Advanced Formula Prediction using Simulated Annealing

Namir Aldawoodi
PhD candidate / Computer Engineer.
University of South Florida
Tampa, FL 33620

Dr. Rafael Perez
Professor & Graduate Program Director
Computer Science Dept.
University of South Florida
Tampa, FL 33620

Abstract

An improved heuristic technique based on a method that we developed in 2003 [1]. The original method was called Formula Prediction using Genetic Algorithms (FPEG) and that method presented an algorithm by which formulas could be generated directly from datasets. The method presented enhances the power and flexibility of the original model resulting a better formula generation tool.

The enhanced method differs from the original by formula structure and expressive power – it has a variable formula structure, a larger alphabet and new independent internal variables. The latter allow for better data pattern recognition and better overall performance. The method presented here uses simulated annealing to generate mathematical equations that fit a set of input data to a function. Data is represented as a set of input and output values collected from a system under consideration. For significantly large numbers of independent variables in the input set, this problem can be intractable and, as such, NP hard. When Advanced Formula Prediction using Simulated Annealing (AFP) was compared against FPEG using the original benchmarks - the results obtained demonstrated that the new algorithm is able to better the original algorithm's performance by 2.65 percentage points on average – or – an average reduction of the error margin by 52.10 percent (which is statistically significant). To keep the comparison valid, the same regression benchmarks were used. In addition, a technique that encodes strings that represent the candidate formulas during the search was enhanced to give it more expressive power.

1 INTRODUCTION

Many data fit problems can be solved using traditional statistical methods. However, there are data sets that cannot be adequately mapped or analyzed using statistical techniques. Specifically of interest, are problems where data is known but the function that generated the data is not. In such cases, one approach is to search for patterns in the data to find clues about the function that generated the data in the first place. The generated or hypothesis function is then tested to see how well it fits the data from which it was derived. If some information is available as to the form of the function (a sin wave, for example), then regression analysis techniques can usually be applied successfully to the problem of generating a function based on observed data.

Still, regression techniques, whether linear or nonlinear, have limitations especially in problems where little or no information is available as to the shape or form of the function that generated data. In these cases, researchers typically use neural networks or fuzzy sets to learn more about the shape of the domain space of the function under consideration [7]. Once that information is collected, regression techniques can then be re-applied. However, there is still no guarantee that regression techniques will be able to generate a function – this can lead to a circular case where there is the need to constantly generate new models and use those to select mathematical functions to conduct regression analysis and when that does not yield results then the cycle starts over with the selection of another model. If there was a tool that can generate functions along with their respective performance values then, that tool can be a time saver when it comes to selecting mathematical models to use in regression analysis.

2 LEARNING

An effective function prediction algorithm must be able to detect or learn patterns hidden in data in order to generate a valid approximation function. Learning can be defined as the ability to predict future values given past experiences or, better yet, the ability to generalize. One way to assess how well an algorithm has learned a given problem is by measuring the rate of error. The less error in future predictions, the better learned the algorithm – and better learning means better performance. Learning also depends on the quality of the data presented. So a good learning algorithm may do poorly if presented with an incomplete data set.

In order to successfully classify data there are some considerations that must be addressed; mainly, whether the data is a representative sample and in this case representative means analyzing the data to determine if it is relevant to making predictions. Also, the error inherent in the data set must be considered (how noisy is the data?). Other considerations relate to how well the dataset is distributed. In addition to the data, the domain from which the data was collected must also be studied. Are there any assumptions that must be made about the domain that would influence how the data is perceived by a learning algorithm looking for patterns? One such concern is a discontinuous function that has no defined or infinite output between some range of input values. This can throw off a pattern detection algorithm. Also, what is the most useful way to represent the output? Options include decision trees, function, or some other representation method.

The error level is also important, a researcher must determine ahead of time the acceptable level of error at which point the simulated function is considered valid-enough for the application at hand. Certainly, building an airplane has little error tolerance but predicting the weather is another matter – what if an algorithm was right 60% of the time while an existing radar-based weather algorithm had a performance of 50%? In this case the new algorithm would be better –even with a 40% error margin.

There are many ways to learn; indeed the type of learning that an algorithm would have to do should be identified. Learning based on past knowledge falls under the category of supervised learning. This category of learning assumes that observed data values and their corresponding output values are provided in the beginning of the search [8].

3 LEARNING CATEGORIES

Generally, there are three categories of learning; they can be classified by the output provided to the learner.

3.1 SUPERVISED LEARNING

In this type of learning the algorithm is supplied with a set of training inputs and their corresponding outputs. Examples of this type of learning include classification and regression. There are three sub-types of supervised learning.

3.1.1 Classification Learning

Expected output is a Boolean: true/false. Hence the output is discrete and not continuous. This type of learning is referred to as a binary classification problem.

3.1.2 Regression (function learning)

In regression learning, the range of the problem is real numbers the output is continuous (real values).

3.1.3 Preference Learning

In this type of learning the range of the problem is an ordered space. Where two objects are compared and if they are not equal then one of them is selected (Permutation).

3.2 UNSUPERVISED LEARNING

In unsupervised learning, the aim is to learn patterns inherent in the data and to learn about the clustering of the data. Density estimation and reinforcement learning are examples of this type of learning. The feedback given indicates how well the algorithm is doing. However, the correct output values are not provided, only training inputs. The goal of this type of learning is to understand the process that generated the data. This approach has applications in data compression and classification.

3.3 REINFORCEMENT LEARNING

In reinforcement learning, the goal is to produce actions that change the state of the world around the algorithm. A good example is a chess game where the goal is to maximize favorable moves and minimize unfavorable moves (those that involve loss of pieces or strategic advantage). The correct output is given to the algorithm after a decision (move) has been made.

4 OTHER SEARCH CONSIDERATIONS

4.1 BATCH/ONLINE LEARNING

Another area of importance when learning deals with the way data is supplied to the algorithm. There are two general classifications. The first is batch learning, where all data is supplied to the learning algorithm at once. The second is on-line leaning, where the algorithm is supplied with one example at a time.

4.2 DESIGNING AN EFFECTIVE SEARCH STRATEGY

There are limitations to classical programming techniques, for example, it is not practical to write a program to do character recognition using classical if-then-else control structures; however, it is possible to use a neural network to accomplish that task. On the other hand, neural networks would be a waste of time if they were used to solve linear equations for example. Thus, well-defined number crunching problems are probably best suited for classical programming techniques (if-then-else) [7].

But, in areas where the solution is not defined and it is not known what form that solution should be in, then, algorithms that can extrapolate a solution directly from data have an advantage. When the answer is not known then the search should be as broad as possible, and if assumptions were made, then the algorithm could be limited to considering solutions that were pre-programmed for ahead of time. Hence, a character recognition program using a classical if-then-else approach is not impossible to write; however, it would be difficult to write an effective one, because the program must make assumptions ahead of time as to what patterns constitute a certain letter or number – this can be especially tricky if the letters are handwritten by different people.

So when it comes to deriving a solution that is essentially a guess, some approaches tend to be more practical than others. We considered decision trees, genetic algorithms, regression splines and clustering as possible methods to base AFP on.

The main goal of the method presented is to derive functions with as little bias as possible. Arguably, any method used will have some built in bias that may or may not be unique to it. Also, some bias is needed, otherwise the solution space would become impossibly large to search and the problem would not be much different than searching for every single possible combination that exists. Hence, neither random search nor exhaustive search is desirable. Random search does not have a time guarantee on convergence and exhaustive search takes far too long to be practical.

One of the considerations in developing this method was the need to limit bias when producing a function. Another consideration is that the function should have configurable components or building blocks – so that the search can be biased in a beneficial manner. In order to reduce bias, the selection of mathematical primitives is left up to the researcher using the algorithm. While this approach may hinder search in the sense that the algorithm can only look for answers within that mathematical function set, it would also make search more orderly because the search is now focused on a limited area but not locked into that area and if the algorithm does not make progress with that mathematical set then the researcher could make changes to the mathematical primitives – this ability to readily tune the algorithm makes it flexible and powerful.

Keeping these requirements in mind, it would be difficult and cumbersome to back-solve a neural network into a function and then translate that function into the desired output format. We also considered decision trees because they are relatively fast (faster than version spaces) for a large concept space and disjunction is easier to carry out; however, they were not considered flexible enough to produce the function formats required and it would be too complex to adapt them for use in building formulas that fit a set of data the way we inted to do in AFP.

In addition, a decision tree may not always expain its classification clearly. Another possible method is statistical analysis (regression based approaches) – however, these do not do well when the form of the function that generated the data is not known and there is limited flexibility as to the mathematical primitives that can be used.

On the other hand, simulated annealing and genetic search techniques were a good fit for many reasons; first, the way they search is not mathematically based (no direct calculations on the data). This means that discontinuities, noise and inconsistencies in the data would have little effect on the search algorithm [9]. Second, they search for a solution independently of what the data looks like. Hence, there is no significant bias in relation to how the data may be distributed and the algorithm is free to look for any pattern hidden within that data set. These search strategies are also resistant to getting stuck in local maximas – a big advantage when searching for patterns within a dataset [9]. As such, the algorithm was implemented using simulated annealing to look for a set of mathematical primitives that, when combined, would result in a function that maps the input data to the output data with as little error as possible.

4.3 THE SEARCH DOMAIN

The search domain can be thought of as some unknown system where the inner workings of the system are not known but the input and output values are measurable. Collecting data from such a system would require monitoring the output while inputting a range of values. These values should be selected carefully so as to represent the range of input values that the researcher is interested in modeling. Such a system is demonstrated in figure 1. For sufficiently large number of variables this problem can be NP hard.

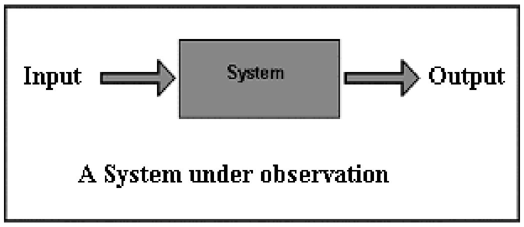


Figure 1: Collecting data from a system.

A search of the literature found no method that generates formulas the way that we propose; however, it is worth noting that there is a class of algorithms that generate formulas based on frequency response. A good example is the Comprehensive Identification from Frequency Responses system or CIFER. NASA developed this software package to model aircraft and it works by measuring the frequency response of a system under study and building a mathematical description of the system.

CIFER works much like reverse-simulation. Simulation requires that assumptions be made ahead of time (a-priori) to allow for the derivation of equations related to a given system and system identification starts with measured vehicle motion and measured responses in order to develop a model that reflects the measured data accurately. CIFER allows designers to skip modeling and get straight to an equation that reflects the data collected from a real system. The method proposed differs from CIFER because its mode of search looks for combinations of mathematical primitives that model the data set using input data. The technique is flexible and since it is based on simulated annealing, it is tolerant to noise and incomplete data sets [12] while CIFER requires that the data be as accurate as possible.

5 THE UPDATED METHOD

The updated method utilizes simulated annealing to generate a function from a set of data. Like FPEG, this method (AFP) is also classified as a supervised regression learning approach that uses batch data. This paper investigates and quantifies the ability of the enhanced simulated annealing search method to find formulas that describe the relationship between a set of observed input and related output data with little or no knowledge of the problem domain and to search with better accuracy than the original FPEG model.

Simulated Annealing search techniques were selected because they offer advantages when searching large, complex domains, the type that we expect to encounter when searching for unknown functions. Simulated annealing is very good at avoiding local maximas [12] [10]. It has been proven as a successful algorithm that is used in many real applications [11]. There has been much research done using Simulated Annealing for large problems with good results [9]. Although the optimality of simulated annealing is not formally defined, the algorithm guarantees that a global maximum will be found if temperature is annealed infinitely slowly [9][10]. Also, simulated annealing is not hindered by discontinuities in solutions expressed by mathematical formulas since it directly manipulate a string representing those formulas.

This updated method can be used to refine and validate functions generated from data. There are advantages and disadvantages to using the approach proposed. It does well in areas where regression can get bogged down. It is also useful in filtering important variables from a large set of independent variables [5].

In applications that require the generation of a meta model the method that we propose is very useful in generating meta-models; this is because commercially available statistical packages have drawbacks, Davis and Bigelow state in a paper published in 2002: "..off the shelf statistical packages are not very useful when there is little knowledge as to the models internal workings." The paper goes on to point out that meta models are useful because they can be used to gain insight as to the inner workings of a large, complex model, and they offer exploratory analysis [6].

If the analysis of the model was over a large part of its domain then a meta model with a limited number of variables – under 15 instead of hundreds makes exploratory analysis more feasible (Davis and Bigelow, 2002) [6].

The function-estimation algorithm can also be used to validate assumptions made about a given problem domain. Hence, a tool whose results can be compared with regression analysis is useful, particularly in validating a model that regression analysis generated.

Thus, the proposed method would search for patterns in data and generate an estimator function. Performance is also important – both in terms of the time it takes to search and in the accuracy of the prediction. The way a search is conducted also matters because it has an impact on how quickly the search converges. The method proposed does not use statistical techniques; rather, it uses simulated annealing because it is likely find patterns in the data that regression may miss due to bias built into the regression methodology itself [12].

Disadvantages in using genetic algorithms and simulated annealing as a basis for generating answers lie in the relative lack of precision [3]. Hence, this approach is not the most efficient approach to use when the domain is well defined and the quality of the data is very good. In these problems types, where the domain is well defined, regression analysis would be an ideal method.

5.1 LIMITATIONS OF REGRESSION

Linear regression has a fixed form as the far the returned formula. Also, linear and nonlinear regression will both return the same answer every time unless a new formula model is used; there is little flexibility once the formula is selected. The method we propose will return a different combination every time. The use of standard statistical methods with a problem that has a large number of variables can be expensive in terms of computation. The parsimony principle states that some factors are more important than others and actually contribute more to the final answer (Myers and Montgomery 1995, Kleijnen 1987). Thus, screening algorithms attempt to find the variables that have most impact on response (Trocine and Mallone, 2000) [5]. Once these variables have been identified then they can be isolated from the rest of the dataset and used as a part of a smaller dataset in statistical analysis.

Other drawbacks of statistical regression techniques are: (1) highly parametric structure and (2) weighty assumptions within a small-data setting. There are some problem classes in which there is no statistical method that can be applied affectively to derive an answer. One such problem was proposed by Ratner [2] where he sites the example of a direct marketing problem that seeks to maximize the response rate of solicitation by identifying customers that are most likely to respond based on collected data. Ratner concludes that there is no standard statistical method that addresses this type of problem adequately.

Linear statistical models make strong assumptions about the structure of data, which often do not hold in applications. The method of least squares is very sensitive to the structure of the data, and can be markedly influenced by one or a few unusual observations. We could abandon linear models and least-squares estimation in favor of non-parametric regression and robust estimation. Unusual data are problematic in linear models that fit by using the least squares method because they can unduly influence the results of the analysis, and because their presence may be a signal that the model fails to capture important characteristics of the data.

5.2 ADVANTAGES OF UPDATED METHOD

The original FPEG algorithm was based on genetic search and simulated annealing to assemble a formula out of an alphabet of primitives supplied by the programmer. The newer method relies primarily on simulated annealing. However, genetic search can still be used as a backup search-method should simulated annealing fail to reach the results expected. So far, experiments that the authors conducted show that that both search approaches return almost identical results – with few exceptions. As such, in order to speed up the search – the genetic algorithm based search is only conducted if simulated annealing fails to find a function with the desired performance.

The algorithm tends to return a unique combination of mathematical primitives each time it is ran. Problems with a large number of near optimal maximas tend to return more diverse functions.

The proposed method does not claim to out-perform regression analysis in every category. Rather, it is presented as a powerful tool to that can perform as well as regression analysis (as indicated by our experimental results) in some areas and, possibly better, in areas where regression analysis has inherent weaknesses. This tool can be used to analyze virtually any system where data can be collected. It also offers alternate answers that can be used and compared to functions or solutions obtained using other methods, case in point: if a complex, well trained, neural network was developed to solve a given problem; AFP can be applied to derive an equivalent function that can be used to gain insight as to the internal operation of the neural network, This is important because mathematical primitives can be plotted and analyzed - there is almost no ambiguity in a mathematical formula – however, a neural network or a fuzzy logic set can be a challenge to analyze mathematically.

Another consideration was the calculation of error. There are many error calculation methods available (as well as the option of defining a new error calculation method). For example, many regression approaches use mean squared error (MSE) to measure how well a generated function ‘fits’ or deviates from the original data set. Another measure of error is the mean absolute percentage error (MAPE). Naturally, the method chosen, can, in a subtle way, affect overall results. One error method may show a lower error reading over another (depending on data and the type of problem). However, early on we decided to select a straightforward error calculation method and use that consistently.

The method chosen is a variation of MAPE. As far as the authors are concerned, there is no error calculation method that has clear advantages over another error calculation method. As one author summed it up: “Data always have some detail, which can trick a method into losing its theoretical advantage. A method may be theoretically preferred over an alternative method for a given problem, but empirical results do not reflect its advantage.”

6 THE ALGORITHM

The new AFP algorithm is designed to return a solution form with the desired mathematical primitives.

6.1 PROBLEM DEFENITION

The problem requires the generation of a function that maps input data to output data from a system under observation. The generated function must meet a minimum (pre-selected) accuracy level or find the best solution within a maximum temperature setting. The fitness measure is calculated for every function that the algorithm generated and that measure is used to indicate how well the generated function approximates the observed data. The algorithm stops searching once the generated function maps the data under the pre-defined minimum error or when the temperature reaches 0.

6.2 DATA FORMAT

Given a system under observation, inputs to that system will be represented by the vector S and observed outputs of the system will be represented by the vector T. Figure 2 demonstrates a system with an observed input vector Si, and an observed output vector Ti. S and T cannot be empty.

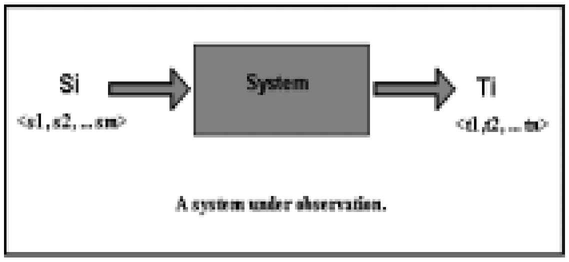


Figure 2: Inputs and outputs to a system

Each input vector **S** must have a corresponding output vector **T**. When collecting data from a system, a collection of input vectors is entered and a collection of output vectors is observed. The algorithm would take the **S** and **T** vectors and produce some function **F** that relates all the input vectors (**S1, S2 S3, … , SN)** to their respective output vectors (**T1, T2, T3, … , TN)** such that***: Tij = Fj(Si())***

where: 1 <= j <= n. and n is the size of **T**.

The average error rate e is calculated as shown in figure 3.

$$\text{Error} = \frac{\prod_{i=1}^{N} \left(\frac{\text{ABS} \left| F(Si()) - Tij \right| * 100}{\text{ABS} \left| Tij \right|} \right)}{N}$$

m = Number of elements in S where 1 =< j =< n
n = Number of elements in T

N = Number of input/output sets ABS = Absolute Value

Figure 3: Error Calculation

Note that the error calculation is applied to one output value at a time. Since **T** holds all the output values for a system, the algorithm produces a separate formula for each element of **T**. The output under consideration (represented by i) is selected from each output vector **T** and considered.

6.3 THE SOLUTION FORM

The algorithm generates formulas using the format shown below. A function with m input variables generated by AFP (Advance Formula Prediction Using Simulated Annealing) has the form:

(*pr)(mul)(op(s1)) comb (pr)(mul)(op(s2)) comb (pr)(mul)(op(s3)) …*

comb (pr)(mul)(op(sm)) dOp Delta

where s1, s2, s3, … , sm are members of the input vector **S**.

The rest of the symbols are as follows:

pr	= Property operator
mul	= Multiplier operator
op	= Operation operator
Comb	= Combination operator
dOp	= Delta operator
Delta	= Delta value

This format is flexible and allows for fine-tuning of the returned function. The algorithm returns a solution that contains the same number of variables as the size of the input vector **S**. AFP takes the members of the input vector **S** and applies to each member the operation, multiplier and property operators. The property operators are variables that may hold any of a number of different mathematical primitives. The values for the operators are derived from sets that will be referred to as the alphabet sets. Once the first set of operators is applied, the resulting (modified) variables are combined using the combination operator. When all these calculations are completed, a delta value is added or subtracted to the resulting sum. The final result is referred to as the calculated value. This technique searches within all the possible combinations of the above operators to find the best combination, that when applied to the input values in **S**, will yield a calculated value that has as little error as possible when compared to the actual value. If the algorithm were to search every combination then the problem would be unmanageable for large alphabets and/or large input sets. Genetic and simulated annealing search methodologies excel in searching large problem domains for combinations that produce the best fit [4].

6.4 AFP RELATED ENHANCEMENTS

While the basic operator structure of FPEG and AFP are similar, there are enhancements that make the new algorithm more effective and accurate. The first enhancement is a layers variable. This variable has a range of 1 through 10. The new variable indicates how many times the basic formula format (from section 6.3) should be repeated. This allows for each variable in the input to appear more than once in the resulting formula with (possibly) different functions applied to it. When the 'layers' variable is set to 1, then each input value in the data appears once and only once in the resulting equation. This means that if a given variable in the input data can influence the range of the problem in more than one way (or dimension), it would have been difficult for FPEG to account for that. However, in AFP, each input variable can appear up to 10 times in the resulting equation.

The second important enhancement to FPEG that AFP makes is through the use of three new random real variables (called companion variables) that are associated with each input variable in S. The companion variables offer a further dimension to making the basic alphabet set more powerful. They are used with the operation operator to derive an almost infinite number of possible calculations from one mathematical primitive. The original FPEG used a limited set of integer values to modify operation operators (x squared for example – the modifier is the 2 – that makes the power operator square the input number). AFP does not use a limited set of values to modify operators; rather, it uses the companion variables to achieve a wide range of values. Thus, the previous example of x to the second power can become x to the value contained in a companion variable.

The third set of enhancement pertains to a ‘time function’. This function is internal to AFP and can be used as a time dimension with the input set S. Thus, each pair of input and output value is assigned a unique time stamp value by the time function. This allows FPEG to search for time-dependent patterns within the data and combine those with the existing patters detected. This adds a time dimension to AFP and makes it sensitive to pattern changes in time – which FPEG never considered.

6.5 OPERATOR DETAILS

The algorithm is flexible because the formula can be adjusted and changed as needed. Also the alphabet sets can contain any mathematical primitive or function and, as such, the length of the sets themselves can be varied. In the next section we will discuss the alphabet values that were chosen for this paper.

6.5.1 The property operator (pr)

This operator is one unit in size. It acts on the multiplier operator and modifies the values that it holds as follows:

Encoding	Action taken
w	1 * (multiplier value)
x	-1 * (multiplier value)
y	1 / (multiplier value)
z	-1 / (multiplier value)

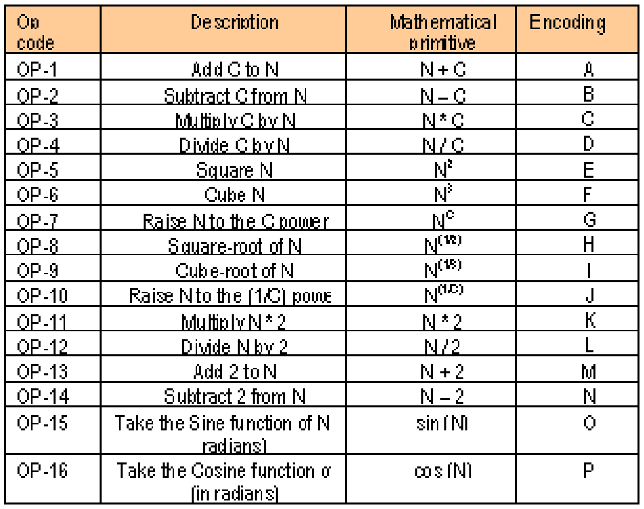
Table 1: The property operator

6.5.2 The Multiplier Operator (mul)

The multiplier operator is 3 units in length; it holds a value that ranges from 000 to 999. The property operator modifies the value within this operator.

6.5.3 The Operation Operator (op)

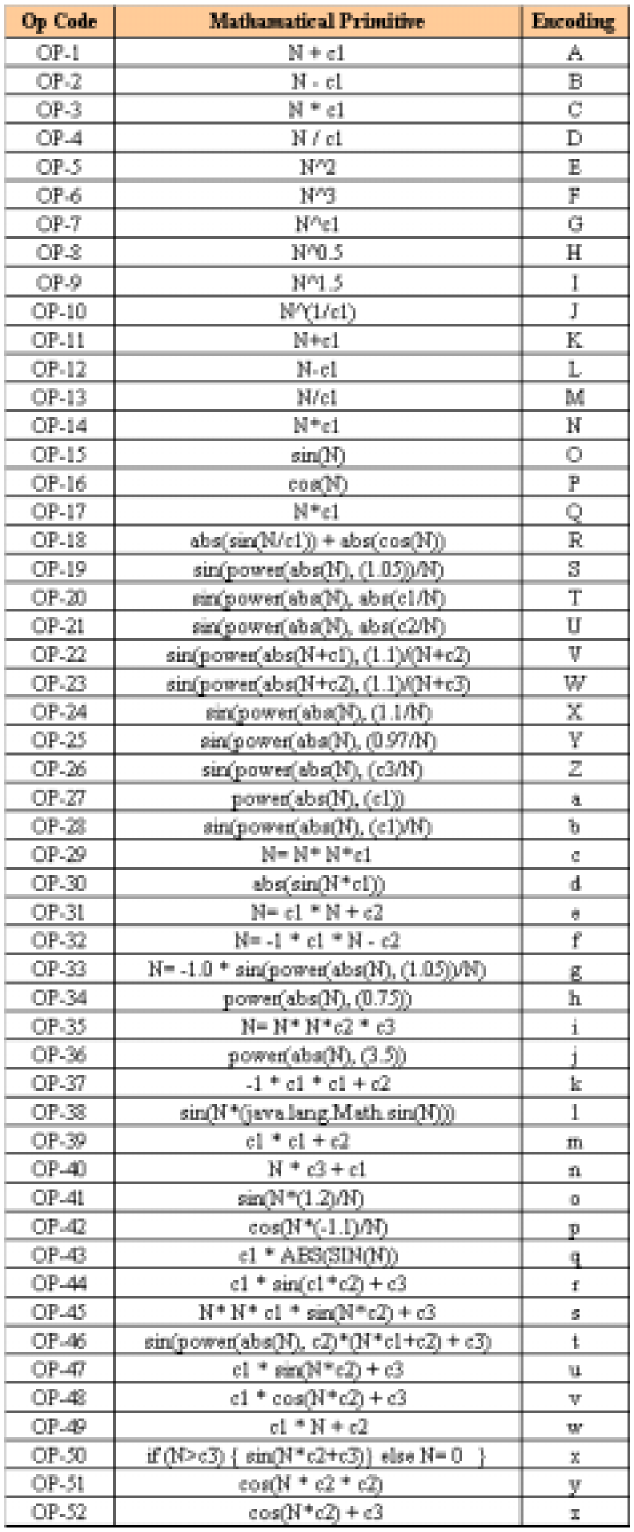
In FPEG this operator holds a character value that ranges from the letter ‘a’ to ‘p’. Even though it is one unit in length, it represents one of 16 possible operations that can be applied to a value. There is a variable, denoted by the letter C that is used to ‘tune’ or adjust the effect of the operation operator. The original FPEG algorithm used a C value of 4. Table 2 summarizes this operator in FPEG:



Op code	Description	Mathematical primitive	Encoding
OP-1	Add C to N	N + C	A
OP-2	Subtract C from N	N – C	B
OP-3	Multiply C by N	N * C	C
OP-4	Divide C by N	N / C	D
OP-5	Square N	N^2	E
OP-6	Cube N	N^3	F
OP-7	Raise N to the C power	N^C	G
OP-8	Square-root of N	$N^{(1/2)}$	H
OP-9	Cube-root of N	$N^{(1/3)}$	I
OP-10	Raise N to the (1/C) powe	$N^{(1/C)}$	J
OP-11	Multiply N * 2	N * 2	K
OP-12	Divide N by 2	N / 2	L
OP-13	Add 2 to N	N + 2	M
OP-14	Subtract 2 from N	N – 2	N
OP-15	Take the Sine function of N radians)	sin (N)	O
OP-16	Take the Cosine function o (in radians)	cos (N)	P

Table 2: The FPEG operation operator

In AFP, changes were made to Table 2. The first change was the expansion of the Op code from 16 to 52 combinations. This was achieved using the alphabet set ‘A-Z’ and ‘a-z’. The C variable was eliminated. This was no longer necessary since each input variables in S had 3 companion variables (the companion variables also undergo perturbation along with the solution form). The new AFP form of table 2 can be seen in table 3. In table 3, N represents the value of an input variable in S. The companion variables are represented as c1, c2 and c3. It important to note that each variable in the input set S has its own unique set of companion variables. In the table the symbol ‘power’ represents raising the first item between the parenthesis (i1) to the power of the second listed item (i2). An example would be: power (i1, i2). The symbol ‘abs’ means absolute value of the expression within the parenthesis. The mathematical primitives listed in table 3 are by no means a fixed set – this is one possible combination that worked in testing the datasets. Through experimentation the values of this table are adjusted and changed to find the best combination that yields the lowest error.



Op Code	Mathamatical Primitive	Encoding
OP-1	N + c1	A
OP-2	N - c1	B
OP-3	N * c1	C
OP-4	N / c1	D
OP-5	N^2	E
OP-6	N^3	F
OP-7	N^c1	G
OP-8	N^0.5	H
OP-9	N^1.5	I
OP-10	N^(1/c1)	J
OP-11	N+c1	K
OP-12	N-c1	L
OP-13	N/c1	M
OP-14	N*c1	N
OP-15	sin(N)	O
OP-16	cos(N)	P
OP-17	N*c1	Q
OP-18	abs(sin(N/c1)) + abs(cos(N))	R
OP-19	sin(power(abs(N), (1.05))/N)	S
OP-20	sin(power(abs(N), abs(c1/N)	T
OP-21	sin(power(abs(N), abs(c2/N)	U
OP-22	sin(power(abs(N+c1), (1.1)/(N+c2)	V
OP-23	sin(power(abs(N+c2), (1.1)/(N+c3)	W
OP-24	sin(power(abs(N), (1.1/N)	X
OP-25	sin(power(abs(N), (0.97/N)	Y
OP-26	sin(power(abs(N), (c3/N)	Z
OP-27	power(abs(N), (c1))	a
OP-28	sin(power(abs(N), (c1)/N)	b
OP-29	N= N* N*c1	c
OP-30	abs(sin(N*c1))	d
OP-31	N= c1 * N + c2	e
OP-32	N= -1 * c1 * N - c2	f
OP-33	N= -1.0 * sin(power(abs(N), (1.05))/N)	g
OP-34	power(abs(N), (0.75))	h
OP-35	N= N* N*c2 * c3	i
OP-36	power(abs(N), (3.5))	j
OP-37	-1 * c1 * c1 + c2	k
OP-38	sin(N*(java.lang.Math.sin(N)))	l
OP-39	c1 * c1 + c2	m
OP-40	N * c3 + c1	n
OP-41	sin(N*(1.2)/N)	o
OP-42	cos(N*(-1.1)/N)	p
OP-43	c1 * ABS(SIN(N))	q
OP-44	c1 * sin(c1*c2) + c3	r
OP-45	N* N* c1 * sin(N*c2) + c3	s
OP-46	sin(power(abs(N), c2)*(N*c1+c2) + c3)	t
OP-47	c1 * sin(N*c2) + c3	u
OP-48	c1 * cos(N*c2) + c3	v
OP-49	c1 * N + c2	w
OP-50	if (N>c3) { sin(N*c2+c3)} else N= 0 }	x
OP-51	cos(N * c2 * c2)	y
OP-52	cos(N*c2) + c3	z

Table 3: The AFP operation operators

6.5.4 The Combination Operator

This operator is one unit in length. It indicates how to combine two variables A and B.

6.5.5 The Delta Operator

This operator indicates what to do with the delta-value. It specifies one of two possible actions: addition or subtraction.

6.5.6 The Delta Value

This operator has been changed to hold a value that ranges from 0.01 to 9.99. It is added or subtracted from the resulting equation depending on the delta operator. This operator is three positions in length.

6.6 CALCULATING THE FITNESS VALUE

The application of these operators to the vector **S** of input variables transforms these variables into a resulting calculated value. This resulting value is compared with the original output value stored in the corresponding result set **T**. As shown in figure 3 earlier, for some data pair **Si():Tij**, the error value is calculated by comparing the calculated value with the actual value. The error value shows how close the function built by AFP comes to approximating the function under study. The fitness value is defined as the average of all the error values for a given output. The error calculated is essentially a percentage difference between the resultant values calculated by the generated function and the supplied (correct) values; hence, the fitness value will vary from 0% error (a perfect match) to very large numbers. Naturally, the lower the error, the better the fitness of that function. The fitness values are expressed in percentile numbers to make it easier to assess how closely the function generated by AFP approximates the function that produced the data set.

6.7 IMPLEMENTATION DETAILS

The algorithm returns a formula that has the same number of variables as there are input variables (input columns) when the 'layers' variable is set to 1. If the 'layers' variable is set to two then the returned number of variables is 2x the input number of variables (each variable is represented twice) and likewise for each value of the 'layers' variable - thus, a layers=10 setting will produce a resultant function with each input variable m represented 10 times (10m). Thus AFP offers a way to configure the number of variables in the resulting function. Next, the algorithm calculates the character string length needed to encode the input data into a function and then it generates a random population (in the case of simulated annealing a single random starting point is chosen). During the search process, a fitness function evaluates the current string (or population of strings). The fitness value is essentially the error rate. The search concludes when the temperature (Simulated Annealing) has reached a terminating state. At that time a function is returned along with three performance parameters: the best, worst and average error values for that function. The best performance value represents the best fit where the retuned value of the function comes very close to the original value in the data set. Likewise, the worst performance value indicates the point where the generated function deviates the most from the data set. The average value indicates the performance of the generated function averaged over the entire data set.

7 TESTING METHODOLOGY

7.1 FUNCTIONS AND SEARCH SPACES

The aim of this algorithm is to derive functions from a given set of data. As we did in FPEG, testing AFP was done after it was tuned using basic algebraic formulas that were assembled for testing purposes. The functions used in tuning the algorithm fell into two classes: the first we refer to as 'In-Alphabet' functions, these are derived from the alphabet sets and use mathematical primitives that already exist in our sets. The second class of functions is referred to as 'Out-of-alphabet' functions; these are equations whose operator variable uses elements that are not in the alphabet sets. Out-of-alphabet functions imply that the operation operator cannot encode the operation in the original function because it is not in the current alphabet. However, there could be a combination of other operators that can yield a close approximation. The second part of the test involved applying the AFP algorithm to the same statistical benchmark sets that were used to test the original FPEG algorithm. These were obtained from the National Institute of Standards and Technology. These benchmark sets were designed to test commercial nonlinear regression software and they were rated by difficulty level. Tests were conducted to evaluate how well AFP evaluates these sets.

7.2 TEST RESULTS

Table 4 below shows the benchmark name, the difficulty level (lower indicates easier to solve), the Class (Exponential or Miscellaneous), the number of parameters, the number of observations, the source (Observed or Generated) and the performance of FPEG and AFP.

Dataset	**Level**	**Class**	**Para (ba)**	**Num Obsv**	**Source**	**FPEG Perf**	**AFP Perf**
Misrala	Lower	Exp	2	14	Obsrvd	4.20%	1.38%
Misralb	Lower	Misc	2	14	Obsrvd	3.59%	2.51%
Nelson	Avg	Exp	3	128	Obsrvd	7.20%	3.29%
Gauss3	Avg	Exp	8	250	Gen	1.88%	0.77%
Rat43	Higher	Exp	2	6	Obsrvd	8.50%	4.75%
BoxBod	Higher	Exp	4	15	Obsrvd	6.59%	1.82%

Table 4: Test results for benchmarks

A 1.38% would indicate an average error rate of about 2% or a 98.62 accuracy rate on average for that function. Next, a comparison was made between AFP and FPEG; the aim was to calculate the improvement that AFP offered over FPEG. The percentage error reduction was compared for AFP. The results are shown in table 5.

Accuracy (FPEG)	**Accuracy (AFP)**	**Improvement AFP over FPEG**	**% Reduction of Error Margin**
95.8	98.62	2.82	67.14
96.41	97.49	1.08	30.08
92.8	96.71	3.91	54.31
98.12	99.23	1.11	59.04
91.5	95.25	3.75	44.12
93.41	98.18	4.77	72.38

Table 5: Improvement AFP vs FPEG

To get an overall idea as to how well AFP does when compared with FPEG, we averaged all the data from table 5 and calculated the average accuracy for AFP and FPEG. We also calculated the average improvement and average reduction of error margin. The results are shown in table 6.

Avarage Accuracy (FPEG)	**Avarage Accuracy (AFP)**	**Avarage Improvement AFPG over FPEG**	**Avarage % Reduction of Error Margin**
94.92	97.57	2.65	52.10

Table 6: Average improvement AFP vs FPEG

8 CONCLUSIONS

A simulated annealing-based algorithm to predict mathematical formulas from observed data was successfully constructed and tested. This new method was based on the FPEG algorithm that was developed by the authors in 2003. The AFP algorithm test data demonstrate that it is capable of generating results that achieve a better accuracy level (or fit) than the original FPEG algorithm. There are many more data sets that this tool could be applied to including market analysis. We have demonstrated an improved tool that can be a valuable aid in research, this tool does not replace regression analysis; however, it offers a unique method that can be used to compare how well a given regression analysis was carried out by comparing results from that analysis with results from AFP. Yet, there are many other uses for this tool, such as in cases where regression analysis fails to yield good results, then this tool can be used an backup method. This tool may offer an alternative to CIFER. Future work may involve benchmarking AFP against CIFER. This application was designed to be used in areas of research where there are a large number of parameters and where the relation between the data is not well understood. The results obtained so far are very promising.

References

[1] N. Aldawoodi, R. Perez (2003), Formula Prediction Using Genetic Algorithms (FPEG), University of South Florida, GECCO A.I. Conference.

[2] B. Ratner (2000). A Comparison of two popular Machine Learning Methods, Mine Tech.

[3] L. Davis, K. DeJong, M. Vose, D. Whitley, (1999), Evolutionary Algorithms, Springer.

[4] T. Back, (1996), Evolutionary Algorithms in Theory and Practice, Oxford University Press.

[5] L. Trocine, L. Malone, (2000), Finding Important Independent Variables Through Screening Designs: A Comparison of Methods, Proceeding of the 2000 Winter Simulation Conference.

[6] P. Davis, J. Bigelow, (2002), Motivated Metamodels, RAND Graduate School.

[7] Q. Wang, T. Aoyama (2001), A Neural Network Solver for Differential Equations, Miyazaki University, Japan.

[8] T. Mitchell, (1997), Machine Learning, WBC/McGraw-Hill.

[9] Lawrence Davis, (1990), Genetic Algorithms and Simulated Annealing, Pitman, London.

[10] P. J. M. van Laarhoven and E. H. L. Aarts, (1987), Simulated Annealing Theory and Applications.

[11] Shaharuddin Salleh, Albert Y. Zomaya, (1999), Scheduling in Parallel Computing Systems – Fuzzy and Annealing Techniques.

[12] D.T. Pham and D. Karaboga, (2000), Intelligenet Optimisation Techniques.