**GPCODE: Genetic Programming for Evolving Ocean Chlorophyll-a Algorithm Generation**

Yuping Liu1, 2, John Moisan3, Guoqing Wang1,2

*1. Ocean Ecology Laboratory, Goddard Space Flight Center, National Aeronautics and Space Administration, Greenbelt, Maryland 20770 USA*

*2. Science Systems and Applications, Inc. (SSAI), Lanham, Maryland 20706 USA*

*3. Wallops Field Support Office, Wallops Flight Facility, NASA Goddard Space Flight Center, Wallops Island, Virginia 23337 USA*

**Abstract**:

Genetic Programming of Coupled Ordinary Differential Equations (GPCODE) is a Genetic Programming (GP) package developed under funding from NASA and The Gordon and Betty Moore Foundation. Through evolutionary computation (EC) GPCODE can generate chlorophyll-a algorithm from in-situ measurements of chlor-a and remote sensing reflectances (Rrs) in visible bands (Ocean Color). In this paper, a series of experiments were implemented with SeaWiFS visible bands Rrs and in-situ Chlorophyll-a data, aiming at examination of how GPCODE performs under different fitness or error metrics by comparing the estimates from GPCODE algorithms to real Chlorophyll-a observations. Statistical analyses were applied to examine their quantitative variations. Considering the possibility of overfitting and bloat of GPCODE estimate with limited training dataset we used resampling technique and bootstrapping (BS) to GPCODE to generalize the algorithm estimates to better accommodate varying environments. The evaluation of the algorithms by GP and BS for different fitness/error metrics was made against SeaBASS in-situ reflectance. Results show that the algorithms derived from genetic programming perform well on most cases compared to conventional OC4 algorithm, but they may present inconsistent skills in the estimation of Chlorophyll-a for different concentration ranges.

GPCODE is written in F90/F95 and parallelized using Intel MPI, currently running on NASA/GSFC NCCS Discover cluster.

**Key Words:**

Genetic Programming, Chlorophyll-a, Remote Sensing Reflectance, Bootstrapping, Overfitting, bloat, Generalization, OC4

**1. Introduction**

Genetic programming (GP) is an evolutionary computation (EC) technique that automatically solves problems without requiring the user to know or specify the form or structure of the solution in advance [1]. As a branch of machine learning, genetic programming (GP) has found its applications in many scientific fields since its first appearing in 1960’s, such as features extraction, image recognition, automatic detector, image differentiation [9], etc. In the ocean color research, genetic programming is also studied to derive algorithms to estimate chlor-a from remote sensing reflectances (Rrs) [3,4]. The GP experiment by Fonlupt [3] tried to find the algorithms for different waters [open water and coastal water]. Chang [4] used gene expression programming and IOCCG data to derive algorithms and compared to SeaWiFS data.

In this paper we present a genetic programming evolutionary computation software system, GPCODE, it has following features: 1) it is written in object-oriented Fortran 90/95; 2) it is parallelized by intel MPI and run on NASA/GSFC NCCS supercomputing cluster; 3) It can utilize hundreds and thousands of computer cores in NCCS discover cluster, making it possible to have more individuals in the population and more generations in the run; 4) it has a few different fitness selections for generation of best algorithms, they are SSE (sum of square error), MAPE (mean absolute percentage error), MAE (mean absolute error). They provide the criteria for the changing environments to give appropriate measurement of the fitness between the results produced by algorithm and real data; 5) The GPCODE provides the function of bootstrapping to generalize the output algorithms.

The basic components of GPCODE are composed of generating initial individual models of population; computing the fitness of all individuals of population; reproducing next generation of individuals from parents, by asexual reproduction, crossover reproduction and mutation methods; generating optimized model parameters by combined run of genetic algorithm/lmdif routine; the GPCODE has a set of functions of 23 from +, -, \*, / to more complex log and exponential functions. They are the basis for constructing complex ocean color algorithm.

To avoid overfitting and bloat with specific training dataset, bootstrapping is added in GPCODE to generalize algorithm, resampling technique of training dataset is used to produce multiple datasets for training.

GPCODE uses a control file which provides users with flexibilities to set up flags or control variables such as individuals in population, number of generations, percentages of available selection operators, maximum levels of the tree etc.

In Section 2, we’ll present technical details of our GPCODE genetic programming for ocean color. Section 3 will examine the GPCODE experimental results, analysis of its performance and its comparisons with the estimates from other method. Finally, Section 4 will summarize GPCODE’s performance and future in ocean color application.

**2. Genetic Programming and Implementation of GPCODE**

**2.1 A brief description of Genetic Programming**

In genetic programming, a population of computer programs or equations are evolved generation by generation from its initial individuals to a certain point where the programs will be output as final resultant programs. In the whole process from initialization of the program population every step is composed of a random transformation of previous population to a new or better hopefully generation of programs.

By randomness, generic programming mimics the natural process of selections, such crossover, mutation, survival of the fittest etc., to solve the problem in real world. It can avoid some defects introduced by deterministic methods and solve problems in a novel and unexpected ways [1].

As an evolutionary computation method genetic programming possesses following key components [8]:

1. Population of individuals (potential solutions to the problem)
2. Fitness for selection of fitter individuals from the population (like Darwinian principle of survival)
3. Genetic operators which modify selected parent individuals to produce new children individuals

A basic control flow for genetic programming is presented in Figure 1, the survival fittest is used to be the solution [1].



Figure 1. Basic control flow of genetic programming.

The basic computation steps in genetic programming are shown in Table 1.

Table 1. General Algorithm of Generic Programming

|  |
| --- |
| a) randomly generate initial individual programs of population.  b) run programs and compute each individual’s fitness.  c) Randomly select individuals from population with certain bias towards higher fitness  d) apply genetic operators to the selected individuals to create new individual programs with pre-defined probabilities.  e) repeat b) to newly generated individuals.  f) stop when certain criteria reached. |

**2.2 Implementation of GPCODE**

GPCODE is a realization of evolutionary computing (EC). It automatically generates optimized program to obtain the ocean color algorithm without knowing the structure of program and what components are explicitly needed pre-hand. The relative blindness to the solution and randomness in the formation of solution differentiate genetic programming from other evolutionary computation. In this Section we present the details in our GPCODE system.

**2.2.1 Representation of programs/equations in GPCODE**

In GPCODE, programs/equations are the basic operation unit. The population in GPCODE is composed of hundreds to thousands of these programs. They are expressed as binary trees or syntax trees instead of lines of code. A binary tree is composed of a root node, some interior nodes and terminal nodes. The interior nodes, including root node are functions from function set (this will be presented later). Terminal nodes or leaf are variables and constants. Variables are the inputs or remote sensing visible bands reflectances. Figure 2 shows an example of binary tree program representation. Actual trees can have different depths and shapes which are randomly created. The depth of the trees is set up by users in GPCODE control file. The depth indicates the maximum depth/levels a tree can reach up, but not every tree in GPCODE is of full depth.

A picture containing watch

Description automatically generated

Figure 2. Tree representation of a program.

**2.2.2 Initialization**

Initialization randomly generates initial population of programs. A larger initial population provides a broader equation search space (or diversity) for GPCODE to operate and produce globally optimized programs. This is one of the most important elements in GPCODE or any other GP system. Meantime large population generally need more computing resource and may cause the solution to overfit to the training data. To overcome overfitting problem a bootstrap method is used in GPCODE, which will be described later.

Based upon computing resources available to users, the population in GPCODE is normally between 1000 to 5000. Larger population needs more computing time for evolutionary computation with each iteration of genetic operation. The total iterated generations with large population may reach an acceptable solution in a less iterations than that of small population.

**2.2.3 Error Metrics and Fitness**

The fitness of programs is calculated from the errors between the outputs computed from programs in population and the training data set. This error shows the distance or closeness between the outputs by GP programs/equations and training data.

There are two types of fitness which can be applied in evolutionary computing: they are absolute fitness and relative fitness. The absolute fitness is the point where evolutionary computing stops when its fitness reaches the preset absolute fitness value. The relative fitness pursues a higher fitness from generation to generation until the evolution comes to a preset number of generations or fulfills a computation time.

In GPCODE there is no absolute fitness. The fitness is given from three error metrics available for users to choose. They are:

Sum of Squared Error (SSE): SSE = ∑(ai – xi)2

Mean Absolute Percent Error (MAPE): MAPE = 1/n\*∑ (ai – xi)/xi|

Mean Absolute Error (MAE): MAE = 1/n\*∑|ai – xi|

Where:

n: number of data points

xi: data

ai: GP solution

Fitness in GPCODE is given as follows:

Fitness = SSE0/Error (SSE/MAPE/MAE)

SSE0sse = ∑xi2

SSE0mape = ∑100\*|xi – xavg|/(n\*xi)

SSE0mae = ∑|xi-xavg|/n

Xavg = ∑xi/n

**2.2.4 Selection and Genetic Operations**

In GPCODE, we use three (3) types of genetic operations by random selection and their probabilities which are controlled in control file:

1. GP Fitness-Proportionate Asexual reproduction: Replace randomly chosen GP individual with another randomly chosen GP individual which may have a higher fitness.
2. GP Tournament Style Sexual Reproduction (crossover): Randomly select two (2) parents from GP Individuals using tournament style selection and crosses the parameter strings to create two new ‘children’ parameter strings.
3. GP Mutation: randomly modifies GP tree nodes with randomly chosen values, terminals will be optimized later on using genetic algorithm’s lmdif subroutine.

In GPCODE we have also an Elitist reproduction which directly copy the programs with best fitness to next generation according to preset probability.

GP departs significantly from other evolutionary algorithms in the implementation of the operators of crossover and mutation (1). In GPCODE the crossover and mutation are both subtree crossover and subtree mutation.

Crossover operation is performed by selecting two (2) parents randomly and exchange their branches or subtrees between the two parents. Figure 3 presents an example of crossover in genetic programming.

Figure 3. Example of Crossover in genetic programming.

Diagram

Description automatically generated

The mutation in GPCODE is to randomly selects a mutation point in a tree and substitutes the subtree rooted there with a randomly generated subtree. This is illustrated in Figure 4.

Diagram

Description automatically generated

Figure 4. Example of Mutation in genetic programming.

**2.2.5 Parameter Optimization**

In GPCODE the primitives include interior function nodes and terminal variables and parameters (constants). The initial constants in programs are randomly generated. In the later recombination of crossover and mutation the constants in programs are optimized by some random process and recombination rules. These constant parameters are handled mainly in those GA subroutines.

**2.2.6 Function Set for Interior Nodes**

As we mentioned before all interior nodes in a GP tree are functions from the pre-determined function sets for GPCODE. The variable leaves are the inputs of remote sensing reflectance (Rrs) and constant leaves are numerical values for the program. All nodes are randomly selected from the function set. In GPCODE’s function set there are total 23 functions available for a GP tree to select from. The whole list of all the functions in the function set can be found in Appendix 1.

It is not necessary for GPCODE to use all of them in the implementation. In our experiments the conditional functions are not used, and all the elementary functions are selected for use in control file.

**2.2.7 Bootstrapping**

The solution in GPCODE is obtained by evolutionary computation based on a training dataset. While the solution produces satisfactory estimates with the training dataset it may depart significantly or lose its consistency when applied to other unseen data [2, 12]. This is the so-called overfitting or lack of generalization; it describes a situation that the solution behaves extremely well only with the training data but other observed data. Obviously, this type of solution may be less useful and reliable in real world applications.

Bootstrap theory is a computerized implementation of basic statistical concepts and was first introduce by Efron as a computer-intensive resampling technique [11]. To mitigate the impact to solution caused by limited training data in GPCODE evolutionary computing we adopted the technique of bootstrapping. This technique is supposed that it can obtain reliable standard errors, reduces the overfitting and increase the quality of the solution. The kernel of the technique is to use a resampling method to overcome the limited training data available. The resampling method resamples the original dataset many times with replacement. Each resampled dataset has same size or same number samples as the original one, some of the data point may be repeated.

These resampled data are used to be against the GPCODE solution estimates and generate a series of differences or errors. We use these errors to produce a bootstrap standard error (BSE) which is similar to the methods in [11] and combine it (BSE) into the errors of solution estimates against original data. Following presents the calculation method for BSE in GPCODE and its use as a bootstrap approach:

First, calculate the mean error of each resample:

Meanb = ∑Errorbi(b,i)/n\_time\_steps [1]

Here Meanb is mean error of all data point of bootstrap replicates b; Errorbi(b,i) is differences of GPCODE solution against bootstrap replicate b at each data point I:

Errorbi(b,i) = solutiongp(i) - dataib

n\_time\_steps is point number of training dataset

Next, calculate the variance of resamples:

Varanceb = ∑(Errorbi-Meanb)2 [2]

Among them Varianceb is Variance of bootstrap replicate b;

Now is the Standard Deviation of the resample:

Stdb = SQRT(Varianceb/(n\_time\_steps-1) [3]

Stdb is Standard deviation of bootstrap replicate b

With above statistics of each bootstrap resample, we can compute following values:

Mean = ∑Stdb/(n\_BT) [4]

Var = ∑(Stdb – Mean)2 [5]

BSE = SQRT(var/n\_BT -1) [6]

Their meanings are as follows

Mean: Mean of Stdb of all bootstrap replicates

Var: Variance of Stdb against Mean

BSE: Standard deviation of Var

Finally, we have the adjusted array of errors of GPCODE estimates against original training data:

Fvec0 = Fvec0 + BSE/(n\_time\_steps)

Fvec0: Array of error between training data and GPCODE solution

n\_BT: number of bootstrap replicates

**3. Experiments and Performance**

**3.1 Training Data and Some Key Control Variables Setup**

In the GPCODE experiment presented here, the training and validation data used are global bio-optical in situ dataset obtained by HPLC from Valente et al. (2019). SeaWIFS remote sensing reflectance (Rrs) at 6 bands are the satellite inputs for ocean color algorithm. The SeaWiFS Rrs and in situ Chla matchups for SeaWiFS construct all the original inputs for the generation of GPCODE ocean color algorithm. The SeaWiFS Chla estimated by standard algorithm OC4 (O’Reilly et al. 1998, 2019) are used to comparison with GPCODE algorithm [16].

As a parallel software package GPCODE is run on NASA GSFC’s Discovery cluster. Its operation is distributed on multiple cluster nodes which can be assigned in Slurm script file. This high-performance computing environment makes it possible for GPCODE to run with complex function set and program structure; large population and generations. A larger population means more individuals are available to participate in the evolution and higher possibility to generate more accurate or complex algorithm.

In our experiment, we set the cycle run time to 12 hours. Each cycle time can have different generations because of real time system resource available. Table 2 shows the setting of key control variables in the experiment.

Table 2 Control Variables Setting

|  |  |
| --- | --- |
| N\_gp\_indivisuals | 1000 |
| N\_gp\_generations | 500 (may not be the actual number) |
| GP\_Elistist\_Probability | 0.0 |
| GP\_Asexual\_Reproduction\_probability | 0.10 |
| GP\_Crossover\_Probability | 0.85 |
| GP\_Mutation\_Probability | 0.05 |
| N\_levels | 11 |
| Eval | 0/1/2 (SSE/MAPE/MAE) |
| Bootstrap | 0/integer number greater than 0 |

The sum of all probabilities in table 2 should be 1.0. Their selection and recombination try to follow the diversity and evolution in biological world to produce some good algorithms from amounts of individual algorithm/equations. The different percentages of types will affect the final outputs.

The number in Bootstrap can be 0 or any other integer: number 0 turns off the bootstrap, 1 or above integer will activate bootstrapping function. In our GPCODE run with bootstrapping it is set to 50. The resampling number 50 here is not of any specific, users can choose a smaller or larger number. A larger number will take longer time to evolve.) ----This paragraph may be moved or deleted.

**3.2 Algorithms with Different Error Metrics**

In above section 2.2.3, we have described the error metrics available to use in GPCODE and the relation between fitness and error metrics. In the following we mainly present the GPCODE solutions and our analyses based on the error metrics of MAPE and SSE.

Figure 5 and 6 illustrate the Chlor-a estimates by GPCODE evolved genetic programs using SeaWiFS’s 6 bands reflectances Rrs against in situ Chla, with error metrics of MAPE and SSE respectively. In the GP Chla vs Measured Chla correlation, solution with MAPE and solution with SSE have comparable correlation coefficients. MAPE GP Algorithm has a small MAPE, a large bias and SSE while SSE GP algorithm has a small SSE, bias and a larger MAPE. Different fitness or error metric in the GP genetic computing do generate different solution algorithms. From these two figures we find the solution algorithm with MAPE shows good skills consistently from low Chla values to high values. The solution algorithm with SSE behaves good only from Chla >= 1, it is of no sensitive to Chla values of less than 100.

One thing that needs to keep in mind is because of the randomness of GP genetic programming each GPCODE run will very possibly generate a somewhat different algorithm, but they are not far from each other. This is the basis we work on the experiment. We will have a specific experiment to show this.

Chart, scatter chart

Description automatically generated

Figure 5 GP Chla estimates vs Measured Chla with error metric of MAPE

Chart, scatter chart

Description automatically generated

Figure 6 GP Chla estimates vs Measured Chla with error metric of SSE

Figure 7 and figure 8 present the histogram of MAPE and SSE with increase of generations. From the two figures MAPE and SSE errors approach to their steady values both around generations of 160. Their frequencies also reach their highs in about same generations. A visible difference in their histogram is SSE GP solutions show some larger variations.

Chart

Description automatically generated

Figure 7. GPCODE estimate MAPE errors histogram with generations

Graphical user interface, chart

Description automatically generated

Figure 8. GPCODE estimate SSE errors histogram with generations

Figure 9 shows the 10th generation of entropy from GPCODE algorithms with MAPE, and figure 10 the 370th generation of GPCODE solutions. In these GPCODE solution entropy figures, the entropy/X axis represents the functions (by categories, +, -, \*, / etc.) used in each individual summed and normalized in each bin from 0-5 with bin size of 0.1. A larger entropy means a wider use of functions in the function set. In contrary, a small entropy means a less part of functions in function set are used. The vertical axis (y) of “relative number” represents a relative or normalized number from all individuals in the specific generation. We can see that in the initial stages, the solution population has a more diversified function collection, which is what we expected and liked. After hundreds of evolutions the functions used by categories in GP solutions become smaller than its original solutions in a generation. But this doesn’t mean the structures of the later solutions are simpler, instead they become more complicated (see figure 13, 14). This means after a long evolution the solutions in the population is inclined to converge with a limited functions. Figure11 and 12 are the 10th and 400th generations of GP solution entropy with error metric of SSE. They present similar features as those with MAPE.

Chart, histogram

Description automatically generated

Figure 9 GPCODE solutions entropy of 10th generation for MAPE

Chart, histogram

Description automatically generated

Figure 10 GPCODE solutions entropy of 370th generation for MAPE

Chart, histogram

Description automatically generated

Figure 11 GPCODE solutions entropy of 10th generation for SSE

Chart, histogram

Description automatically generated

Figure 12 GPCODE solutions entropy of 400th generation for SSE



Figure 13 Tree representation of best one of GPCODE solutions in initial population



Figure 14 Tree representation of best one of GPCODE solutions in population of last generation (409)

Because of the inherent properties of genetic programming each repeated (all control variables are exactly the same) individual GPCODE run will not generate exactly same algorithm/program. Now the problem arises, in what extent/how well does the chlorophyll-a algorithm generated by GPCODE run represent the regression relation between the SeaWiFS visible bands Rrs and Chlorophyll-a. To answer this question, we made 20 repeated GPCODE runs with exactly same control variables and plot the results into Taylor diagram in Figure 15. From the figure we can see that all the 20 GPCODE algorithms estimates have consistent correlations of 0.8 or a little more than 0.8 with observation data, showing all algorithms possess the skill to estimate the variations of Cholorophyll-a. The figure also shows these algorithms estimates can have a wide range of standard deviations and RMSEs. This means these algorithms may have quite different bias in their estimates.

The Taylor plot in Figure 16 illustrates two GPCODE estimates with MAPE and SSE. Both run’s evolutionary generations are over 800. The a-little-far one from data center is estimate of GPCODE run with MAPE, the closer one is GPCODE run with SSE. Though SSE seems to have a better RMSE than MAPE, both of them present verifying skills in estimating Chlorophyll-a. From above figures and our analysis, it is safe to say that both error metrics/fitnesses in GPCODE run can produce good enough algorithms/programs for estimation of Chlorophyll-a after a certain generations. One thing that needs to point out is the best algorithm/program is not necessary the one in last generation. It may appear in any part of the evolutionary run.

Chart

Description automatically generated

Figure 15 Taylor plot of 20 GPCODE runs with SSE and OC4s

Chart

Description automatically generated

Figure 16 Taylor plot of GPCODE runs with MAPE and SSE and OC4s

(Case21 BS SSE, Case25 mape NoBS, Case37 mape BS dif2, case55-58; comparison with in situ data, bitmaps, Taylor plot ) ------to be removed later

**3.3 Resampling and Bootstrapping**

Bootstrap is a computer-intensive resampling technique that draws a large number of resamples from an initial dataset repeatedly with a computer. As computing power is increasing rapidly, computer-based statistical methods are more and more attractive. The purpose of applying bootstrapping is to reduce possible over-fitting in the GP algorithm because of limited training dataset in GPCODE run.

The figure 17 and 18 show the estimated Chlorophyll-a vs measured Chlor-a concentrations with bootstrapping. Figure 17 shows fitness with error metric of MAPE and figure 18 shows fitness with error metric of SSE. Figure 17 using MAPE error metric looks not very different from the estimates of no bootstrap GPCODE run. The GPCODE algorithm in figure 18 illustrate better skill in low Chla below 1. mg/m3 than its no bootstrapping one in figure 6. The following 2 figures 19 and 20 show their histograms of error variations with GPCODE run generations. We can see that the MAPE error is relatively smoother than the one with no bootstrap. Both errors reach almost same steady error level. The SSE error histogram in figure 20 shows a little more of saw teeth shape while the no bootstrapping error histogram shows a little step wise shape in figure 8. But its SSE errors stabilize at a higher value of around 40000, about 15000 higher than its no bootstrapping estimates.

Figures 21 and 22 present the entropies of GPCODE algorithm at its initial generation and last generation. It can be seen that the entropy in last generation is obviously shifted to a lower value than its entropy in initial generation.

Figures 23 and 24 show their corresponding tree representations of their algorithms. With or without bootstrapping the final algorithm is generally more comprehensive than its initial ones.

The figure 25 is the Taylor plot of estimates with two different error metrics. They are very close to each other in the graph and present better performance in correlation, RMSE and standard deviation than OC4.

Chart, scatter chart

Description automatically generated

Figure 17 GP Chla estimates vs Measured Chla with error metric of MAPE and bootstrap

![Chart, scatter chart

Description automatically generated]()

Figure18 GP Chla estimates vs Measured Chla with error metric of SSE and bootstrap

Graphical user interface

Description automatically generated with medium confidence

Figure 19. GPCODE estimate MAPE errors histogram with generations using bootstrap

Chart

Description automatically generated

Figure 20. GPCODE estimate SSE errors histogram with generations using bootstrap

Chart, histogram

Description automatically generated

Figure 21 GPCODE solutions entropy of 10th generation for SSE using bootstrap

Chart, box and whisker chart

Description automatically generated

Figure 22 GPCODE solutions entropy of 390th generation for SSE using bootstrap



Figure 23 Tree representation of best one of GPCODE solutions in initial population



Figure 24 Tree representation of best one of GPCODE solutions in population of last generation

Chart, radar chart

Description automatically generated

Figure 25 Taylor plot of GPCODE runs with MAPE and SSE and OC4s using bootstrap (56/58)

**3.4 Comparison of GPCODE Algorithms with other models**

The comparison of GPCODE algorithm estimate with other models like OC4 are made using real data. In many cases the GPCODE estimate is no less than OC4. See our cases in figure 13,14,21. They show better statistics in fitting real data. The GPCODE algorithms may not be always consistently fitting well for all Chla concentration. From the cases of our experiment a 500 or more of generations GPCODE run and a population of 1000 individuals or more will find a good enough algorithm to estimate Chla. OC4 generally showed less biased with in-situ observation.

**4. Summary**

Genetic programming is an artificial intelligent method which imitate the biological evolution by natural selection to generate proper programs/equation from hundreds or thousands of initial trees through genetic operations of Crossover, mutation etc. In the loop of generations each program of population is calculated against training data to find their fitness.

We carried out a series of experiments with GPCODE genetic programming software system aiming at assessing its performance and validate its estimation of Chlorophyll-a concentration from remote sensing reflectances.

Using SeaWiFS visible bands Rrs as training data, our study showed that under our setting of control variables GPCODE can generate algorithms to well match the real Chlor-a data. Under different error metrics (MAPE and SSE), GPCODE may generate somewhat different algorithms or estimates, but they always give consistently good correlations with in-situ Chlora data. The algorithms of different GPCODE runs may have different biases. In our experiments the algorithms with SSE as error metric seem to have less biases and MAPE estimates present larger biases. Comparisons of the GPCODE algorithms to OC4 algorithm show that the former in many cases presents higher correlation with real data in Chlor-a concentration. GPCODE algorithm with bootstrapping experiment shows it may produce more stable algorithms and reduce the over-fitting in the GP evolutionary computing.

Overall, the GPCODE experimental results are very encouraging. Further study should be conducted using other remote sensing training data and assess their performances in different waters. Also, it may be helpful to develop some new fitness for progressive computing of Chlorophyll-a.

**Acknowledgements**

The work is funded by.

**Appendix I**

|  |  |  |
| --- | --- | --- |
| **model DATA** | **# DATA** | **denotes that this is a GPCODE run to evolve a single equation/algorithm** |
| GA\_Crossover\_Probability | 0.8 | # Probability of "Tree Crossover" in the GA process |
| GA\_Mutation\_Probability | 0.05 | # Probability of "Tree Mutation" in the GA process |
| GA\_Rand\_Recruit\_Probability | 0.06 | # Probability of "Random Recruitment" in the GA process |
| GA\_save\_elites\_Probability | 0.09 | # Probability of saving the elite trees in the GA process |
| ga\_tournament\_style | 0 | # Use [1] vs no-use [0] of "Tournament Style" Section in the GA process |
| n\_GA\_Generations | 30 | # Total number of Generations in the GA process |
| n\_GA\_Individuals | 18 | # Total number of Individual in the GA process |
| n\_gp\_individuals | 3000 | # Total number of Individuals in the GP process |
| n\_gp\_generations | 200 | # Total number of Generations in the GP process |
| random\_scale\_small | 1.5 | # |
| random\_scale\_large | 1000.0 | # |
| random\_scale\_fraction | 0.5 | # |
| GP\_all\_summary 2 |  | # Output Flag: GP\_all\_summary ==> [0 = false; 1 dumps GP\_last\_gen\_summary\_file; 2 dumps both GP\_Gen\_Trees\_All & GP\_last\_gen\_summary\_file] |
| print\_equations 1 |  | # Output Flag: print equations [0 = false; 1 = true] |
| GP\_output\_parameters 1 |  | # Output Flag for GP\_output\_parameters file: [0 = false; 1 = true] |
| GP\_log 1 |  | # Output Flag for GP\_log file: [0 = false; 1 = true] |
| GPSSE\_log 1 |  | # Output Flag for GPSSE\_log file: [0 = false; 1 = true] |
| GP\_Tree\_Probability 1.0 |  | # set to 1.0 for single algorithms, and 0.5 for models. See code for explanation. |
| GP\_Elitist\_Probability | 0.00 | # Probability of keeping Elite (Best Tree) [These four values must add to 1.0] |
| GP\_Asexual\_Reproduction\_Probability | 0.05 | # Probability of Asexual Reproduction [These four values must add to 1.0] |
| **model DATA** | **# DATA** | **denotes that this is a GPCODE run to evolve a single equation/algorithm** |
| GP\_Crossover\_Probability | 0.90 | # Probability of Sexual Reproduction [These four values must add to 1.0] |
| GP\_Mutation\_Probability | 0.05 | # Probability of Mutation [These four values must add to 1.0] |
| number\_gp\_child\_prints 2 |  | # |
| n\_input\_vars 6 |  | # total number of input variables in the algorithm training dataset |
| n\_levels 11 |  | # high number of allowable levels in the trees |
| prob\_no\_elite 0.3 |  |  |
| selected\_function 1 |  | # Addition: = a + b [note: a = lhs of tree; b = rhs of tree] |
| selected\_function 2 |  | # Subtraction: a – b |
| selected\_function 3 |  | # Multiply: a \* b |
| selected\_function 4 |  | # Protected Divide: a / b, for b .ne. 0, else 0.0 |
| selected\_function 5 |  | # Ivlev Grazing Function: (1 - e^-abs(a\*b)) |
| selected\_function 6 |  | # Michaelis-Menton Term: (abs(b) / (abs(a) + abs(b))) |
| selected\_function 7 |  | # Mayzaud-Poulet Grazing Function: abs(a\*b)\*(1 - e^-abs(a\*b)) |
| selected\_function 8 |  | # Power: a ^ b |
| selected\_function 9 |  | # EXP: exp(-abs(a\*b)) |
| selected\_function 10 |  | # Minimum: min(a,b) |
| selected\_function 11 |  | # Maximum: max(a,b) |
| # selected\_function 12 |  | # IF a .ne. 0 THEN b ELSE 0 |
| # selected\_function 13 |  | # IF a .GT. b THEN 1 ELSE 0 |
| # selected\_function 14 |  | # IF a .GE. b THEN 1 ELSE 0 |
| # selected\_function 15 |  | # IF a .LT. b THEN 1 ELSE 0 |
| # selected\_function 16 |  | # IF a .LE. b THEN 1 ELSE 0 |
| **model DATA** | **# DATA** | **denotes that this is a GPCODE run to evolve a single equation/algorithm** |
| selected\_function 17 |  | # EXP\_LP: exp(a) |
| selected\_function 18 |  | # EXP\_RP: exp(b) |
| selected\_function 19 |  | # EXP\_LM: exp(-a) |
| selected\_function 20 |  | # EXP\_RM: exp(-b) |
| selected\_function 21 |  | # Log\_rhs: Log10(a)/Log10(b) |
| selected\_function 22 |  | # Mult\_1: a \* 1.0 |
| selected\_function 23 |  | # Square: a ^ 2 |
| USER\_INPUT\_RANDOM\_SEED 0 |  | # 1388171815 |
| n\_partitions 16 |  | # Total number of partitions |
| gp\_para\_lmdif 1 10 |  | # |
| eval 1 |  | # Error Metric Flag [0 ==> SSE 1 ==> MAPE 2 ==> MAE] |
| Bootstrap 50 |  | # Bootstrap Flag [0 ==> .false.' 1-50 ==> number of resampling+1] |
| replace\_larger\_sse\_only |  | # |
| caseOutput\_dir gpcode\_gen3kmape\_invBSE\_BS |  | restart 850964654 1726724523 |

**5 References**

[1] A Field Guide to Genetic Programming

Riccardo Poli, William B. Langdon, Nicholas F. McPhee with contributions by John R. Koza. March 2008

[2] Jeannie Fitzgerald, R. Muhammad Atif Azad, Conor Ryan, A Bootstrapping Approach to Reduce Over-fitting in Genetic Programming. GECCO’13 Computation. July 6-10, 2013, Amsterdam, The Netherlands

[3] C. Fonlupt. Solving the cean Color Problem Using a Genetic Programming Approach. Applied Soft Computing 1 (2001) 63-72

[4] Chih-Hua Chang, Development of Ocean Color Algorithms For Estimating Chlorophyll-a Concentrations and Inherent Optical Properties using Gene Expression Programming (GEP). 9 Mar 2015|Vol.23, No.5|DOI:10.1364/OE.23.005417|OPTICS EXPRESS 5417

[5] Ruixiang Sun, Fugee Tsung, Liangsheng Qu, Combining Bootstrap and Genetic Programming for Feature Discovery in Diesel Engine Diagnostics. International Journal of Industrial Engineering, 11(3), 273-281, 2004.

[6] Dimitris N. Politis, Computer-intensive methods in statistical analysis, IEEE Signal Proc. Mag. in Jan 1998, pp 39-55.

[7] Isaiah Lankham, Matthew Slaughter, Simple and Efficient Bootstrap Validation of Predictive Models Using SAS/STAT® Software, SAS Global Forum 2020, Paper 4647-2020

# [8] [Adam Marczyk](mailto:adam@ebonmusings.org), Genetic Algorithms and Evolutionary Computation. Copyright © 2004

[Posted: April 23, 2004]

[9] [David J.Lary](https://www.sciencedirect.com/science/article/pii/S1674987115000821" \l "!)[a](https://www.sciencedirect.com/science/article/pii/S1674987115000821" \l "!)[Amir H.Alavi](https://www.sciencedirect.com/science/article/pii/S1674987115000821" \l "!)[b](https://www.sciencedirect.com/science/article/pii/S1674987115000821" \l "!)[Amir H.Gandomi](https://www.sciencedirect.com/science/article/pii/S1674987115000821" \l "!)[c](https://www.sciencedirect.com/science/article/pii/S1674987115000821" \l "!)[Annette L.Walker](https://www.sciencedirect.com/science/article/pii/S1674987115000821" \l "!)[d](https://www.sciencedirect.com/science/article/pii/S1674987115000821" \l "!), Machine learning in geosciences and remote sensing [**Geoscience Frontiers**](https://www.sciencedirect.com/journal/geoscience-frontiers), [Volume 7, Issue 1](https://www.sciencedirect.com/journal/geoscience-frontiers/vol/7/issue/1), January 2016, Pages 3-10. <https://doi.org/10.1016/j.gsf.2015.07.003>

[10] Li Chen, Keh-Chia Yeh, Hsiao-Ping Wei, Gin-Rong Liu, An improved genetic programming to SSM/I estimationtyphoon precipitation over ocean. Hydrological Processes Hydrol. Process. 25, 2573 – 2583 (2011)Published online 10 May 2011 in Wiley Online Library (wileyonlinelibrary.com) DOI: 10.1002/hyp.8132

[11] Bradley Efron, Robert J. Tibshirani, An Introduction to Bootstrap, ©Springer Science+Business Media Dordrecht 1993

[12] Helmuth, T., N. F. McPhee, E. Pantridge, and L. Spector. Improving Generalization of Evolved Programs through Automatic Simplification. In GECCO '17: Proceedings of the 2017 Genetic and Evolutionary Computation Conference. July 2017. ACM

[13] A. Shintemirov, W. Tang, and Q. H. Wu, Power Transformer Fault Classification Based

on Dissolved Gas Analysis by Implementing Bootstrap and Genetic Programming. IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C: APPLICATIONS AND REVIEWS, VOL. 39, NO. 1, JANUARY 2009, p69-p79

[14] John R. Koza, Human-competitive results produced by genetic programming. Genet Program Evolvable Mach (2010) 11:251–284. DOI 10.1007/s10710-010-9112-3

[15] P. Jeremy Werdell, Bryan A. Franz, Sean W. Bailey, Gene C. Feldman, Emmanuel Boss, Vittorio E. Brando, Mark Dowell, Takafumi Hirata, Samantha J. Lavender, ZhongPing Lee, Hubert Loisel, Stéphane Maritorena, Fréderic Mélin, Timothy S. Moore,Timothy J. Smyth, David Antoine, Emmanuel Devred, Odile Hembise Fanton d’Andon, and Antoine Mangin. Generalized ocean color inversion model for retrieving marine inherent optical properties. APPLIED OPTICS / Vol. 52, No. 10 / 1 April 2013

[16] NASA Ocean Color, Airborne and Field Data Workshop on March 29th - 30th, 2022 from 11am - 5pm ET

[17] W. B. Langdon, Genetic programming convergence. Genetic Programming and Evolvable Machines, https://doi.org/10.1007/s10710-021-09405-9