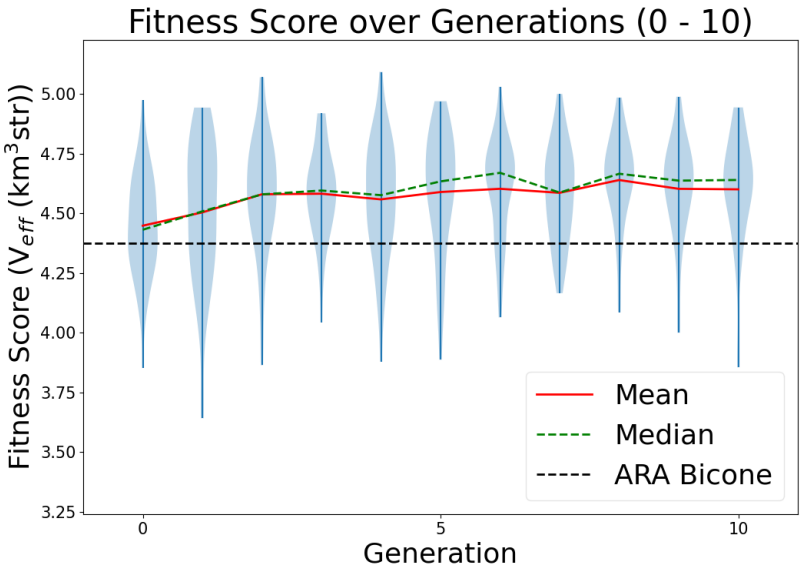


Figure 3.29: Initial results for the evolution of the second order nonlinear bicone. The current ARA bicone fitness is shown as the horizontal dotted line. Figure by Alex Machtay.

PAEA project has begun setting up a new software loop to optimize other types of antennas, such as the horn antennas used by ANITA. We have begun integration of the ANITA antenna simulation software, IceMC [50], as well as modifying the genes and XFdtd to produce horn antenna geometry.

We will also be adding intricacies in design required for printing and testing, such as adding coaxial cables and impedance matching circuits to the GA. Additionally, the GENETIS group will be 3D printing antenna prototypes through additive manufacturing at The Ohio State University Center for Design and Manufacturing Excellence. Anechoic chamber testing will be conducted to measure the printed antenna response. If sensitivities are improved by more than a factor of 2 over the ARA bicone, the antennas will be deployed in-ice for further testing.

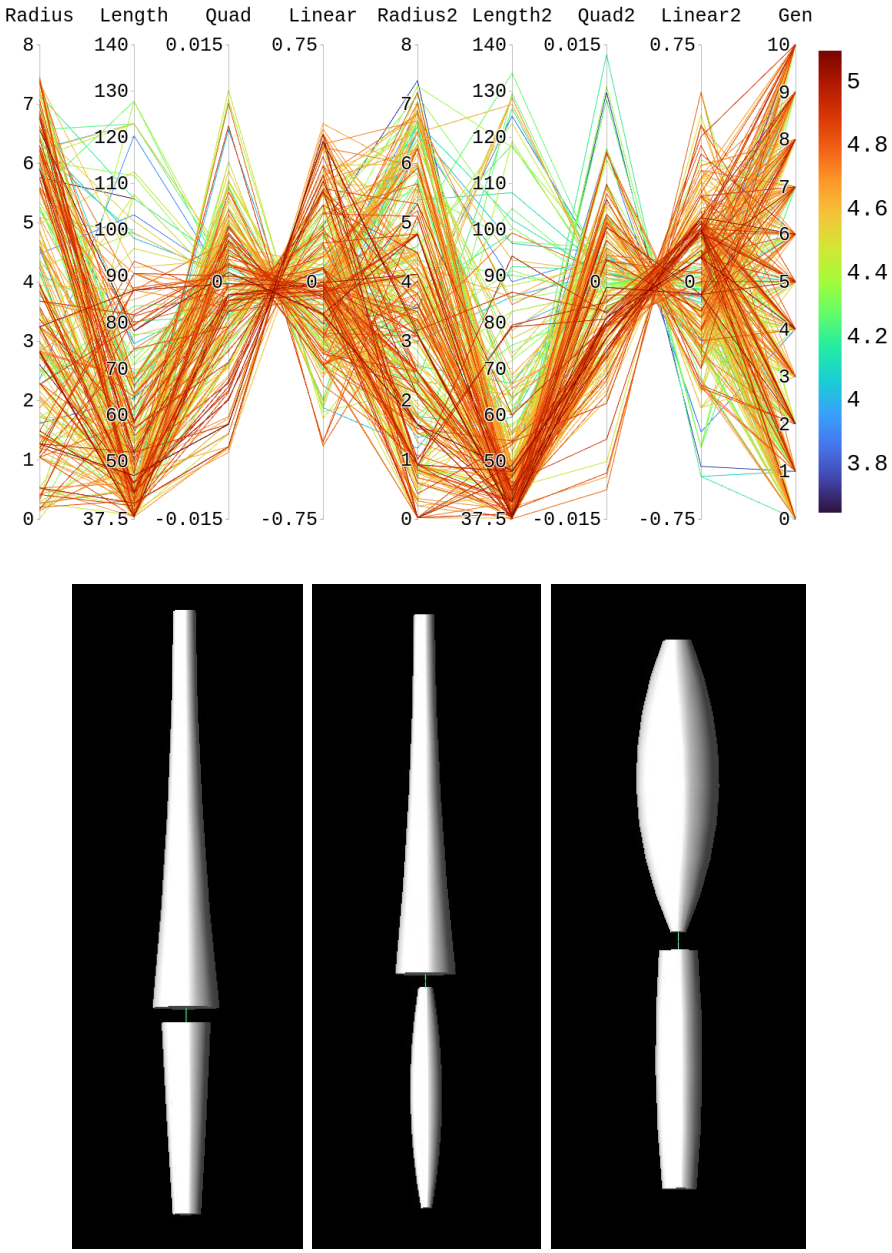


Figure 3.30: Top panel: Evolution of the antenna parameters radius, length, linear coefficient, and quadratic coefficient for each cone, showing trends toward preferred features (red being most fit). Bottom panel: CAD models of the best individuals from the second order non-linear bicone evolution. Figure by Alex Machtay.

Another exploration for GENETIS, which is currently in progress, is using a neural network (NN) to reduce AraSim computation times. The NN recently developed at OSU, with Ben Sipe as a notable contributor, is intended to supplement AraSim, ideally sending some of the individuals from a generation through AraSim itself and the rest being sent through our NN to predict their fitness score based on past AraSim calculations and bypass the computationally heavy simulations. Some of the individuals tested with AraSim will join the training data to continue to train the NN as the evolution progresses.

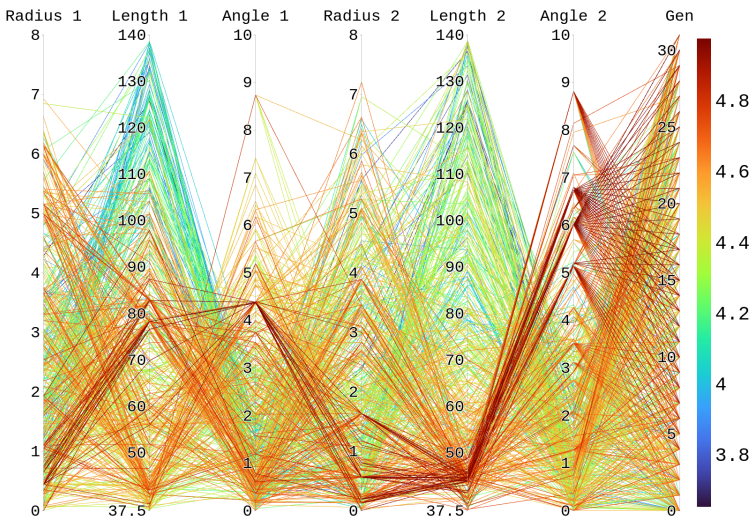


Figure 3.31: Evolution of 31 generations and 50 individuals using only the NN trained on past fitness scores calculated using XF and AraSim and the GA; XFdtd and AraSim was not used. Figure by Ben Sipe.

In order to test this NN, we have previously trained it on the genes and corresponding fitness scores from a past run, particularly the asymmetric bicone run discussed in Chapter 3.4.2, to make predictions of the fitness score based on the

genes. The NN was trained on 85% of the data and then tested on the remaining 15%. This NN utilizes 5-fold cross-validation so that it is both trained and tested on all individuals. For this asymmetric bicone run, there were a total of 32 generations that the NN trained on, each having 50 individuals. All of this information, including the hyper-parameters, are subject to change as we optimize the NN. For the run seen in Fig. 3.31 and Fig. 3.32, the mean actual error was $0.1\,\mathrm{km}^3/\mathrm{sr}$, which means that on average, the NN would give a fitness score $0.1\,\mathrm{km}^3/\mathrm{sr}$ away from the actual fitness score. This result showed the ability to predict fitness scores using a NN; it was not expected to improve upon the prior result, because the NN had a static training set. The static training set resulted in a cap to the maximum fitness score, which is seen in Fig. 3.32 as a cutoff around $5\,\mathrm{km}^3/\mathrm{sr}$.

3.5 Antenna Response Evolution Algorithm (AREA)

The subsequent GENETIS investigation involves the evolution of antenna radiation patterns to explore: (1) the optimal neutrino sensitivity attainable and (2) the potential for optimizing a directional beam pattern. The purpose of this project is to explore what improvement to the neutrino sensitivity is possible due to improvements in antenna responses alone, without regard to what physical design might be needed to bring about that response In this section, I will be discussing the work conducted by our previous collaborators at California Polytechnic State University, San Luis Obispo, as well as the continuation of this work at The Ohio State University.

3.5.1 AREA Loop

The loop software for AREA utilizes all of the same software elements as PAEA, except it no longer includes the involvement of XFdtd; since we are now producing the

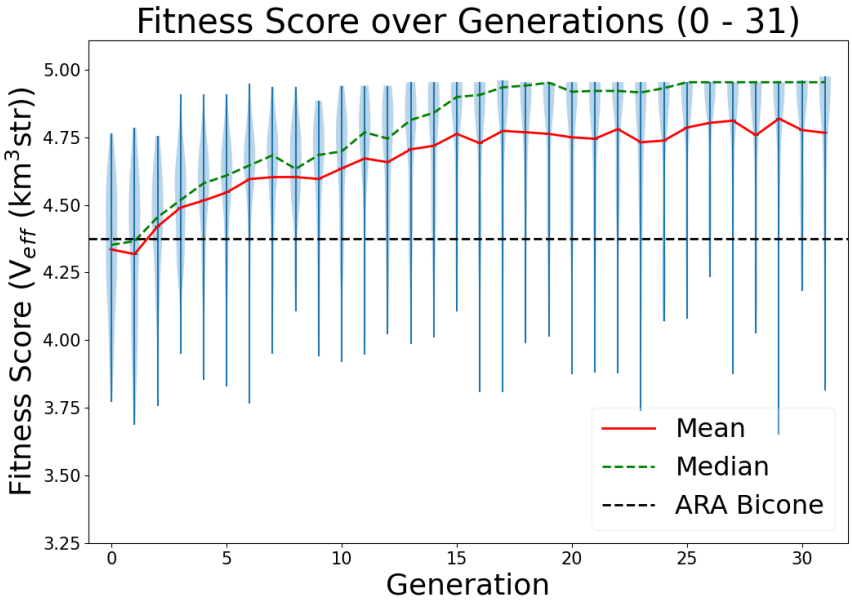


Figure 3.32: Initial results for the evolution of the asymmetric bicone using a NN instead of AraSim to produce the fitness score (effective volume). The fitness score for the current ARA bicone fitness is shown as the horizontal dotted line. The apparent cutoff is due to a lack of individuals with higher fitness scores for the NN to train on. Figure by Ben Sipe.

antenna gain patterns as our solutions, they no longer need to be simulated and our individuals and their genes are now parameters that describe the antenna response pattern itself. This is accomplished by evolving 13 azimuthally symmetric spherical harmonic functions as genes of each individual to give the equation below.

$$G(\theta, \vec{a}) \approx 2\sqrt{\pi}Y_0^0(\theta) + a_1 Y_1^0(\theta) + ...a_{12}Y_{12}^0 \tag{3.11}$$

A diagram of the software for the AREA loop can be seen in 3.33.

Past runs for this project have been done using 50 individuals, throwing 10,000 neutrinos. All individuals are constrained to enforce the conservation of energy on

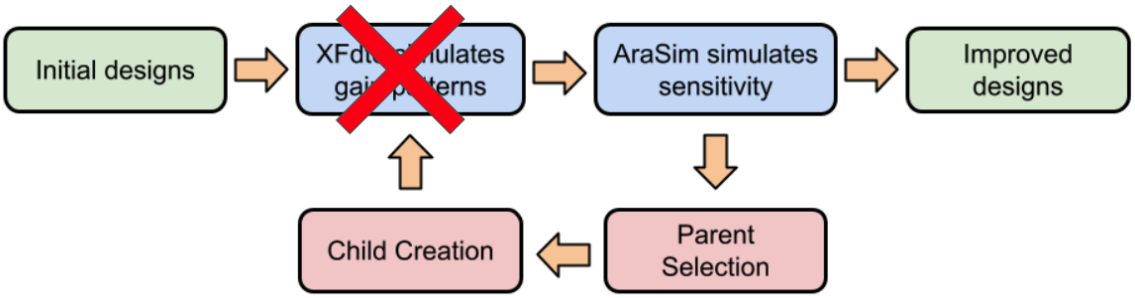


Figure 3.33: A diagram of the AREA software loop used to evolve antenna response patterns, which follows the same structure as the PAEA loop, but without XFdtd simulations.

radiation patterns. New generations were built with the following selection methods and genetic operators.

- One half of new individuals are made by roulette selection of two parents then continuous crossover
- One sixth of new individuals are made by roulette selection of one parent then a Gaussian mutation
- One sixth of new individuals are made by 6-way tournament selection of two parents then continuous crossover
- One sixth of new individuals are made by 6-way tournament selection of one parent then Gaussian mutation

In the runs completed by Cal Poly [63], which are discussed later in this section, they constructed two simplified versions of AraSim to be tested, called AraSimLite and AraSimLite2. These were constructed to cut down the run time in these earlier

stages, as AraSim required approximately 30-45 minutes per radiation pattern to run on the Cal Poly machines per 10,000 neutrinos. AraSim's slow computation time meant multi-day simulations in order to execute multiple generations. AraSimLite, however, takes approximately 0.3 seconds for the same simulation. Computational specs between each Monte Carlo simulation version can be seen in Fig. 3.34.

Computation	AraSim	AraSimLite	AraSimLite2
Models earth absorption	■	■	■
Accounts for signal spreading loss	■	■	■
Evaluates single frequency		■	■
Accounts for neutrino travel direction and Cherenkov Cone	■		■
Models ice characteristics	■		
Accounts for noise	■		
Accounts for signal polarization	■		
Evaluates full frequency range: $f = 83.3\ MHz$ - $1083.5\ MHz$	■		

Figure 3.34: Computational specifications between AraSim, AraSimLite, AraSimLite2. The red squares indicate the included computations, whereas white boxes indicate that those computations are excluded. Table from [63].

AraSimLite simplifies the fitness assignment by omitting the software's ray-tracing, noise waveforms, signal polarization, ice modeling, and uses a neutrino energy of 10^{18} eV. It simulates the neutrino-nucleon interaction cross-sections and samples potential interaction points in the ice [63]. Then, it randomly distributes neutrino directions over 4π str and generates corresponding weights and particle trajectories. By omitting the more intricate details of AraSim, AraSimLite is able to simulate 10,000 neutrinos 6000 times faster. However, these details are crucial for a complete understanding of an optimized gain pattern. Consequently, AraSimLite is an intermediate step that will eventually be replaced with the full AraSim.

AraSimLite creates the fitness score in two main steps. First, the number of neutrinos desired are simulated, resulting in a location and weight for all of them. The location is the coordinates of the event interaction relative to the detector. The weight is the probability that the neutrino event occurs based on the probability of the interaction occurring and the neutrino being absorbed. In the next phase, the radiation pattern is used in conjunction with an array of locations and weights to calculate the fitness score. The fitness score is given by the function:

$$\text{Fitness Score} = \sum_{\mathrm{i}=1}^{\mathrm{N}} \begin{cases} w_i, \text{ if } \frac{\mathrm{g}(\theta_\mathrm{i})}{\mathrm{R}_\mathrm{i}^2} > \mathrm{r_{th}} \\ 0, \text{ otherwise} \end{cases}$$

Where w is the weight, R and θ is the event location in polar coordinates, g is the antenna gain, and $\mathrm{r_{th}}$ is a threshold value set by the minimum detectable signal by the ARA electronics. The value for $\mathrm{r_{th}}$ in this analysis was $150\,\frac{1}{\mathrm{m}^2}$.

AraSimLite2 begins to introduce more complexity by adding the neutrino trajectory and Cherenkov cone into the simulation. There is an added constraint that the viewing angle is between 56.3° and 57.3°. The viewing angle is defined as the angle

between the neutrino trajectory and the vector between the detector and the event. This accounts for the angle of the Cherenkov cone of 56.8° and a one degree viewing angle of the detector. This additional angle restriction heavily reduces the number of detected events and thus increases computation time significantly. As a result, approximately 0.7% of generated events pass the new criteria. The fitness score of the AraSimLite2 analysis is given below. It is also worth noting that both AraSimLite and AraSimLite2 run independent of frequency, whereas AraSim utilizes gain patterns at 60 different frequency steps between 83.33 MHz and 1.066 GHz. Thus, we are only looking at one gain pattern assumed at one frequency with AraSimLite and AraSimLite2.

$$\text{Fitness Score} = \sum_{\mathrm{i}=1}^{\mathrm{N}} \begin{cases} w_i, \text{ if } \frac{\mathrm{g}(\theta_\mathrm{i})}{\mathrm{R}_\mathrm{i}^2} > \mathrm{r_{th}} \text{ and } 56.3° < \theta_{\text{view}} < 57.3° \\ 0, \text{ otherwise} \end{cases}$$

3.5.2 Preliminary Test Cases

This section will very briefly go over some of the proof-of-concept runs conducted at Cal Poly. As a preliminary study, two tests were conducted: (1) the ability for the GA to evolve omnidirectional radiation patterns, and (2) the ability to evolve a directional radiation pattern. The GA utilized the same specs as mentioned previously; however, the fitness function was modified to motivate the evolutions accordingly. In the first case, the fitness was maximized by a radiation pattern that had no directivity. Being equal in all angles, this pattern should form a circle. The test evolution was successful, achieving a fitness score 99% of the perfect solution. Final results can be seen in Fig. 3.35. In the second case, the fitness function maximized the gain toward 90° and 270°, forming a rectangular radiation pattern. The evolved solutions tended

toward the desired result but failed to match closely due to the limited number of spherical harmonics. Full results can be seen in Fig. 3.36.

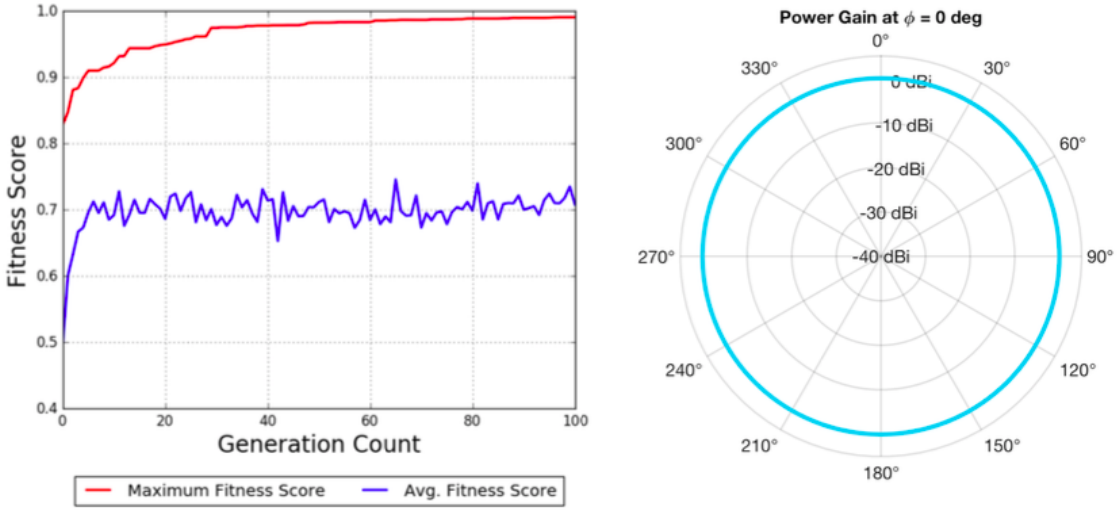


Figure 3.35: Evolution results (left) of the omnidirectional beam pattern run and radiation pattern of the fittest individual (right) [63].

3.5.3 Antenna Response Optimization with AraSimLite

Fig. 3.37 shows the individual with the highest fitness score from all runs using AraSimLite. This individual evolved 500 generations, and has a maximum gain of 5.42 dBi at 133 degrees and a fitness score of 27,120. Genes for the highest-scoring individual can be seen in Tab. 3.6. A population size of 50 individuals and 10,000 neutrinos were used.

3.5.3.1 AraSimLite2 Results

Fig. 3.38 shows the individual with the highest fitness score from all runs using AraSimLite2. This individual evolved 500 generations, has a maximum gain of

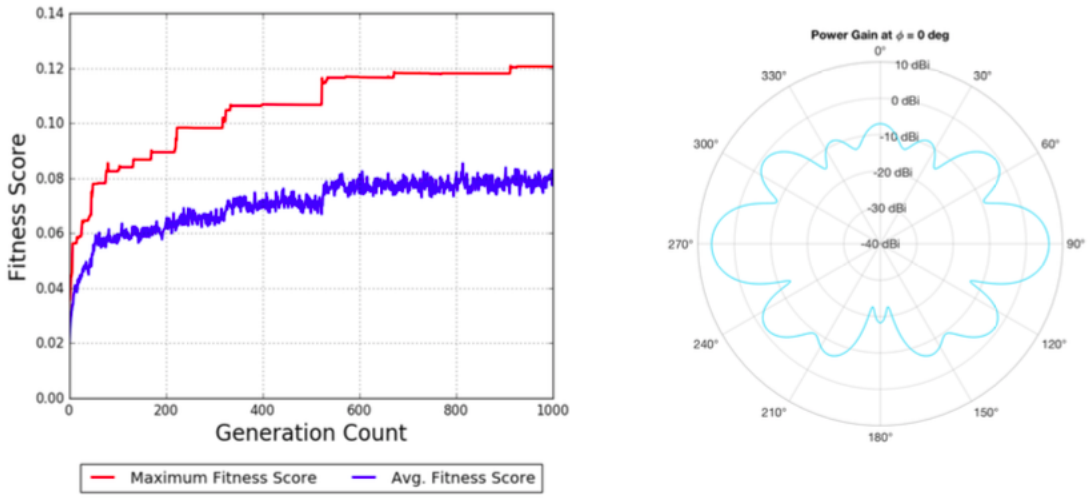


Figure 3.36: Evolution results (left) of the directional beam pattern run and radiation pattern of the fittest individual (right) [63].

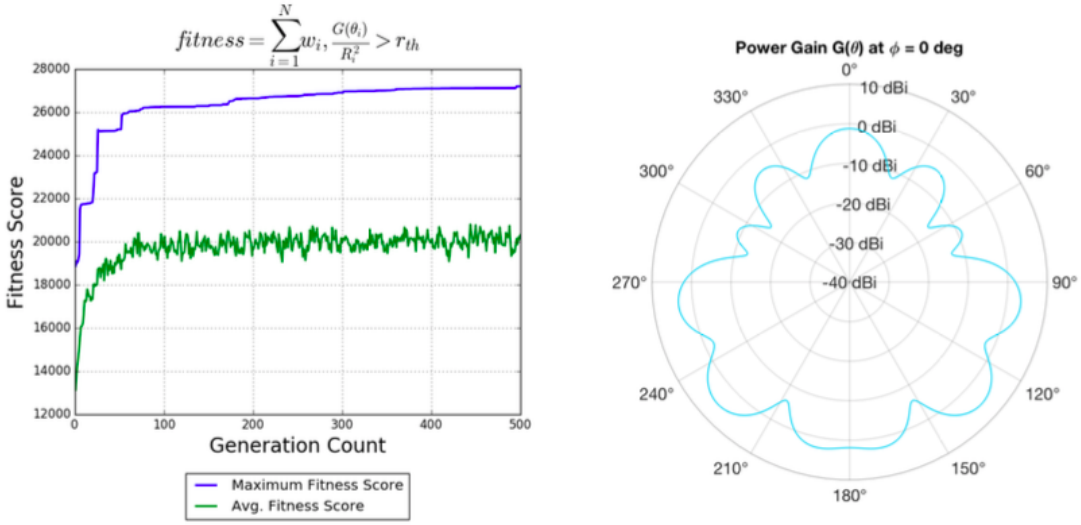


Figure 3.37: Summary of the full evolution (left) and most fit individual from that evolution using AraSimLite (right) [63].

Table 3.6: Genes from the most fit solution in the AraSimLite run.

x_1	x_2	x_3	x_4	x_5	x_6
-2.291	-0.287	1.265	-0.210	0.019	-0.497

x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
-0.265	1.556	-1.141	-0.087	1.077	-0.487

3.82 dBi at 116 degrees, and a fitness score of 150.5. Genes for the highest-scoring individual can be seen in Tab. 3.7.

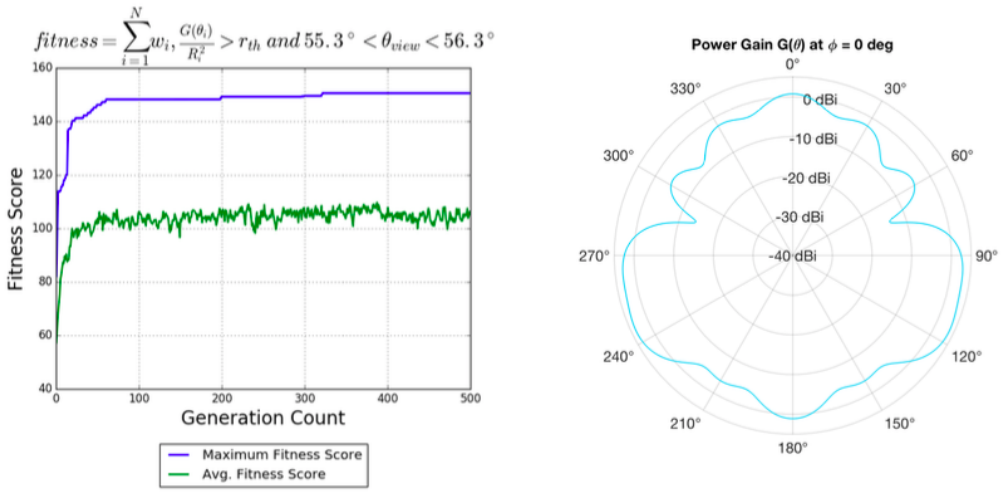


Figure 3.38: Summary of the full evolution (left) and most fit individual from that evolution using AraSimLite2 (right) [63].

3.5.3.2 Result Discussion

The current AREA results show promising evolution and improvement in the fitness function. However, the highest-scoring gain patterns failed to perform well in a full AraSim simulation. Comparing the best evolved individual gain pattern to the

Table 3.7: Genes from the most fit solution in the AraSimLite2 run.

x_1	x_2	x_3	x_4	x_5	x_6
-1.211	-1.327	1.810	0.456	-1.043	0.308

x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
0.108	0.111	0.071	-0.332	-0.030	0.476

gain pattern of the ARA bicone antenna, the evolved pattern had a fitness score 40% higher in AraSimLite2, but 20% lower when running AraSim. Unfortunately, this suggests that the simplifications used in AraSimLite2 were too drastic to be used to evolve an accurate gain pattern.

Full implementation of AREA with AraSim should allow the loop to accurately produce a gain pattern that is optimized and can exceed the fitness of the ARA bicone. The GENETIS team is actively pursuing this project, and preliminary results will be discussed in the next section.

3.5.4 Antenna Response Optimization with AraSim

Investigations at The Ohio State University, with notable contributions by Ethan Fahimi, include (1) a test run of the GA with full AraSim, and (2) a test evolution of isolated frequencies. These runs utilize the same GA as described in Chapter 3.5.1.

The test run of the GA with full AraSim was a trial run to ensure that the full AraSim software would run, without error, in the AREA software loop. Trial runs had evolution specs set to 50 individuals, 10,000 neutrinos, and ran for 36 generations. Results of this run can be seen in Fig. 3.39. These results show poor evolution, as AraSimLite versions were frequency independent. Since the Full AraSim is frequency dependent and requires 60 different antenna response patterns at varying frequencies,

this evolution generated all 60 frequencies for each of the 50 individuals and evolved them independently of each other. Thus, each individual had 60 disunited frequencies that were all run in the same AraSim job, which gave very low fitness scores that were unable to improve through each evolution. Nonetheless, this test showed that the integration of AraSim into the AREA software was successful.

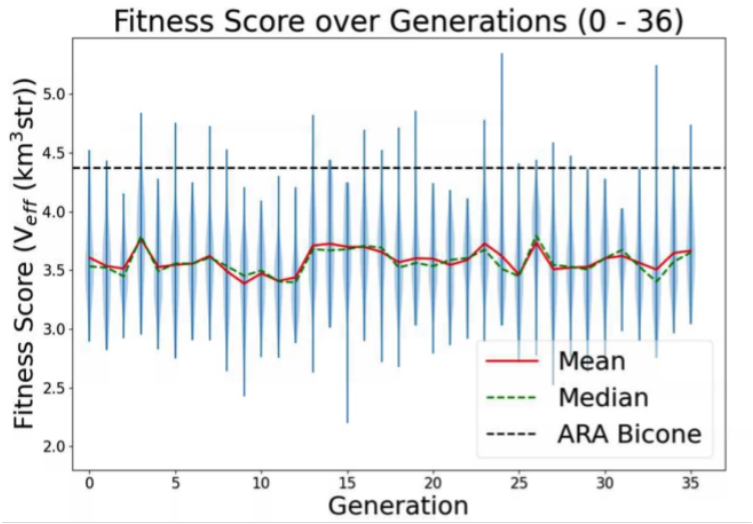


Figure 3.39: Summary of the test AREA run with full AraSim. Figure by Ethan Fahimi.

The next goal with AREA is to achieve meaningful runs with full AraSim. Because of the difficulty in implementing a range of frequencies into an evolution, we are beginning by isolating our evolution to use the same beam pattern across all frequencies; this means we need to modify the GA outputs to trick AraSim into thinking it has the proper input file for all 60 frequency steps by duplicating the same beam pattern for all of the AraSim inputs. This investigation is currently in progress, and we expect results shortly.

3.5.5 Future AREA Work

Future goals for AREA include expanding the number of correlated frequencies we can include after we have completed a single-frequency evolution. Additionally, this GA has not yet been optimized. More diversity in selection methods and operators can be explored before optimization of the AREA GA. Further, once AREA successfully optimizes the antenna response for the ARA project neutrino detection, we can merge AREA and PAEA by using the optimized AREA gain patterns as a measure of fitness when evolving to optimized antenna designs and find out whether it improves on the results using AraSim to generate fitness scores in the main loop.

3.6 Conclusion

This chapter contains the results for all of the main GENETIS projects since the group's founding (and my involvement). GENETIS is pioneering the use of GAs to design experiments by using a physics outcome as a measure of fitness and laying the foundation for future detector optimizations. The GENETIS collaboration is currently working on a number of improvements to the loop that could continue to improve the computational efficiency, convergence speed, and maximum fitness. First, we are introducing more complex antenna geometries, such as bicones with nonlinear sides. Initial testing of the GA using bicones with nonlinear sides is underway, and the evolution of other types of antennas is also in development. We are also exploring the use of additional and more advanced selection methods and genetic operators. In the future, the GENETIS project will expand beyond antenna design and explore other aspects of experimental design and analysis, including detector layouts, trigger, and bandpass filter optimization. The GENETIS project will also expand to utilize other

types of computational intelligence techniques in other experimental applications, as well as expand to optimizing antennas for other neutrino radio experiments such as ANITA and PUEO.

The successful deployment of GA-designed detectors could pave the way for additional applications of computational intelligence for the design of scientific instruments. Expanded research in this area will streamline the optimization of the design of many types of experiments across fields for superior science outcomes.

Chapter 4: ARA Event Classification with Genetic Programming

4.1 Introduction

Many UHE neutrino experiments involve collecting a large amount of data and then searching that data for events of interest. Background is often eliminated in a series of cuts until only events of interest remain, or a flux limit is determined if no events are found. GP is one potential way to handle the discrimination between, or separation of events from background. This chapter will discuss my efforts to use GP to classify background and neutrino signals in the ARA collaboration data. As ARA has yet to detect a neutrino, it is necessary to simulate waveforms in AraSim and inject them into a background data set. At the outset, the goal of this analysis was to achieve at least a background rejection, or true negative rate (TNR), of 99.9% with an AraSim event acceptance rate, or true positive rate (TPR), of 90%.

I will first introduce a brief literature review of GPs used in data classification and then define some important terminology used in classification algorithms. The introduction will also further describe the acquisition and structure of ARA data.

The following section will describe Karoo GP, a software suite built by GENETIS collaborator Kai Staats, used in this analysis [108, 34, 107, 109]. A discussion of the

software's functionality and a brief description of how to perform Karoo GP runs are included. Also described is the construction and testing of a fitness function that was not already present in Karoo GP and had to be custom-built. More details on Karoo GP are given in Appendix C.

Next, I will describe the process of an initial investigation using one antenna channel, including variable exploration and initial results. From these results, an additional variable was extracted to improve the classification accuracy further. In the final section, I will increase the complexity of the analysis by repeating the procedure while utilizing the data from all antenna channels through coherently summed waveforms.

4.1.1 Literature Review

Karoo GP was originally created for a classification analysis of data from the Square Kilometre Array (SKA) in South Africa [108]. The goal was to generate solutions to classify data as signal or radio frequency interference (RFI). As an exploration of Karoo GP's potential, the creator conducted significant work testing different GP parameters and comparisons to other ML algorithms. Karoo GP was able to perform similarly to other algorithms.

The analysis conducted in this Chapter is loosely based on another study investigating the classification of LIGO data with Karoo GP [34]. In this study, the authors examined 1.47 days of single-interferometer LIGO data, injected with simulated gravitational waves produced by four different models. They utilized 11 variables to describe each event, including the signal-to-noise ratio, duration, frequency, and bandwidth. After 200 different Karoo GP runs, the resulting solution was able to

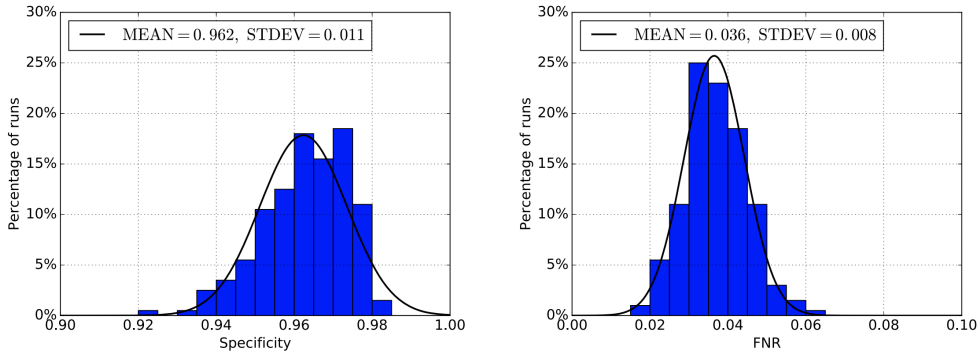


Figure 4.1: Specificity or TNR (left) and false negative rate or FNR (right) of a classification analysis of LIGO data injected with simulated events. The GP was run 200 times with the same parameters, and the evaluation of the best solution was done to determine the TNR and FNR. Figure from [34].

.

identify 96.2% of noise transients while misclassifying only 3.6% of simulated signals. Histograms displaying the specificity (True Negative Rate, TNR) and False Negative Rate (FNR) of the 200 runs in this analysis are presented in Fig. 4.1.

The authors also repeated this analysis using only a high SNR noise sample, demonstrating that the GP could be tuned to target a particular type of background. Finally, they performed a Bayesian analysis to determine the probability of an event labeled as a signal actually being a real signal, based on the repeatability of that label.

Computational techniques have become more prevalent in Physics analyses; however, the use of GPs is lacking compared to other fields. The KM3NeT collaboration, an optical Cherenkov experiment based in the Mediterranean Sea, has begun using random forests, boosted decision trees, and neural networks in various analyses, including event identification, energy/direction estimates, and signal/background discrimination [43]. The use of convolution neural networks was examined for signal

background classification in IceCube and ATLAS in [25]. The potential for deep learning searches for exotic particles from colliders was investigated in [26]. Similar to the above GP LIGO analysis, neural networks have also been applied to explore gravitation wave signals from core collapse supernovae [35]. A review of machine learning in high energy physics and neutrino physics can respectively be found in [112] and [97]. One example of the use of GPs in Physics is a study where GPs were used to model nucleon-nucleon collisions and predict the distributions of the shower particles [47]. Another example is the creation of software similar to Karoo GP, called PhysicsGP, where the authors demonstrate the ability of the software to search for new particles [41].

Applications of GPs have been conducted in a number of other fields to solve a variety of complex problems. In the medical field, GPs were used for the identification of Cancer based on the classification of microarray gene expression data [111] and for the diagnosis of Parkinson's disease [103]. In [88], a GP was used to establish a connection between seas' surface temperatures and rainfall that could be utilized for long-term forecasting. A review of GP applications and future possibilities in Civil Engineering is presented in [116].

4.1.2 GP Evaluation Terminology

To reduce overfitting, the models are often evaluated on a separate testing sample from the training sample. In addition to the fitness score (described in detail below), a confusion matrix can describe a classification model's performance, which helps visually define the model's performance. Note that the convention of the indexes

may be different. The format of the confusion matrix used in this analysis is given below:

$$\begin{bmatrix} \text{True Negatives (TN)} & \text{False Positives (FP)} \\ \text{False Negatives (FN)} & \text{True Positives (TP)} \end{bmatrix}$$

In the context of two group classifications this is simply:

$$\begin{bmatrix} \text{Count of Group 0 Correct (TN)} & \text{Count of Group 0 Incorrect (FP)} \\ \text{Count of Group 1 Incorrect (FN)} & \text{Count of Group 1 Correct (TP)} \end{bmatrix}$$

A more detailed list of classification terminology is presented in Appendix B.

4.1.3 ARA Data Acquisition and Structure

A brief discussion of the ARA stations was given in Chapter 1.5.2. This section presents a deeper discussion of how data is acquired in the ARA experiment, the data structure, and how to access waveforms and variables.

After a signal induces a voltage in an in-ice ARA antenna, it goes through a 150-850 MHz band-pass filter and a 450 MHz notch filter before an amplifier. The signal is then converted to a fiber-optic signal at the Downhole Transition Module (DTM) to travel to the surface. Once there, the Fiber Optic Amplification Module (FOAM) amplifies the signal by another 40 dB before passing it to the Data Acquisition (DAQ) box. Next, the DAQ splits the signal to the Digitizing Daughter Boards (DDAs) and the Triggering Daughter Boards (TDAs). The DDAs digitize the signal while the TDAs handle the triggering.

The ARA Triggering and Readout Interface (ATRI) board receives the signal from the DDA and TDA boards. If the input from the TDAs indicates a trigger, the ATRI board reads out the signal from the DDAs. Triggering occurs when three out of the eight VPol or HPol antennas detect a signal greater than about 5 times the

average thermal noise within 170 ns (the time for an EM wave to cross the array). The exact threshold value is automatically adjusted via a servo to trigger at a rate of approximately 5 times per second. A triggered event is sent through a fiber-optic network to the IceCube Counting Laboratory. The final event is the waveforms for each antenna and general event information that is eventually stored in a ROOT file.

ARA searches involve a blinding scheme with a 10% burn sample used for initial analysis and a 90% sample used for final analysis. The burn sample is used to determine cuts and estimate background without biasing the final analysis. Since a low number of neutrinos are expected it is unlikely, but not impossible, that a neutrino event could exist in the burn sample. Several runs and time frames contain bad data due to calibrations, broken equipment, and sizeable anthropomorphic noise. These anomalies were removed from the data set.

For each run, which is just a sequential directory, one ROOT file is created. ROOT is an Object-Oriented C++ framework built by CERN for handling large data sets. ROOT files contain TTrees, which are a type of column-oriented data set. For ARA each row in the TTree contains the waveforms for each antenna and additional information about an individual event. Each TTree stores the events from one run, totaling about 225,000 per run. The ARA collaboration has built a set of C++ libraries called ARA ROOT, which is used to read and analyze the data files.

Once the TTree has been properly read, one can loop through the events of the run to access the waveforms. ARA ROOT handles the basic calibration of events through the UsefulAtriStationEvent class with the kLatestCalib flag (or UsefulIcrrSationEvent for Testbed and early A1 events), which turns the raw event into a "useful" event. This includes pedestal subtraction, which is the process of testing for a DC offset at regular

intervals and correcting the drift. This process also conducts voltage calibration, timing calibration, and cable delay correction. The useful event is created with the following code.

```
UsefulAtriStationEvent *realAtriEvPtr = new UsefulAtriStationEvent(
    ↪ rawAtriEvPtr, AraCalType::kLatestCalib);
```

For each useful event, a quality check is also done, and events that fail are removed. This cut is based on a rudimentary glitch detection, including waveforms with too few samples, sample timestamps that are not sequential, and other known hardware glitches. Events are skipped with the quality cut using the following code.

```
AraQualCuts *qual = AraQualCuts::Instance();
bool this_qual = qual->isGoodEvent(realAtriEvPtr);
if(!this_qual){continue;}
```

Once the useful event is created and the quality cut is done, the waveform and spectrum can be created as TGraph objects.

```
// First the waveform is constructed
TGraph *waveform = realAtriEvPtr->getGraphFromRFChan(0);
//Next, an interpolated waveform is created
TGraph *interpolated_waveform = FFTtools::getInterpolatedGraph(waveform,
    ↪  0.5);
// The padded waveform adds 0V samples to the beginning
// and end of the event
TGraph *padded_waveform = FFTtools::padWaveToLength(
    ↪ interpolated_waveform,2048);

// Finally the spectrum is made from the padded waveform
TGraph *spectrum = FFTtools::makePowerSpectrumMilliVoltsNanoSecondsdB(
    ↪ padded_waveform);
```

With the TGraphs, it was then possible to find descriptive variables that could potentially be used for the GP. For this analysis, the interpolated waveform and spectrum were typically used. ARA ROOT also has a number of built-in functions that were used. For custom functions, one can loop through the individual points of the waveforms and then use the GetPoint function.

```
interpolated_waveform->GetPoint(i, x_value, y_value);
```

The Monte Carlo simulation software AraSim (used in the GENETIS GA) can produce waveforms depicting how the ARA detector would have seen neutrino events. Finding these waveforms and obtaining the variables is done in the same process as the ARA data, except that the AraSim events need to be filtered to only view events that would have triggered the detector.

4.2 Karoo GP Introduction

Karoo GP is a Python-based GP application suite developed by GENETIS collaborator Kai Staats as part of his graduate studies [108]. Karoo GP can conduct GP evolutions of tree structures with large sets of data, thanks to the multi-core and GPU support through the TensorFlow library [109]. Karoo GP allows the user to set various GP parameters and input data for regression and classification analysis. This section will describe how Karoo GP works, modifications made to Karoo GP, and finally, the process for conducting runs. Karoo GP can be downloaded from the GitHub site, and instructions are contained in the user manual.

Karoo GP evolves tree-based functions in one of three modes: matching, regression, and classification. *Matching* attempts to determine the exact form of an expression. *Regression* is a minimization function that aims to produce equations that accurately predict a solution. Finally, *classification* is a maximization function that attempts to divide the provided data into two or more groupings. Karoo GP is built with an example for each of these modes, solving a simple expression for matching, Kepler's third law of motion for regression, and the Iris flower problem for classification. The remainder of this discussion will focus on the classification mode, as this was the core focus of this chapter.

4.2.1 The Karoo GP Loop

As with all evolutionary algorithms, Karoo GP follows the same structure described previously: initialization, fitness evaluation, creation of new generation, an iterative loop, and termination. The user is able to alter all of the evolutionary parameters as needed and in the middle of runs, which allows customization and fine-tuning of the GP on the fly. The initialization begins by simply constructing the given number of hypothesis trees (the individuals) of the predefined size and shape.

For the classification kernel, the default fitness function is simply the sum of correct predictions. For each row in the training data set, the equation representing the individual is evaluated, giving a resulting number, which acts as a hypothesis. If this number is less than or equal to 0, then the individual tree predicted the row should be group 0. If the number is greater than 0, the tree predicted the row should be group 1. The fitness function then compares the prediction to the actual group given in the solution column. This calculation is done for each row in the data set, and the sum of the correct predictions is the final fitness function for the single individual. The process must then be repeated for all individuals to obtain fitness scores for each of them.

For building a new generation, Karoo uses the same steps of parent selection and genetic operations. While the GENETIS loop employs multiple selection methods, Karoo GP only uses tournament selection. As described before, tournament selection is when a group of individuals is chosen from the population, and the highest-scoring individual from the group is picked to be a parent used in generating a new population. Karoo GP defaults to tournament groups consisting of 7 individuals and increases the

size by 7 for each additional 100 individuals in the population after the first 100 [107]. An individual can be chosen multiple times to be a parent for the next generation.

The user defines what percentage of each genetic operator is used in creating the new generation. The operators are not done in succession, and only one is used when creating the new child. By default, Karoo GP uses 10% reproduction, 0% point mutation, 20% branch mutation, and 70% crossover, although the user is able to change each of these parameters, both before and during a run. Point mutation is set to zero by default since branch mutation can have the same effect when a terminal node is selected. Changing these parameters during a run allows the user to interactively help the algorithm out of a local minimum and explore the parameter space more fully.

Karoo GP runs the evolutionary loop a set number of generations. After the run concludes, the user can see the results and decide to end the run completely or continue for additional generations (with or without changing parameters). For a complete discussion of how to run Karoo GP, including the data file format, various parameters, and steps of conducting a run, see Appendix C.

4.2.2 Karoo GP Modifications

After initial testing of Karoo GP, it became apparent that some modifications would be necessary for the desired classification analysis. First, TensorFlow, the Python library that acts as the basis for Karoo GP, was recently upgraded, and many of the functions used had been depreciated. Therefore, it was necessary to work through the Karoo GP source code and update the TensorFlow functions as necessary.

A significant endeavor was undertaken to introduce a new fitness function for classification, which I called weighted classification. This was necessary because of the inherent equal importance built into classifying each group in the default fitness function. If an individual tree correctly predicts a row's group, the fitness score increases by 1, regardless of the group. This makes sense for the famous Iris classification problem, where there is no difference in importance between each group. However, there are classification problems where it is desirable to favor one group over the other. In the case of the analysis presented here, where background data is contrasted with simulated event data, either group could be viewed as more important.

Suppose one wanted to use Karoo GP for an early cut to eliminate most background but keep all the potential signal events. In that case, the function could be set up to favor actual signal and thus predict signal events extremely well and be less concerned if the background was incorrectly classified. Conversely, if the analysis hoped to determine with high confidence if an event was signal and not background, it could favor background events and thus be able to more confidently eliminate all background events even if some signal events were also incorrectly classified. Note that this could be artificially done with Karoo GP by using a data set that contains an unequal number of each group; however, building a new function allowed for more flexibility and control.

With this rationale in mind, the goal was to build a new fitness function within Karoo GP that would allow the user to set the importance of each group. The simplest way to do this was to alter the fitness score, so that correct predictions caused an increase by a different amount for each group. During the Karoo GP initialization, a parameter was added where the user would enter a weight, w_0 between 0 and 1 for

Group 0. The weight of Group 1, w_1 would then be automatically set to 1 - w_0, so the sum of the weights is 1. The fitness score then becomes:

$$\text{Fitness Score} = \text{C}_0 \times \text{w}_0 + \text{C}_1 \times \text{w}_1 \tag{4.1}$$

where C_i is the total number of correct predictions for group i, and w_i is the corresponding weight.

Karoo GP is open source and well-commented, with some instructions on how to implement new fitness functions. The most significant challenge was learning and implementing the code using TensorFlow. The following code excerpt is a simplification of the final code to update the fitness function. Keeping with the terminology of Karoo GP, the "solution" is the actual group of the row, and the "result" is the prediction of the individual.

```
# First the Group 0 rows are evaluated
    # rule11 is true if the row is in Group 0
    rule11 = tf.equal(solution, 0)

    # rule12 is true if the result (prediction) is less than or
    ↪ equal to 0
    rule12 = tf.less_equal(result, 0)

    # rule13 is true if both the above lines are true
    rule13 = tf.logical_and(rule11, rule12)

    # Next the True/False values are converted to 1/0
    r13 = tf.cast(rule13, tf.int32)

    # The number of correct Group 0 predictions are summed # and
    ↪ converted to a float
    G0_Correct = tf.cast(tf.reduce_sum(r13), tf.float16)


# Next the Group 1 rows are evaluated
    # rule21 is true if the row is in Group 1
    rule21 = tf.equal(solution, 1)

    # rule22 is true if the result is greater than 0
    rule22 = tf.greater(result, 0)

```

```
25      # rule23 is true if both the above lines are true
26      rule23 = tf.logical_and(rule21, rule22)
27
28      # Next the True/False values are converted to 1/0
29      r23 = tf.cast(rule23, tf.int32)
30
31      # The number of correct Group 0 predictions are summed # and
        ↪ converted to a float
32      G1_Correct = tf.cast(tf.reduce_sum(r23), tf.float16)
33
34  # The weights are then determined
35      # weight_0 is set to the user inputted value
36      weight_0 = self.w0
37
38      # weight_1 is set to 1 - weight_0
39      weight_1 = 1-weight_0
40
41  # The new fitness function is then calculated
42      new_fitness_score= (G0_Correct*weight_0) + (G1_Correct*weight_1)
43
44  # For comparison and debugging, the normal classify fitness
45  # score is also determined:
46      old_fitness_score_classify = G0_Correct + G1_Correct
```

4.2.2.1 Validation of New Fitness Function

In order to ensure the custom fitness function was operating correctly, several tests were conducted. Primarily, 8 runs were conducted with different Group 0 weights, ranging from 0.1 to 0.9. These runs evolved solutions over twenty generations with data containing only one variable. The results are presented in Fig 4.2, which illustrates that altering the weight correctly favors one group and allows adjustment of the final result to suit specific needs.

With a single variable, the equation Karoo GP finds can be set equal to zero and solved to determine a constant, which acts as a single variable cut on the data. By plotting the density curves for the data and vertical lines for the constants found at some example weights, we can see how the different weights lead to a shift in the cuts to maximize the fitness score, as illustrated in Fig 4.3. With the Group

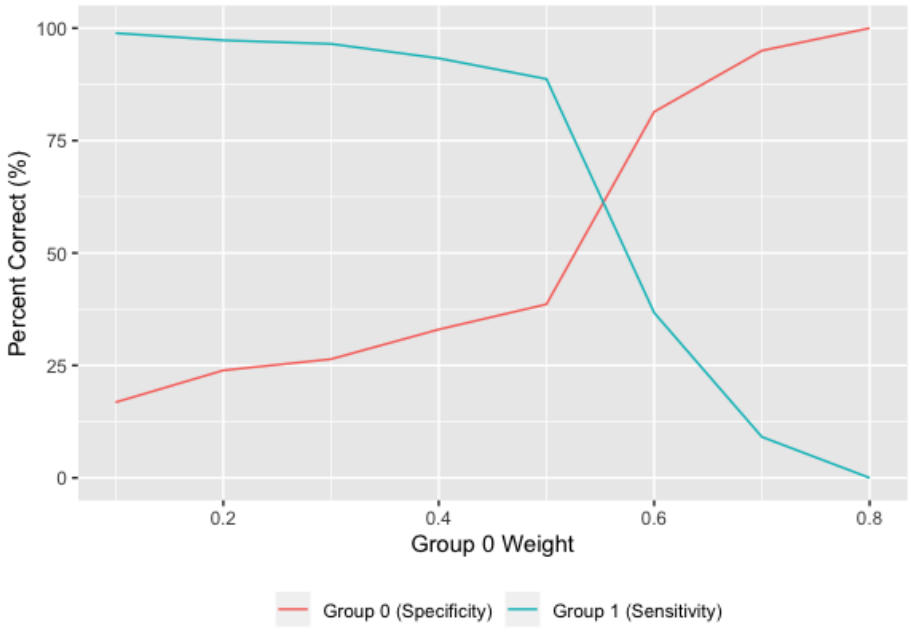


Figure 4.2: Customized fitness function check, showing the effect of the weight on the TNR (specificity) and TPR (sensitivity). When the weight is adjusted, the rate of change between TNR and TPR is dependent on the variable(s).

0 (Background Events) weight set to 0.1 and the Group 1 (AraSim Events) weight set to 0.9, the algorithm determined that a cut at -105 mV (shown in blue) would maximize the fitness score by classifying all events to the right of the vertical line as Group 1. Because the incorrect Group 0 classifications are less costly, this weight ensures that most signal events are classified as Group 1, with only a small portion as Group 0. Conversely, the red vertical line shows the cut when the Group 0 weight is set to 0.8, and thus, the background events are worth more to the fitness score. Here all the events less than 230 mV are classified as Group 0. In this case, 100% of events were classified as background, because there was no value in which the AraSim events exceed the background at the sides of the distributions. The black line shows the cut when both groups are weighted equally, here the value is equivalent to the Karoo GP's default classification fitness function. The cut occurs at the intersection of the two curves, where the maximum number of events will be classified correctly.

The effect of the weight is somewhat dependent on the classification abilities of the variables. For a variable that is not very discriminatory, such as the integrated voltage used in the charts above, changing the weight causes significant shifts within a narrow band of weights. However, variables that are more discriminatory may require large or extreme weights to shift the threshold sufficiently.

4.3 Classification Analysis

The following investigation loosely follows the steps of the LIGO data analysis mentioned above [34]. The basic premise is to combine actual ARA data (that is almost surely all background) with simulated neutrino events from AraSim, and then

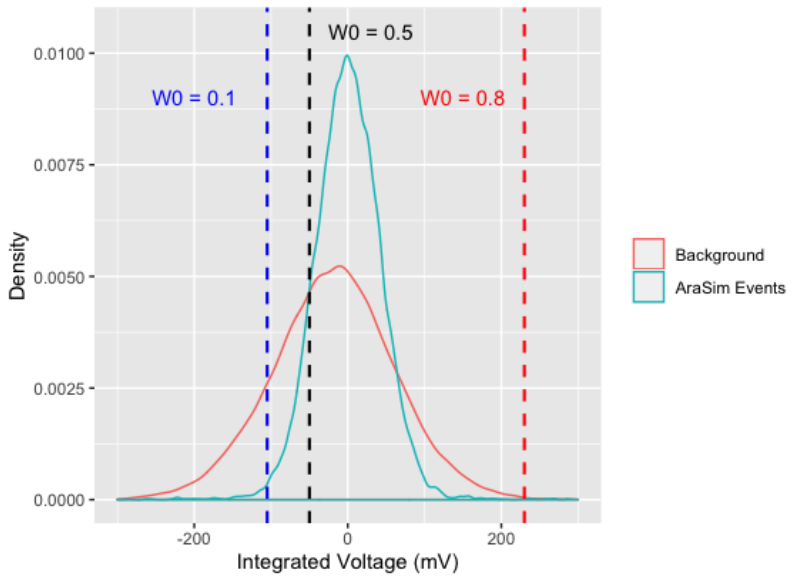


Figure 4.3: Density distribution of the Integrated Voltage, demonstrating how the change in weights of the fitness function alters the location of single variable cuts. W0 represents the weight of the Group 0 or background group.

use Karoo GP with the modified classification kernel to generate functions to predict the groups. The general procedure is outlined below.

1. Extract variables from the ARA and AraSim data
2. Prepare the data files
3. Variable Exploration
4. Run Karoo GP to obtain classification functions and explore results

This procedure was first conducted on an initial set of variables. In order to improve the accuracy of the classification, a new variable was extracted, and the procedure was repeated. The following sections describe those results.

4.3.1 Analysis with Initial Variables

1. Extract variables from the ARA and AraSim data

The data for this investigation comes from the 2013 operation of the A2 detector. The initial results only utilized a single detector channel. The 100% data set was used, as there was no need to burn the 10% that is typically done for a search as there is no need to find cuts or background estimates before the classification analysis. Since higher SNR (HSNR) background events are more difficult to distinguish from neutrino events than low SNR (LSNR) background events, this investigation focused on high SNR background events, which are hereafter referred to as background events, unless otherwise specified. The threshold between high and low SNR events was chosen to be an SNR of 5.5, which causes the HSNR events to represent about 12% of the data. This value was chosen somewhat arbitrarily but is an approximate cutoff that is an estimate of other cuts used by ARA. Additionally, data was only taken from the summer months (October-February) because these months tend to have more noise. The data extraction was done using the Ohio Supercomputer Center and the final data set contained 50 runs totaling 10.3 million events, with 1.2 million calibration pulsers and 111,000 thousand HSNR events. The initial investigation used the following list of variables, which were extracted using the procedure outlined above.

- **Peak voltage** - The highest voltage obtained in the waveform
- **Peak time** - The number of nanoseconds since the start of the event to the peak voltage
- **Sum voltage** - The sum of the voltage for all data points in the waveform
- **Integrated voltage** - The integrated area under the waveform

- **Peak power** - The highest power obtained in the spectrum
- **Peak frequency** - The frequency that the highest power occurred at in the spectrum
- **Integrated power** - The integrated area under the spectrum
- **SNR** - The signal to noise ratio of the waveform

The same variables were extracted from an AraSim data set that was generated using a neutrino energy of 10^{18} eV. The total number of triggered AraSim events was 12,320.

2.Prepare the data files

Next, the ARA data and simulated events need to be combined. A complete flow chart of the data can be seen in Fig. 4.4. Due to a limitation in TensorFlow, the maximum data set that could be run on Karoo GP is 150,000 rows. Karoo GP also requires an equal number of rows for each group being classified. Consequently, 75,000 of the HSNR events were extracted from the full data set using the R sample function (which uses a uniform distribution rejection method). The remaining 36,000 were saved as an external sample for testing after Karoo GP runs were completed. To match the HSNR sample, the AraSim data set was duplicated 10 times, and then 75,000 events were extracted in the same matter. Duplication of the simulated events was not ideal, but necessary due to time constraints. Similarly, 36,000 AraSim events were saved from the remaining duplicated data set for the external testing sample. A new "s" column was made in each data set containing a 1 for the AraSim events and a 0 for the HSNR sample. This is used by Karoo GP in the internal training and evaluating steps to see if an individual prediction was correct. Karoo GP uses

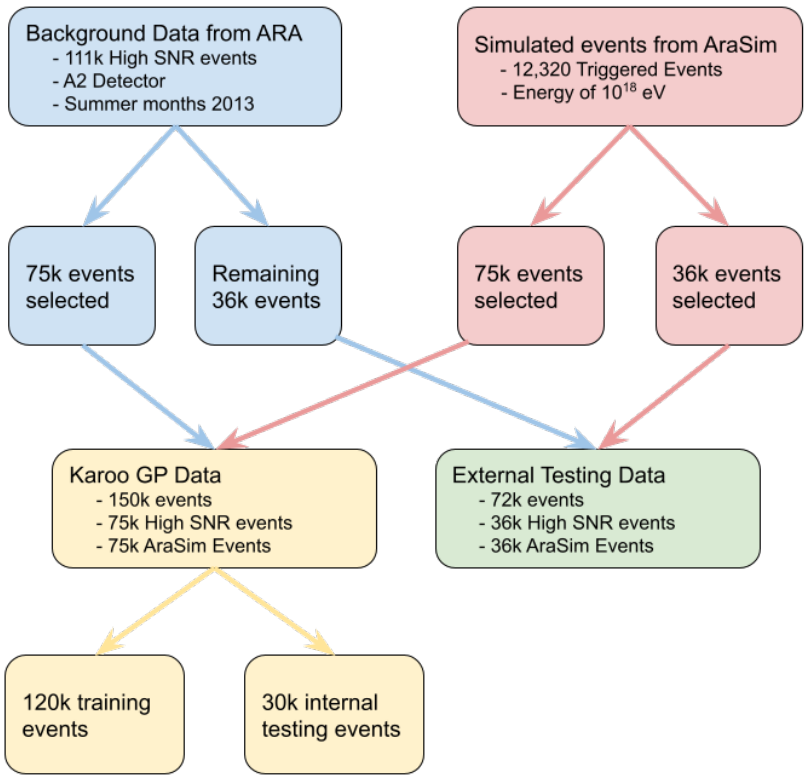


Figure 4.4: Illustration of how data is divided between throughout the GP workflow. Shown is the ARA background data (blue) and AraSim simulated data (red), which are combined to run Karoo GP (yellow) and a post GP blind testing (green).

80% of the data provided for training the algorithm and 20% for internal evaluation. Each time Karoo GP is run, the data is split again. Finally, the training sample was created by combining the 75,000 events from each group and reordering them. The blinded test sample was created by combining the 36,000 event groups.

3. Variable Exploration

Before running Karoo GP, the density distributions of each variable were plotted to better understand how the different groups were described by that particular variable. The results are seen in Fig. 4.5.

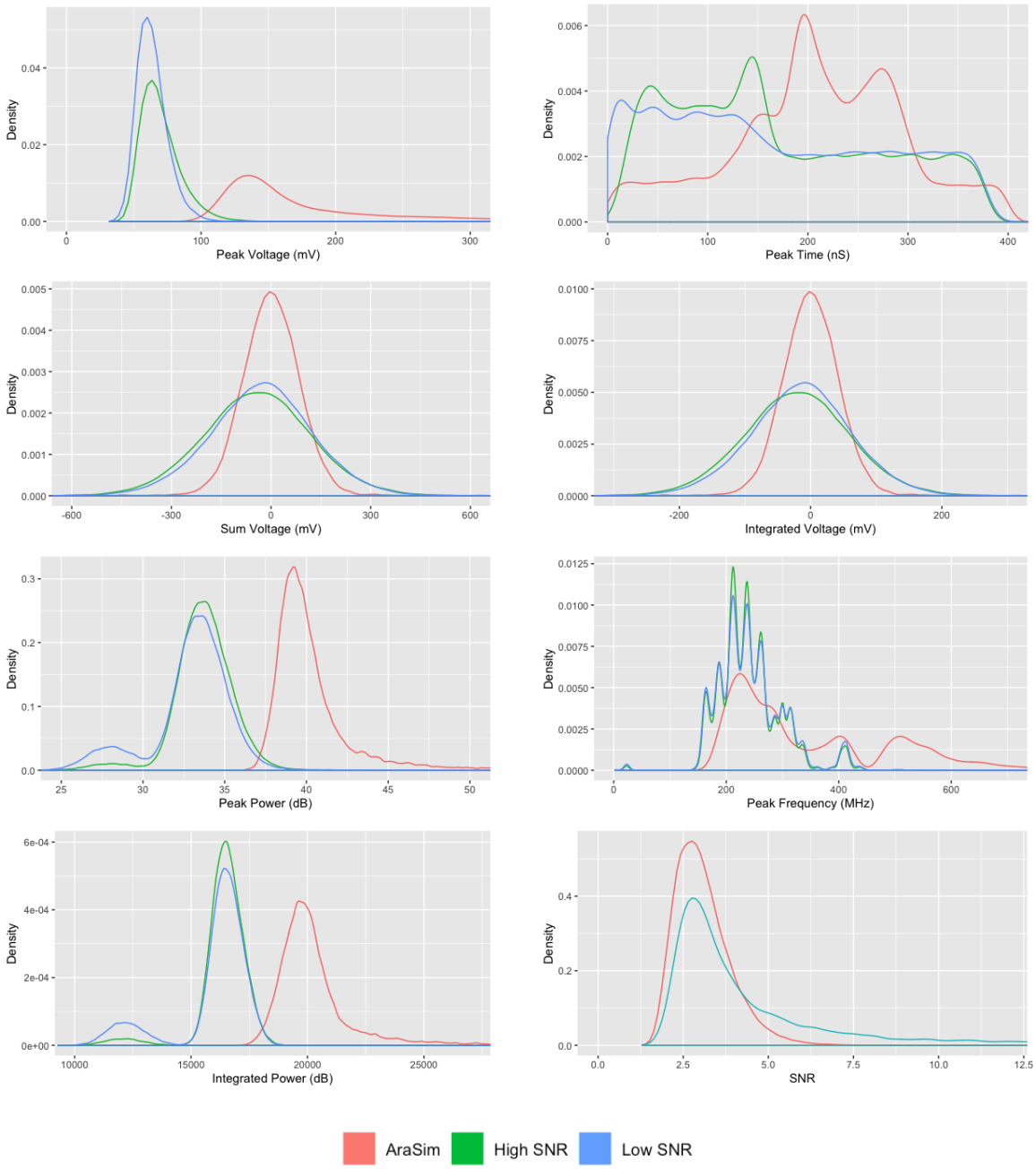


Figure 4.5: Comparison of the density distributions for each of the initial variables. Note the SNR plot only shows two groups: AraSim (red), and the combined HSNR and LSNR background (teal).

Fig. 4.5 provides some interesting insight that can inform the investigation. First, from these initial variables, peak voltage, peak time, peak power, and integrated power are the most descriptive of the class. Furthermore, the high SNR group has more overlap with the AraSim events than the low SNR group. This means that the results of using high SNR and AraSim events can be used for low SNR classification. The sum voltage, integrated voltage, and peak frequency variables do not discriminate between the groups well independently. However, beyond single variable uses, the GP could find combinations of the seemingly poor variables that are effective at classification. The SNR variable only shows the AraSim events in red and the combined high SNR and low SNR background in teal. SNR was excluded from further analysis since when comparing high SNR events and AraSim events, the SNR cutoff provides an artificial comparison between the groups. Moving forward in this dissertation, "background" will only refer to the selected high SNR events.

In order to test the efficacy of the GP, each variable was individually run through Karoo GP with equal weight for each group (high SNR and AraSim). The expected result is the intersection of high SNR and AraSim density curves that maximizes the number of AraSim events to the right of the line. Karoo GP was able to exactly determine these values for each variable accurately.

Furthermore, plotting 2D distributions of a combination of two variables, as shown in Fig. 4.6, helped to see whether any could potentially work together to discriminate between the groups. Two variables that appeared to be able to discriminate between the groups well were integrated power and peak voltage, which were used in the first multi-variable Karoo GP runs as shown in the figure.

4. Run Karoo GP to obtain classification functions and explore results

A Karoo GP run was conducted on the test sample with only integrated power and peak voltage (given that the 2D distributions seemed promising) with a tree depth of 5 and equal weight for 50 generations with 100 individuals. The highest scoring individual of generation 50 was defined by the equation $\frac{3P_{\text{int}}}{25} + 4V_{\text{p}} - 2598$, where P_{int} is the integrated power and V_{p} is the peak voltage. The resulting confusion matrix is given below, which amounts to an accuracy of 98.9% with a TNR of 99.2% and a TPR of 98.7%.

$$\begin{bmatrix} 14,801 & 116 \\ 202 & 15,031 \end{bmatrix}$$

Fig. 4.6 shows an example of the distribution of the groups using the integrated voltage and the peak voltage. The solid line shows the threshold created by the highest scoring individual. The vertical and horizontal lines show how the integrated power and peak voltage respectively, would separate the groups alone. This result demonstrates high levels of accuracy with only a few key variables. However, it is still short of the initial goal.

Next, Karoo GP was run with all the initial variables to see if any better hypotheses could be evolved. The default Karoo GP parameters were used with 100 individuals over 50 generations and a tree depth of 8. The highest-scoring individual was defined by the equation $\frac{-9P_{\text{int}}}{100f_{\text{p}}} - 2P_{\text{p}} + V_{\text{p}} - 76/3$, where f_{p} is the peak frequency, and P_{p} is the peak power. The resulting confusion matrix is given below, which amounts to an accuracy of 97.3% with a TNR of 98.4% and a TPR of 96.2%. The resulting confusion matrix is given below.

$$\begin{bmatrix} 14,563 & 583 \\ 228 & 14,626 \end{bmatrix}$$

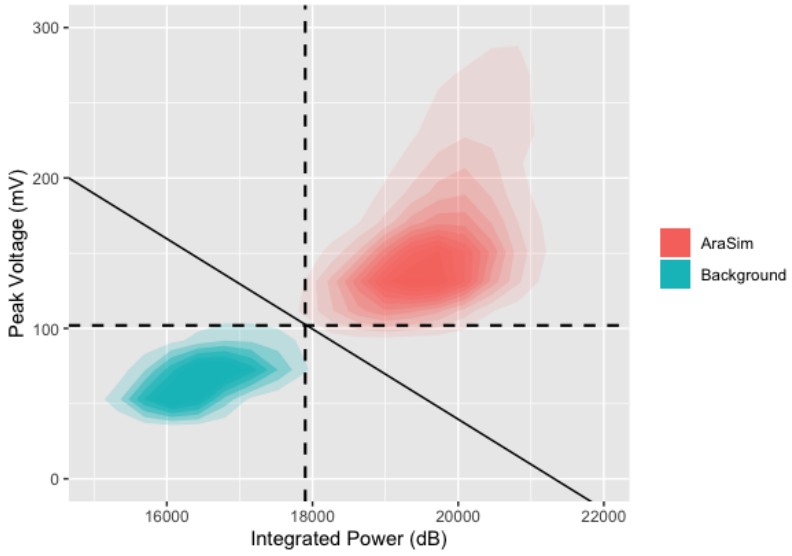


Figure 4.6: Result of two variable GP analysis showing clear discrimination. The solid line is the threshold created by the equation $\frac{3P_{\text{int}}}{25} + 4V_{\text{p}} - 2598 = 0$ found by GP using equal weights. The dashed lines show where the single variable would evenly separate the groups.

This result contains several interesting insights into classification algorithms. This result with all variables performed worse than the two variable (integrated power and peak voltage) result given above, despite only having additional variables (the two variables used above were still present). First, this is an example of overfitting, where the resulting model is more complicated than necessary. The confusion matrix is built from the 20% of the data Karoo GP saves for evaluation, so the model was built on a different set of data and became more complicated to match the training set too closely. Second, this illustrates the importance of the initial settings of the run. By allowing trees to grow more complicated (larger depth), overfit results are more likely. Additionally, the more variables, the more challenging it is to search the parameter space exhaustively. Given enough generations, the GP would have found a result

that exceeded or matched the two variable result; however, the overfitting could have been even worse. Using the same number of generations as the two variable run, the many variable run was unable to equal the results of the simpler run. Finally, this demonstrates the stochastic nature of GPs, and how even repeating the same parameters and data multiple times could give different results.

Running the algorithm again, but with a smaller maximum tree depth, the GP converges to $\frac{-4P_{\text{int}}}{100}+7V_{\text{p}}-8$. The resulting confusion matrix is given below, which gives a TNR of 94.7% and a TPR of 97.7% (accuracy of 96.2%). Of particular interest is that the algorithm converged to the two variables used alone above. This highlights the GP's ability to determine which variables are most discriminatory. However, this result is worse than when only those two variables were used because a larger parameter space was being searched in the same number of generations. Having not yet reached the goal it was clear that additional variables would be needed.

$$\begin{bmatrix} 14,308 & 792 \\ 331 & 14,569 \end{bmatrix}$$

4.3.2 Variable Extraction and Results

In order to improve the classification results and reach the goal of a TNR of 99.9% and a TPR above 90%, it was apparent that additional variables were necessary. In order to find new variables, events from both groups were visually inspected and compared. Fig. 4.7 shows four representative waveforms from each group. The High SNR waveforms were explicitly chosen to identify the events that the peak voltage alone would not classify. In other words, high SNR waveforms where the peak voltage exceeded 100 were examined in particular.

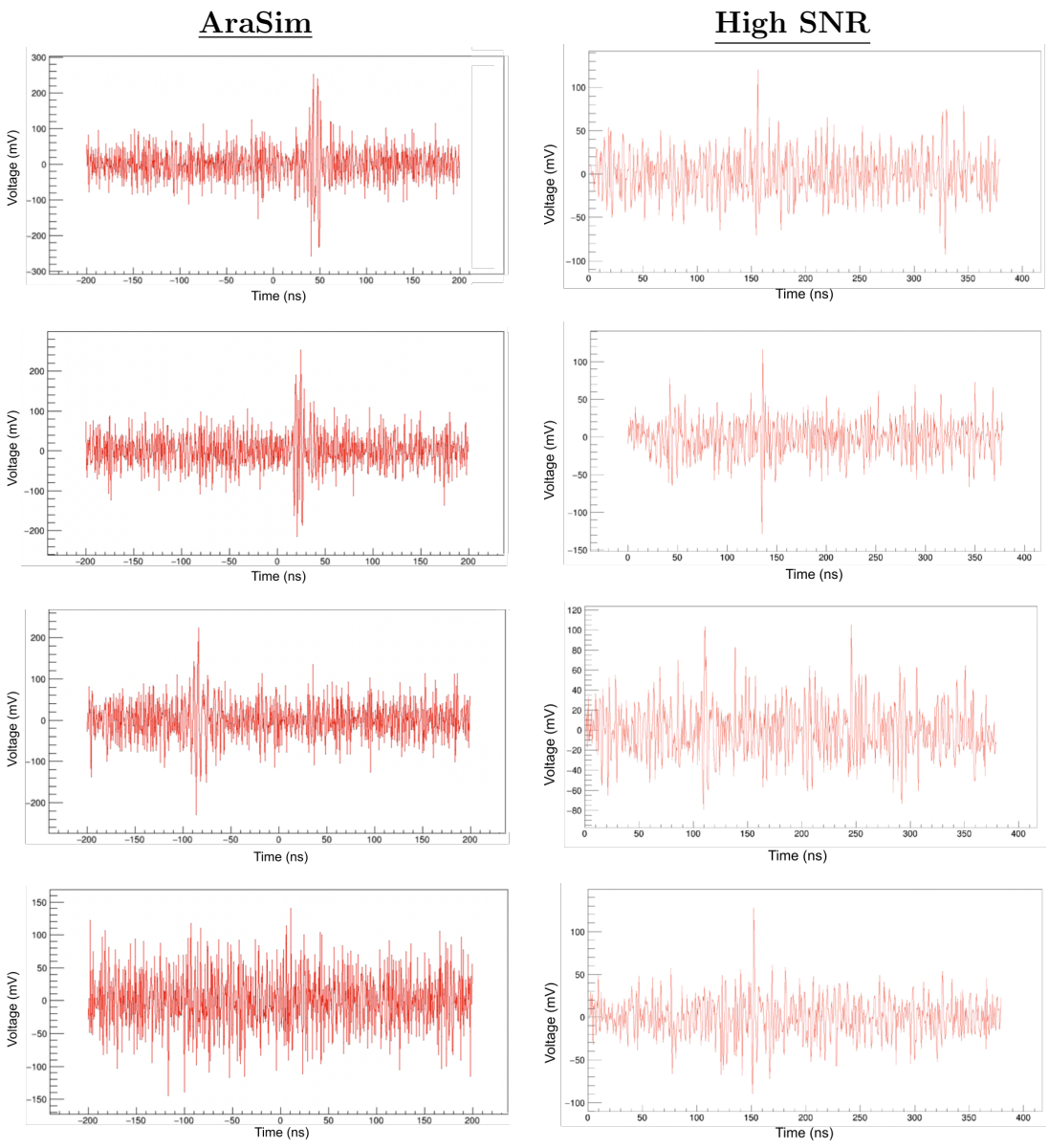


Figure 4.7: Example waveforms from AraSim (left column) and High SNR events with a peak voltage above 100 (right column).

Examination of the waveforms in Fig. 4.7 reveals some additional features that may help the GP discriminate better. The most straightforward takeaway was that the AraSim events typically had multiple high points near the peak voltage. In contrast, the peak voltage in high SNR background events was typically a single point rising above the noise (or multiple high points spread throughout the waveform). Another noticeable feature was that more symmetry existed in the AraSim events; near the peak voltage there were also many negative points that loosely mirrored the higher positive points mentioned above. In other words, the envelope of the AraSim events had more vertical symmetry about the horizontal axis than the background events. The fourth AraSim plot does seem to be an exception to this rule, which is most likely due to the event triggering from hitting channels other than the one being investigated. However, this event does have elevated voltages throughout the entire waveform.

In order to make a single variable that described these observations, I first found that the width of the elevated voltage sections in AraSim averaged around 50 ns (or $\pm$25 ns around the peak). I then explored different thresholds in the range around each peak and found that the most discriminatory variable was counting the number of samples whose absolute value exceeded 40 mV, which I defined as the *count of elevated voltage near peak*, C_{evnp}. Since the interpolated waveform was used with 0.5 ns interpolation, this variable could have values between 1 (the peak voltage) and 101 (if all points were elevated). This variable succeeds in capturing (1) the presence of additional high values near the peak in AraSim events, and (2) the symmetry of AraSim events, because the absolute value was taken. The density distribution for this variable with AraSim events, high SNR background, and low SNR background

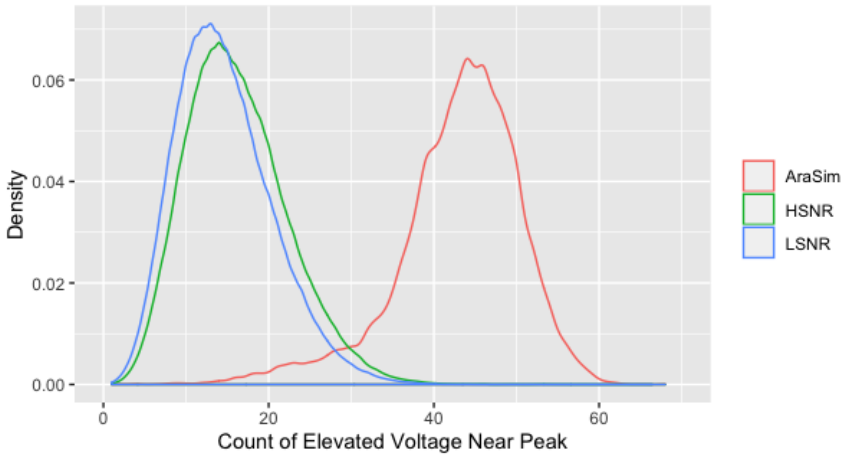


Figure 4.8: Density distributions of the new variable, which counted the number of samples whose absolute value exceeded 40 mV with 25 ns of the peak voltage.

is shown in Fig. 4.8. Armed with a new promising variable, the data was extracted and combined again in the manner described above for additional Karoo GP runs.

4. Run Karoo GP to obtain classification functions and explore results

First, the new variable was combined only with the peak voltage variable and ran with Karoo GP, with a maximum depth of 8. The highest-scoring individual was defined by the equation $C_{\text{evnp}} + 9 - \frac{2304}{V_{\text{p}}} - \frac{207552}{V_{\text{p}}^2}$, where C_{evnp} is the elevated voltage near peak term. The individual seems overly complicated, despite the excellent performance, with a TPR of 99.6% and a TNR of 99.2%. Additional testing showed that overfitting continued to be an issue at that depth. Running the GP again with a lower depth of 5 resulted in a simpler equation with almost the same performance: $4C_{\text{evnp}} + 2V_{\text{p}} - 334$ with a TNR of 99.1% and a TPR of 99.6% (accuracy of 99.4%). The confusion matrix is given below.

$$\begin{bmatrix} 14,898 & 135 \\ 54 & 14,913 \end{bmatrix}$$

Next, an additional variable, the integrated power, was added. Integrated power was chosen because it had previously shown promising results paired with peak voltage. Interestingly, after each test run (with a depth of 5), the integrated power term was dropped out of the equations, illustrating the GP's ability to eliminate unnecessary variables. The top equation was given by $C_{\text{evnp}} + \frac{V_{\text{p}}}{2} - 84$ and the confusion matrix is given below.

$$\begin{bmatrix} 14,832 & 81 \\ 117 & 14,970 \end{bmatrix}$$

At this point, the new fitness function allowing the weight of each group to be changed was implemented to favor correct background (negative) selections. Tab. 4.1 shows the results of adjusting the weight while using only the elevated voltage near peak and the peak voltage terms. The same parameters were run ten times for each weight, and an average TNR and TPR were taken. Each run had a tree depth of 5, with 100 individuals evolved over 50 generations. As expected, there is a clear trend of improving TNR and a reduction in TPR as the weight increases. At a group 0 (background) weight of 0.98, the averages exceeded the initial goals of 99.9% TNR and above 90.0% TPR.

An example of an individual that exceed the goal is $2C_{\text{evnp}} + V_{\text{p}} - \frac{584}{3}$ with a TNR of 99.93%±0.021 and a TPR of 94.85%±0.18 (accuracy of 97.4%). The confusion matrix is given below.

$$\begin{bmatrix} 14,872 & 10 \\ 778 & 14,340 \end{bmatrix}$$

The same individual was tested against the external blinded data set removed before Karoo GP, achieving a TNR of 99.92%±0.015 and a TPR of 95.04%±0.11 (accuracy of 97.5%). Fig. 4.9 shows the density distributions for each group for the

Table 4.1: TNR and TPR resulting from varying the weight when evolving individuals using peak voltage and elevated voltage near peak.

Test Number	Background Weight	AraSim Weight	TNR (%)	TPR (%)
1	0.5	0.5	98.02	99.75
2	0.6	0.4	98.28	98.77
3	0.7	0.3	98.85	98.87
4	0.8	0.2	99.41	97.62
5	0.9	0.1	99.81	97.81
6	0.95	0.05	99.86	96.07
7	0.98	0.02	99.93	96.81

testing sample and blinded evaluation samples, with the above equation plotted in both. The confusion matrix is given below.

$$\begin{bmatrix} 36,052 & 29 \\ 1,788 & 34,293 \end{bmatrix}$$

Using the weight of 0.98 for group 0, the GP was run 300 total times to generate the following histograms, demonstrating the expected range of results if one were to run the GP. The result is presented in Fig. 4.10, which demonstrates that 71% of runs resulted in a TNR greater than the target of 99.9%.

4.4 Classification of Coherently Summed Waveforms

In this section, the signals from multiple antennas are combined to extract single variables containing information from every channel and, consequently, classify even better. Coherently Summed Waveforms (CSWs) are created by offsetting each signal to maximize the correlation between the channels. The signals are then summed to provide one waveform. This analysis was conducted using only VPol antennas and with events that triggered the VPol antennas. The general procedure is given below.

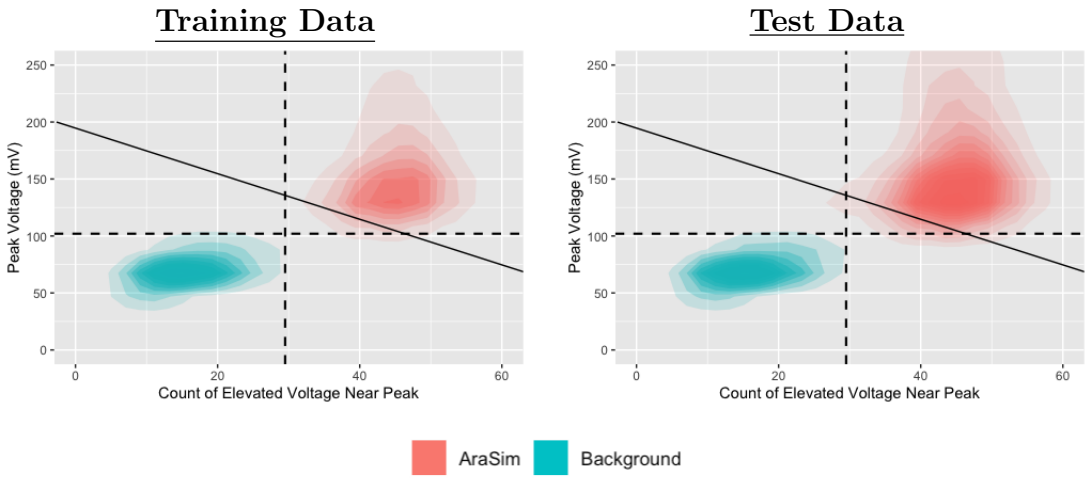


Figure 4.9: Result of two variable GP analysis (peak voltage and elevated voltage near peak) showing clear discrimination. The left panel shows the distribution of the testing data that was fed into the GP. The right panel shows the distribution of the external testing data. The solid line is the threshold created by the equation $2C_{\text{evnp}} + V_{\text{p}} - \frac{584}{3} = 0$ found by GP. The dashed lines show where the single variable would best separate the groups.

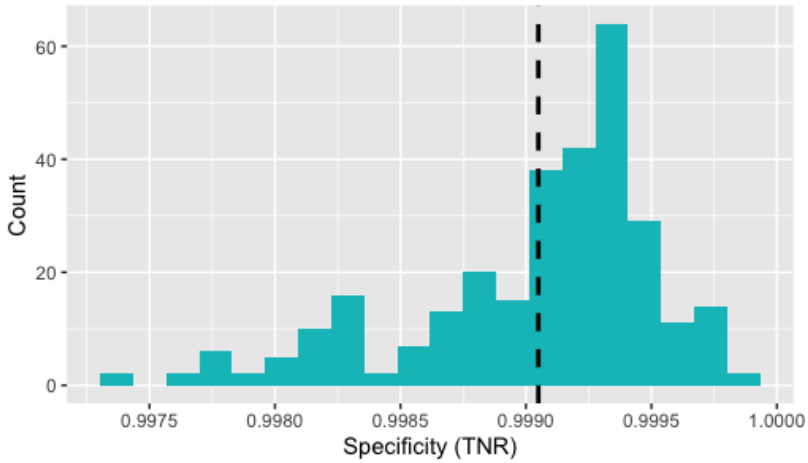


Figure 4.10: Histogram of TNR result of best individual from 300 Karoo GP runs using peak voltage and the elevated voltage near peak with a background weight of 0.98. The mean is given by the dashed line.

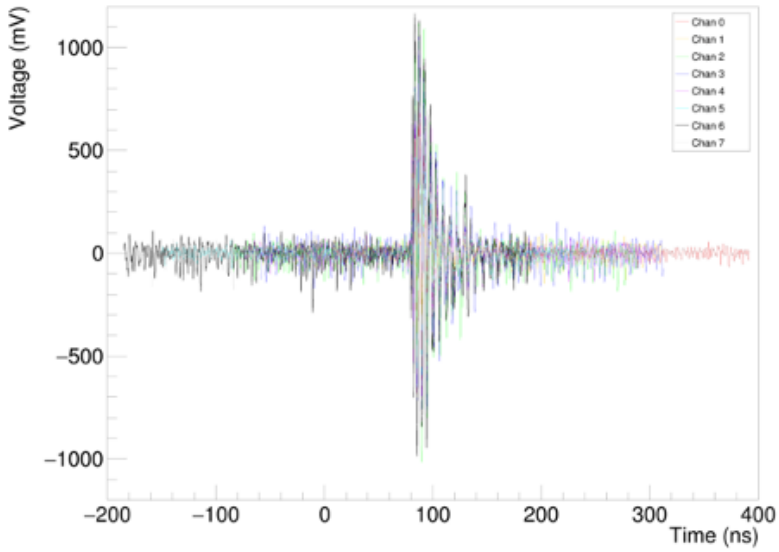


Figure 4.11: Translated waveforms of an AraSim event. The amount of translation is based on of the correlation function from each channel to the base, which in this case is channel 6.

4.4.1 Data Preperation

First, the event is checked to see if it triggered the VPol antennas (requiring 3 of the 8 antennas having an SNR greater than about 5.5). Next, the antenna with the highest peak voltage is set to be the base channel. A correlation function is found that compares the base channel to each of the other channels. Maximizing the correlation function provides the time that the signals are most highly correlated. That time acts as a delay that translates each waveform to align with the base channel. An example showing the translated waveforms of an AraSim event is given in Fig. 4.11. The translated waveforms are then summed at each point to produce the CSW shown in Fig. 4.12.

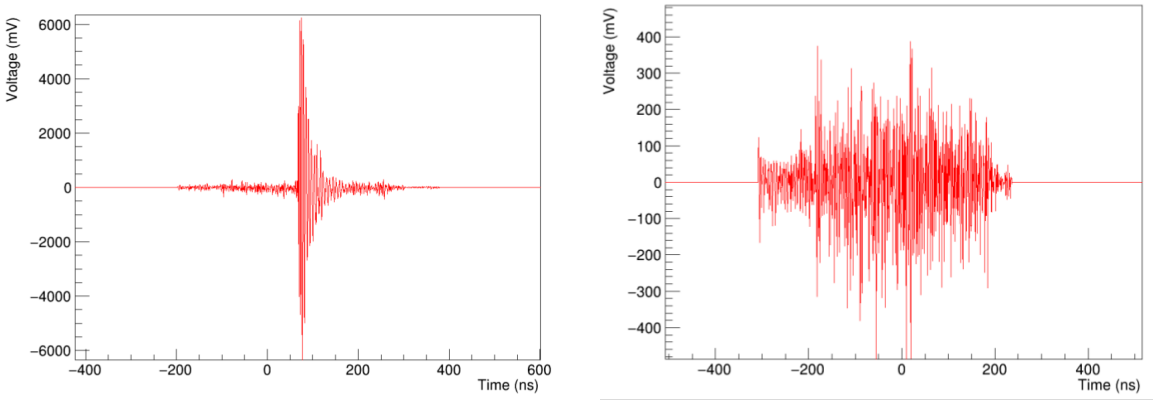


Figure 4.12: Example CSWs. Left: CSW created from the AraSim event translated waveforms shown in Fig. 4.11. Right: Example background CSW. Note the significantly different vertical scales.

Fig. 4.12 demonstrates the potential usefulness of CSWs in the classification of these events. The CSW for thermal noise and other common background will have low signal-to-noise ratios. Conversely, AraSim events will have a high correlation, so the waveforms will constructively add, producing waveforms with a much higher magnitude than the background.

Peak voltage and the elevated voltage near peak variables derived from the CSWs were both explored using Karoo GP. To account for the increase in magnitude of the CSWs compared to the single channel waveforms, the elevated voltage near peak was redefined as the count of samples within 25 ns of the peak voltage with at least a voltage magnitude of 320 mV (eight times the previous definition). The individual distributions of both variables are given in Fig. 4.13.

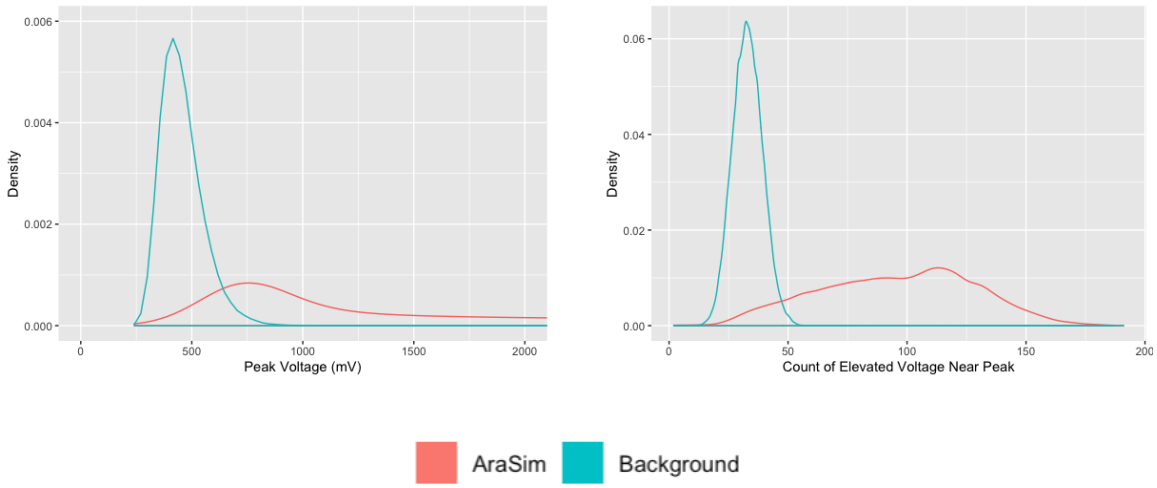


Figure 4.13: Comparison of the density distributions for the peak voltage and elevated voltage near peak for the CSWs.

4.4.2 Classification Using CSWs

The elevated voltage near peak and peak voltage were extracted and input into Karoo GP using the same procedure as before. This test was repeated 25 times. Running with additional variables did not improve the results below. Interestingly, there appeared to be a local maximum where the GP converged about 70% of the time to an equation that only used the elevated voltage near peak variable. An example individual is given by $\frac{C_{\text{evnp}}}{4} - 12$, which corresponds to a cut at 48. The confusion matrix for that individual was:

$$\begin{bmatrix} 13,370 & 102 \\ 1206 & 12,488 \end{bmatrix}$$

which corresponds to a TNR of 99.2% and a TPR of 91.2% (accuracy of 95.2%).

In the remaining 30% of runs, the GP found individuals with both variables that were able to distinguish between the groups better than previously achieved. An example is given by the individual $C_{\text{evnp}} + \frac{V_{\text{p}}}{50} - 83$, which had a TNR of 99.97%±0.015

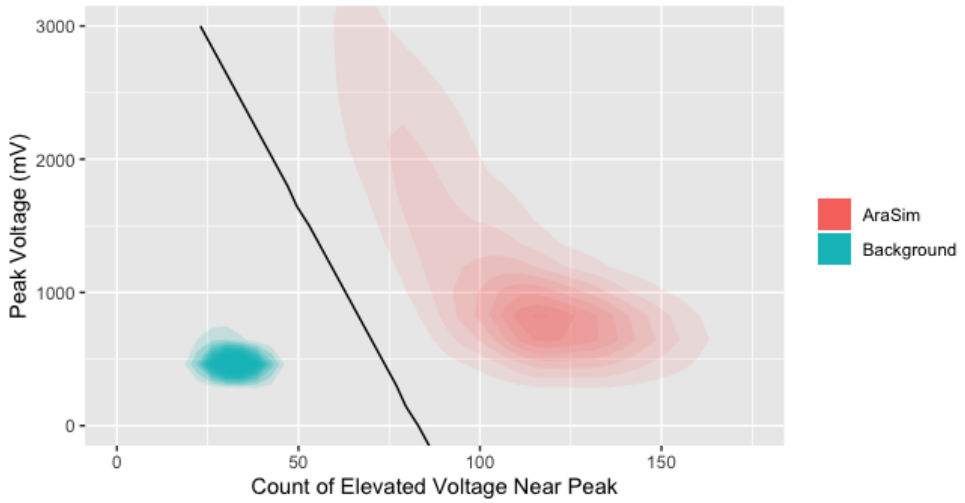


Figure 4.14: Result of CSW GP analysis using peak voltage and elevated voltage near peak. The solid line is the threshold created by the equation $C_{\text{evnp}} + \frac{V_{\text{p}}}{50} - 83 = 0$ found by GP.

and a TPR of 99.90%±0.03 (accuracy of 97.4%). The line produced by this individual is displayed on the 2D density plot of each variable in Fig. 4.14.

$$\begin{bmatrix} 13,578 & 4 \\ 16 & 13,584 \end{bmatrix}$$

4.5 Conclusion

This chapter outlines the use of genetic programming to classify ARA background and simulated neutrino events. The software suite Karoo GP was modified to allow the variation of weights for each group. Various standard variables were explored, with the most promising being the peak voltage and integrated power used in conjunction. A custom variable, the elevated voltage near peak, was extracted and found to be very discriminatory. The goal of a 99.9% classification of background was first reached using the peak voltage and elevated voltage near peak, with a background weight of

0.98. This result was further improved by creating coherently summed waveforms and running the same two variables with equal weight. This investigation highlights the potential for GP in data analysis for astrophysics.

Chapter 5: Conclusions and Future Work

This thesis has discussed two main applications of evolutionary algorithms in UHE neutrino astrophysics. Chapter 3 describes the use of GAs to design antennas for UHE neutrino experiments. Chapter 4 presents an investigation on using GP to classify background and simulated events for the ARA experiment.

The GA of Chapter 3 was created through the efforts of the GENETIS group. The overarching goal of the GENETIS group is to use evolutionary algorithms to optimize astroparticle physics experiments and ultimately advance the field. As a first project, the GENETIS group has worked to evolve the design of antennas used in UHE neutrino detection experiments. This required the integration of XFdtd and AraSim to provide fitness scores based on the science outcome of the individual.

Chapter 3 provides an overview of all projects the GENETIS team has worked on since its inception in 2017. The earlier projects provide a proof of concept. Over time, each project built on the lessons (and code) of prior work to reach the final results in the evolution of both asymmetric bicones with linear sides and with nonlinear sides. The GENETIS group will continue to add additional complexity to the design of the antennas while working towards improvements in the GA.

In addition to working directly on the GA, as a leader in the group, I mentored and managed the work of many students on the project. In that capacity, I developed

tutorials and training to facilitate the on-boarding of the constant stream of new students. I also created the GENETIS Manual described in Appendix A.

Chapter 4 provides details on the use of GP to classify UHE neutrino events, with the ultimate goal of achieving 99.9% correct classification of background and greater than 90% classification of simulated events. The analysis was built using the software suite Karoo GP. By combining actual ARA background data with simulated AraSim neutrino events, a GP could be used to build functions that classify the data.

The initial investigation involved readily available variables from the waveforms. While the initial runs were promising, I needed to complete two additional steps to reach the goal. First, a new variable was extracted that took advantage of both the envelope width and symmetry that was present in the simulated events but not in the background events. Second, a new fitness function was designed and added to Karoo GP that allowed for different priorities to be placed on the different groups being classified. These changes allowed the goal to be reached with a limited number of variables. Coherently summed waveforms (CSWs) were also constructed and the variables derived from them were examined with Karoo GP. These waveforms allow information from all 16 antennas in a station to be used in the analysis. Using CSWs, Karoo GP was even more effective at separating AraSim-generating events from high SNR events. The results of Chapter 4 provide strong evidence for the use of GP in data analysis in astrophysics and the classification of large data sets in any field.

Evolutionary algorithms are a powerful area of computational intelligence that is only recently being explored in physics. This thesis explores some uses of evolutionary algorithms in UHE neutrino astrophysics and help expand their use. The results of these projects show a promising future for evolutionary algorithms and present an

argument for their continued use in the optimization and analysis of astroparticle physics experiments.

Appendix A: The GENETIS Manual

This manual is to help anyone joining the OSU GENETIS group or anyone interested in utilizing our work. Please read Chapters 2 and 3 before going through this appendix section. As a note, it is advisable that you get an account with the Ohio Supercomputer Center (OSC) before you begin digging into running or modifying this code. All code is set to run on OSC; the job submission scripts for AraSim and directories must all be modified if this software is adapted to other supercomputers. Additionally, (1) this software is too slow to run locally, and (2) XFdtd is quite costly and needs to be purchased/installed and is already present on the OSC clusters (XFdtd used for PAEA loop, not AREA). The OSU GENETIS group runs primarily through their supercomputers. Anyone joining the OSU Connolly group can request an account. Details in setting up your account are addressed later in this section.

A.1 Introduction to the GENETIS Project

There is a future and a history of this code; both are relevant. Our first presentation of results was in April of 2018, where this group presented at April APS displaying (1) the success of the evolution of a quarter wavelength dipole antenna (essentially used as a sanity check to show our software was properly evolving) and (2) that we can evolve physical gain pattern to maximize or minimize gain in a specific

direction. Note that the old code for the dipole evolution can be found on the following GitHub account: https://github.com/hchasan/XF-Scripts. Though that is a relevant part of this project, the contents of this section are focused mainly on later works, such as the bicone evolution (PAEA project) and AREA. For now, the purpose of the bicone evolution package is to genetically evolve the inner radius, length, and the coefficients of a polynomial for each side of the cone. The purpose of the AREA software is to evolve a radiation pattern at a single isolated frequency. Our future goal is comprised of three main things (1) show we can evolve more complicated antennas (PAEA bicone loop), and (2) evolve a gain pattern to both find the maximum effective volume possible (AREA loop), and (3) use the optimized beam pattern to guide us in producing an antenna. This means that eventually, we would be able to create antenna geometry based on it evolving toward a specific set of gain patterns.

A.1.1 Tutorials

Before modifying or working with the code, skills in bash, python, some C++, the XFdtd coding language (xmacros), and Git are required. In order to help gather the skills you need to be familiar with the way this software was coded, we highly suggest the following tutorials after you have read through Chapter 2 and 3 of this thesis:

Intro to Bash

Intro to Python

Intro to Git

Intro to OSC

As we will discuss later in this section of the appendix, XFdtd code is in a file type specific to their software, called ".xmacros." This language is somewhat similar to java; however, it still differs significantly. We will discuss how to work within the GUI of XFdtd later in this appendix, where information apropos the coding language can be found in the "help" section. Though there are not any formal tutorials or introductory documentation from Remcom/XFdtd or the OSU group on the .xmacro language, this "help" function within the GUI is the most viable resource. For more information on how to access this, see section A.3.1.2

It is also recommended that anyone interested in learning more read chapter 3 of the book "Introduction to Evolutionary Computation" by Eiben and Smith; it is accessible as an electronic copy through the OSU library.

After you have completed the tutorials, please finish reading through the rest of this appendix, as it will teach you more intimate details of the software and setting it up. Once you have completed the items from this section, as well as reading this appendix, you can complete the following practice assignment aimed at reinforcing your knowledge of our code structure: Practice Getting to Know the Loop. The goal of this practice assignment is to teach you where to find each piece of software, as well as to help you master the bash scripts.

A.1.2 Where our Info Exists

If you are new to our group, we have a slack, ELOG, Dropbox, and a Google drive and can give access upon request. For historical use, here is our outdated GitHub for dipole evolution.

To use the PAEA loop, you need the following software installed:

1. XFdtd: note you need this to be purchased through Remcom if you are not an OSU Connolly group researcher. If you are working with the GENETIS team, this is already installed on the supercomputers we use.
2. PAEA Github link//
3. AraSim package//

To use the AREA loop, you need the following software installed:

1. AREA Github link//
2. AraSim package//

If you are a part of the GENETIS project, you do not need to add a copy of any of this software to your user from GitHub. Instead, we run a communal copy of PAEA on the project space PAS1960; this allows us to not have to change the directories in the main bash script each time we pull a version that was recently pushed to GitHub. Additionally, we have a limited number of XF licenses, and a maximum number of jobs that can be submitted at one time within the Connolly project folder. This makes it impossible for multiple users to be running the loop at the same time, even if the software were installed on their individual user's home directors versus the general Connolly project space; thus, as a part of the GENETIS team you will not need to install this, and you will run on a communal copy in a general project space on OSC (project space PAS1960). Proceed forward for information on how to do so.

If you have correctly set up your .bashrc as instructed in section A.2.3, you can access the PAEA directory by typing "GE60" to access the bicone folder on PAS1960.

For Connolly group users, the AREA software is set up on another researcher's account. Please reach out to the group on how to proceed until it is transferred over to PAS1960.

Set the GE60 alias with the following, as seen in section A.2.3:

```
alias GE60='cd /fs/ess/PAS1960/BiconeEvolutionOSC/BiconeEvolution/
    ↪ current_antenna_evo_build/XF_Loop/Evolutionary_Loop/'
```

If you are not part of the Connolly group, you will need to set up your bash script with the appropriate alias, and the rest of this section may not apply perfectly for you; however, it should still give you some general guidance.

A.2 Setup

A.2.1 Getting an OSC Account

The software for this project is currently housed and run on the Ohio Supercomputing Center (OSC). If you are associated with OSU, you will need an OSC account to get access. To do this, email or talk to Dr. Connolly about getting an OSC account. Include the following information in your email:

First and last name

Date of birth

Email address

Phone number

Once you request access, you'll receive an email stating you've been invited to join the project, PAS1960. Follow the link provided in the email to register your account. If you don't already have an OSC account, you'll also receive an email telling you

that a request has been made to make one for you and asking you to verify your email address. As a reminder, you won't be able to run locally and will need to run on OSC, as you will need access to XFdtd, which is purchased and installed on OSC for our usage.

A.2.2 Logging on to OSC

Once you have an account, you can log in remotely to OSC. If you're running Mac OS or Linux, simply open the terminal. If you're running Windows you will need to get bash set up to work on Windows before proceeding to the next step.

Open the terminal and type:

```
ssh -XY your_username@pitzer.osc.edu
```

This will log you into Pitzer. Input your password and hit enter.

A.2.3 Setting up your .bashrc

Once you are able to log onto OSC, you need to set up your .bashrc file. This should be done before running any GENETIS code. If you are unfamiliar with what a .bashrc is, it is a bash shell script that the computer automatically runs each time you log in to OSC. You can put any alias to commands in this file (refer back to the Bash tutorial in section A.1) and the system will automatically learn/follow these abbreviations each time you open a new terminal window. The idea in utilizing your .bashrc is to set up your variables, functions, and aliases. Below is the contents that need to be added to your OSC .bashrc when you set up your account. Note that this is only specific to OSU students working in the Connolly group; this will not work for anyone not authorized to run within this project space. Additionally, note that

.bashrc files are hidden files in your home user directory. In order to see it you need to enter the command "ls -a" (instead of "ls") and it should appear. You can edit this using any editor you prefer (vim, emacs, etc).

```
# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

#For GENETIS
module load gnu/7.3.0
module load mvapich2/2.3
module load fftw3
module load python/3.6-conda5.2
module load cmake
export CC=`which gcc`
export CXX=`which g++`

#For running ARASim
source /cvmfs/ara.opensciencegrid.org/v2.0.0/centos7/setup.sh
source /fs/project/PAS0654/BiconeEvolutionOSC/new_root/
    ↪ new_root_setup.sh

#Bicone GENETIS directory shortcut alias to PAS1960
alias GE60='cd /fs/ess/PAS1960/BiconeEvolutionOSC/BiconeEvolution/
    ↪ current_antenna_evo_build/XF_Loop/Evolutionary_Loop/'
```

A.3 The PAEA Software

In this section, we will be breaking down the stages of the evolutionary loop. The loop is defined to be the entire software package which contains the code responsible for the genetic algorithm, antenna simulation software (XFdtd), AraSim, and fitness function. Fig. A.1 shows a diagram of the stages of the loop, as well as the responsibility of that stage. Note that all stages (A-F) are run by the main Bash script, which calls separate sub-Bash scripts for each of the stages to be executed.

A.3.1 Bash Script

Our loop is composed of a multitude of different programs and languages (XFdtd xmacros, .cpp, and python) that are all intertwined; in order to run them without user involvement, we use a Bash script. In other words, a Bash script controls when every software and code gets implemented, controls the movement of data files in the system, and provides the input parameters for every code automatically. Our code is set up with one main Bash script that runs all of our software; the main Bash script contains sub-Bash scripts that act as functions to call all of the stages (A-F) of the loop. Each sub-Bash script is responsible for running a different software. The software is setup in this way to allow users to more easily find and edit parts of the loop. A depiction of the function of both the main Bash script and the sub-Bash scrips can be seen in Fig A.2. In this section, we will focus on the main Bash script, while the upcoming sections will discuss details of stages A-F and their contents.

The main bash script, called Asym_XF_Loop.sh, contains all variables and paths, as well as contains sub-Bash scripts that run all of the stages shown in Fig 3.6. The intent is that if any of this changes, we only have to modify these variables or paths in one place; thus, variable values and paths are passed to all other software called in the loop from this main Bash script. Below are some of the details of the variables and what they do. Note that most of these don't change often, except the "RunName" variable, as each different run must be named something unique so old data does not get overwritten.

Notable variables from the main Bash script (Asym_XF_Loop.sh):

RunName This is the name of the run. Please include the date, your name, and details of the run in the name so we can properly document and track runs. This variable should change each time we start a new run. Note that if we are continuing a current run, this name should not be changed.

TotalGens This is the number of generations (after initial) to run through. Note that it starts at generation 0.

NPOP This is the number of individuals per generation. Usually, we utilize 50 individuals as a standard.

NNT This is the number of neutrinos generated/simulated in each AraSim job. The higher this number, the slower the software, but the lower the error. The lower this number is, the faster it runs, but the higher the error is. We have been running about 300k neutrinos consistently, which means if you do NNT=15,000 and Seeds=20, this would be a way of hitting 300k neutrinos.

Seeds This is how many AraSim jobs will run for each individual. In order to speed up our software, we have modified our AraSim jobs so that, for each individual, we evenly divide the total number of neutrinos generated into smaller jobs and run them in parallel; thus, if NNT=15,000 and Seeds=20, this means that we would submit 20 jobs of 15,000 neutrinos for each individual; thus, the jobs would be running in parallel to hit an overall NNT of 300k. As mentioned, this dramatically speeds up our run time.

FREQ This is the number of frequencies being iterated over in XFdtd (currently, this only affects the output.xmacro). Currently, our software, including AraSim,

expects 60 frequency steps from 83.33MHz to 1.066GHz, as our signal is broadband.

exp This is the exponent of the energy for the neutrinos in AraSim. Currently, we run at exp=18.

RADIUS, LENGTH, ANGLE, A, B This is an indicator that tell us whether or not we are running for the symmetric or asymmetric case; thus, are our genes repeated for both sides of the cone? If we want the run to be for a symmetric bicone, we set it to 0; if we want it to be an asymmetric bicone, it is set to 1.

SEPARATION This variable is used to tell the software whether or not we are evolving the spacing between the two cones. If 0, the separation stays constant; if 1, the separation distance is evolved and becomes another gene for each individual. Note that the standard setting is for the separation distance to stay constant (SEPARATION=0).

NSECTIONS This variable determines whether the bicone design being evolved is symmetric or asymmetric. This variable needs to correlate directly with [RADIUS, LENGTH, ANGLE, A, B]. If we make any of these asymmetric, this would need to follow. For example, if this is entirely symmetric, i.e., the genes of the length, radius, and theta are all copied directly to both sides of the cone, NSECTIONS=1. If, say, we decided to allow this to be asymmetric by allowing the length to vary with each cone, then NSECTIONS=2.

database_flag Our software is set up to utilize a database. The goal with the database is to save lots of gain patterns from previous runs. If our individual

has genes closely resembling that of the genes of a previously run individual, we would pull from a database instead of recreating the gain pattern through XFdtd; this is to improve the run time; however, it is not currently operational for anything other than the symmetric, linear case. This variable should be equal to 0 if we are not using the database and 1 if we are using the database. Since it is not yet operational, this should always be 0.

CURVED This setting tells us whether we are running the linear or nonlinear bicone. If CURVED=1, this executes a second-order nonlinear bicone run. If CURVED=0, this executed the linear (first-order) bicone run.

The final thing to mention is that the main Bash script is also responsible for running what we call our "save-state". The savestate.txt is a file that stores information apropos the last executed generation and section of the loop – i.e. it saves the generation, part (A-F) in the loop, etc, as each section and generation finishes. This way, if our software is stopped for any reason, for example the VDI ends before we hit the target number of generations, the Bash script can read this file and know where to pick up from the previous session (see more about VDIs in section A.4). After each of the sub-Bash scripts A-F is called in the main Bash script, you should see the line "SaveState_Protype.sh $gen $state $RunName $indiv", which writes to the save state telling it that the section it just executed has completed.

A.3.1.1 Sub-Bash Script Part (A): Genetic Algorithm and Its Data

The first sub-Bash script (our part (A)) called by the main Bash script is called "Part_A_With_Switches.sh" if it is linear and "Part_A_Curved.sh" if it is nonlinear. As seen in Fig A.2, the sub-Bash script for part (A) does two things: (1) Runs the

GA and saves the DNA to a .csv file, and (2) moves the GA outputs and renames the .csv file so it isn't overwritten.

First, let's talk about the GA. The genetic algorithm is the process that evolves our antennas between generations; thus, it generates genes of the individuals for the first generation, and selects parents and produces children in the subsequent generations. See Chapter 2 for more details on what GAs do. See Chapter 3.4.2 for details on what the linear, asymmetric bicone GA contains in terms of selection methods, operators, and constraints, and see Chapter 3.4.3 for details on what the nonlinear, asymmetric bicone GA contains in terms of selection methods, operators and constraints.

Part (A) can run two potential GAs for the linear case depending on what we set the variable NSECTIONS to in the main Bash script. So, if NSECTIONS=1, this compiles improved_GA.cpp and runs its executable. If NSECTIONS>1, then this compiles Latest_Asym_GA.cpp and runs its executable. Note that these GAs have to be different because the symmetric and asymmetric runs have a different number of genes for each individual (as mentioned in Chapter 3).

Similarly, we have two potential GAs for the nonlinear case ('Part_A_Curved.sh"). In the case where it is linear and NSECTIONS=1, it compiles "improved_GA.cpp". In the case where it is nonlinear and NSECTIONS>1, it compiles "Elite_GA.cpp".

Second, this Bash script moves the GA outputs and renames the .csv file. To do so, the code saves the genes to generationDNA.csv, copies that file, and renames it to indicate what generation those individuals (and their DNA) are from (saves it as gen_generationDNA.csv). After this, the main Bash script moves on to parts (B1) and (B2).

A.3.1.2 Sub-Bash Script Part (B1) and (B2): XFdtd Simulation Software

The second sub-Bash script called by the main Bash script for part (b) utilizes XFdtd. XFdtd (XF) is a computational electromagnetism simulation software developed by REMCOM using the finite difference time domain method for calculations. The antenna and its properties are simulated in XF by hitting an artificial burst of radiation on the antenna to calculate its gain patterns. For more information on what XFdtd does, please see section 3.3.

In this section, we will discuss the two sub-Bash scripts that are responsible for generating the antenna response pattern with XFdtd. Part (B1) and (B2) are responsible for three things: (1) Prepares output.xmacro with the parameters (antenna type, population number, and grid size), (2) Prepares simulation_PEC.xmacro with information on each individual's antenna parameters, and (3) runs XFdtd and loads it with both .xmacro files. There are five pieces of code associated with XFdtd: simulation_PEC.xmacro and its two skeletons, and ouput.xmacro and its skeleton. The skeletons are the text that is constant for each script.

Part (B1) is set up to call either "Part_B_GPU_job1.sh" or "Part_B_job1_sep.sh". Which one of these we run depends on what we set "NSECTIONS" to. Since NSECTIONS>1 means we are running an evolution with the asymmetric bicone, we need a different script to set up the CAD models of these individuals than for the NSECTIONS=1 (symmetric) case. Additionally, It could also call "Part_B_Curved_1.sh" if it is nonlinear. Part (B1) is responsible for running and setting up simulation_PEC.xmacro and its two skeletons, which builds the antenna, creates the waveform, and queues and runs each simulation.

Part (B2) is set up to call "Part_B_GPU_job2_asym_array.sh". This part runs the ouput.xmacro and its skeleton, which write the XFdtd simulation data to an output .uan file.

For more information on XFdtd, reference the XFdtd manual. Additionally, Xfdtd has many new resources that are accessible through their GUI under the "Help" drop-down menu. To access the XFdtd GUI you will need a VDI as seen in section A.4. Once you have requested the VDI, open the desktop and select the terminal icon (second icon) as seen in Fig. A.3. Enter "module load xfdtd" and then "xfdtd &". This should open up the XFdtd GUI as seen in Fig A.4. Select the "help" drop-down and then "Scripting API documentation" as seen in A.5. This provides more information on the scripting language.

A.3.1.3 Sub-Bash Script Part (C): XFdtd Output Conversion Code

Part (C), which calls "Part_C.sh" is responsible for converting our XFdtd output format. In order to run AraSim we need to make the files that XFdtd outputs readable by AraSim. This means converting the .uan files from XF into .dat files that AraSim can read. This is done in the XFintoARA.py file. Once this is done we move them into the AraSim directory.

A.3.1.4 Sub-Bash Script Part (D1) and (D2): AraSim Execution

Sub-Bash scripts for parts (D1) and (D2) are responsible for executing AraSim. AraSim is a neutrino simulation software that simulates the environment the ARA antennas experience in Antarctica when taking data. AraSim generates neutrino events independent of each other, with interaction point locations chosen with a uniform density in the ice. For computational ease, neutrinos are generated within a 3-5 km radius

around the center of a single station for neutrino energies from $E_\nu = 10^{17}eV - 10^{21}eV$, with the larger radii used for higher energies. AraSim then performs ray tracing and attenuation on the signal and calculates what electromagnetic radiation reaches the station. At the station, the gain and phase data from XF are used to calculate the sensitivity of the antenna to neutrinos. This value is extracted as an effective sensitive ice volume (called the 'effective volume'). This effective volume is our fitness measurement, called the fitness score, for all individuals. For more information on what AraSim does, please see Chapter 3.3.2.2.

Before going into the detail of these scripts, it's important to note that to run AraSim, we need to submit jobs in order to fully utilize the computational power of the supercomputing cluster. To submit a job, a short script is run that contains a number of parameters for the cluster. Jobs allow the user to specify the number of cores to use, the number of GPUs to use, and a time limit. You are also more easily able to run many jobs in parallel, which means we can run multiple simulations at once instead of waiting for each individual simulation to run before starting the next. Note that running a program through the command line instead of submitting a job will work, but it will be significantly slower and may time out before completion. Jobs allow the cluster to allocate cores and time to users appropriately, so you may have to wait for a job to start, especially if you are requesting significant computational power. Please see more about job submissions on the OSC site.. Though this may seem intimidating, our software is set up to automatically submit AraSim jobs; the user bares no responsibility in submitting the jobs manually. This, as well as the remainder of discussions apropos AraSim, is noteworthy information that can assist

in properly understanding our software for future modifications; however, it is not essential to starting a run if no modifications are needed.

One more note-worthy mention is that because these jobs take a long time, we have modified our run so that, for each individual, we break up the number of neutrinos into smaller jobs and utilize the ability to run simulations in parallel. For example, if our variable NNT=30,000, making Seeds=10 means that we divide a total of 300k neutrinos into 10 jobs that will run in parallel; this dramatically speeds up our run time.

In the rest of this section, we will discuss the two sub-Bash scripts that are responsible for running the simulation of our antennas in Antarctica using AraSim. Part (D1) is responsible for running two things: (1) it moves each .dat file into a folder AraSim can access, while changing it to a .txt, which is what AraSim reads, and (2) it runs AraSim for each individual and moves the output into the Antenna_Performance_Metric folder. It also makes a directory for all errors and output files from AraSim to be dumped. Part (D2) is responsible for telling our loop to wait until AraSim is finished running. This will check the completed files and make sure they all appear before moving on to the next part. This checks that both (1) the jobs finished and (2) that they were successful. If it is unsuccessful, it will resubmit the job. It does so by using the "grep" command to search the files for segmentation violations. If it sees one, it resubmits that job. Note that we also submit a job for the actual ARA bicone during the first generation so it can be compared against in our run.

A.3.1.5 Sub-Bash Script Part (E): Fitness Score Generation

Part (E) calls "Part_E_Asym.sh" if it is linear and "/Loop_Parts/Part_E/Part_E_Curved.sh" if it is nonlinear. Now that AraSim has successfully run, we want to take the data recorded and extract the fitness scores. The fitness scores determine how well each specific antenna performed; we can use this information to compare it with other antennas with different parameters. This way, we can determine which antenna performed the best. The AraSim data for each generation is concatenated into one text file, where each AraSim antenna output is separated by a space. This data is fed into fitnessScores.exe, which will generate an individual fitness score for each antenna based on the effective volume of ice observed. Finally, gensData.py will extract useful information from the fitness scores and write to maxFitnessScores.csv and gensData.csv, which give results about which performed the best.

For NSECTIONS=1 it compiles fitnessFunction_ARA.cpp. For NSECTIONS>1 it compiles fitnessFunction_ARA_Asym.cpp. Also, if we decide to evolve the separation, we can do so, but only for NSECTIONS=1 and it would compile fitnessFunction_ARA_Sep.cpp. The software knows which to compile and run based on the settings indicated in Asym_XF_Loop.sh. Each of these determines the maximum effective volume output by AraSim and uses this to figure out which antennas should be parents for the next generation.

One more thing to note is that this part is also set up to run an radius, length, theta plotting software, but is not currently functional. This plotting software would be better served for part (F) and should be transitioned there in the future.

A.3.1.6 Sub-Bash Script Part (F): Plotting

Part (F) is primarily responsible for running all of our plotting software. It is called "Part_F_asym.sh" for the linear case. For the nonlinear case, it is called "Loop_Parts/Part_F/Part_F_Curved.sh". More specifically, this part plots the fitness scores of all individuals, in 3D and 2D, of current and previous generations' scores. It then sends them to an email address that is set to automatically upload plots to our Dropbox. This allows easy access to plots as each generation finishes without the need for OSC access.

A.3.2 Running the PAEA Loop

Now that all of the details of our software have been presented, we can take a look at how to run the loop; however, before we do we will discuss a few important notes.

Note 1: The process differs for starting a new run versus continuing an old one.

First, let me explain why you would need to pick up an old run. Each time we start a run, we need to request something from OSC called a Virtual Desktop Interface (VDI); these allow you to run without needing to forward actions through to your computer's terminal. XFdtd has a GUI that cannot be suppressed and, when we run without a VDI (on the command line, allowing forwarding through to the terminal), XF runs usually time out or fail; thus, each time we run the loop we have to request a VDI, which is requested for a set block of time. Once that set block of time ends, the loop stops. It takes days to weeks to run our software 20+ generations and the more time we request for a VDI, the longer it takes to get one. So if I ask for, say, 5 hours of VDI time, I won't get very far in an evolution, but I will only have to wait for a

handful of minutes for a VDI request to be granted. On the contrary, if I requested 80 hours for a VDI this could take a longer time for a VDI to open up. This means that we usually request blocks of time from 5-10 hours when we run and when the VDI ends, the run stops.

Because our runs will stop dozens of times before completing, we have to restart the run often. This is important because the way we start a run and continue a run is slightly different. This will be explained in more detail later in this section.

Note 2: Watch our instructional video for running the PAEA loop

Though this section will be going through all of the contents in detail, you should also utilize our instructional video, as it comes with visual aids for guiding you. Before you watch it, note that some things are now out of date. First, we are officially using slurm now, which is mentioned in the video when they request an interactive job (instead of a VDI). Please pay close attention to not utilize the batch job command and instead request a VDI (which is explained in Appendix A.3.2.2).

Second, we no longer apply a penalty for an antenna exceeding the borehole size. In the video, it mentioned that we have two fitness scores: (1) the effective volume from AraSim, and (2) the effective volume for each individual with penalties for antennas that are too large. We no longer do this; instead we restrict the antennas so that they are not produced at sizes larger than the borehole. This means the regular effective volume coming out of AraSim is the final fitness score of each individual.

Note 3: We are not using the database.

Third, make a special note that the database is not currently set up for any of the runs we are currently doing (asymmetric, non-linear, and AREA). If you were to accidentally activate this in the main Bash script variables, it would not be effective; that database does not exist.

In the rest of the section, we are going to show you how to run the loop. This is specific to OSC Connolly group users; however, I will add as much detail as possible so those hoping to follow along can understand our reasons for running the code the way we do.

A.3.2.1 Step 1: Setting your Variables

As I mentioned, the process in starting a new run versus continuing a previous run differs. Below gives information apropos both scenarios. Please read and understand both methods, as they will both be pertinent to executing a run.

Starting a brand new run: Before starting the loop for the first time, you need to open up the main Bash script (Asym_XF_Loop.sh) and check your variables. If you are starting a new run, you need to change the run name to Name_YYYY_MM_DD_RunDetails. For example:

```
Julie_2021_09_19_NonlinearTest
```

Note that usually we run a total of 300k neutrinos divided up into 10-20 seeds and set runs for 50+ generations. Remember that you can always end a run early, but it's difficult to restart if you completed all the generations assigned in this variable and want to keep going. For the time being we run at exp=18 and NPOP of 50.

Continuing a previous run: In this case the run has already been initialized, but the VDI has ended. You should check where the save-state recorded the last completed state to be by looking at savestate.txt. This can be found in:

```
/users/PAS0654/osu9348/BiconeEvolutionOSC/BiconeEvolution/
    ↪ current\_antenna\_evo\_build/XF\_Loop/Evolutionary\_Loop/
    ↪ saveStates
```

There should be a file called Name_YYYY_MM_DD_RunDetails.savestate.txt, named after your variable RunName. The three numbers in this file are the generation, state, and individual (in this order). If this looks correct and you see data correctly written from the state before this, you can leave everything alone. If this looks incorrect and you need to move back to an earlier state, you can do so by changing the "state". Note that state=0 refers to part (A), state=1 refers for part (B1), etc.

If you need to jump back a generation, this is a little bit more complicated. In this case, you will need to change both the state and replace the latest generationDNA.csv and fitnessScores.csv files in the Generation_Data/ directory with the ones from the generation you are stepping back to. For example, if you are on generation 10 and want to step back to generation 9, you will need to edit the save-state file to revert the generation to 9, as well as replacing generationDNA.csv and fitnessScores.csv in Generation_Data with the text in 9_generationDNA.csv and 9_fitnessScores.csv.

In practice, there are cases where this is excessive (though not harmful). For example, if you have not progressed beyond the AraSim part (Part D) of generation 10, then the new fitnessScores.csv file hasn't been created yet. Nevertheless, it is good practice to follow these same steps any time you need to step back.

Stepping back is most complicated if you have finished AraSim in the current generation and want to step back to the previous generation to a point before AraSim.

Note that the XF portion (Part B) automatically checks if the files it is supposed to create have already been created and removes them if so, meaning that there is no adjustment needed to be made to the files in the XF directory. Once you have done this, you can proceed to the next step.

Another important note: never change variables mid-run. If you want to change any variables, you will have to start a new run.

A.3.2.2 Step 2: Requesting a VDI

To request a VDI, visit the OSC OnDemand VDI page. Once you log in, you should see a screen similar to Fig. A.6. You can then request the number of hours needed for the VDI and hit "launch". You should then see a virtual desktop open up. In order to run these scripts, we now have to access the terminal. On the bottom of the screen, there is a terminal icon as seen in Fig A.7. Click on that icon. This should pull up the terminal. You are now ready to start the software!

As a quick note, it's common practice in our group to share your VDI with the group via our VDI_links Slack channel. Once the link is shared, anyone can click on it and view the terminal and, based on the outputs hitting terminal, determine how the run is doing. To generate a shareable VDI link, you can click the "View Only(Share-able Link)" button on the VDI launch screen as seen in Fig A.8.

A.3.2.3 Step 3: Starting the Script

Starting the script is fairly straight forward. Once you have the VDI terminal open, navigate to the proper directory using GE60. You should now see the Asym_XF_Loop.sh script and can run it to begin the loop by using the command "./Loop_Scripts/Asym_XF_Loop.sh".

Note when using the terminal through the VDI to start a run, you no longer need to use the "ssh" login command (from section A.2.2); it is already logged into OSC. The "ssh" command used for remote login is for logging in to OSC directly from your terminal on your home computer and is needed when making software edits. VDIs are necessary for operations that require more computing power, which is not necessary for modifying code (only for running it).

Once the loop starts, it spits out the save-state to the terminal. Check it and make sure it is correct and then press any key. You only need to do this when you are starting a brand new run or restarting a run that is a handful of generations in but ended due to the VDI wall-time ending, not every generation. This is just cross-checking the save-state.

Ignore all of the pop-ups until it gives you the pop-up in Fig. A.9. You need to select "no." You only need to do this for the first generation when you start a new run or restart an existing run.

Note that sometimes XF will show an error saying XF is not responding, as seen in Fig. A.10. You should not click "wait" or "force quit." This is a nonsense error and should be ignored.

You are going to see lots of things outputting to your terminal, some with errors. This is normal. Some of these errors are nonsense and should be cleaned up in

the future. While we wait for those errors to be cleaned up, we can tell if errors are concerning based on the data. You should always be checking the outputs throughout runs.

As a note, as AraSim is running you should see "waiting for AraSim to finish..." many times and it will not update super frequently. Please be patient; the loop is not stuck. AraSim takes a long time. You can always check the AraSim output and error files. The error files are in

```
/fs/ess/PAS1960/BiconeEvolutionOSC/BiconeEvolution/
    ↪ current_antenna_evo_build/XF_Loop/Evolutionary_Loop/
    ↪ Run_Outputs/THENAMEOFYOURRUN/GENNUMBER_AraSim_Errors
```

and the output files are in

```
/fs/ess/PAS1960/BiconeEvolutionOSC/BiconeEvolution/
    ↪ current_antenna_evo_build/XF_Loop/Evolutionary_Loop/
    ↪ Run_Outputs/THENAMEOFYOURRUN/GENNUMBER_AraSim_Outputs
```

Note that GENNUMBER is the generation you are currently on. For example, if this is the first generation it would be

```
/fs/ess/PAS1960/BiconeEvolutionOSC/BiconeEvolution/
    ↪ current_antenna_evo_build/XF_Loop/Evolutionary_Loop/
    ↪ Run_Outputs/THENAMEOFYOURRUN/0_AraSim_Outputs
```

"THENAMEOFYOURRUN" should be replaced with the actual name of your run that you set as the variable "RunName" in the main bash script.

After AraSim completes, the effective volume of every AraSim run for every individual will be printed to terminal, as well as other details. You don't need to do anything with this. You'll see plots pop up and disappear quickly; those plots are sent directly to our group DropBox. Once you have made it this far, congrats; you've made it through one generation! The code should now run on it's own until the VDI wall-time ends. Once it ends, you'll need to request another VDI to continue the run

using the same command used to begin it (./Loop_Scripts/Asym_XF_Loop.sh). Make sure you are checking your results often, and you are good to go!

A.4 The AREA Software

In this section, we will break down the AREA software. As mentioned in section 3.5, this loop does not utilize XFdtd, since we are evolving the gain patterns as our individuals. For details on what to expect in the software loop, see Fig. 3.33. To run this package, you will need the AREA software and AraSim only.

A.4.1 Bash Script

GA_controller_job.sh is the main Bash script. This one does not run like the PAEA loop, as it is not separated into additional Bash scripts that are run by this one. Instead, all software is run directly from this script. In this section, we will break down what the Bash script does for the AREA project.

First, it runs the GA; more specifically, it runs the executable named "GA". It is important to note that the executable is created from the GA code, which is called ga.cpp. The code ga.cpp exists in /AREA/GA/. It is incredibly important that you recompile ga.cpp any time you make edits to the GA and that you save the executable as the name "GA"; otherwise, this will not run properly. The command to recompile the GA after edits is "g++ -o (name-for-executable) source.cpp"; thus "g++ -o GA ga.cpp" would be the command to compile it.

Next, it submits the AraSim jobs via another sbatch (this one is called within the Bash script). The job submission script is called "oscRun.sh". Similar to the PAEA scripts, the Bash script will wait for AraSim jobs to finish by making sure all of the output files have been written properly (flag files).

Note, you do not need a VDI to run this loop since XFdtd isn't being used anymore. VDIs were used for loops utilizing XFdtd, because we were unable to suppress its GUI.

Finally, the bash script runs a script called stringReplacement2.py, which concatenated all of the AraSim outputs for each individual. Remember that we have multiple AraSim outputs for one individual due to this SEEDS variable used to divide up the AraSim run.

Notable variables from the Bash script (GA_controller_job.sh):

RunName The RunName works the same in this script as it did in the bicone code. The RunName should include your name, the date, and a detail apropos the run so it can be identified later.

SEEDS Seeds work the same as in the bicone script. This divides the AraSim jobs up so that we can run more jobs with fewer neutrinos in order to not drastically increase our run time by adding more neutrinos or by increasing our error by lowering the number of neutrinos thrown.

#SBATCH -t 00:00:00 This sets the amount of time requested for the sbatch job for the loop to run. Note that this is different from the AraSim jobs. More details are below. This is an OSC-specific variable.

#SBATCH -A PAS0654 You want to set this for whatever project space you are on with OSC. If you are on PAS1960, it should be changed as such. This is an OSC-specific variable.

USER=$ After the $ you want to put your OSC user name.

GAPATH This is the path where the GA exists. It is in \AREA\GA; however, if you are running on OSC you'll need the information from the full path, i.e. \users\PAS1960\rolla\AREA\GA.

ARAPATH This is the path where your AraSim is installed. Note that you will need to install AraSim if you are not running on OSC as a part of the Connolly research group; however, if you are with the Connolly research group, we already have versions installed that you can utilize on PAS1960, though you are also welcome a version on your user for running AREA.

MAINPATH This is the path where the main loop exists, i.e. \AREA\, or on OSC\users\PAS1960\username\AREA\

See section A.3.1 for further details on the variables listed. It is worth noting that the RunName directory gets made and saved in

```
users\PAS1960\username\AREA\GA\Runs\RunName\
```

where RunName is what you call the variable above and username is your personal OSC username.

Notable variables from setup.txt:

NNU You also need to check the setup.txt file for AraSim located in \AREA\setup.txt. This is the file that is read into AraSim to determine the details for the start of the run. You do not see this with the PAEA bicone code because this is set in the main Bash script, and that is passed to setup.txt prior to AraSim running; however, for AREA, our software is not as advanced at the moment and this needs to be done manually. To do so, you can open setup.txt with an editor and

look at the variable called NNU. Remember that NNU indicated the number of neutrinos thrown for each job. So if SEEDS=4 and NNU=10000, the total number of neutrinos thrown for one individual will be 40000.

Notable variables when starting the run and passing in flags: The following are variables the Bash script needs to be defined via flags set at the command line, not edited directly in the Bash script. This means that the Bash script expects values to be given for these variables; however, instead of wanting you to declare them in the script, they are passed via the command prompt along with the command executing the code. With these variables passed as flags, it should look like this: "sbatch –export=ALL,A=1,B=1,C=1,D=1,E=1,F=10 GA_controller_job.sh"

A This is the number of individuals you want run through roulette cross-over.

B This is the number of individuals you want run through roulette mutation.

C This is the number of individuals you want run through tournament cross-over.

D This is the number of individuals you want run through tournament mutation.

E This is to pass the number of generations you want to this evolution to run for. Remember that it's always better to declare more generations you'll need and end the run early than to have to restart a run that has completely concluded.

F This is responsible for telling the script what generation we are on. This is a simplified version of the save-state from the PAEA bicone software. Here you can only jump back a generation, not a single step in the generation. If you are doing a brand new run, you will have this at 0. If you are picking up a run that has ended before the full number of generations have concluded, you will

put the number for the generation you want to start on and it will restart from the first part of the loop in that generation. Reasons for needing to continue a run that has stopped is if the wall-time is reached on the sbatch job you have submitted for the loop. Don't forget that our first generation starts at 0.

We know what each of those are from the following line in the main Bash script:

```
# Input arguments for this controller script are:

# \$1 = Roulette cross-over
# \$2 = Roulette mutation
# \$3 = Tournament cross-over
# \$4 = Tournament mutation
# \$5 = number of generations to run
# \$6 = index number of the starting generation, 0 for a new run
```

So, if you wanted to run, say, 50 individuals, you would make sure that the sum of all of the numbers passed into A, B, C, D are 50. For example, you could have 25 individuals go through roulette cross-over, 8 through roulette mutation, 9 through tournament cross-over, and 8 through tournament mutation, which equates to 50 total individuals. Now let's say you decided the number of generations to be 100 (variable E), and you are starting on the first generation (variable F=0). Your line to run the software with these numbers passed in to satisfy those variables would be: sbatch –export=ALL 25, 8, 9, 8, 100, 0 GA_controller_job.sh

A.4.2 Running the AREA Loop

Once you modify the variables listed in the last section, you are ready to run. If you are familiar with the PAEA loop, there are a few differences worth noting:

(1) Because this loop does not utilize XFdtd, we don't need to run using a VDI.

(2) In order to make sure we don't time out, we don't use ./GA_controller_job.sh. Instead, we submit it as a job at the command line by using the sbatch command

(you'll see this in just a moment). We don't need to worry about OSC timing us out with PAEA because we are running using a VDI; similar to sbatch jobs, this times out when the wall-time (requested job time) is met.

Now, for starting the run, after all of the variables have been checked in both GA_controller_job.sh and setup.txt, go to the command prompt and enter the following line with the proper variables/flags for A, B, C, D, E, and F:

"sbatch –export=ALL,A=1,B=1,C=1,D=1,E=1,F=10 GA_controller_job.sh".

Remember that you should be checking over the results frequently throughout the run. You can check them by looking at the slurm output files labeled "slurm-runid#.out". These files contain everything that would be written to the terminal throughout the run but are saved to this file instead since it is run as a job submission. To determine the run ID, you can type "squeue -u OSCusername", where "OSCusername" is your actual OSC username. This will spit out details of the job, including the job ID. Additionally, you can check the AraSim files and outputs by going into the /AREA/GA/Runs/RunName/gen_#/ directory and looking at the "temp" and "child" files.

It's also important to note that if an AraSim run has an issue, it is handled differently for PAEA and AREA. Instead of the script looking for segmentation violations and resubmitting failed jobs (as done in PAEA code), the AREA code waits for flag files to be written before moving on. If the files are not written, it will not move to the next part of the loop and you will see "waiting for the first batch of jobs to complete"

written in the slurm file continuously without moving on to any other output. In this case, you will need to resubmit that generation.

Finally, it's worth noting that the AREA software is less developed than PAEA. It requires you to monitor if AraSim fails instead of it flagging the segmentation violations and resubmitting the jobs. Additionally, it requires the user to modify the setup.txt file for AraSim and does not allow you to jump back to stages in the loop using the save state (only full generations). This is something we will be developing shortly. If you are looking at running this, please keep up to date on this section, which will be added to the AREA and PAEA GitHub repositories soon.

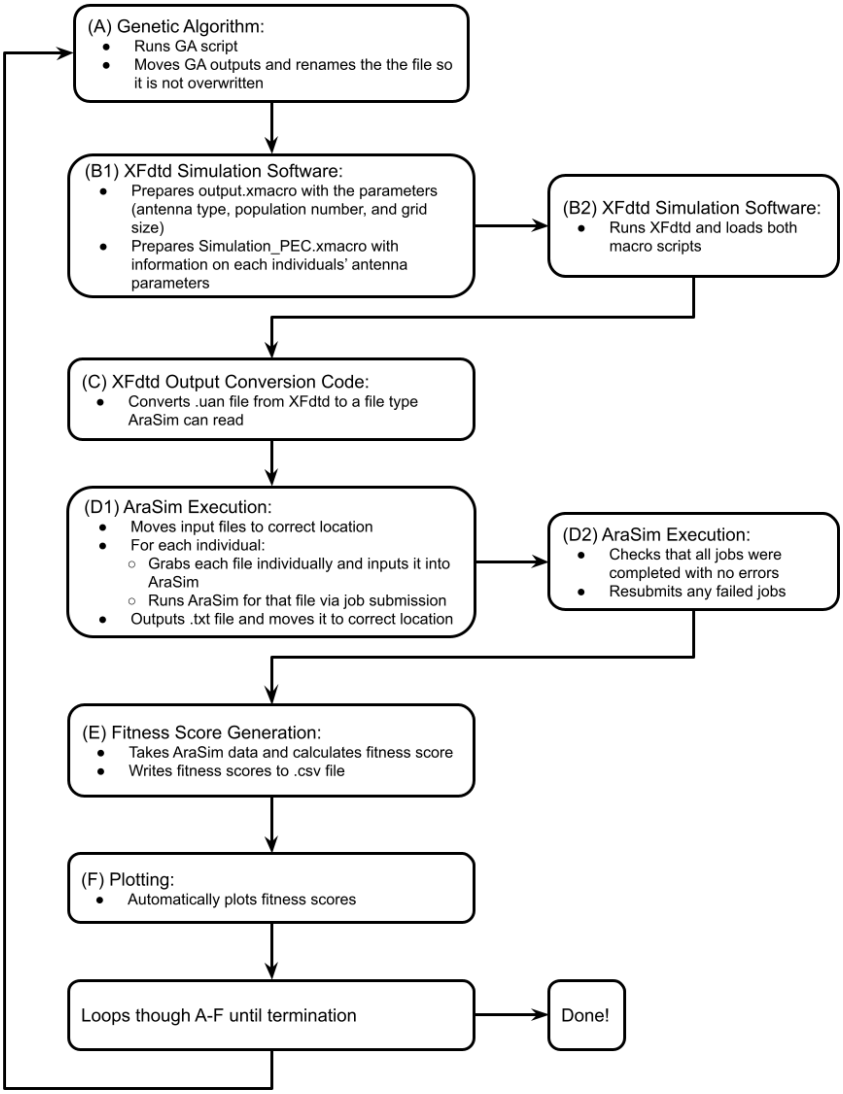


Figure A.1: Evolutionary Loop software breakdown, which is all run in this order by a Bash script.

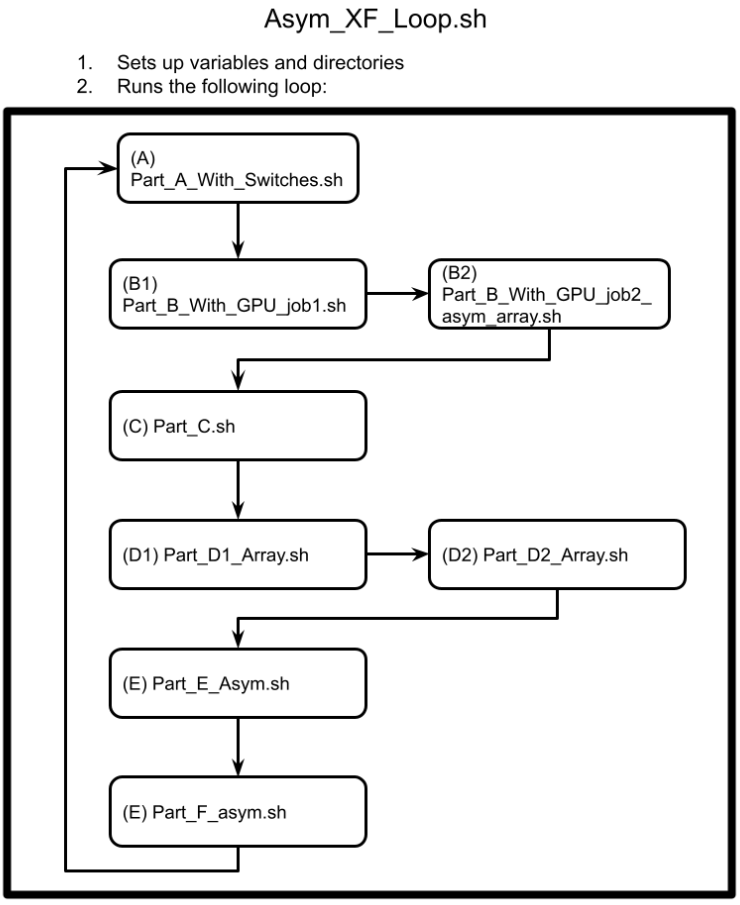


Figure A.2: Map of the function of the main Bash script.

Figure A.3: VDI Terminal

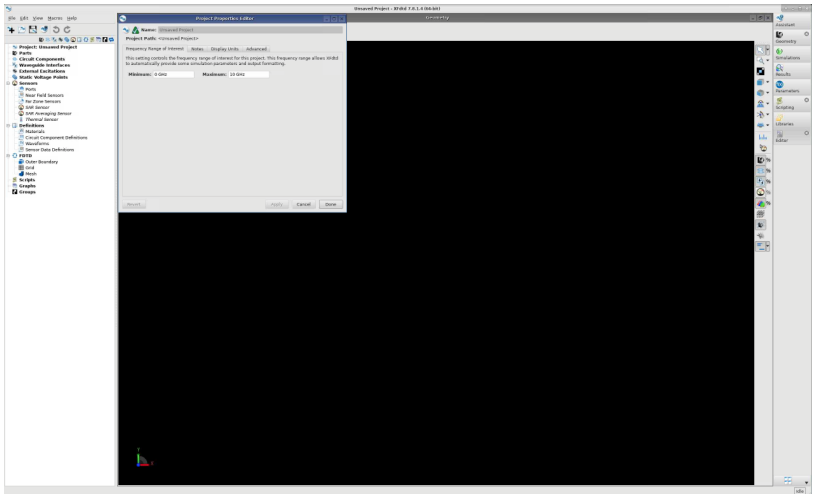


Figure A.4: XFdtd GUI

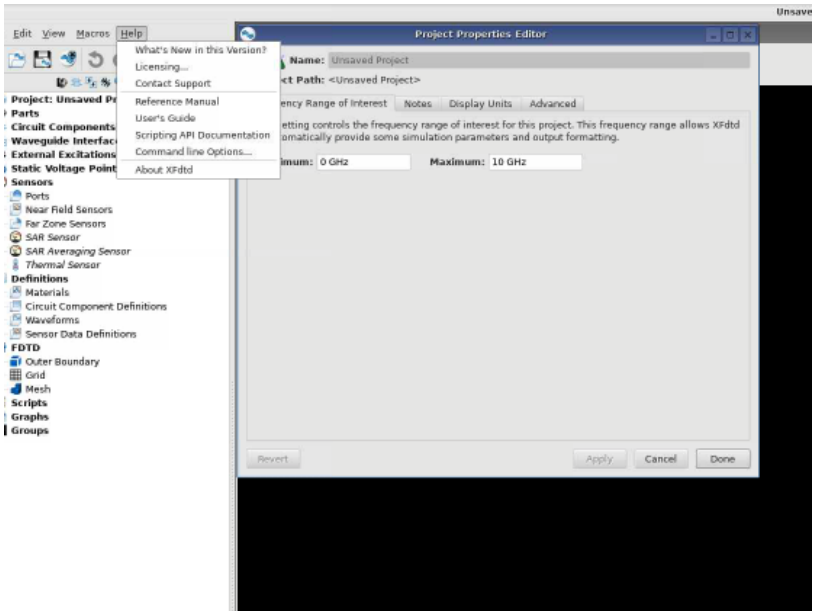


Figure A.5: XFdtd Help Menu

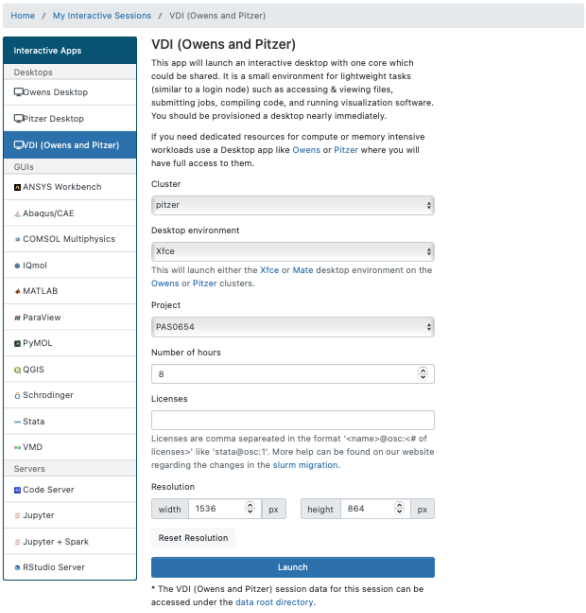


Figure A.6: VDI Request screen.

Figure A.7: Virtual desktop terminal found at the bottom center of the desktop.

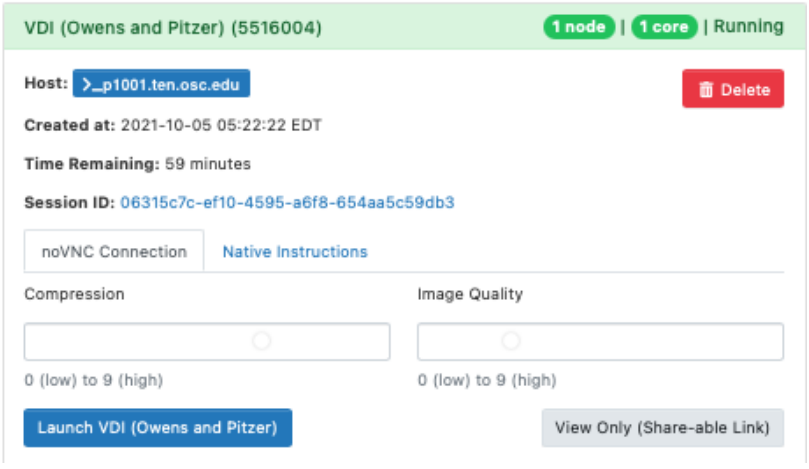


Figure A.8: VDI launch screen with the VDI share-able link.

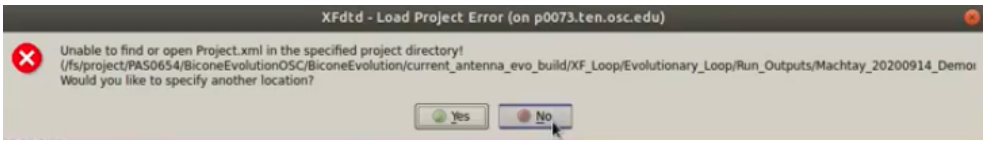


Figure A.9: This is an expected popup from XF that happens each time you start a new run or restart an existing run.

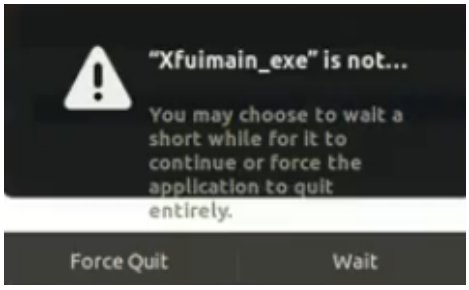


Figure A.10: Error given by XF.

Appendix B: Classification Algorithm Terminology

There are a number of definitions that are important for understanding the output of classification algorithms that can be derived from the confusion matrix. Some of these definitions are presented below. In the case of this analysis, a positive (1) is a AraSim simulated neutrino event, and a negative (0) is a background event.

Positive count (P): the actual number of positive cases.

Negative count (N): the actual number of negative cases.

True Positive (TP): The number of correctly identified positive cases.

False Negative (FN): The number of positive cases, incorrectly identified as negative. This is called a Type I Error.

True Negative (TN): The number of correctly identified negative cases.

False Positive (FP): The number of negative cases, incorrectly identified as positive. This is called a Type II Error.

Sensitivity or true positive rate (TPR): The rate of positive events correctly identified: $TPR = \frac{TP}{P}$. Also called the recall or hit rate.

Specificity or true negative rate (TNR): The rate of negative events correctly identified: $TNR = \frac{TN}{N}$. Also called the selectivity.

Precision or positive predictive value (PPV): The ratio of correct positive predictions to all positive predictions: $PPV = \frac{TP}{TP+FP}$.

Negative predictive value (NPV): The ratio of correct negative predictions to all negative predictions: $NPV = \frac{TN}{TN+FN}$.

False Negative Rate (FNR): The ratio of false negatives to total positives: $FNR = \frac{FN}{P}$.

False Positive Rate (FPR): The ratio of false negatives to total negatives: $FPR = \frac{FP}{N}$.

Accuracy: is the total number of correctly identified events over the total number of events: $A = \frac{TP+TN}{P+N}$

Another important term that aims to give a holistic evaluation of the model is the **F1 Score**. The F1 Score is the harmonic mean of the precision and recall. The F1 score ranges from 0 when there is completely incorrect precision or recall, to 1 for perfect precision and recall. teh F1 Score is given by

$$F1 = 2\,\frac{PPV \times TPR}{PPV + TPR} \tag{B.1}$$

which is equivalent to

$$F1 = \frac{2TP}{2TP + FP + FN} \tag{B.2}$$

Appendix C: How to run Karoo GP

For a complete description of downloading and running Karoo GP, see the user guide and source code available on GitHub.

In order to run the classification mode of Karoo GP, the data must be provided in a comma-separated values (CSV) file with a precise format. Each row describes one event or sample, with the first row containing the column names. The first columns contain descriptive variables. The next columns contain constants. These are input by naming the columns the numeric value of the constant and inputting zero for all rows. For this analysis, constants of 1 through 9 were used. Since the constants can interact within a tree, this allows for a wide range of constant terms. Finally, a column labeled "s" indicates the solution column. The value of this column is a 0 for group 0 or a 1 for group 1. Karoo GP does allow more than two groups, which would take subsequent integer values, although this feature was not utilized in this investigation. An example data set is presented in Tab C.1.

In Tab. C.1, the first four columns contain the descriptive variables a, b, c, and d. The next five columns contain the constants 1 through 5. Finally, the last column contains which group the individual row belongs to. In this case, rows 2, 3, and the final row belong to Group 0, and rows 1 and 4 belong to Group 1.

Table C.1: Example data structure for a Karoo GP run.

a	b	c	d	1	2	3	4	5	s
5.5	-3.0	240	0.05	0	0	0	0	0	1
6.2	2.7	360	0.09	0	0	0	0	0	0
3.1	1.5	189	0.2	0	0	0	0	0	0
2.8	4.1	125	0.4	0	0	0	0	0	1
...									
7.8	-8.0	388	-0.07	0	0	0	0	0	0

The simplest way to run Karoo GP after installation is by inputting the following line in the terminal.

```
python3 karoo_gp.py filepath/data_filename.csv
```

This begins the Karoo GP interface and loads the data. Note that running the above line without a file allows you to learn Karoo GP with the built-in examples. The first thing Karoo GP does is divide the data into a training and a testing sample. Based on industry standards, the training sample consists of 80% of the data, with the testing containing the remaining 20%.

Table C.2: Initial user-inputted parameters to run Karoo GP.

Parameter	Values	Default
Mode (or Kernel)	Classification, Regression, Matching, Weighted, Play	Classification
Tree Type	Full, Grow, Ramped	Ramped
Depth of initial population	3 - 10	3
Maximum tree depth	3 - 10	3
Minimum number of nodes	3 or larger	3
Number of individuals	10 -1000	100
Number of generations	1 - 100	10
Number of generations	1 - 100	10

Next, Karoo GP asks the user to define a number of GP parameters presented in the Tab C.2. The options describing trees and populations give the user valuable control over the possible solution space and allow for an exploration of possible underfitting or overfitting. If the depth of the initial population is too large, the GP will struggle to not overfit because the starting individuals will have many terms. If the depth is too small, it may take longer for the GP to find an adequate solution, if at all. A similar challenge is present with the maximum tree depth. Consequently, the program should be run with different values of this parameter, to find a solution with the smallest depth to reduce the possibility of overfitting. The minimum number of nodes allows the user to restrict how simple the trees can be. In general, this should remain low to prevent overfitting and elitism in the GP. Finally, the number of trees and the number of generations are self-explanatory. More trees allow for greater genetic diversity but requires more computational power. After the set number of generations is complete, Karoo GP allows the user to add more generations, so it is useful to run in increments of 10 while exploring parameter settings and testing how many generations it takes for the GP to converge. Note that all of these parameters can be entered in the command line when starting Karoo GP; see the user manual for more details [107].

Once the run is started, Karoo GP will display various information based on the setting selected by the user. When the chosen number of generations is completed, the highest-scoring individuals are displayed with the fitness score calculated from the training data set. Note this can be a little confusing, as the presented array only shows the highest scoring tree up to that individual. For example, Individual 1 will always appear because up to that point, it is always the highest scoring individual.

The next tree presented is the next individual who has a fitness score higher or equal to individual 1, and so on. So for the output:

```
8 trees [1 3 8 17 37 67 72 85] offer the highest fitness scores.
```

Tree 85 has the absolute highest score from the population, and the earlier trees listed could only be equal or worse.

At this point, the user is able to perform a number of actions, of which I will only describe a few. Of particular note, the "l" command **l**ists the actual equations of the highest fitness score individuals that were output at the end of the generation. The next crucial command is "e", which **e**valuates the individual against the internal test data and presents the corresponding confusion matrix. Since the training data is used for the evolution process, it is important to validate the results of the run with the test data. You can also continue the run with more generations using the "add" command. In addition to these commands, you can adjust more advanced parameters such as the ratio of selection methods and genetic operators. The user may end the run with the "q" command, at which point Karoo GP saves the population information to various CSV files.

Appendix D: Smith Chart

Presented on the following page is a full Smith chart for reference.

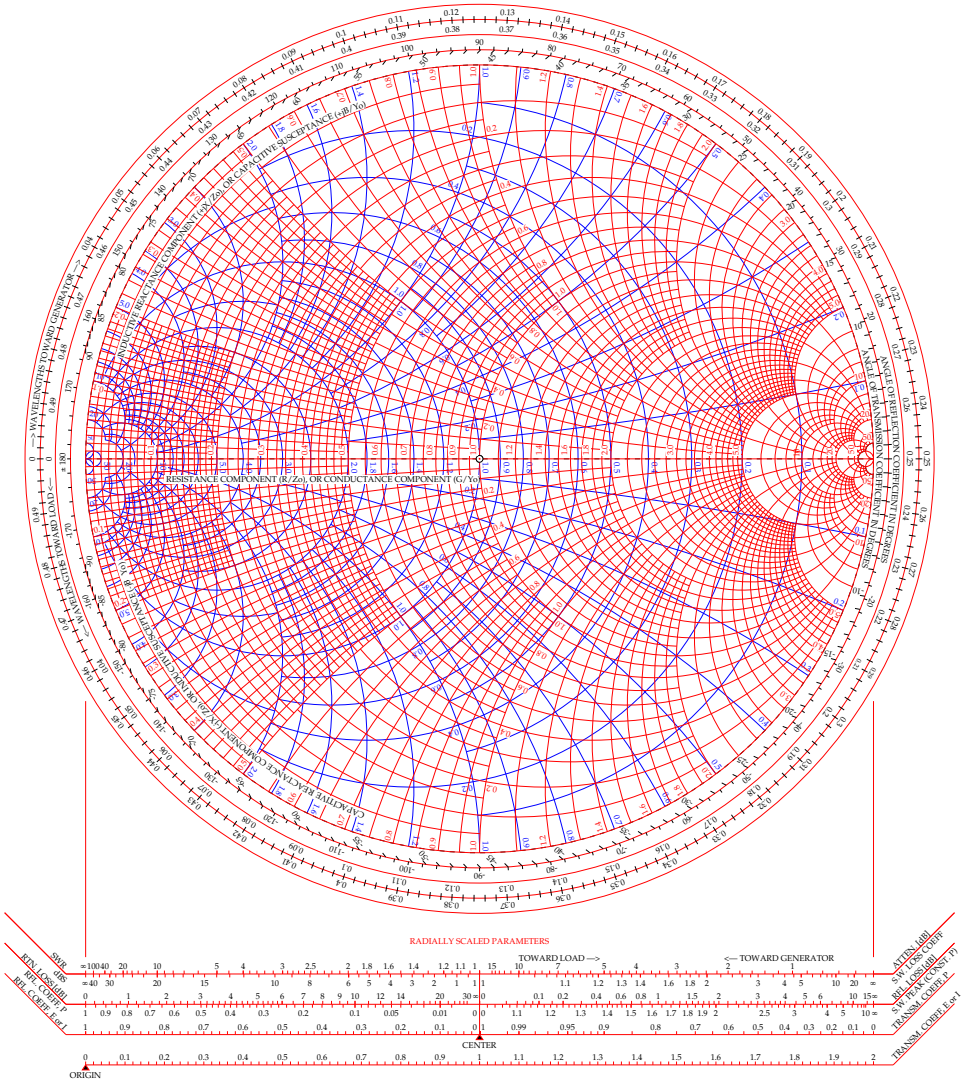


Figure D.1: Template from [89].

Bibliography

[1] Alexander Aab et al. Improved limit to the diffuse flux of ultrahigh energy neutrinos from the Pierre Auger Observatory. *Phys. Rev. D*, 91, 2015.

[2] M G Aartsen, R Abbasi, M Ackermann, J Adams, and Et al. Energy Reconstruction Methods in the IceCube Neutrino Telescope. *Journal of Instramentation*, 2016.

[3] M G Aartsen, K Abraham, M Ackermann, J Adams, J A Aguilar, and et al. Constraints on ultra-high-energy cosmic ray sources from a search for neutrinos above 10 PeV with IceCube. *Physics Review Letters*, 119, 2017.

[4] M G Aartsen, M Ackermann, J Adams, J A Aguilar, and et al. Neutrinos and Cosmic Rays Observed by IceCube. *Advances in Space Research*, 2017.

[5] M G Aartsen, M Ackermann, J Adams, J A Aguilar, and et al. The IceCube Neutrino Observatory: Instrumentation and Online Systems The IceCube Collaboration. *Journal of Instrumentation*, 2017.

[6] M. G. Aartsen et al. Evidence for Astrophysical Muon Neutrinos from the Northern Sky with IceCube. *Phys. Rev. Lett.*, 115(8):081102, 2015.

[7] M G Aartsen et al. Differential limit on the extremely-high-energy cosmic neutrino flux in the presence of astrophysical background from nine years of IceCube data. *Phys. Rev. D*, 98, 2018.

[8] Mark Aartsen, Markus Ackermann, Jenni Adams, Juan Antonio Aguilar, Markus Ahlers, Maryon Ahrens, Imen Al Samarai, David Altmann, Karen Andeen, and et al. Neutrino emission from the direction of the blazar txs 0506+056 prior to the icecube-170922a alert. *Science*, 361(6398):147–151, Jul 2018.

[9] M.G. Aartsen et al. Observation and Characterization of a Cosmic Muon Neutrino Flux from the Northern Hemisphere using six years of IceCube data. *Astrophys. J.*, 833, 2016.

[10] Q. Abarr et al. The payload for ultrahigh energy observations (pueo): A white paper. *Journal of Instrumentation*, 2021.

[11] S Abdullin. Genetic algorithm for susy trigger optimization in cms detector at lhc. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 502(2):693–695, 2003. Proceedings of the VIII International Workshop on Advanced Computing and Analysis Techniques in Physics Research.

[12] J. Aguilar et al. The Next-Generation Radio Neutrino Observatory. *Decadal Survey on Astronomy and Astrophysics*, 2020.

[13] J.A. Aguilar and et al. Design and Sensitivity of the Radio Neutrino Observatory in Greenland (RNO-G). *JISNT*, 16, 2021.

[14] Markus Ahlers et al. Multimessenger observations of a flaring blazar coincident with high-energy neutrino icecube-170922a. *Science*, 361(6398), 2018.

[15] Markus Ahlers and Francis Halzen. Minimal cosmogenic neutrinos. *Phys. Rev. D*, 86, 2012.

[16] Markus Ahlers and Francis Halzen. Opening a new window onto the universe with IceCube. *Progress in Particle and Nuclear Physics*, may 2018.

[17] P. Allison, S. Archambault, J.J. Beatty, M. Beheler-Amass, D.Z. Besson, M. Beydler, C.C. Chen, C.H. Chen, P. Chen, B.A. Clark, and et al. Constraints on the diffuse flux of ultrahigh energy neutrinos from four years of askaryan radio array data in two stations. *Physical Review D*, 102(4), Aug 2020.

[18] P Allison, J Auffenberg, R Bard, J J Beatty, and et al. First Constraints on the Ultra-High Energy Neutrino Flux from a Prototype Station of the Askaryan Radio Array. *Astroparticle Physics*, 70, 2015.

[19] P. Allison et al. Design and initial performance of the askaryan radio array prototype eev neutrino detector at the south pole. *Astroparticle Physics*, 35:457–477, 2012.

[20] P. Allison et al. Performance of two Askaryan Radio Array stations and first results in the search for ultra-high energy neutrinos. *Phys. Rev. D*, 93, 2016.

[21] P. Allison et al. Recent results from the askaryan radio array, 2019.

[22] A. Anker et al. Targeting ultra-high energy neutrinos with the ARIANNA experiment. *Advances in Space Research*, 2019.

[23] Simon Archambault et al. Understanding of the performance of ARA antennas in ice. *35th International Cosmic Ray Conference*, 2017.

[24] GA Askaryan. Excess negative charge of an electron-photon shower and its coherent radio emission. *Sov. Phys. JETP*, 14:441–443, 1962.

[25] Venkitesh Ayyar, Wahid Bhimji, Lisa Gerhardt, Sally Robertson, and Zahra Ronaghi. The use of convolutional neural networks for signal-background classification in particle physics experiments. *EPJ Web of Conferences*, 245, 2020.

[26] P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5, 2014.

[27] Volker Baum. *Search for Low Energetic Neutrino Signals from Galactic Supernovae and Collisionally Heated Gamma-Ray Bursts with the IceCube Neutrino Observatory*. PhD thesis, Johannes Gutenberg-Universität, 2017.

[28] V S Beresinsky and G T Zatsepin. Cosmic Rays at Ultra High Energies (Neutrino?). *Physics Letters B*, 28(6), 1969.

[29] E G Berezhko and H J Völk. Hadronic versus leptonic origin of the gamma-ray emission from Supernova Remnant RX J1713.7-3946. *Astronomy & Astrophysics*, 2008.

[30] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, 1996.

[31] A. Boag, Amir Boag, E. Michielssen, and R. Mittra. Design of electrically loaded wire antennas using genetic algorithms. *IEEE Transactions on Antennas and Propagation*, 44(5), 1996.

[32] G. E. P. Box and Mervin E. Muller. A Note on the Generation of Random Normal Deviates. *The Annals of Mathematical Statistics*, 29(2):610 – 611, 1958.

[33] M. Calviani, S. Di Luise, V. Galymov, and P. Velten. Optimization of neutrino fluxes for future long baseline neutrino oscillation experiments. In *Nuclear Physics B Proceedings Supplement*, 2014.

[34] M Cavaglia, S. Gaudio, T. Hansen, K. Staats, M. Szczepanczyk, and M. Zanolin. Improving the background of gravitational-wave searches for core collapse supernovae: a machine learning approach. *Mach. Learn.: Sci. Technol*, 1, 2020.

[35] Man Leong Chan, Ik Siong Heng, and Chris Messenger. Detection and classification of supernova gravitational wave signals: A deep learning approach. *Phys. Rev. D*, 102:043022, Aug 2020.

[36] Nagesh S Chauhan. Build an artificial neural network.

[37] Brian Clark. *Optimization of a Search for Ultra-High Energy Neutrinos in Four Years of Data of ARA Station 2*. PhD thesis, The Ohio State University, 2019.

[38] Shirit Cohen. *GZK Neutrino Search with the IceCube Neutrino Observatory using New Cosmic Ray Background Rejection Methods*. PhD thesis, École Poly-Technique FéDérale de Lausanne, 2013.

[39] Amy Connolly, Robert S. Thorne, and David Waters. Calculation of high energy neutrino-nucleon cross sections and uncertainties using the martin-stirling-thorne-watt parton distribution functions and implications for future experiments. *Physical Review D*, 83(11), Jun 2011.

[40] Amy L Connolly and Abigail G Vieregg. Radio Detection of High Energy Neutrinos. In *Neutrino Astronomy – Current Status, Future Prospects*, chapter 1. World Scientific, 2016.

[41] Kyle Cranmer and R. Sean Bowman. Physicsgp: A genetic programming approach to event selection. *Computer Physics Communications*, 167(3):165–176, 2005.

[42] L Davis. *Handbook of genetic algorithms*. Van Nostrand Reinhold, 2016.

[43] De Sio, Chiara. Machine learning in km3net. *EPJ Web Conf.*, 207:05004, 2019.

[44] R. Deepika, P. Manikandan, and P. Sivakumar. Optimization of pyramid horn antenna using genetic algorithm and evolution strategy. *2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering*, 2017.

[45] J. Deng, X. Chen, R. Yu, and X. Wen. A broadband cage antenna optimized by genetic algorithm. *2014 International workshop on Antenna Technology: "Small Antennas, Novel EM Structures and Materials, and Applications*, 2014.

[46] D. Eclercy, A. Reineix, and B. Jecko. Fdtd genetic algorithm for antenna optimization. *Microwave and Optical Technology Letters*, 16:72–74, 1998.

[47] El-Sayed El-Dahshan, Amr Radi, and Mahmoud El-bakry. Genetic Programming Modeling For Nucleus-Nucleus Collisions. *International Journal of Modern Physics C*, 20(11), 2009.

[48] Ralph Engel, David Seckel, and Todor Stanev. Neutrinos from propagation of ultrahigh energy protons. *Physical Review D*, 64(9), Oct 2001.

[49] Aartsen2013 et. al. "Measurement of the multi-TeV neutrino cross section with IceCube using Earth absorption". *Science*, 2017.

[50] L. Cremonesi et. al. "The Simulation of the Sensitivity of the Antarctic Impulsive Transient Antenna (ANITA) to Askaryan Radiation from Cosmogenic Neutrinos Interacting in the Antarctic Ice". *arXiv:1903.11043v2*, 2019.

[51] Ke Fang, Kumiko Kotera, M Coleman Miller, Kohta Murase, and Foteini Oikonomou. Identifying Ultrahigh-Energy Cosmic-Ray Accelerators with Future Ultrahigh-Energy Neutrino Detectors. *Journal of Cosmology and Astroparticle Physics*, 2016.

[52] Yu Fengrui, Fu Xueliang, Li Honghui, and Dong Gaifang. Improved Fitness Proportionate Selection-Based Genetic Algorithm. In Yarlagadda, P, editor, *Proceedings of the 2016 3rd International Conference on Mechatronics and Information Technology (ICMIT)*, volume 49 of *ACSR-Advances in Computer Science Research*, pages 136–140. ATLANTIS PRESS, 2016.

[53] Eric B. Flynn and Michael D. Todd. Optimal placement of piezoelectric actuators and sensors for detecting damage in plate structures. *Journal of Intelligent Material Systems and Structures*, 21(3):265–274, 2010.

[54] Thomas K. Gaisser, Todor Stanev, and Serap Tilav. Cosmic ray energy spectrum from measurements of air showers, 2013.

[55] Jacob Gordon. "A Binned Search for Ultra High Energy Neutrinos with Data from the Third Flight of the Antarctic Impulsive Transient Antenna". *Retrieved from https://etd.ohiolink.edu/*, 2018.

[56] P. Gorham et al. Constraints on the ultra-high energy cosmic neutrino flux from the fourth flight of ANITA. *Phys. Rev. D*, 99, 2019.

[57] P W Gorham, P Allison, O Banerjee, L Batten, and Et al. Constraints on the diffuse high-energy neutrino flux from the third flight of ANITA. *Physical Review D*, 98, 2018.

[58] J. Gregenstette and J. Baker. How genetic algorithms work: A critical lo ok at implicit parallelism. *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[59] Kenneth Greisen. End to the cosmic-ray spectrum? *Phys. Rev. Lett.*, 16:748–750, Apr 1966.

[60] M. Gulati, S. Siddhartha, Y. Vedi, and M. Susila. Genetic-algorithm based planar antenna design. *2018 International Conference on Wireless Communications, Signal Processing and Networking*, 2018.

[61] Francis Halzen. Astroparticle Physics with High Energy Neutrinos: from AMANDA to IceCube. *European Physics Journal*, 2006.

[62] Peter Hancock. Selection Methods for Evolutionary Algorithms. In *Practical Handbook of Genetic Algorithms*, chapter 3. CRC Press, 1995.

[63] C. Harris, L. Lewtin, and J. Trevithick. Optimization of Antennas in the Askaryan Radio Array Using Genetic Algorithms. *Senior Thesis, California Polytechnic State University, Adviser: S. Wissel*, 2018.

[64] Randy L. Haupt. Antenna design with a mixed integer genetic algorithm. *IEEE Transactions on Antennas and Propagation*, 55(3):577–582, 2007.

[65] R.L. Haupt. Adaptive crossed dipole antennas using a genetic algorithm. *IEEE Transactions on Antennas and Propagation*, 52(8):1976–1982, 2004.

[66] T. Heuge and D. Besson. Radiowave Detection of Ultra-High Energy Neutrinos and Cosmic Rays. *Progress of Theoretical and Experimental Physics*, 2017.

[67] E.S. Hong, A Connolly, and C.G. Pfendener. Simulation of the ARA Experiment for the Detection of Ultrahigh Energy Neutrinos. *International Cosmic Ray Conference*, 33, 2013.

[68] Tzung-Pei Hong, Hong-Shung Wang, Wen-Yang Lin, and Wen-Yuan Lee. Evolution of appropriate crossover and mutation operators in a genetic process. *Applied Intelligence*, 16:7–17, 01 2002.

[69] G. Hornby et al. Automated Antenna Design with Evolutionary Algorithms. *American Institute of Aeronautics and Astronautics*, 2006.

[70] E.A. Jones and W.T. Joines. Design of yagi-uda antennas using genetic algorithms. *IEEE Transactions on Antennas and Propagation*, 45(9):1386–1392, 1997.

[71] M Kadler, F Krauß, K Mannheim, R Ojha, and et al. Coincidence of a high-fluence blazar outburst with a PeV-energy neutrino event. *Nature Physics*, 12, 2016.

[72] Doddy Kastanya. Adore-ga: Genetic algorithm variant of the adore algorithm for rop detector layout optimization in candu reactors. *Annals of Nuclear Energy*, 46:160–168, 2012.

[73] U F Katz and Ch Spiering. High-Energy Neutrino Astrophysics: Status and Perspectives. *Progress in Particle and Nuclear Physics*, 2011.

[74] N. Kleedtke, M. Hua, and S. Pozzi. Genetic algorithm optimization of tin–copper graded shielding for improved plutonium safeguards measurements. *Nuclear Inst. and Methods in Physics Research, A*, 988, 2021.

[75] K. Kotera, D. Allard, and A.V. Olinto. Cosmogenic neutrinos: parameter space and detectability from pev to zev. *JCAP*, 2010.

[76] I. Kravchenko. Rice limits on the diffuse ultra-high energy neutrino flux. *Physical Review*, 01 2006.

[77] Mike Kroll. *Propagation of cosmic rays : studying supernova remnants as sources for the galactic flux and the influence of the solar magnetic field*. PhD thesis, Ruhr-Universität Bochum, 2017.

[78] K.S. Kunz and Luebbers R.J. *The Finite Difference Time Domain Method for Electromagnetics*. CRC Press, 1993.

[79] C. Laohapensaeng and C. Free. An adaptive antenna using genetic algorithm. In *Asia Pacific Microwave Conference-Proceedings*, volume 62, pages 425–431, 2005.

[80] J.A. Lima, N. Gracias, H. Pereira, and A. Rosa. Fitness function design for genetic algorithms in cost evaluation based problems. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 207–212, 1996.

[81] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A*, 391:2193–2196, 2012.

[82] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196, 2012.

[83] A. Liu, A. Bross, and D. Neuffer. Optimization of the magnetic horn for the nustorm non-conventional neutrino beam using the genetic algorithm. *Nuclear Inst. and Methods in Physics Research, A*, 794:200–205, 2015.

[84] Bin Liu et al. Optimization of the design of Gas Cherenkov Detectors for ICF diagnosis. *Nuclear Inst. and Methods in Physics Research, A*, 897:54–58, 2018.

[85] R. Lovestead and A. Safaai-Jazi. Optimum design of helical antennas by genetic algorithm. *Microwave and Optical Technology Letters*, 62:425–431, 2020.

[86] R. Luebbers. Xfdtd and beyond-from classroom to corporation. In *2006 IEEE Antennas and Propagation Society International Symposium*, pages 119–122, 2006.

[87] Darragh McCarthy, Neil Trappe, J. Anthony Murphy, Créidhe O'Sullivan, Marcin Gradziel, Stephen Doherty, Peter G. Huggard, Arturo Polegro, and Maarten van der Vorst. The optimisation, design and verification of feed horn structures for future cosmic microwave background missions. *Infrared Physics & Technology*, 76:32–37, 2016.

[88] Ali Danandeh Mehr, Vahid Nourani, and Bahrudin Hrnjica. A binary genetic programing model for teleconnection identification between global sea surface temperature and local maximum monthly rainfall events. *Journal of Hydrology*, 55:397–406, 2017.

[89] Microwaves101. Smith chart in color.

[90] Brad L. Miller and David E. Goldberg. Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Complex Systems*, 9:193–212, 1995.

[91] Nuruzzaman. Cherenkov Radiation and Neutrino Detection, 2015.

[92] A.V. Olinto, K. Kotera, and D. Allard. Ultrahigh energy cosmic rays and neutrinos. *Nucl. Phys. Proc. Suppl.*, 217:231–236, 2011.

[93] Chris Persichilli. *Performance and Simulation of the ARIANNA Pilot Array, with Implications for Future Ultra-high Energy Neutrino Astronomy*. PhD thesis, University of California - Irvine, 2018.

[94] Tutorials Point. Genetic algorithms - crossover. *Retrieved from https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm*, 2016.

[95] Tutorials Point. Genetic algorithms - mutation. *Retrieved from https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm*, 2016.

[96] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008. (With contributions by J. R. Koza).

[97] Fernanda Psihas, Micah Groh, Christopher Tunnell, and Karl Warburton. A review on machine learning for neutrino experiments. *International Journal of Modern Physics A*, 35(33), 2020.

[98] M. M. Raghuwanshi and Omprakash Kakde. Survey on multiobjective evolutionary and real coded genetic algorithms. In *Complexity International*, volume 11, 2004.

[99] Remcom. Fdtd method. *Retrieved from: https://www.remcom.com/xf-fdtd-method*, 2021.

[100] J Rolla et al. Evolving antennas for ultra-high energy neutrino detection. In *36th International Cosmic Ray Conference*, 2019.

[101] H.M. Schellman. Lbnf beamline. In *The 39th International Conference on High Energy Physics*, volume 340, 2018.

[102] A. Shuckla et al. Comparative review of selection techniques in genetic algorithm. *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management*, 2015.

[103] Stephen L. Smith, Patrick Gaughan, David M. Halliday, Quan Ju, Nabil M. Aly, and Jeremy R. Playfer. Diagnosis of parkinson's disease using evolutionary algorithms. *Genetic Programming and Evolvable Machines*, 8:433–447, 2007.

[104] Devin Soni. Supervised vs. unsupervised learning, Jul 2020.

[105] Christian Spiering. "Towards High-Energy Neutrino Astronomy". *arXiv:1207.4952*, 2012.

[106] Christian Spiering. Towards High-Energy Neutrino Astronomy A Historical Review. *The European Physical Journal H*, 2012.

[107] Kai Staats. Karoo gp user guide.

[108] Kai Staats. Genetic Programming Applied to RFI Mitigation in Radio Astronomy. Master's thesis, University of Cape Town, 2016.

[109] Kai Staats, Edward Pantridge, Marco Cavaglia, Iurii Milovanov, and Arun Aniyan. Tensorflow enabled genetic programming, 2017.

[110] Jorge Torres. *Neutrino Astrophysics with the Askaryan Radio Array.* PhD thesis, The Ohio State University, 2021.

[111] HanRui Wang, KeSen Li, and KunHong Liu. A genetic programming based ecoc algorithm for microarray data classification. In Derong Liu, Shengli Xie, Yuanqing Li, Dongbin Zhao, and El-Sayed M. El-Alfy, editors, *Neural Information Processing*, pages 683–691, Cham, 2017. Springer International Publishing.

[112] Shimon Whiteson and Daniel Whiteson. Machine learning for event selection in high energy physics. *Engineering Applications of Artificial Intelligence*, 22:1203–1217, 2009.

[113] L. Xie, Y. Jiao, G. Wei, G. Zhao, and F. Zhang. A compact uwb slot antenna optimized by genetic algorithm. *Microwave and Optical Technology Letters*, 53:2135–2139, 2011.

[114] GT Zatsepin and VA Kuzmin. Upper limit of spectrum of cosmic rays. *JETP LETTERS-USSR*, 4:78, 1966.

[115] R. Zebulum et al. *Evolutionary Electronics*. CRC Press, 2018.

[116] Q. Zhang, K. Barri, P. Jiao, et al. Genetic programming in civil engineering: advent, applications and future trends. *Artif Intell Rev*, 54:1863–1885, 2021.

[117] Jinghui Zhong, Xiaomin Hu, Min Gu, and Jun Zhang. Comparison of performance between different selection strategies on simple genetic algorithms. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, pages 1115–1121, 2005.