Proceedings of the Fifth Workshop on
Mathematical Modelling
of Environmental and Life Sciences Problems
Constanţa, Romania, September, 2006, pp. 35–42

Meteorological Data Analysis and Prediction by Means of Genetic Programming

Andrei Băutu[*♮] and **Elena Băutu**[**♮]

Weather systems use extremely complex combinations of mathematical tools for analysis and forecasting. Unfortunately, due to phenomena in the world climate, such as the greenhouse effect, classical models may become inadequate mostly because they lack adaptation. Therefore, the weather prediction problem is suited for heuristic approaches, such as Evolutionary Algorithms. Experimentation with heuristic methods like Genetic Programming (GP) can lead to the development of new insights or promising models that can be fine tuned with more focused techniques. This paper describes a GP approach for analysis and prediction of data and provides experimental results of the afore mentioned method on real-world meteorological time series.

1. Introduction

Since the beginning of mankind, people have always been interested in weather and have tried to find various methods that would allow them to know the weather in advance. These methods evolved considerably from the time of Admiral Robert Fitzroy (one of the pioneers of weather forecasting). Today, state of the art weather systems use satellite cameras that can zoom into local areas, Doppler radar that uses sound waves, and real-time computer-based data analysis. However, most serious climatologists nowadays agree with Robert A. Heinlein's quote: "climate is what you expect, weather is what you get". This quote expresses very well the close relationship, and also the difference, between climate and weather.

Weather is a description of natural conditions over a short period of time, a "snapshot" of the atmosphere at a particular time. In one form or another, weather

[*] "Mircea cel Bătrân" Naval Academy, Constanţa, Romania, e-mail: `abautu@anmb.ro`
[**] "Ovidius" University of Constanţa, Romania, e-mail: `ebautu@univ-ovidius.ro`
[♮] the paper was supported by CNMP CEEX 05-D11-25/2005 Grant.

affects our activities, because it affects the main ingredients of life on Earth: water and heat. To ensure our livelihood, planners need to anticipate variation in weather.

Climate is the statistics of weather over a long period of time, i.e. a synthesis of the weather recorded for a specified window of time at a particular place. It offers valuable information about the average conditions, extremes, or frequencies of events. Although climate is not weather, it is defined by the same terms, such as temperature, precipitation, wind, and solar radiation. Many businesses use climate and weather data to make informed economic decisions.

Climate varies from one place to another (spatial variation), and, more important, it varies from season-to-season, year-to-year, decade-to-decade, and so on (temporal variation), due to the natural climate variability and the human-caused (anthropogenic) climate change. Differences in the timing, intensity and duration of seasons can have a huge impact on the environment and people, as climate change impact assessments concerning agriculture, forests, water resources, etc. Results from general circulation models usually provide neither the most likely scenario nor the full range of possible outcomes.

The main artificial intelligence methods for weather prediction currently in use include model output statistics, fuzzy logic, and expert systems. Some research has focused on using genetic algorithms for various aspects of weather prediction.

2. Genetic Programming

In his famous 1950 paper, Turing suggested one of three directions for artificial intelligence research should be automatic program design by means of an evolutionary inspired approach. He was the first to envision evolutionary techniques that evolve computer programs for problem solving. In Turing's approach, an initial solution is transformed over time using mutations; each mutation is judged by a human expert who decides which mutations are accepted based of their effects. Turing never implemented his technique, but his belief was that programs could learn intelligence from their human judges.

Among the first implementations of Turing's ideas was Holland's Genetic Algorithm (GA) [4][2]. GA maintains a set of candidate solutions and evolves them through time. Holland proposed that the merits of a solution be automatically evaluated by a fitness function, instead of the human expert. The evolution process involves a kind of natural selection and genetically inspired operators of mutation and crossover. The chances of each individual to survive into the next generation are proportionate to its fitness.

In the early 1990s, Koza [5] introduced a variation of GA closer to Turing's idea of evolutionary computation, called Genetic Programming (GP) [6][7]. GP is an evolutionary algorithm that uses a functional encoding of solutions—initially, the candidate solutions were encoded by Lisp S-expressions. The individuals are computer programs encoded as syntax trees with varying size and shape. The internal nodes of the tree are labeled with functional symbols. The number of child branches of each node matches the arity of the function that labels it. Nodes on the frontier of the tree are labeled with constants or variables (they are considered to be 0-arity nodes).

The first step of the algorithm involves the creation of the initial population. There are many initialization procedures described in the literature. We use the Ramped Half-and-Half initialization procedure introduced by Koza in [5]. With this procedure, an equal number of individuals are initialized for various depth levels. For each depth level considered, half of the individuals are perfectly balanced trees with all the branches of the same length, and the other half are unbalanced trees, possibly with some branches much longer than others. The population of trees resulting from this initialization method is very diverse, with balanced and unbalanced trees of several different depths. The inner nodes of the trees are closed forms of classical mathematical operations (addition, subtraction, multiplication, division, exponential, logarithm, etc), whereas the terminal nodes are constants or references to independent variables and values in the past of the dependent variables.

After the initialization process, a cyclic process begins, with individuals evolving due to genetic operators and a selection scheme. The genetic operators maintain diversity in the population of individuals by combining features of existing the individuals and/or adding new features. The most important GP operators are tree-crossover and tree-mutation. In tree crossover, random nodes are chosen from both parent trees, and the respective branches are swapped creating two offspring. There is no bias toward choosing internal or terminal nodes as the crossing sites. In tree mutation, a random node is chosen from the parent tree and substituted by a new random tree created with the terminals and functions available.

Each genetic operator is applied in the population with some probability. We use an automatic adaptation procedure of these probabilities based on [1]. The algorithm uses a FIFO-like repository to maintain information regarding each child produced, like its parents, the operator used, its relative fitness. Each child receives a credit value based on this information, and a percentage of this credit is attributed to its ancestors. From time to time, the performance of each genetic operator is calculated as the average credits of the individuals from the repository created by that operator. The probability of operators that have been performing badly will drop, while the probability of operators that have been performing well will raise. The probability of operators that haven't been able to produce any children since the last adaptation will increase.

Although we do not impose any restriction on the shapes of the individuals, we do restrict the size the individuals may have. In this context, size can refer to the depth or to the number of nodes of the tree encoded by the individual. These restrictions are meant to avoid bloat, a phenomenon consisting of an excessive code growth without the corresponding improvement in fitness. The standard way of avoiding bloat is by setting a maximum size on individuals being evolved. An individual that exceeds the maximum size may be dropped or may be "shaved" to the allowed size. We use a mixed approach of hard and soft size limits. An individual that breaks the hard limit is dropped and one of its parents is used instead. An individual that breaks the soft limit is accepted only if its fitness is greater than the fitness of the best individual found so far. The soft limit is initially set to a low value, and it is updated every time a better individual is found.

After the application of genetic operators, the algorithm continues with the evaluation phase. The function encoded by each individual in the population is evaluated on the fitness cases set. For each input set, the result of the function is compared with the expected output of that fitness case and the fitness of the individual is adjusted according to the results. In this paper, the fitness of an individual is computed as the sum of the absolute difference between the expected output value and the value returned by the individual, over all fitness cases:

$$fitness = \sum_{c \in C} |expected_c - computed_c| \tag{1}$$

The best individuals are the ones that return better approximations of the expected values—the ones with a lower fitness.

After evaluation, the algorithm uses a selection mechanism to choose the individuals that will survive in the next generation. We use a selection mechanism that implements lexicographic parsimony pressure, a technique that has shown to effectively control bloat in different types of problems [8]. This method chooses each parent by randomly drawing a number of individuals from the population and selecting only the best of them. An individual A is better than another individual B if the fitness of A is lower than the fitness of B, or fitness of A is equal to the the fitness of B and A has less nodes than B.

The evolution-evaluation-selection cycle is repeated until a stopping criterion is met. We use a mixed stopping criterion: the algorithm stops when a maximum number of generations have been reached, or when an acceptable individual was found. The solution designated by a run of the GP algorithm is the best individual throughout the generations.

3. Experiments

Our experiments were carried out in Matlab 6 with GPLab 2 and MeteoLab toolboxes. GPLab is a highly customizable toolbox for genetic programming in Matlab created by Sara Silva. Although it provides the basic means for a quick start with genetic programming, it also has a plug and play architecture that allows scientists to fine-tune every aspect of the GP algorithm. It provides us with all the tools we need for our experiments (automatic adaption of operator probabilities, hard and soft limits for the sizes of the individuals, lexicographic parsimony tournament selection, etc). GPLab also offers real-time plots for the statistics on the state of the algorithm (population diversity and complexity, fitness evolution, operators probabilities, etc).

MeteoLab was developed by Antonio S. Cofińo, Rafael Cano, Carmen Sordo, Cristina Primo, and José Manuel Gutiérrez as a companion software for their book on weather prediction[3]. It is a collection of numerical weather prediction (NWP) algorithms, raw data of weather observations (precipitation, pressure and temperature observations) for european stations from the Global Climate Observing System (GCOS) Surface Network (GSN), and information of reanalysis projects. MeteoLab also includes algorithms for generation of simulation data and a routines for the representation of climate data.

Table 1. GPLab parameters used in experiments

Parameter	Value
Population size	100
Number of generations	50
Terminals set	$[0, 1) \cup \{\text{history values}\}$
Functions set	$\{+, -, \times, \text{div}, \sin, \cos, \log, \exp\}$
Hard size limit	7
Soft size initial limit	4
Initialization size limit	7

For our experiments we set up the GPLab parameters as presented in table 1. The first experiment uses a data set of temperature observations collected every 10 days between January 1st and December 31st, 1999, in Rennes, France. We compared our GP-based approach with the AR model implementation of Matlab and with results obtained by the neural networks toolbox Netlab. For this data set, the arfit function from the ARFit Matlab toolbox automatically selected an order of the model of 2. In order to compare the GP results with the ARFit results, both GPLab and NetLab were setup to use a history window of size 2.

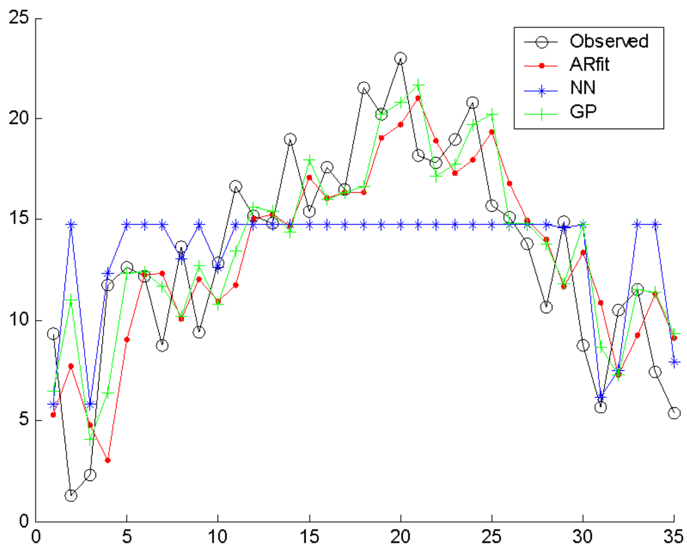

Fig. 1. Results for Rennes data set.

Figure 1 contains the plots of the models found by ARfit, Netlab, and GPLab. In this case, the mean error of the AR model is 2.9731, and the mean error of Netlab's solution is 3.3289, while the mean error of the GP model is 2.5873.

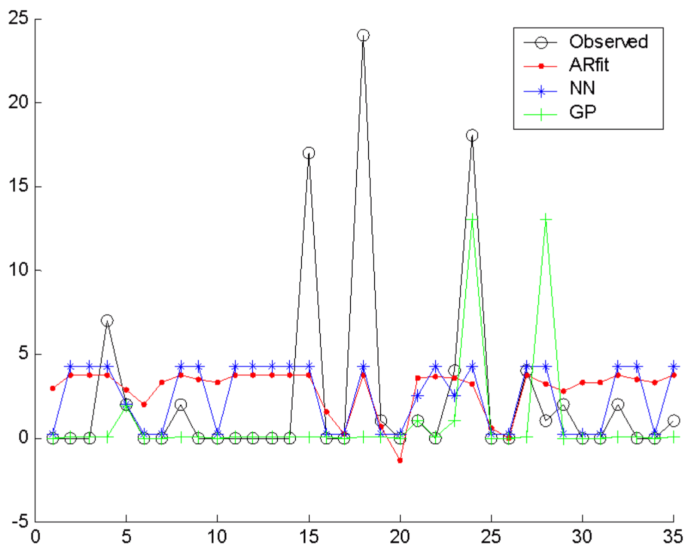

Fig. 2. Results for Ostersund Froson data set.

The next experiment presents the results for precipitation analysis in Ostersund Froson, Sweden, during 2001, within a similar experimental setup as the previous experiment. In this case, the observed data has many null values (corresponding to dry days), with occasionally high peaks, making this a difficult data set.

Figure 2, which contains the plots of the models found by ARfit, Netlab, and GPLab, clearly shows the high quality of the GP model. In this case, the mean error of the AR model is 3.5215, and the mean error of Netlab's solution is 3.0390, while the mean error of the GP model is 2.2928.

4. Conclusions

This paper shows that genetic programming can outperform classical and modern methods (based on artificial intelligence) for the discovery of models in weather data. Moreover, the self adaptability of GP allows it to work in an unsupervised manner and to discover relations hidden inside the data set. However, GP is not an adversary of the other approaches, as models found with GP can be further optimized with other techniques.

References

[1] L. Davis, *Adapting operator probabilities in genetic algorithms*, In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69, 1989.

[2] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[3] J.M. Gutiérrez, R. Cano, A.S. Cofińo and C. Sordo. *Redes Probabilsticas y Neuronales en las Ciencias Atmosféricas*, Monografas del Instituto Nacional de Meteorologa, Madrid, 2004.

[4] J.H. Holland, *Adaption in Natural and Artificial Systems*, MIT Press, Cambridge, MA, 1992.

[5] J.R. Koza, *Genetic programming: On The Programming of Computers by Means of Natural Selection*, Cambridge MA, MIT Press, 1992.

[6] W.B. Langdon, *Data Structures and Genetic Programming: Genetic Programming + Data Structures = Automatic Programming!* Kluwer, Boston, 1998.

[7] W.B. Langdon and R. Poli, *Foundations of Genetic Programming*, Springer-Verlag, New York, 2002.

[8] S. Luke and L. Panait, *Lexicographic parsimony pressure*, In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 829–836, New York, 9-13 July 2002. Morgan Kaufmann Publishers.