Peer Reviewed

Title:
Intra-hour Direct Normal Irradiance solar forecasting using genetic programming

Author:
Queener, Benjamin Daniel

Acceptance Date:
2012

Series:
UC San Diego Electronic Theses and Dissertations

Degree:
M.S., Electrical engineering (Applied ocean sciences)UC San Diego

Permalink:
http://escholarship.org/uc/item/3g06n4dp

Local Identifier(s):

Abstract:
The development and utilization of solar energy has resulted in increased interest in solar irradiance forecasting. Ground level insolation has a natural variability due to atmospheric processes that are directly tied to the local meteorological conditions. Independent System Operators (ISOs) find that forecasting errors for small timescales are highly dependent on the characteristics and dynamics of the local cloud cover. This work seeks to explore the use of Genetic Programming to develop forecasting programs that surpass the performance of persistence forecasting. Specifically, our interest lies in forecasting a 30-second average Direct Normal Irradiance with a time horizon of five minutes. The GP-produced forecasting programs will be compared to the performance of persistence forecasting in the terms of Root Means-Squared Errors (RMSE). These proof-of-concept experiments have demonstrated that GP is a promising approach, producing forecasting programs with a 10% performance improvement over persistence forecasts

UNIVERSITY OF CALIFORNIA, SAN DIEGO

Intra-hour Direct Normal Irradiance Solar Forecasting Using Genetic Programming

A Thesis submitted in partial satisfaction of the

requirements for the degree

Master of Science

in

Electrical Engineering
(Applied Ocean Sciences)

by

Benjamin Daniel Queener

Committee in charge:

Professor Carlos F.M. Coimbra, Chair
Professor Gert Lanckriet, Co-Chair
Professor William Hodgkiss

2012

The Thesis of Benjamin Daniel Queener is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2012

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank all the brilliant professors and students at the Scripps Institute of Oceanography for all the help they have given me pushing through my program in such a short time. I will always be grateful to everyone at SIO. I would not have made it had I not been surrounded by so many good people.

I would especially like to thank everyone from the Coimbra Research group for supporting me through my thesis process. Your willingness to answer questions, have general discussions or just listen to my stories helped me keep my footing and continue moving forward one step at a time.

ABSTRACT OF THE THESIS

Intra-hour Direct Normal Irradiance Solar Forecasting Using Genetic Programming

by

Benjamin Daniel Queener

Master of Science in Electrical Engineering

(Applied Ocean Sciences)

University of California, San Diego, 2012

Professor Carlos F. M. Coimbra, Chair

Professor Gert Lanckriet, Co-Chair

The development and utilization of solar energy has resulted in increased interest in solar irradiance forecasting. Ground level insolation has a natural variability due to atmospheric processes that are directly tied to the local meteorological conditions. Independent System Operators (ISOs) find that forecasting errors for small timescales are highly dependent on the characteristics and dynamics of the local cloud cover. This work seeks to explore the use of Genetic Programming to develop forecasting programs that surpass the performance of persistence forecasting.

Specifically, our interest lies in forecasting a 30-second average Direct Normal Irradiance with a time horizon of five minutes. The GP-produced forecasting programs will be compared to the performance of persistence forecasting in the terms of Root Means-Squared Errors (RMSE). These proof-of-concept experiments have demonstrated that GP is a promising approach, producing forecasting programs with a 10% performance improvement over persistence forecasts.

Chapter 1

General introduction

We are experiencing unprecedented growth in the use of renewable resources. Improving technology combined with the benefits of manufacturing economies of scale is accelerating the adoption of renewable resources. Solar energy has seen considerable gains but its acceptance has been impeded by the inability of large-scale utilities and Independent System Operators (ISOs) to predict the availability of solar resources. Utilities and ISOs require accurate forecasts over wide temporal ranges in order to confidently manage operational planning and budgeting. The natural variability of irradiance received at the ground-level creates a significant hurdle to widespread adoption of solar power. Direct Normal Irradiance (DNI) is of particular interest in the context of power generation because DNI is the irradiance component that plays the greatest role in applications requiring solar concentration. The variability of local ground-level solar irradiance is strongly tied to the cloud cover and its meteorological dynamics. This makes it necessary to consider meteorological conditions in short time-horizon forecasts.

There are many methods being developed to forecast solar irradiance. This work will explore whether the general intelligent search method of Genetic Programming (GP) might serve as a method for producing forecasting programs that are competitive when compared to the established forecasting benchmark of

persistence forecasting. In that capacity, we will analyze the application and results of the GP forecasting to discern trends, strengths and weaknesses.

This work is a first attempt in exploring the application of GP to address solar forecasting. Research using machine learning and Artificial Intelligence (AI) methods has been applied to solar forecasting along with various evolutionary computation strategies and genetic algorithms [2, 10, 11, 22]. However, GP has not been used as a means of forecast creation or optimization. Additionally, this work seeks to forecast Direct Normal Irradiance (DNI) at a high resolution on short timescales, an area which has seen relatively little research. The work will focus on forecasting the 30-second average DNI at a 5-minute horizon, in contrast to the bulk of the literature on forecasting solar irradiance, which has concentrated on forecasting Global Horizontal Irradiance at longer timescales.

This project intends to fill a forecast niche that would be especially useful for generating and dispatching electrical power in expeditionary or remote settings. The cost of replenishment and resupply can easily become the predominant cost associated with generating power in remote settings. A Pentagon report obtained by the Wall Street Journal indicated the total cost of delivering fuel in Afghanistan to remote bases was over $400 per gallon [9]. A typical portable diesel generator requires one gallon of diesel for every 10 KW-hours of power produced with a start time on the order of minutes. Solar power could provide an attractive and secure alternative to the potentially high operational costs associated with remote internal-combustion power generation. However, the variability of ground-level insolation coupled with the cost

and weight associated with a typical battery storage system makes a stand-alone remote solar power system less attractive. A possible compromise would couple solar power generation with an ancillary backup diesel generator. With an accurate micro-scale forecast, a generator could be queued up to compensate for the loss of power due to the variations in solar irradiance or an increased load beyond the capacity of the solar power generation. Long-term solar forecasts could be used to properly size a micro-forecasting enabled solar power installation in order to reduce cost associated with deployment and operation in an expeditionary or remote situation.

The results from the GP forecast programs are promising in that they clearly demonstrate an ability to outperform the reference model of persistence forecasting. They also seem to compare well with the current research in ground- level solar irradiance prediction on similar timescales [23].

Chapter 2 covers a general background of solar forecasting, including some of the dynamics and current research. Chapter 3 covers Genetic Programming and the specifics of the GP implementation used in this work. Chapter 4 covers this work's experimental setup and application of GP to solar forecasting of DNI on short timescales. Chapter 5 analyzes the results of the project. And Chapter 6 states the project's conclusions.

Chapter 2

Solar Forecasting Background Information

Current research indicates that accurate forecasts are necessary in order for the large variable capacity of renewable resources, including solar energy, to achieve economically viability and competitiveness [12, 13, 14].

2.1 Irradiance Data

The three typical measurements of ground level solar irradiance are Direct Normal Irradiance (DNI), Global Horizontal Irradiance (GHI) and Diffuse Horizontal Irradiance (DHI). DNI is the irradiance that comes in a straight line from the sun at its current position in the sky. DHI is the irradiance that has been scattered and is received from all directions. GHI is the total amount of irradiance received from above by a surface horizontal to the ground. GHI is the sum of the DHI and the DNI scaled by the cosine of the angle of incidence of the beam with reference to the horizontal surface. A high DNI is indicative of a very clear day, whereas high DHI relative to DNI indicates an overcast day.

2.2 Different types of forecasts

Solar forecasting can be implemented for a variety of specific temporal and spatial windows. The amount of time that the forecast looks ahead is called the forecast horizon. The forecast resolution is the window size at the forecast horizon.

For instance, a 5-minute horizon with a 30-second resolution would refer to a forecast that looks five minutes ahead to predict the average value over a 30-second period. The inputs and forecasting methodology might change significantly under different spatial and temporal scales as the relative importance of different environmental dynamics changes with the forecasting horizons and resolutions. For example, long term forecasts depend on the orientation of the earth's axis and the macro-weather dynamics associated with the changing seasons. Accurate long-term forecasting methods typically require data-mining and analysis of past records. Medium-term forecasts that look weeks ahead may often use satellite images to consider large scale weather patterns. In contrast, this work studies a very small temporal window and spatial resolution. Accurate forecasts with short time horizons require local information with a high resolution. Irradiance instruments installed at the forecast site can be used to observe local patterns associated with changes in solar variability. Any drop in GHI with an associated rise in DHI would indicate an increasing cloud cover and a probable drop in DNI. To produce an accurate short term forecast, there must be some method to discern detailed information regarding the local meteorological conditions. Research by Kleissl and Lave [25] indicates that local information in the form of sky images enable accurate GHI forecasts on a timescale of 5 minutes with a spatial resolution of a few kilometers.

The Total Sky Imager (TSI) has been used to produce information about the local cloud cover in the form of minute-by-minute images. Using these sky images, researchers have been able to reduce the forecast error by 50 to 60% on a 30-second

forecast horizon as compared to persistence forecasting [4]. The TSI has also been used previously to provide images to a Radial Basis Function Neural Network (RBFNN) to classify pixels that were then fed into an Artificial Neural Network GHI solar forecaster [2]. Typically, some information about the identification and movement of the cloud cover has been extracted from the sky images. Many analytical techniques have been explored to discover methods to properly interpret information from sky images [5, 6, 7, 8]. The TSI has also been used to obtain cloud indices using built-in cloud classification algorithms and analyze cloud field propagation [4], demonstrating that TSI is useful for forecasting GHI at time-horizons of 15 minutes. Research using images like those from the TSI indicates that the information from the immediate local weather conditions, especially cloud cover, are important components in solar forecasting models.

2.3 Persistence Forecasting

The persistence forecast model is a simple forecast method that is, despite containing no information on the future, surprisingly difficult to outperform in the short-term. Persistence makes the assumption of no change from moment-to-moment. As the forecast horizon gets shorter, persistence forecasting typically becomes more accurate since the error introduced by variation of the diurnal solar cycle reduces. Consequently, persistence forecasting is particularly accurate on characteristically clear days, where there is low solar variability. Additionally, as the forecast resolution tightens, clear day forecasts will be more accurate as the drift introduced by averaging the irradiance measurements is reduced. Persistence forecasting provides additional

information about the forecast accuracy relative to solar variability. High Root Mean Square Error in persistence forecasts correlate with high variability of solar irradiance [27]. These qualities make persistence forecasting a relatively stable benchmark of solar forecast performance. Any improvement over the persistence model reduces random variability and indicates a genuine improvement in the forecasting ability of the model [26].

2.4 Clear-sky Models

The Clear-sky model used in this project was developed by Ineichen [16] to model location-specific irradiance based off past turbidity records. Figure 2.1 shows the Clear-sky forecast for the day used in Validation Set 1. The supposed scaling error seen in Figure 2.1 could be the product of an inaccurate Clear-sky model, or it could be an instrument-generated calibration error.

The Clear-Sky model acts as a memory for the forecasting programs by providing the forecasts with information about the probable irradiance when there is no cloud cover. This is an important piece of information for the genetic programs in this experimental setup because they use a very small time-window of inputs to produce their forecasts. Without the Clear-sky model, there is no information available to indicate the correct irradiance during an abrupt transition from a low DNI to a high DNI.

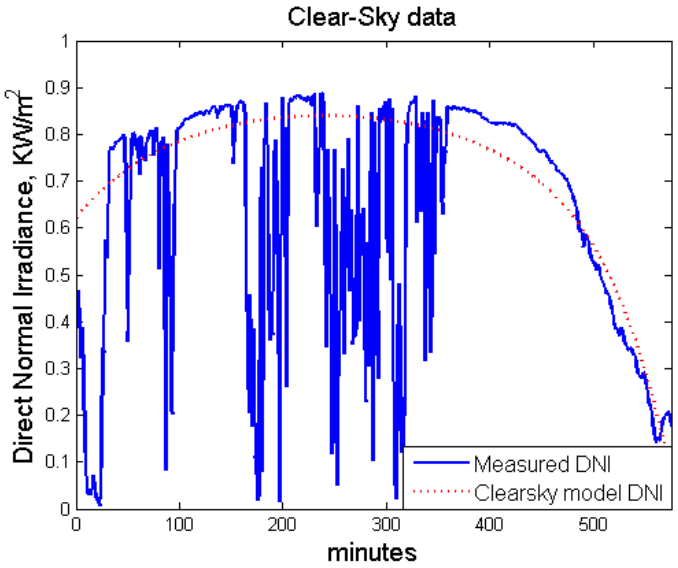

Figure 2.1 The Clear-Sky: This figure shows the Clear-Sky model plotted along with the measured DNI taken from Validation Set 1 on September 28, 2012

2.5 Evaluation of forecasting skill

There are many ways to describe the performance of different forecasting models. Unfortunately, the use of a variety of evaluation metrics makes it difficult to compare the performance of different forecasting methods.

In order to understand the evaluation of the forecast program we must clearly state how the error is quantified. The Root Mean Square Error (RMSE), as seen in Equation (2.1), is typically used because it describes a measure of the average spread of errors. While this is a good way of describing the error in the forecast, it does not account for the difficulty of forecasting under different meteorological conditions. Methods have been proposed to evaluate the accuracy of the forecast models through analysis of solar resource variability and forecast uncertainty [26]. However, most

forecast models use some variation of the RMSE to evaluate the performance of their forecasts. Many forecasts use a relative RMSE that represents the RMSE with regard to the mean observed value as seen in Equation (2.2).

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - t_i)^2} \tag{2.1}$$

$$rRMSE = \frac{RMSE}{\bar{t}} \quad where\ \bar{t}\ is\ the\ average\ DNI\ observed\ for\ the\ set \tag{2.2}$$

2.5.1 GHI versus DNI

Direct Normal Irradiance is typically harder to predict than Global Horizontal Irradiance. DNI can change abruptly as the sun is obscured by cloud cover. GHI will show a more subdued change in its values when DNI drops as DHI increases to compensate for the loss of DNI. Additionally, the relative contribution of DHI to GHI increases at times farther from the solar apex as the cosine scaled DNI reduces. Rapid changes in irradiance indicate changing meteorological conditions. Specifically, the changes are probably due to the moving cloud cover. The abrupt changes in irradiance due to cloud cover will cause an attenuation of 60 to 70% of GHI and 100% in DNI [28]. This characteristic makes DNI more dependent on the changing local meteorological conditions. Since, the dynamics of the local meteorological conditions are difficult to model, DNI is harder to predict. Research comparing solar forecasting of DNI and GHI has shown that DNI is typically about twice as difficult as predicting GHI [22].

2.5.2 State-Of-The-Art Performances

Research by Kostylev and Pavlovski has attempted to quantify the industry standards of solar power forecasting performance [23]. By observing the forecasting performance of different geographic locations under different meteorological conditions they created a basis for what would be the characteristic performance of satellite-based GHI forecast at different time intervals. Figure 5.9 shows a graph of this guideline with the performance of the GP forecasts marked. It is possible to argue that a 20% rRMSE GHI forecast with a five minute horizon would be considered to have good performance. By taking the additional information that DNI is typically twice as difficult as GHI [22], we can propose that the GHI boundarys indicating a typical forecast between 20 to 30% could be scaled to 40 to 60% rRMSE for DNI. It is also worth noting that this satellite-based forecasting used a one-minute time resolution. This gave it an additional level of stability derived from the averaged forecast value when compared to GP DNI forecasts produced in this work, which used a sharper time resolution of 30-seconds. There is inadaquate research on DNI forecasts for high resolutions and short time horizons to make clear comparisons of performance, requiring some level of inference in the analysis of results.

Chapter 3

Evolutionary Computation

An evolutionary system has three characteristics that give it the capacity to optimize a process: variation of features, selection of good features and retention of good features over time. Evolutionary computation, and it's sub-discipline of Genetic Programming use the paradigm of evolution to create solutions to problems. The key characteristic of evolutionary computation that sets it apart from other types of optimization is that the population acts as an orchestrated parallel search using interaction of individuals through evolutionary operators like selection and crossover as a means of optimization. Evolutionary computation is non-deterministic; therefore every run will produce slightly different results. It exhibits a complex adaptive behavior and can evolve novel and intricate solutions.

The process of evolutionary computation can be described as a search starting at random points in a large solution space. The process of selection chooses the points that exist at more successful points of fitness. Crossover can be thought of as a lattice that connects the structure of the individual points within the search space. Mutation can be described as a probability cloud that surrounds the individual points in the search space. In each generation, genetic operations are performed on the individual programs causing the distribution of individual points to converge towards the more optimal areas in the solution space.

3.1 Evolutionary Computation Foundation

Finding answers to most complex problems requires a broad and sustained effort to review many possible solutions. Sequential search methods can be very time consuming and inefficient. The effective use of parallelism, where multiple permutations are simultaneously considered, can be used to enable an efficient search for many potential solutions. However, a parallel search requires some method to accurately evaluate and guide the exploration through the sequences of solutions.

Often, the nature of complex problems requires the solutions to be adaptive in order to continue to perform well in a changing environment. Usually, seemingly intractable problems require successful solutions to demonstrate some level of innovation. The paradigm of evolution is essentially a process of parallelized search through numerous possibilities for innovative solutions to complex problems. Evolutionary biology uses genetic sequences as its means of encoding the possible solutions, with the hopeful result being a "fit" organism that can compete well against its peers and pass on its genes. Fortunately, the macro-level mechanisms behind evolution are easy to quantify and implement. The average fitness of a species evolves though natural selection alongside a continual variation and retention of features and fitness through the evolutionary operators of mutation and crossover.

A "search space" is a key concept in optimization problems. This is sometimes referred to in a more mathematical terms as a hyperspace, with each potential point of variation in the individual serving as a hyper-plane. In evolution, the search space is

considered to be all the possible genetic permutations of an individual species. Another conceptual visualization useful to evolutionary computation is that of the "fitness landscape." The fitness landscape was defined by Sewell Wright as the representation of all the possible fitness and genotype combinations of a population [29]. Wright described the ways in which evolution pushed populations to migrate along a fitness landscape. He interpreted the biological process of adaption of a population as a drift towards a local peak in the fitness landscape. In evolutionary computation, the operations of mutation and crossover may be interpreted as the processes which move the collection of parallel solutions around the "fitness landscape" that has been defined by the fitness function.

3.1.1 Schema and Building Blocks

The theoretical foundation of evolutionary computation was developed by John Holland in the early 1970s. Holland hypothesized that a natural or artificial adaptive system must continually identify, evaluate, and integrate structural features that are thought to provide incrementally increasing performance [30]. The concept postulates that good solutions are composed of good building blocks and the inclusion of good building blocks in individual solutions correlates with a higher fitness. Holland formalized the expression of these structural features or building blocks as schemas. The Schema Theorem implies that the process of selection causes the representation of good schemas in the population to increase or decrease with respect to their fitness in each new generation. Selection causes good schemas to undergo an increasing number of evaluations, consequently focusing the search of the population into subsets of the

search spaces where schemas with above average fitness may be found. This will effectively bias the parallel sampling and evaluation to better solutions over generations. While the process of crossover seeks to preserve and propagate better schemas, the mutation operator acts as a method of preserving the loss of diversity within the population. It is worth noting that there must be some structure in the fitness landscape and individuals in order for adaption to be possible. If the fitness landscape is sufficiently random, then adaptation becomes impossible.

Using schema analysis, Holland described how genetic algorithms implicitly estimate the average fitness of a much larger sample size than the population while only calculating the fitness for the individuals in the population [30]. This interesting phenomenon of "implicit parallelism" does not require additional computational resources and serves as a powerful method for navigating a fitness landscape.

In the context of Schema Theory and evolutionary computation, adaption may be seen as a contest between the pressure of exploration and exploitation. Exploration is the movement of the population across the fitness landscape, in search of new and useful adaptations. Exploitation is the use, incorporation, and spread of these adaptations within the population. A proper evolutionary system will have the flexibility to continue to try new and novel possibilities; otherwise the solution will over-adapt to a subset of problems and become stuck at a local optima. Holland demonstrated through schema analysis that a properly formulated adaptive system should achieve an optimal balance between exploration and exploitation.

While evolutionary computation serves as an effective method for exploring a search space for solutions, it is not a panacea for all computational problems. If the search space is small, then an exhaustive search might be a simpler and more effective way to solve a problem. Evolutionary computation requires a certain amount of planning and expertise, and that creates a hidden computational overhead. Evolutionary Computation is not guaranteed to find the global optimum and might converge at a local optimum. If the fitness landscape is smooth or convex, then gradient ascent algorithms (e.g. steepest ascent hill-climbing) will probably require less computational effort. If the problem is well-understood, then already existing analytical methods or domain-specific heuristics will out-perform evolutionary computation.

In Summary: Evolutionary computation typically out-performs other intelligent search methods in situations where the fitness landscape is not smooth or convex. The implicit parallelism enables evolutionary computation to navigate a noisy fitness solution where a single point hill-climbing method would be ill-served. Evolutionary computation is a way to find solutions where the dynamics of the problems are not well understood or where the solution only needs to be “good enough” and not globally optimal.

3.1.2 Genetic Programming

Genetic Programming is a subset of Evolutionary Computation. Genetic Programming (GP) uses individual programs as a means of exploring the solution space of a problem.

The Schema Theorem describes the schemas as binary bit-strings. Comparisons between binary strings and multiple character strings [32] demonstrate that multiple character strings outperform a binary encoding. The schema in Genetic Programming is typically the actual code for the program. Instead of bit strings, the individual is usually encoded as a series of programming operations and input terminals. Research by Koza [32] and Cramer [34] propose a tree structure representation of a program in the GP genome. Koza [35] went on to develop genetic programming and empirically prove that GP was applicable to a wide variety of problem in a variety of fields.

3.1.3 Epistasis

Epistasis is an important consideration in Genetic Programming. Epistasis is used to describe how the expression of an individual's encoding is related to the inter-dependence of the individual genes. If the epistasis is high, then small changes to an individual's encoding will cause large changes in the individual's expression of behavior and fitness. Correspondingly, low epistasis describes how small changes to the individual's encoding to result in small changes in the expression of behavior and

fitness. In problems with very high epistasis, the solution space has more variability. A high epistasis fitness landscape can be thought of as consisting of very steep and narrow hills with little correlation to the global optimum of the search space. High epistasis can make optimization of programs using GP very difficult and computationally intractable. Research by Lipsitch has shown that evolutionary computation finds a solution space with many local optima more difficult than a solution space with a few local optima [36]. Epistasis is important to this work's research as it might explain some of the mediocre results.

3.2 Push Programming Language

The GP was implemented with a modified version of PushGP coded in the Scheme programming language. PushGP uses the Push programming language as the operational language of the individual programs.

Push is a programming language developed by Lee Spector specifically for use in GP [37]. Push is an incredibly flexible and expressive language implemented with tree-based S-expressions and multiple data-stacks that enables many advanced features not found in other genetic programming implementations such as multiple data-types, automatically defined functions, and the ability to manipulate its own code. Push has the ability to use any data type without consideration of syntactic restrictions. Push's capacity to manipulate its own code and execution cycle enables it to support recursion and create evolved functional modules, such as, macros or recursive subroutines. This gives it the capacity to mimic automatically-defined functions found

in other genetic programming implementations without the need for additional genetic programming architectural overhead [19].

Push gains the ability to use multiple data-types from its stack-based architecture. By creating multiple stacks to represent each data-type, Push programs carry out any arbitrary operation without the need to check the preceding instructions to ensure that the current operation has been properly passed the correct data-types of the proper arity. If an instruction cannot find the necessary data in the stacks, it acts as a 'no-operation' instruction. Push3, the latest version of Push [38], has six types of data-stacks: Floating-point, Integer, Boolean, Execution, Code and Name. The first three types are self-explanatory. The Execution stack stores expressions, instructions, and literals that are queued up to be executed by the Push interpreter. The Code stack is similar to the Execution stack except that its instructions are static data unless called to execute by Code type instructions. Many of the Code and Execution Instructions mimic the list processing instructions found in the LISP programming language. In Push3, the Name stack is used to bind literals to a body of Code or Execution instructions. In this way, the Name data-types can be used to store literals that are treated as instructions. The Push instruction set is quite large and explaining all the operations is beyond the scope of this work. However, the table below is a sample of some of the types of operations. Additional information can be found at the Push3 website [38].

Table 3.1 A Sample of Push3 Instructions

Type of operation (data-type)	**Sample Instructions**
Stack Manipulation instructions (all types)	POP, SWAP, YANK, DUP, STACKDEPTH
Math (Integer, Floating-point)	+, -, /, *, >, <, MIN, MAX
Logic (Boolean)	AND, OR, NOT, FROMINTEGER
Code manipulation (Code, Name)	CAR, CDR, CONS, INSERT, LENGTH, LIST
Control and Manipulation (Code, Execute)	DO*, DO*COUNT, DO*RANGE, DO*TIMES, IF

The syntax for Push is simple: An instruction is the push program; a literal is a push program; a parenthesized sequence (i.e. a list) of zero or more push programs is a Push program. This format of Push enables the creation of robust programs that execute instructions without regard to syntactic constraints. This also facilitates manipulation of code by genetic operations like crossover and mutation without artificially constraining the sequence of instructions in order to match data-type and arity. One of the only internal constraints used by a Push interpreter is the evaluation limit. It is possible that a push program could manipulate its own code resulting in a loop. To compensate for this, the Push interpreter keeps track of the number of instruction evaluations that a program carries out. A Push program will continue to execute its instructions until the program ends or reaches the evaluation limit. When the program has finished running, the output is considered to be the collective data left in the stacks. It is up to the user to define what part of this information may be considered to be the "solution" to the program. Typically, for a problem demanding the return of a floating point number, the value from the top of the Floating-point stack will be considered the return value.

3.3 PushGP

PushGP is a Genetic Programming implementation that uses the Push programming language as its operational language. The robust nature of Push enables the PushGP to undertake evolutionary operations on the Push code without constraint on code generation or manipulation. The PushGP environment contains the interpreter for the Push language along with the architecture necessary to carry out evolutionary operations using the Push language.

PushGP uses the typical evolutionary operators of crossover and mutation during reproduction events. Additionally, there is a simplification operator. The simplification operator takes an individual program and performs a sequence of re-evaluations while deleting random branches of the program. If the individual program's fitness upon re-evaluation is the same or better, the deletion is maintained. Otherwise then the deleted branch is restored. The probability of simplification and the number of simplification evaluations per reproductive event are set in the GP initiating parameters. Simplification can serve as an effective way to remove introns from the code. Introns are pieces of code that do not directly contribute to the overall fitness of an individual. Introns are typically redundant schema within an individual or "junk" code that does nothing or duplicates a function found elsewhere in the code. Removing introns creates individuals with highly efficient code. Efficient coding becomes an important consideration when faced with an individual maximum-size constraint.

Historically-Assessed-Hardness (HAH) is an added capability incorporated into the latest version of PushGP. Klein and Spector found HAH was a useful operation in PushGP [39]. HAH scales the fitness of individuals to reward individuals who solve problems in the training set that have historically low solution rates. This gives a mechanism to emphasis the importance of evolving new processes. A program that solves difficult instances of a problem in a training set might not be globally optimal. HAH allows these individual to compete with more globally fit individuals, and consequently spread the novel problem-solving code within the population through crossover.

Chapter 4

Experimental Setup

4.1 Data

The GP runs were supplied with input data and solutions for training and validation sets from instruments located at the University of California, San Diego campus. The irradiance input data consisted of the three irradiance measurements and values from the Clear-sky model seen in Figure 4.1. The programs were also given information derived from sky images. The solution to the training and validation sets was simply the DNI value, 5-minutes from the forecast point.

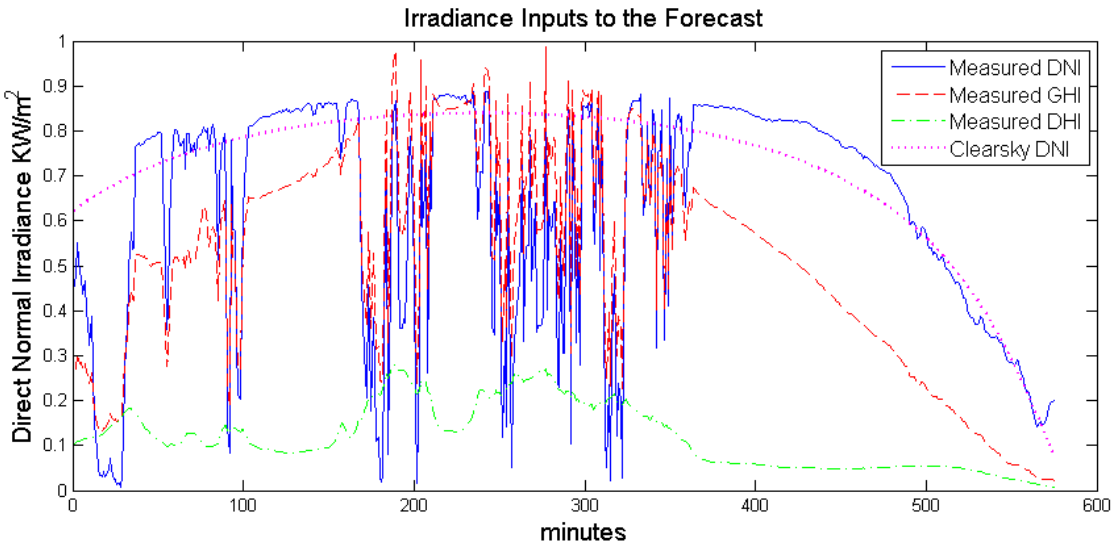

Figure 4.1: Irradiance Input Graph: This is a graph of the irradiance data and Clear-sky model for Validations Set 1. Notice that the DHI increases during times when the DNI drops, showing the probable increase of DHI during periods of DNI variability when a moving cloud cover periodically obscuring the sun.

4.1.1 Instruments

There are a variety of meteorological instruments that provide information from measurements of environmental conditions. Correlations can be found between variations in solar radiation and environmental characterization of conditions like dew point, wind speed, humidity and temperature. For this forecasting experiment, the information was restricted to irradiance measurements and sky images.

4.1.1.1 Vivotek camera

The images were taken using a Vivotek FE8171V high-resolution fish-eye security camera mounted at the University of California, San Diego campus. The camera uses a 3.1 Megapixel CMOS sensor and a 1.27 mm 180^0 Fisheye lens, allowing it to capture images with resolutions of 1536x1536. The camera was set up to produce minute-by-minute time-stamped images. The camera meets the EN 50155 Standard for embedded systems has an IP66-rated weather proof housing allowing it to be deployed in locations exposed to extreme environment conditions. The images were saved as 1536x1536 time-stamped JPEGs on an integrated on-board MicroSD/SDHC/SDXC card. Additionally the F8171V can be remotely access through a standard RJ 45 Ethernet interface. The camera was placed with the lens pointing directly upward and then oriented with the top of the image at true north. Figure 4.2 shows the emplacement of a Vivotek camera in the field.

Figure 4.2: The Vivotek FE8171V, camera used to gather sky images.

4.1.1.2 PSP

The Global Horizontal Irradiance data was collected by a Precision Spectral Pyranometer (PSP) from Eppley Labs. The Eppley Labs PSP is a World Meteorological Organization First Class Radiometer. The PSP is designed specifically for the measurement of the 30-second average solar irradiance in units of KW/m^2. It uses a multi-junction wire-wound Eppley thermopile covered by a hemisphere of clear WG295 glass, which is characteristically uniform in transparency in wavelengths from 0.285 to 2.8 micro-meters. The multi-junction thermopile converts energy into voltage using the Peltier Effect. These sensors offer good sensitivity of $9uV/Wm^{-2}$ and a flat spectral response at the typical operational temperatures of the PSP.

4.1.1.3 Shaded Disk PSP

The Diffuse Horizontal Irradiance data was collected with an Eppley Lab PSP with a shaded disk. This allows the GHI to discount the Direct Normal Irradiance component of the GHI, leaving only the Diffuse Horizontal Irradiance.

4.1.1.4 NIP

The Direct Normal Irradiance data was collected by a Normal Incidence Pyrheliometer from Eppley Labs. The Eppley Labs NIP is a World Meteorological Organization First Class Pyrheliometer. The NIP produces a measurement of the 30-second average solar irradiance in units of KW/m^2. The NIP also uses a multi-junction wire-bound Eppley thermopile. Additionally, the NIP was mounted on an Eppley SMT-3 Solar Tracker. The SMT-3 can orient in a two-axis (azimuth/elevation) plane in order to keep the NIP at a normal angle of incidence to the sun. The SMT-3 tracks the sun using built-in tables that take into account the time and location of the instrument. The tracker automatically follows the position of the sun through the day and resets itself during the night. Figure 4.3 shows a combination of the NIP and Shaded Disk PSP mounted on a SMT-3.

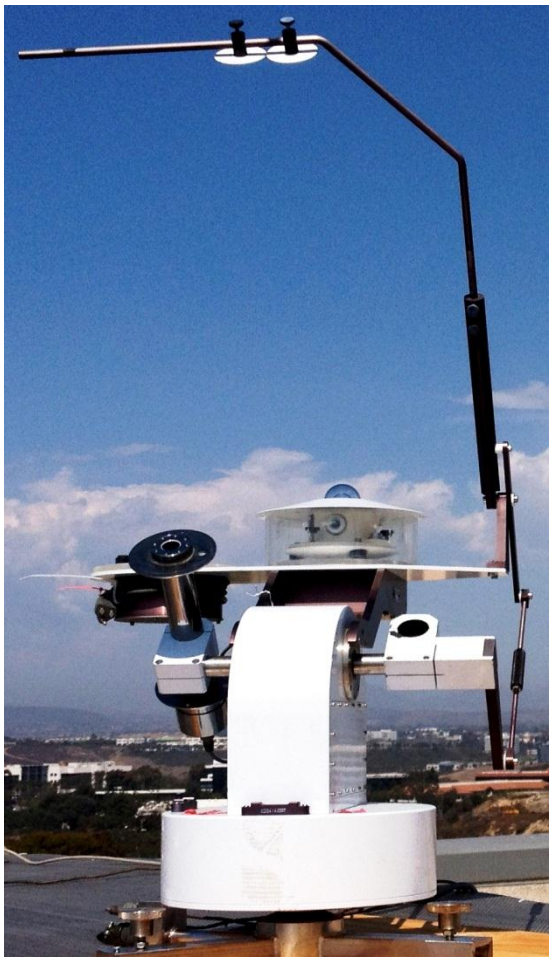

Figure 4.3: Irradiance Instruments. Combined NIP and Shaded disk PSP mounted on a SMT-3 Solar Tracker.

All of the DNI, GHI and DNI data were available by remote access. It is important to note that the time stamps of the irradiance data and the image data did not exactly match. There was some drift in both of the timestamps, including several instances when the time stamps were out of sync by 30 seconds. This is worth noting because the Genetic Programming used only 1 minute of data to make a forecast with a 5 minute horizon. A 30 second drift within a 1 minute set is a significant deviation. This can be expected to introduce a level of error into the training and validation sets.

4.1.2 Training and Validation Sets

The data for the training set was taken on June 09, 2012. Figure 4.1 shows the measured DNI of the training set. June 9th included a variety of meteorological conditions including overcast, cloudy and clear skies. The validation data was taken on September 28, 2012. Figure 4.2 shows the measured DNI for the day when the validation data set was derived. September 28 could be classified as a mixture of clear skies interspersed with periods of light clouds. Two validation sets were created out of this validation data. Validation Set 1 covered the entire day, while Validation Set 2 covered a portion of the day that could be characterized as cloudy.

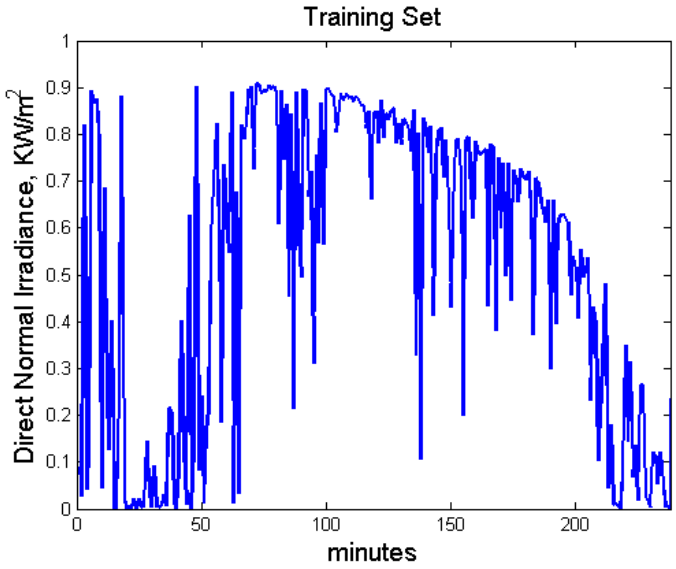

Figure 4.4:Training Data Set. The DNI of the training data used from 9 June 2012. The data from the training set covers periods of the day with light clouds, heavy clouds and overcast conditions.

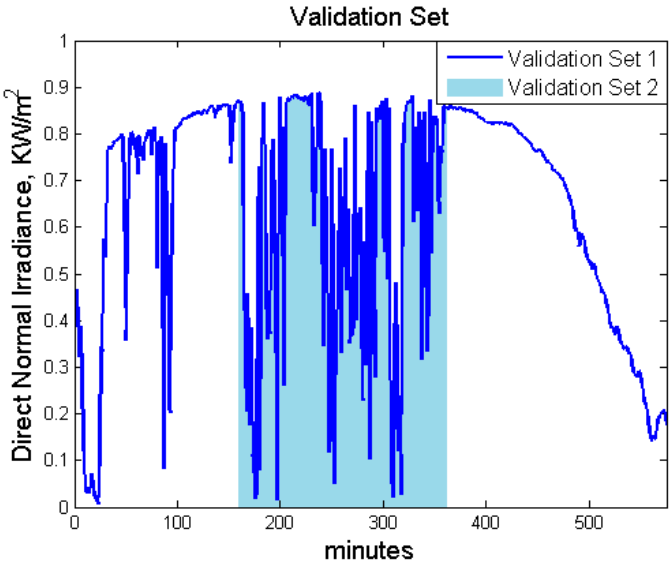

Figure 4.5: Validation Data Set. The DNI of the Validation Data from 28 Sep 2012. Both sets come from the same day. Validation Set 2 covers a portion of the day that is characteristically cloudy.

4.2 Pre-processing Images

Even though greater image resolution offers the potential for more accurate predictive capability, the amount of raw data in a single image can overwhelm a simple GP implementation. Intelligent systems designed to work with imagery data typically have domain specific operators and methods to transform raw image data into useful symbols and quantities. The purpose of this project was to explore the limits of the application of a general purpose GP system to solar forecasting. In order to bring the dimensionality of the images down to a practical level of computational effort, the images were processed in a way that would encapsulate some of the relevant information. The images were reduced from 1536x1536 matrices to 24x24 matrices using a Lanczos2 resampling method. The Lanczos method uses the windowed sinc filter seen in equation (4.1) where the value ‘a’ controls the size of the

convolution kernel. The value of 'a' was set to 2, producing a slightly sharper image, which was important for clearly defining the edges of the clouds in the reduced images. The images were resampled using the Lanczos resampling formula in Equation (4.2).

$$L(x) = \begin{cases} sinc(x)sinc(x/a) & -a < x < a \\ 0 & otherwise \end{cases} \tag{4.1}$$

$$\widehat{I}(x_0, y_0) = \sum_{i=\lfloor x_0 \rfloor - a + 1}^{\lfloor x_0 \rfloor + a} \sum_{j=\lfloor y_0 \rfloor - a + 1}^{\lfloor y_0 \rfloor + a} I(i,j) L(x_0 - i) L(y_0 - j) \tag{4.2}$$

The smaller images from the sky camera were then converted from a TIFF format to a grayscale bitmap format. In an attempt to encapsulate the change in the images over time in a single matrix, the previous minute's image matrix was subtracted from the current image matrix as seen in Equation (4.3). This could not be considered an image because the matrix included negative and positive values. However, it was effective in encapsulating the change in cloud cover as an array. Any increase in cloud cover would show up as a positive edge in the matrix, while any decrease in cloud cover would show up as a negative edge. Figure 4.6 illustrates these first image processing steps at a higher resolution. The final matrix was scaled to properly fit a grayscale image.

$$I_{final} = I_t - I_{(t-1)} \tag{4.3}$$

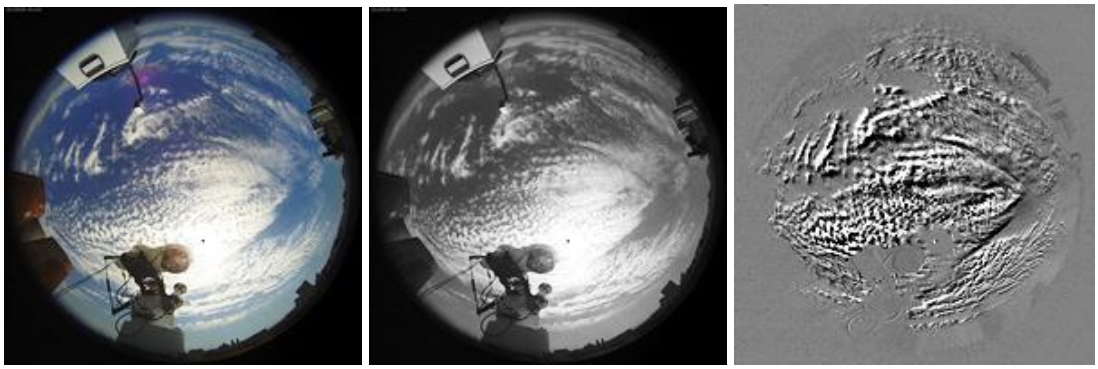

Figure 4.6: Sky image processed to encapsulate cloud cover movement. The sequence of photos shows the transformation of the images into data that can be used to model the change in the cloud cover. The original color image on the left is transformed into the grayscale image in the center. Then the grayscale image from the previous minute was subtracted from the current grayscale image to produce the figure on the right. The movement of the lighter clouds across the darker sky creates a pattern where the brighter edges indicate the direction of the cloud movement and the darker edges indicate where the cloud cover is retreating. From the normalized grayscale picture on the right, the clouds cover appears to be drifting to the top of the image, which corresponds with a northern movement.

Push programs do not have built-in instructions for operations on arrays. This requires the image to be formatted into a vector that can be loaded into a data-stack. This presented an opportunity to further reduce the amount of data in the image. The original images are rectangular images with a fish-eye image in the center. The useful image information is contained only in the circular-lensed area. Additionally, the outer ring of the lensed image could be considered irrelevant to the sky image as it mostly contained images of the earth at the horizon. The pixels in the circular area of the image were concatenated into a single vector. The combination of these pre-processing operations reduced the image data from 7,077,888 points to 305 points. While this is a drastic reduction in detail, it is still enough for the GP to create programs that use image operations to make modest increases in the performance of ground level solar forecasting over the persistence model.

4.3 Configuration of PushGP

The PushGP architecture was modified by removing the Name and Code data-type and instructions and adding an Image data-type and instructions. The Code data-type created programs that were prone to exponential growth. The Execution data-type instructions had many of the same capabilities and did not demonstrate exponential growth behavior. The Image data-type had all the operations of a regular Floating-point stack with the basic statistical operators in Table 4.1 added.

Table 4.1: Added Statistical Image data-type Instructions.

Operator	Description
image.mean	Returns the mean of the image vector.
image.median	Returns the median of the image vector.
image.standard-deviation	Returns the standard deviation of the image vector.
image.maximum	Returns the maximum of the image vector.
image.minimum	Returns the minimum of the image vector.
image. maximum-index	Returns the index of the maximum value of the image vector.
image. minimum-index	Returns the index of the minimum value of the image vector.

Table 4.2 shows the basic parameters used to initialize the run. A major consideration when choosing initialization parameters was the computational effort needed to see a trend towards optimization. Typically, larger populations will yield more population diversity and better results while being less prone to premature convergence at a locally optimal point. Multiple runs with a population of 4000 typically demonstrated a valid improvement over persistence forecasting before over-fitting to the training data started to occur. Evolutionary pressure was lowered by using a tournament size of 2, enabling a more thorough exploration of the fitness landscape with such a relatively small population. The population diversity was

maintained by setting the mutation operator probability equal to the crossover reproduction operation.

Table 4.2 Evolutionary Parameters

Parameter	Value
Population	4000
Max Points	200
Crossover Probability	48%
Mutation Probability	48%
Simplification Probability	4%
Maximum Mutation Points	45
Maximum Simplifications	20
Tournament Size	2

In order to explore the effects of the inclusion and exclusion of different data types, two different types of evolutionary runs were carried out, of which one was performed without the image data.

The fitness function ran individual forecasts through each minute of the evaluation set. If there was no value on the Floating-point stack at the end of the program evaluation, a large fitness penalty was assigned to the evaluation by PushGP. The absolute error of each minute was logged and fitness was judged as the sum of the total amount of errors over the entire evaluation set. An individual with lower cumulative errors was considered to be more fit and processed for reproduction.

4.3 Data to PushGP format

The Genetic Programming implementation was setup to use inputs from a very narrow time frame to make the forecast. Data from the different inputs were loaded into the PushGP and stored as global variables. For each instance of evaluation, the data was loaded onto the appropriate data-type stack before the individual forecasting program was initialized. The image vector was loaded onto the custom Image stack.

The inputs from the DHI, GHI and DNI consisted of two points of data taken at 30 second intervals from each measurement. Additionally a single Clear-sky predictive value targeted at the forecast horizon was made available. These values were appended in order from GHI, DHI and Clear-Sky to DNI. This resulted in a single vector with 7 values. The values were taken from their respective individual data files, appended into a single vector, and then loaded onto the Floating-point stack at the beginning of the program evaluation.

There is an interesting side effect to this loading order. The fitness function considers the value on top of the Floating-point data stack as the forecasted value. Since the latest DNI value is the last value loaded on the data stack, any program that does nothing will return the same value as a persistence forecast. We already know that the persistence forecast is an excellent benchmark for solar forecasting. This creates an implicit stepping-stone for any GP forecast. Even though it is possible to create programs that perform badly compared to persistence forecasting, there will be an inherent bias that any code executed by the forecasting GP will have to outperform the values produced by simply doing nothing.

Chapter 5

Results

The analysis of the results will first make a comparison of the performance of the GP forecasts to with standard benchmark of persistence forecasting. Evolutionary computation typically uses the best individual for each generation as its indication for the overall success of the evolutionary run. Table 5.1 includes the results from the two different types of evolutionary runs. GP Run 1 used only the data available from the Clear-sky model and the irradiance instruments. GP Run 2 additionally included the pre-processed data from the sky images. The Push code for the best GP forecasting programs for the two types of runs can be found in Appendix A.

Table 5.1 Results. This table presents the results of the best GP forecast and the persistence forecast for the evolutionary run using the image data (GP Run 2) and the run only using the irradiance data (GP Run 1)

Characteristic value	**GP Run 1**	**GP Run 2**
Validation Set 1, Persistence RMSE	.2380	.2380
Validation Set 2, Persistence RMSE	.2982	.2982
Validation Set 1, RMSE	.1975	.2443
Validation Set 2, RMSE	.2687	.3037
Validation Set 1, Best RMSE ratio	.0407	.0164
Validation Set 2, Best RMSE ratio	.0989	.0323
Validation Set 1, Mean DNI	.6194	.6194
Validation Set 2, Mean DNI	.6081	.6081
Validation Set 1, relative RMSE	.3188	.3945
Validation Set 2, relative RMSE	.4419	.4995

5.1 GP Run Using Image Data

Figure 5.1 illustrates the typical values seen from a GP run that used the additional image data. It can be seen that in the earlier generations the validation sets seem to improve along with the training set. However, around generation 90, there was a divergence of improvement from the training set and the improvement from the validation sets. This behavior indicates that the forecasting programs were over-fitting to the training data. Essentially, the programs were getting better at predicting only the training set of values, and not necessarily creating useful forecasting algorithms and processes. Consequently, supposedly successful forecasting programs performed much worse in the validation sets. One possible conclusion is that the imagery data in its encoded form requires complex algorithms for proper analysis. Limiting image operation instructions to relatively primitive operators creates a problem with very high epistasis. Evolving proper image operational modules within the genetic programs using only the primitive operators becomes more difficult compared to simply over-fitting the training data. One possible way of addressing this weakness would be to create a very large training data set. This would make it difficult for the forecasting programs to find specific patterns of past forecasting instances. However, in order to lower the computational effort needed for the evaluation of the population, a smaller number of training evaluations out of the larger training set should be randomly chosen for each generational or individual evaluation. A possible pitfall of this approach is that larger training sets tend to create noise in the solutions and programs are correspondingly less accurate. For this experiment, the inclusion of

imagery data accompanied by primitive operators was shown to be unsuccessful in producing broadly optimal genetic programming forecasts.

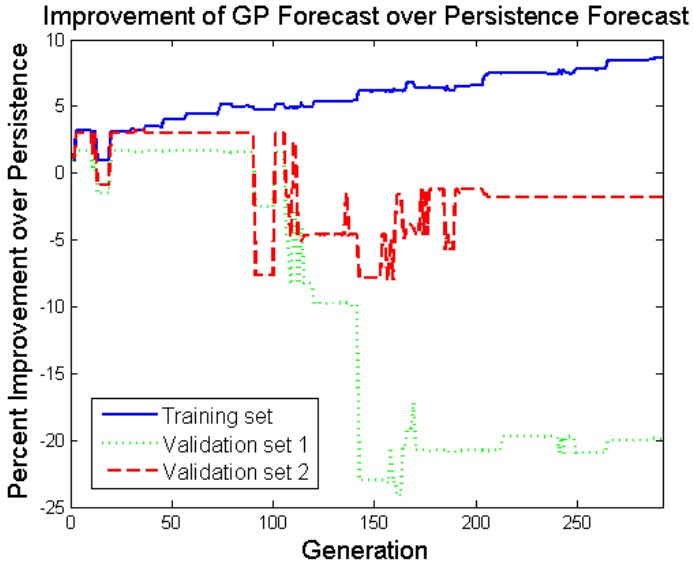

Figure 5.1: Improvement of the GP Run including image inputs.

5.2 GP Run Using Only Irradiance Data

Much better results were observed in the evolutionary run that excluded the imagery data and operators. Figure 5.2 illustrates the improvement of the genetic programming forecast over time. Validation Set 2 achieved a non-trivial 10% improvement in forecasting ability over persistence forecasting during characteristically cloudy times. Validation Set 1 showed an improvement over time, however its performance when compared to persistence forecasting was only marginally better. This is reasonable considering that persistence forecast performance is better during characteristically clear days and Validation Set 1 included clear periods. It is interesting to note that there was a relatively flat improvement from

generation 400 to generation 900 as seen in Figure 5.2. When reviewing the best individuals saved from each generation, there was relatively little that was changed in the code of the best program during the span from generation 400 to 900. However, around generation 900 the best individual program changed, and it appeared that this program had become a stepping stone for the improvement within the rest of the population. At the end of the evolutionary run, we observed an acceleration of improvement in Validation Set 2, while Validation Set 1 seemed to have been negatively impacted from this change. This indicated that whatever programming changes occurred around generation 900, they introduced some mechanism that was more suited for predicting DNI on characteristically cloudy days rather than a mix of characteristically clear and cloudy periods.

The response of the performance increase in the training and validation sets seems to be tightly coupled. The jumps in terms of relative changes in performance appeared to happen around the same generation. This may indicate that the changes made to the best individual programs correlate with a trend towards some generally optimal point. The negative response in Validation Set 1 at the end of the run might indicate that this is a generally optimal point associated with cloudy days.

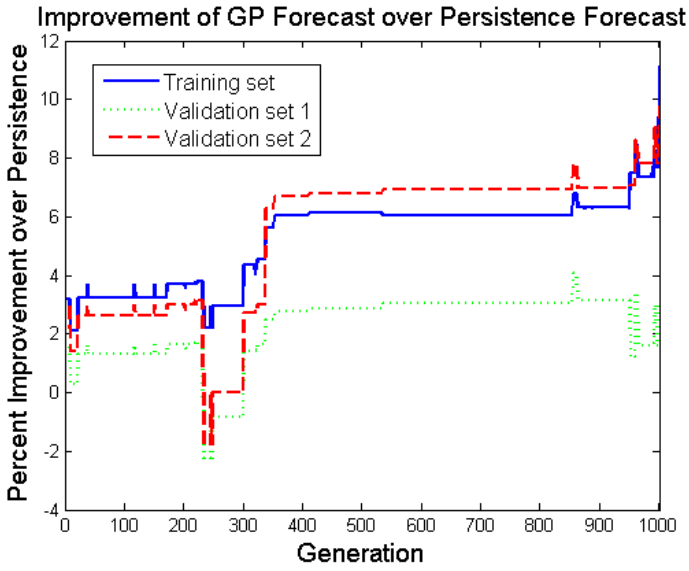

Figure 5.2 Improvement of GP Run not using image inputs. This evolutionary run used only the irradiance and Clear-sky data.

Figure 5.3 shows the decrease in scaled errors from the best individual and the median individual on the training run. It appears that median individual's performance slightly lags behind the best individual during the entire evolutionary run. This indicates that for an evolutionary run with these parameters, the population is very homogenous or at the very least, the performance of the population is very homogenous. The fact that the median individual slightly lags before it converges with the best individual during the entire run indicates that any improvement in the genetic forecast is quickly adopted by the rest of the population. In fact, it appears that it only takes around 20 generations for any improvement in the best individual to be adopted by the median individual. This can indicate a problem in the initialization parameters. There should be greater population diversity when the selection pressure is set low by using a tournament size of 2. It is likely a problem with seeding random individuals at the startup of the evolutionary run. If large portions of the population are removed in

the early generations, the remaining population would quickly loose its diversity. Consequently the parallelized search for solutions would be conduction on a large number of programs that are clustered in a small area of the fitness landscape. The most likely mechanism responsible for this loss of divergence is the fitness penalty given to a forecasting program with an empty Floating-point stack. This mechanism ensures the viability of the programs but has the side effect simultaneously culling population diversity. Perhaps making the penalty part of the reproductive process ensuring that all new programs introduced to the population are viable rather than removing the individuals through selection pressure would counteract the loss of diversity.

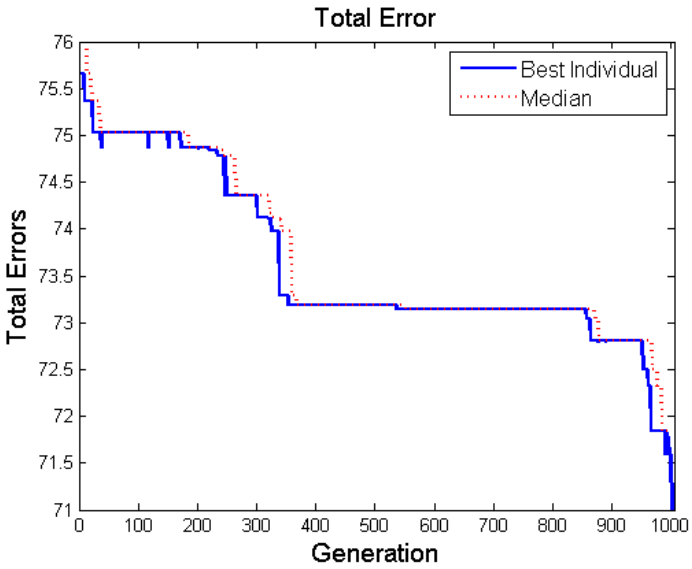

Figure 5.3: Reduction of Total Error in GP forecast. This figure shows the reductions of error in the best and median programs over generations.

Figure 5.4 illustrates the size of the best individual program along with the average size of the population over the evolutionary run. The programs were

artificially constrained to have a Maximum Size of 200 points. By reviewing the code for the best individuals of each generation after the evolutionary run, it was noticed that the rapid downward spikes of the size of the best individual appears to be an effect of the simplification operator. Many times, the forecasting programs included introns, and "dead" code. The simplification operator essentially groomed the best individual of any code that did not directly impact successful performance of the best individual. With a maximum size constraint, the building block hypothesis would have us infer that opening up "space" in the best individuals would create room for additional building blocks that may positively affect performance. Additionally, this would lead us to believe that whatever code is left from a simplification operation will be a much more efficient schema and will be more easily incorporated as an operation module on other programs.

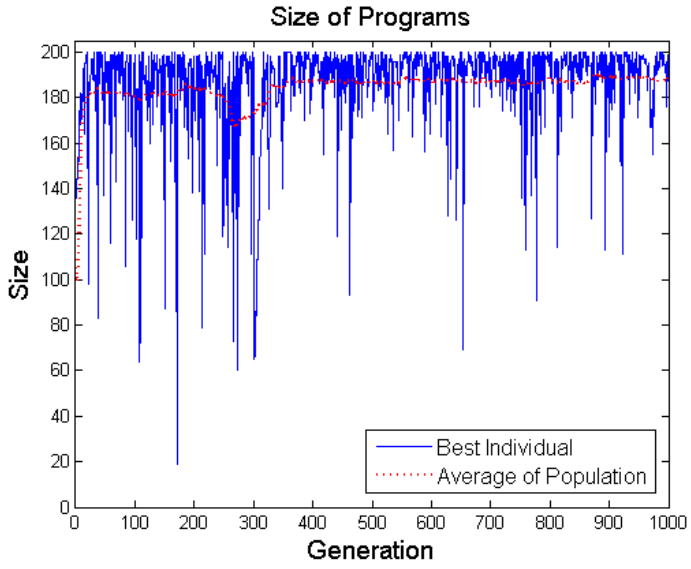

Figure 5.4: The Size of the GP Programs. This graph shows the sizes of the best individual and the average of the population through the generations.

5.3 Interpreting the GP Forecast for Validation Set 1

The top graph in Figure 5.5 below shows us the best GP forecast alongside the measured DNI in in Validation Set 1. The bottom graph in Figure 5.5 compares the absolute error of the persistence and GP forecasts.

From the upper graph in Figure 5.5, it is possible to interpret the behavior of the genetic program forecast. It appears that the forecast programs use the Clear-sky model as an upper limit during periods of the day that are characteristically clear. The GP forecast does not seem to compensate for the fact that the measured DNI was slightly higher than what the Clear-sky model would indicate. It could be argued that if the Clear-sky model was correctly scaled, then the GP forecast would show a significant increase in performance with respect to persistence forecasting, especially during the clear periods of the day. However, this assumes that the error is in the Clear-sky forecast. The error is within the 5% margin of possible error by the NIP. The absolute DNI error graph on the lower portion of Figure 5.5 shows that during the characteristically clear portions of the day, the persistence forecast outperformed the GP forecast. The behavior of the GP forecast seems to mimic the persistence forecast by lagging behind the measured DNI.

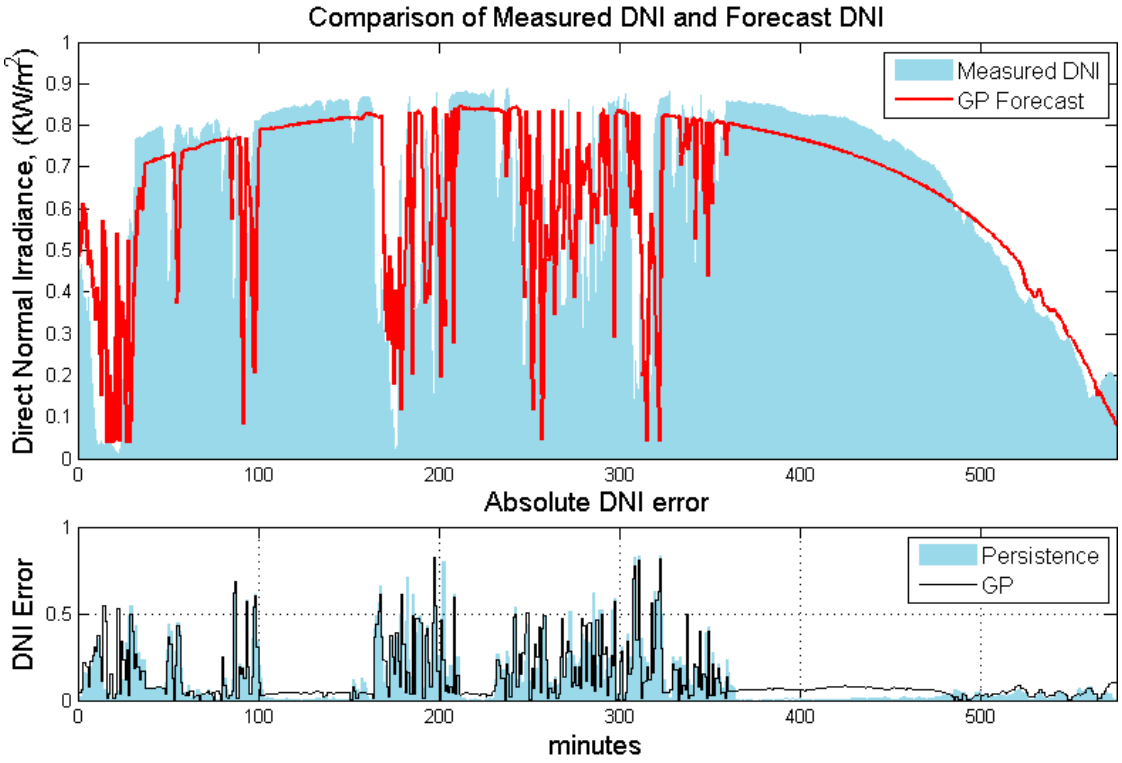

Figure 5.5: Comparison of Measure DNI vs GP forecast of Validation Set 1 and the absolute DNI error for persistence and GP forecasts.

Figure 5.6 compares the persistence and GP forecasts in their predicted versus measured values for Validation Set 1. A perfect forecast would appear as a perfectly linear line from the bottom-left corner to the top-right corner. The persistence and GP forecast both exhibit generally similar trends up to DNI values of about 0.6 KW/m^2. Between 0.7 and 0.9 KW/m^2 however, we see that the GP forecast slightly underestimates the measured value while the persistence forecast overestimates. A quick glance at the data does not indicate where the GP programs are out-performing the persistence forecast. The GP forecast performed only 4% better than the persistence forecast for Validation Set 1. A small incremental improvement of 4% might not be easily detected from a scatterplot such as the one in Figure 5.6.

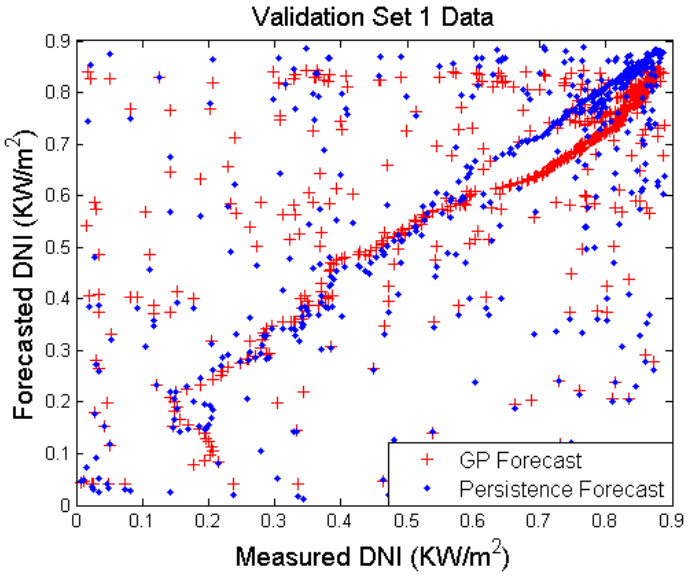

Figure 5.6: Forecast vs Measured for Validation Set 1.

5.4 Interpreting the GP Forecast for Validation Set 2

The top graph in Figure 5.7 shows the performance of the best GP and persistent forecast alongside the measured DNI for Validation Set 2. The behavior of this GP forecast also seems to have the scaling errors correlated with the Clear-sky model. Additionally in the center portion between minutes 80 and 140 it appears that the GP forecast is clipping the forecasted DNI at around 0.6 KW/m^2. It could be that this value is a safe average during a characteristically cloudy day and the GP forecast is uses its forecasting algorithms to switch between discrete forecasting values in some situations.

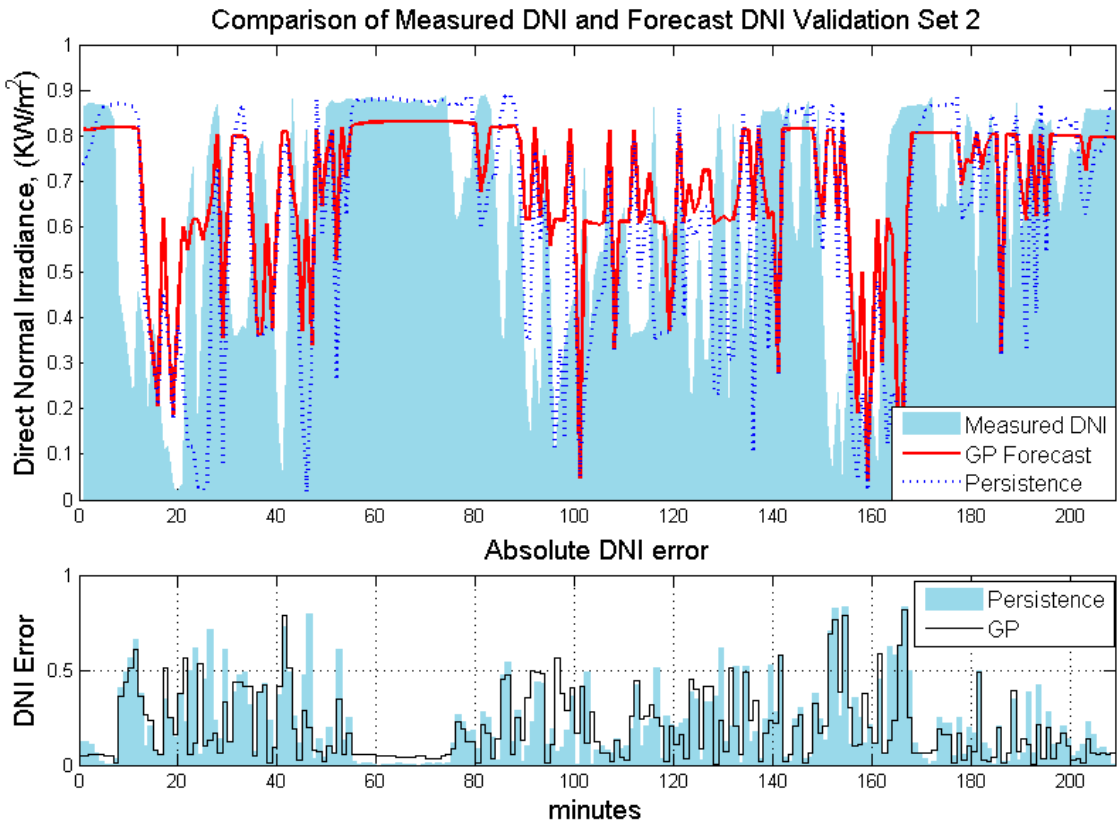

Figure 5.7: Comparison of Measure DNI vs GP forecast of Validation Set 2 and the absolute DNI error for the persistence and GP forecasts.

The scatterplot in Figure 5.8 of compares forecasting patterns of the GP forecast to persistence forecasting. There is a clear difference in the pattern of the values produced by the GP forecast. The GP forecast appears to have a density of forecasting at a DNI of 0.6 or 0.8 KW/m^2, while the persistence forecast shows a more random spread. The GP forecast at 0.8 KW/m^2 matches up with the Clear-sky forecast. The proposed behavior of switching between discrete forecasting points appears to be more obvious in Figure 5.8.

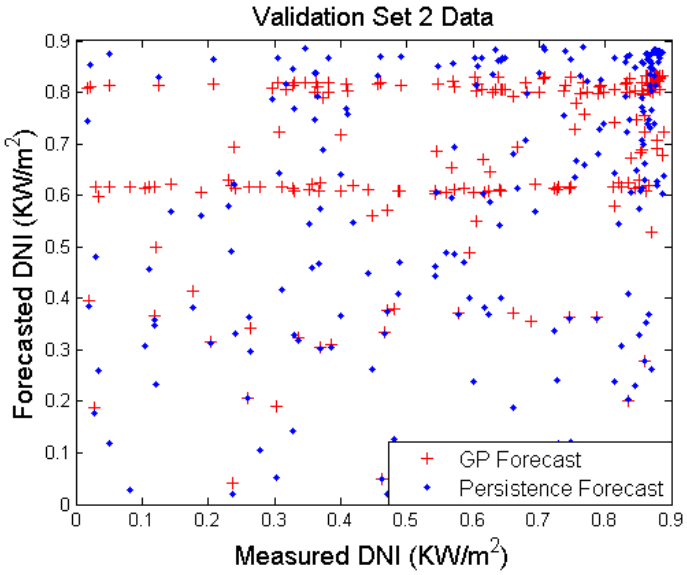

Figure 5.8: Forecast vs Measured for Validation Set 2.

5.5 Comparing Forecast Quality to Standards

Solar forecasting is a relatively new and growing area of research and it is difficult to quantify "good" forecast. It would be useful to establish how well the GP forecast performed relative to industry standard forecasts. Kostylev and Pavlovski proposed a general guideline for comparing solar forecasting [23]. They developed the graph in Figure 5.9 in order to visualize the comparison of forecasting considering the meteorological conditions in addition to the time horizon of the forecast. A study in DNI forecasting found that the rRMSE for same-day forecasts were in the range between 28 to 35% [22]. This research correlated with other studies that used satellite based data [24]. If we consider DNI to be twice as hard at forecasting GHI, then we can scale the rRMSE of the best GP Forecasting programs and plot them in the area of the graph closest to their forecasting horizon. All of the best GP forecasts outperform persistence forecasting, which is intrinsically significant. When those forecasts are put

in the context of the proposed industry guideline in Figure 5.9, all the GP forecasts are seen to perform within the proposed industry standards. It is especially important to note that the GP forecast used days that are considered characteristically cloudy. Every GP forecasting program performed much better than the logarithmic "mostly cloudy" line, and in fact, the best program performed as well as the proposed standard threshold for a mostly clear day.

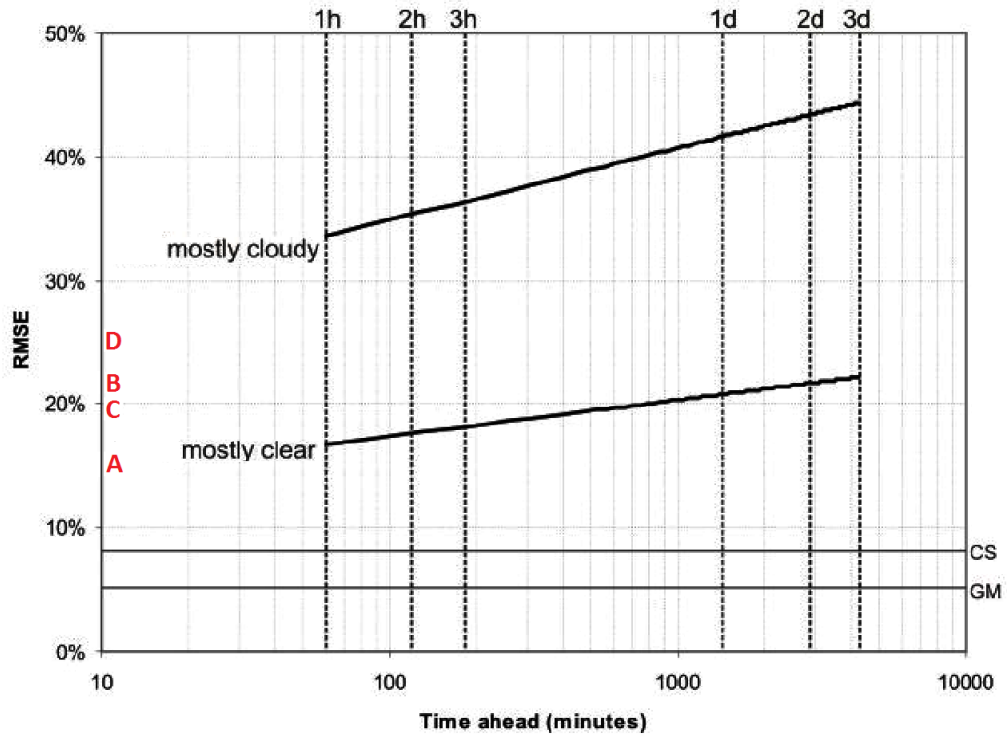

Figure 5.9: Conceptual guideline for Solar Power Forecasting Performance [23]. This was developed to illustrate the general trend of GHI forecasting. The research showed a logarithmic temporal trend with regard to the accuracy of the GHI forecasts. The horizontal GM line indicated the uncertainty inherent in ground instrument measurements. The CS line indicates the relative RMSE of a high quality Clear-sky model on a clear day. The lines marked by mostly cloudy and mostly clear were derived from the normalized mean forecasting performance at multiple locations. A, B C and D mark the interpreted GP forecasting performance as referenced in Table 5.2.

Table 5.2 The Interpreted GHI Values of Figure 5.9. These values represent the DNI values that have been scaled GHI values to reflect the relative hardness of DNI forecasting when compared to GHI.

Genetic Programming Forecast	**rRMSE**
A (no image inputs , Validations Set 1)	16%
B (no image inputs , Validations Set 2)	22%
C (image inputs, Validation Set 1)	19%
D (image inputs, Validation Set 2)	25%

Chapter 6

Conclusion

6.1 Summary Conclusion

This work was able confirm that the general intelligent search method of Genetic Programming was able to create forecasting programs with performance comparable to persistence forecasting. This first attempt to use Genetic Programming highlighted key technical hurdles while providing an indication of the potential behind a more complex implementation.

Although both GP forecasting programs modestly out-performed persistence forecasting, there was a clear trend that the GP forecast without image data outperformed the GP with the image data. Additionally, it appeared that the GP run without image data was still optimizing when the evolutionary run completed itself. Allowing a bigger population with more generations might produce better forecasting programs. Both types of evolutionary runs would show a significant increase of performance over persistence forecasting if the Clear-Sky scaling input was addressed. Representation of the image data as an array data-type and new matrix-based image instructions may allow the GP to interpret the image data in a useful way. This will remove the burden of evolving useful image analysis functions away from the GP.

A particularly important behavior to note is that the GP Forecasts “discovered” the tools of the persistence and the clear-sky models and developed a method of switching between the two models to suit the forecast. The broader implication of such behavior is this work demonstrated that GP used the available data and instruction set to create novel solutions that resembled an improved application of previously known analytical forecasting methods.

DNI is much harder to predict than GHI. For a short 5-minute forecast range, any improvement over persistence forecasting is significant. A 10% forecasting improvement over persistence forecasting is non-trivial. Both evolutionary approaches were able to make a measureable performance improvement over persistence forecasting. Additional expanded explorations of solar forecasting with GP are expected to yield improved forecasts.

6.1 Future work

Reducing the error and variability in the training set is expected to improve the forecasts. The image data could be presented as a sun-centered image. The original fisheye image could be warped and scaled to create a full image spatially normalized to the sun, consequently removing the necessity of the GP forecast to create methods of identifying the sun’s position with respect to the clouds.

In order to take advantage of the input from sky images, established Digital Image Processing algorithms and analytical processes need to be incorporated into the Image data-type instruction set.

Efforts need to be made to reduce the computational effort or at least the time necessary to run an evolutionary sequence. There is incredible potential for parallelization of the process. The program, as it was implemented in the Scheme programming language, was written and compiled in a way that used only a single processing core. Coarse grain parallelism could be implemented by allowing the evolutionary run to process more than one individual at the same time. Finely grained parallelism could be achieved by making use of GPU processing for image and vector operations. The evolutionary process could be expanded to take advantages of advanced evolutionary concepts such as population demes, migration and dynamic selection pressure. Finally, the population from each GP run should be easily saved and reincorporated into future runs, to allow for dynamic parameter changes. As the program is currently written, each population must start with a new generation of completely random individuals. The GP would benefit from saving and reusing past populations so processing isn't lost when the run parameters need to be changed.

There is a tremendous potential for improvement and optimization of the process. By incorporating domain-specific instructions and analytical operators, parallelization of the process and normalized data, Genetic Programming would become a powerful tool for the discovery of innovative and novel solar forecasting methods.

Bibliography

[1] R. Marquez and C. F. M. Coimbra, “Intra-hour DNI based on cloud tracking image analysis", *Solar Energy*, October, 2012.

[2] E. M. Crispim, P. M. Ferreira, and A. E. Ruano, “Prediction of the solar radiation evolution using computational intelligence techniques and cloudiness indices,” *International Journal of Innovative Computing, Information and Control*, vol. 4, no. 5, pp. 1121-1133, 2008.

[3] R. Marquez, V. Gueorguiev , and C. F. M. Coimbra, “Forecasting solar irradiance using sky cover indices,” *ASME Journal of Solar Energy Engineering,* vol. 135, pp. 0110171-0110175*,* 2012.

[4] W. C. Chow, B. Urguhart, M. Lave, A. Dominquez, J. Kleissl, J. Shields, and B. Washom, “Intra-hour forecasting with a total sky imager at the UC San Diego solar energy testbed,” *Solar Energy*, vol. 85, no. 11, pp. 2881-2893, 2011.

[5] J. Huo and D. Lu, “Cloud determination of all sky images under low-visibility conditions,” *Journal Of Atmospheric And Oceanic Technology*, vol. 26, no.10, pp. 2172-2181, 2009.

[6] Q. Li, W. Lu, and J. Yang, “A Hybrid Thresholding Algorithm for Cloud Detection on Ground-Based Color Images,” *Journal of Atmospheric and Oceanic Technology*, vol. 28, pp. 1286-1296, 2011

[7] C. N. Long, J. M. Sabburg, J. Calbo, and D. Pages, “Retrieving Cloud Characteristics from Ground-Based Daytime Color All-Sky Images,” *Journal Of Atmospheric And Oceanic Technology*, vol. 23, no. 5, pp. 633-652, 2006

[8] M. Neto, S. Luiz, A. von Wangenheim, E. B. Pereira, and E. Comunello, “The Use of Euclidean Geometric Distance on RGB Color Space for the Classification of Sky and Cloud Patterns,” *Journal Of Atmospheric And Oceanic Technology,* vol. 27, no. 9, pp. 1504-1517, 2010.

[9] N. Hodge, “U.S.’s Afghan headache: $400-a-gallon gasoline,” *Wall Street Journal,* 2011.

[10] H. Pedro, C.F.M. Coimbra, “Assessment of forecasting techniques for solar power production with no exogenous input,” *Solar Energy*, Volume 86, Issue 7, July 2012, Pages 2017-2028.

[11] A. Sfetsos, A.H. Coonick, "Univariate and multivariate forecasting of hourly solar radiation with artificial intelligence techniques," *Solar Energy*, Volume 68, Issue 2, Pages 169-178, February 2000.

[12] D. Lew and R. Piwko, "Western Wind and Solar Integration Study," *Technical report,* National Renewable Energy Laboratories, 2010.

[13] G. D. Rodriguez, "A Utility Perspective of the Role of Energy Storage in the Smart Grid," Power and Energy Society General Meeting*, IEEE*, pp. 1-2, July 2010.

[14] V. Sundar, "Integration of renewable resources: Operational requirements and generation fleet capability at 20 percent RPS," *Technical report,* California Independent System Operator (CAISO), 2010.

[15] G. Reikard, "Predicting solar radiation at high resolutions: A comparison of time series forecasts," *Solar Energy*, vol. 83, no. 3, pp. 342-349, 2009.

[16] B. Molineaux, P. Ineichen, and J.J. Delaunay, "Direct luminous efficacy and atmospheric turbidity: improving model performance," *Solar Energy,* vol. 55, no. 2, pp. 125–137, 1995

[17] S. Alam, S. C. Kaushik, and S.N. Garg, "Computation of beam solar radiation at normal incidence using artificial neural network," *Renewable Energy,* vol. 31, no. 10, pp. 1483-1491, 2006.,

[18] F. S. Tymvios, C. P. Jacovides, S. C. Michaelides, and C. Scouteli, "Comparative study of Angstrom's and artificial neural networks' methodologies in estimating global solar radiation," *Solar Energy,* vol. 78, no. 6, pp. 752-762, 2005.

[19] L. Spector and A. Robinson, "Genetic programming and Autoconstructive evolution with the push programming language," In *Genetic Programming and Evolvable Machines,* vol. 3, no. 1, pp. 7-40, 2002.

[20] R. Johnson, J. Shields, and T. Koehler, "Analysis and interpretation of simultaneous multi-station whole sky imagery," *Marine Physical Laboratory, Scripps Institution of Oceanography*, University of California San Diego, SIO 91-3, PL-TR-91-2214, 1991.

[21] J. Shields, M. Karr, A. Burden, R. Johnson, and W. Hodgkiss, "Continuing support of cloud free line of sight determination including whole sky imaging of clouds," Final Report for ONR Contract N00014- 01-D-0043DO #13*, Marine Physical Laboratory*, *Scripps Institution of Oceanography*, University of California San Diego, Technical Note 273, 2007.

[22] R. Marquez and C. F. M. Coimbra, “Forecasting of global and direct solar irradiance using stochastic learning methods, ground experiments and the NWS database," *Solar Energy*, vol. 85, no. 5, pp. 746-756, 2011.

[23] V. Kostylev and A. Pavlovski, “Solar power forecasting performance towards industry standards,” In *Proceedings of the 1st International Workshop on the Integration of Solar Power into Power Systems,* Aarhus, Denmark, 2011.

[24] F. Vignola, P. Harlan, R. Perez, and M. Kiniecik, “Analysis of satellite derived beam and global solar radiation data,” *Solar Energy,* vol. 81, no. 6, pp. 768–772, 2007.

[25] M. Lave and J. Kleissl, “Solar variability of four sites across the state of Colorado,” *Renewable Energy,”* vol. 35, no. 12, pp. 2867–2873, 2010.

[26] R. Marquez and C. F. M. Coimbra, “A novel metric for evaluating solar forecasting models," *ASME*, Paper no. ES2011-54519, pp. 1459-1467, 2012.

[27] T. E. Hoff and R. Perez, “Quantifying pv power output variability,” *Solar Energy*, vol. 84. no. 10, pp. 1782-1793, 2010.

[28] T. Cebecauer, M. Suri, and C. Gueymard, “Uncertainty sources in satellite-derived direct normal irradiance: How can prediction accuracy be improved globally?,” In *Proceedings of the SolarPACES Conference*, Granada, Spain, pp. 20-23, 2011.

[29] S. Wright, “Evolution in mendelian populations,” *Genetics,* vol. 16, no. 2, pp. 97-159, 1931.

[30] J. H. Holland, “Adaptation in natural and artificial systems,” Second Edition, *MIT Press,* 1992.

[31] A. H. Wright, “Genetic algorithms for real parameter optimization,” in G. Rawlings, ed, Foundations of Genetic Algorithms*, Morgan Kaufmann*, 1991.

[32] C. Z. Janikow and Z. Michalewicz, “An experimental comparison of binary and floating point representations in genetic algorithms,” in R.K. Belew and L.B. Booker, eds, In *Proceedings of the Sixth International Conference on Genetic Algorithms,* pp.205-218, 1991.

[33] J. R. Koza, “Hierarchical genetic operating on populations of computer programs,” In *Proceedings of the 11th International Joint Conference on Artificial Intelligence,* San Francisco, vol. 1, pp. 768-774, 1989.

[34] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," In *Proceedings of the International Conference on Genetic Algorithms and their Applications, [CMU],* pp.183-187, 1985.

[35] J. R. Koza, “Genetic programming: On the programming of computers by natural selection,” *MIT Press*, Cambridge MA, 1992.

[36] M. Lipsitch, “Adaptation on rugged landscapes generated by local interactions of neighboring genes,” *Technical Report, 91-02-01, Santa Fe Technical Report* pp. 128-135, 1991.

[37] L. Spector, “Autoconstructive evolution: Push, pushGP, and pushpop,” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001),* pp.137-146, San Francisco, California, 2001.

[38] L. Spector, C. Perry, J. Klein, and M. Keijzer, “Push 3.0 programming language description,” *Technical Report,* Hampshire College School of Cognitive Science, 2004.

[39] J. Klein and L. Spector, “Genetic programming with historically assessed hardness,” In *Genetic Programming Theory and Practice VI*, New York: Springer-Verlag, 2008.

Appendix A

Appendix A.1 Best Program GP Run 1

(((((float.tan) exec.dup) ((float.yankdup (float.swap) (float.tan (float.tan float.cos) (exec.dup) float.tan exec.dup)) ((exec.dup boolean.frominteger (integer.> (float.sin float.sin integer.min (((boolean.rot integer.>) integer.- (integer.* (boolean.pop) integer.<) boolean.yank (0.041978706310403284 float.max (boolean.rot exec.s)) ((integer.swap) exec.rot)) (exec.s)) (boolean.=) 0.7505782542564621 float.min ((boolean.flush) integer.yank boolean.dup) integer.<) boolean.stackdepth exec.do*range))) float.max) (float.min) ((((boolean.frominteger boolean.yank) float.rot integer.yank) (float.tan (float.cos (exec.do*range)) integer.<) ((boolean.or (float.* boolean.not)) exec.swap exec.swap integer.fromboolean) integer.+ (float.rot) float.yankdup)) (boolean.swap) integer.<) exec.s integer.fromboolean (exec.dup)) exec.dup (float.rot ((integer.max) boolean.swap ((boolean.flush) integer.yank boolean.dup) exec.dup (() exec.swap integer.=)) integer.stackdepth float.* ((integer.%) exec.dup) (((exec.dup)) boolean.swap)) ((float.tan) (integer.max (exec.dup) (integer.% integer.> ((float.+) exec.pop integer.%) exec.do*count) exec.dup float.<) (boolean.=) (integer.=) (float.tan (integer.% (boolean.frominteger)) boolean.yank ((integer.fromfloat) 84) float.max (((float.swap float.tan ((float.cos exec.do*range) float.dup exec.y float.swap) float.min) integer.rot boolean.yank float.dup) ((boolean.flush) integer.dup) integer.fromboolean exec.do*count))))

Appendix A.2 Best Program GP Run 2

(((integer.fromboolean) image.minimum-index float.max exec.pop (integer.flush ((exec.noop () () (float.yankdup exec.yankdup ((exec.flush ((exec.do*times image.minimum integer.fromboolean (integer.flush)) ((float.sin 0.6604357500026553 integer.rot (float.flush float.fromimage) (integer.dup (integer.dup) float.frominteger) (boolean.dup)) (image.fromfloat ((boolean.swap) image.stackdepth (integer.swap exec.dup) image.<)) integer.yank)) exec.y) exec.do*times image.yankdup (((float.swap integer.yankdup (exec.stackdepth image.dup) image.=) (exec.do*count (exec.pop (boolean.or) (exec.pop (float.pop) float.max)) image.maximum-index exec.flush image.yank)) ((boolean.dup (((integer./) image.max) (image.fromfloat) exec.rot integer./) integer.*) image.maximum-index exec.flush) exec.stackdepth (float./))) integer.+ boolean.fromfloat (integer.- (exec.if) boolean.and))) integer.* (float.rot) integer.pop) (integer.rot integer.> ((boolean.fromfloat exec.= (integer./ (float.flush boolean.yankdup (image.max))) integer.rot (boolean.stackdepth float.tan)) image.swap (boolean.dup (boolean.fromfloat boolean.pop) integer.fromboolean))) ((exec.yankdup (image.< exec.yankdup) (integer.<))))) (image./) ((image.= (integer.* integer.dup float.stackdepth) exec.if) ((exec.do*count) integer.swap (exec.pop exec.dup integer.flush)) float.min) 62 ((float.fromboolean (((boolean.rot exec.=) float.dup) ((image.rot exec.k exec.stackdepth) boolean.swap) image.+ 82) boolean.= (float.fromboolean (integer.pop)) (image.* (float.fromimage) (float.+ image.minimum) (exec.yankdup (integer.rot integer.> float.max (exec.yankdup)) exec.if (exec.dup) (float.+ boolean.or)))) (integer./ (exec.rot) integer.- integer.dup) image./ (float.= (((image.minimum-index integer.rot)) integer.+) ((exec.do*range exec.do*times) integer.flush (float.-)) ((exec.yankdup) float.dup image.dup))))