Genetic Algorithm for Developing Program Using Backus Naur Form (GAPBNF)

Rasha Shaker Abdul Wahab*

Received on: 3/5/2004
Accepted on:12/7/2005

Abstract

GAPBNF, genotype-phenotype genetic algorithms, is presented here as a technique for the creation of computer program. This technique uses new representation to generate the program which can be executed using any language. In this case the chromosome (genotype) is a list of integer representing production in syntax, which is used to generate the program (phenotype). In order to explain the effectiveness of using this type of representation, GAPBNF is implemented for solving symbolic regression problem. The results show that GAPBNF is an efficient and promising method that can be used to generate any program in any language.

الخلاصة

يتناول هذا البحث عرض GAPBNF وهي احدى تقنيات البرمجة الجينية الحديثة. في هذه التقنية تم استخدام تمثيل جديد لعناصر فضاء البحث . في هذه الطريقة يتم التمييز بين عناصر فضاء البحث وعناصر فضاء الحل. عناصر فضاء البحث والذي هو عبارة عن محتويات المجتمع الجيني عبارة عن سلسلة من الاعداد الصحيحة التي تمثل أرقام الترتيب القواعدي المستخدم والذي يتم اختياره حسب نوع البرنامج المنتج. تم اختيار الترتيب القواعدي للغة باسكال وتطبيق هذه الطريقة على حل مشكلة توليد التعابير الرياضية. لقد أظهرت النتائج الكفاءة في ايجاد الحلول الجبرية بالاضافة إلى التنبوء بمقدرتها في توليد برامج أخرى باستخدام ترتيب قواعدي مختلف.

1. Introduction

Evolutionary algorithms (EA) have been used with much success for automatic generation of program. In many researches, that are concerned with generating a computer program automatically, use lisp language as their target language to represent the program and because lisp s_expression causes some problems such as memory problem and slow execution [1,2],therefore most recent researchers train to use a new direction for representing the program and to guide the EA to generate other languages by using the grammar to the target language.

Evolutionary algorithms area has been populated with numerous different approaches to generate programs with other languages; the following paragraphs represent some of these approaches:

Honer introduced a system called genetic program kernel (GPK) which is similar to standard GP, employs trees to code genes .Each tree is a derivation tree made up from the BNF definition. However, GPK has been criticized for the difficulty associated with generating the first generation (effort must be put into ensuring that all the entire tree represents valid sequences) [2].

Paterson developed an approach to generate C program directly. This

* Dept. of Computer Science, University of Technology

https://doi.org/10.30684/etj.24.6.14

approach uses fixed length chromosome that encodes which production rules are to be applied. But this approach suffers from number of draw backs. In particular as the number of productions grows, the chance of any particular production being chosen by a gene reduces [6].

In this paper, we describe a different technique that uses BNF (Backus Naur Form) definitions and evolves the chromosomes using some constraint. The developed system can be used to produce program in any language. In addition the developed approach adopts the idea of distinction between the genotype and the phenotype (program generated).

This approach is based on the same principle of (GADS) technique [6], and solving the problems that suffering from it. So our main contribution is what we call GAPBNF (Genetic Algorithm for developing Program using Backus Naur Form).

2. Backus Naur Form [2]

Backus Naur Form (BNF) is a notation for expression the grammar of a language in the form of production rules. BNF grammars consist of terminals, which are items that can appear in the language, i.e., +, - ...etc, and non-terminal which can be expanded into one or more terminal and non-terminals. A grammar can be represented by the tuple {N, T, P, S} where N is the set of nonterminals, T the set of terminals, P a set of production rules that map the elements of N to T and S is a start symbol which is a member of N .For example, below, is a possible BNF for a simple expression ,where :

N= {sexp, op, pre_op,var}

T= {+,-, %,*, X,Sin,Cos}

S=<sexp>

And P can be represented as:

```
<sexp>::= <sexp><op><sexp>   (1)
         |<pre_op> (<sexp>)  (2)
         | <var>             (3)
<op>::= +                    (1)
       | -                   (2)
       | /                   (3)
       | *                   (4)
<pre_op>:= Sin               (1)
          | Cos              (2)
<var>::= X                   (1)
```

3. Principle of GABHFP

GAPBNF, a genotype-phenotype genetic algorithm, is a new technique for the creation of computer program.

GAPBNF genotype is a list of integers representing production number of the syntax language used to generate computer program. These numbers are used to generate the phenotype.

As explained above, GAPBNF is a genetic algorithm (like GAs and GP), as it uses population of individuals, select them according to fitness, and introduce genetic operators, so that GAPBNF can be implemented using the traditional genetic algorithm principles and it is developed in order to improve traditional method.

The important differences between GAs, GP and GAPBNF reside in the nature of the individuals in which:

1. GAs individuals are nonlinear strings of fixed length (chromosomes).
2. GP individuals are nonlinear entities of different sizes and shapes (parse trees)
3. GAPBNF individuals are encoded as linear strings of variable length (the genotype of chromosomes) which are afterwards expressed as nonlinear

entities of different sizes and shapes (expression tree).

GAPBNF is the aggregation of different proposals for automatic program generation which are:

1. The first proposal being the change in the representation of the genotype (chromosome).
2. The length of chromosomes will be variable.
3. The storage space is reduced.
4. Evaluating programs using any language.

In the following subsection, the most important components used with GAPBNF will be described.

3-1 Initial Population

In this method, the generation of gene in the genotype simply depends on some constraints. If a1, a2, a3 .....aL is the genotype, the selection of a gene a2, for example, is not done randomly. This gene is generated based on the cases of the previous gene or a1 in this case. Therefore, for doing this operation some constraint is used which is represented by the rewriting rules of the CFG (Context Free Grammar) [7].

In GAPBNF, the genotype consists of a linear integer string of variable length of genes .In this method, it will be shown that despite this type of constraints, GAPBNF genotype can reduce the storage space, so in this case there is no loss in the generated genes ,and the losses will be after the program builds (phenotype), i.e., the tail genes.

3-2 The Ontogenic Mapping and Generating the Phenotype

The chromosomes (genotype) will be passes into a suitable generator after their generation. This generator output the program which represents the phenotype. The mapping process from the genotype to phenotype is called the ontogenic mapping.

The ontogenic mapping is represented in this method as an aggregation of a different number of rules that helps to generate the program (phenotype), these rules represent the production rules of the syntax language, therefore the principles of the BNF definition are used to generate the associated phenotype.

The implementation of the ontogenic mapping is done by writing the syntax of the phenotype language as a set of productions of the form: lhs: =rhs, where lhs is one non-terminal symbol of the language and rhs is a concatenation of zero or more symbols (terminate or non-terminal of the language) for example:

$$\langle s\exp\rangle := \langle s\exp\rangle + \langle s\exp\rangle \mid \langle s\exp\rangle - \langle s\exp\rangle$$

The use of | symbol is to indicate an alternative in the rhs, it is a shorthand for several productions which have the same rhs [6].

In GAPBNF the productions are numbered from 0 to n and each production may have several concatenation symbols on the right hand side, therefore in this case the number of each one of the concatenation symbol is appended with the number of the production, for example if the following production has a number 1, then:

$$\langle s\exp\rangle := \langle s\exp\rangle + \langle s\exp\rangle \mid \qquad (10)$$
$$\langle s\exp\rangle - \langle s\exp\rangle \qquad (11)$$

This production has two of the possibilities on right hand side, so the probability of the gene that can be showed based on this production may be 10 or 11.

Even well-formed program in a language has a derivation according to the syntax of that language. A derivation is a sequence of productions which when applied in turn; transform the language's start symbol into the program [6].

Thus any program can be represented by a derivation, and any derivation can be represented by the sequence of integer which corresponds to the production in its derivation [6]. So any program can be represented as a sequence of integer number in the range [0,n] and each range can be represented as an integer number in the range [0,p], where p represents the number of concatenation symbol on the right hand side of each production and n represents the number of productions. The evolutionary design of GAPBNT system is represented in figure (1).

The phenotype may be generated as strings or abstract syntax trees [6]. In our work we use a string which is ordered as a postfix expression (postfix ordering occurs when the operator follows its operands). For example, the following postfix expression [8]: wx*yz+*sqrt is equivalent to the following expression: $\sqrt{}(w*x)*(y+z)$.

One advantage of using postfix expression is that, if one evaluates the postfix expression from left to right, we will always have evaluated the operands to each operate before it is necessary to process the operator. Also in postfix expression no parentheses are needed in postfix notation [5, 8].

In order to evaluate the generated phenotype that has been represented as a postfix expression, we use stack-based implementation which is an easy way to evaluate the postfix expression. Stack-based implementa_ tions in general do not require postfix expression to represent a syntactically correct parse tree. This is a potential advantage of the stack-based approach since the other interpreters which we considered require the syntax to be maintained [8].

At the beginning of the process of converting the genotype to phenotype, an initial value is needed. The initial value of the genotype is usually the start symbol of the language symbol. In more general terms, the initial value may be any partially generated program such as <sexp> which is presented in the production exits in section 2.

4. Experimental Design

This section describes an experiment to test the GAPBNF technique by solving a symbolic regression problem.

4-1 The Problem

Symbolic regression is the process of discovering both the functional form of a target function and all of its necessary coefficients, or at least an approximation to this .symbolic regression is a distinct form, since in other forms of regression such as polynomial regression, we are merely trying to find the coefficient of a polynomial of a pre_specified order. In this case, the system will be provided with a set of input and output pairs and it must determine the function that maps one to another.

That is, we are given a sample of data in the form of 20 pairs (X_i,Y_i), where X_i is a value of the independent variable in the interval [-1,+1] and Yi is the associated value of the dependent variable. The 20 values of X_i were chosen at random in the interval [-1, +1]. For example, these 20 pairs (X_i,Y_i) might include pairs such as (-0.40,-0.2784),(+0.25, +0.3320) and (+0.50,+0.934). These 20 pairs (X_i,Y_i) are the fitness cases (fitness cases are typically only a small finite sample of entire domain space which is usually very large or infinite) that will be used to evaluate the fitness of any individual. The goal is to find a function in symbolic form that is close or perfect to fit the 20 pairs of numerical data points. The solution to this problem is to find a function in symbolic form; this function is viewed as a search for mathematical expression in a space of possible solutions [4].

4-2 The Phenotype Language Syntax

As mentioned earlier, to convert the genotype to phenotype the ontogenic mapping is needed.

The syntax rule (BNF) that has been used for this problem is presented in table (1).

In the beginning, we identify rule number (10) as the initial starting symbol which will be used to generate the phenotype. After that we use the implementation of the stack-based in order to obtain the objective value of this genotype (chromosome) and after that discarding the generated phenotype.

To complete the BNF definition for a Pascal, we need to include the following rule for symbolic regression problem:

```
<Header>:= program symbolic;
var x: real
Begin   f: = <sexp>   End.
```

We used in this case a limited rule, this is because the nature of the problem, and the system can easily generate program in any form that is done by changing the above rule.

4-3 Evaluation Function

After the GAPBNF begins to convert each genotype to phenotype, the fitness value is computed. This value is used to control the application of the operations that modify the population. The fitness function that has been used in this paper is computed as follows [4]:

$$r(X) = \sum_{j=1}^{N_c} \left| s_j - c_j \right| \text{............(1)}$$

$$Fit(X) = 1/1 + r(X) \text{...........(2)}$$

where Nc is the number of fitness cases, s_j is the returned value by an individual program for fitness j and c_j is the correct value for fitness case j.

The next step is to select the best chromosome, if the selected chromosome is considered the correct program then stop otherwise perform the genetic operators on the population genotypes). These opera_ tions are repeated until the required program is found.

4-4 The Initial Population

Each chromosome in the population have a variable length, and in the generated process, each gene in the associated chromosome is based on some constraints which are illustrated by using the rewriting rules of the CFG (context free grammar). To build the grammar, which shows the constraint, we need to assign each production rule number with a specified symbol as follows (based on the production defined in table (1)):

$10 \rightarrow A, 11 \rightarrow B, 20 \rightarrow C,$
$21 \rightarrow D, 30 \rightarrow E, 31 \rightarrow F,$
$32 \rightarrow G, 33 \rightarrow H.$

Thus the CFG used in this problem is shown in figure (2).

4-5 Experimental Condition

Table (2) presents the information needed to solve the problem using GAPBNF. Table (3) shows the experimental constants and variable conditions.

4-6Genetic Operators

The brood recombination operation has been used in this work. This method was devised by Tackett [8]. Tackett attempted to model the observed fact that many animal species produce far more offspring than are expected to live. Although there are many different mechanisms, the excess offspring die. This is a hard but effective way to cull out the result of bad crossover. Tackett created a "brood" each time the crossover was performed. One of the key parameters of his system was a parameter called brood size NB. Figure (3) shows the creation and evaluation of brood where NB=4 [8]. The probability of the crossover and mutation is 1.0 and 0.35 respectively.

Mutation operator involves by selecting a gene randomly from a genotype, generating another gene randomly and replacing the selected gene.

5. The Result

5.1 The Performance

Since GP is a probabilistic algorithm, not all runs are successful in yielding a solution to the problem by generation i. When a particular run of GP is not successful after running the pre-specified maximum number of generations, G, there is no way to find out whether or not the run would ever be successful. Thus, there is no knowable value for the number of generations that will yield a solution.
We can compute the probability of success that the particular run with a population of size M yields the solution by generation i.

Figure (4) presents the performance curves of GAPBNF .The curve is based on 100 independent runs. This curve shows the probability of success of solving the problem by generation i.

5.2 Example

Experiment 1: GAPBNF discovers the target function, where the length of the individual (the genotype) is a shorter length GAPBNF, in one run the individual or target expression is found at generation 3 with the length equal to 26 and contains around 13 symbols. The genotype of that individual is as follows:

10 33 10 30 11 20 10 33 11 20 11 20 10 30 10 32 11 20 11 20 10 33 11 20

the corresponding phenotype is presented as follows (postfix expression).

xxx+%xx*xx+

After converting the postfix expression in to infix order, the corresponding program generated is:

```
Program symbolic;
var x:  real
Begin
f=((x*x)=x)*((x%x)+(x*x))
End.
```

Experiment 2: In another run we found the best individual at generation 10, the genotype and the corresponding phenotype is presented as follows:

10 30 11 10 33 11 20 10 30 11 20 10 33 11 20 10 30 11 20 10 33 11 20 11 20 11 20

the phenotype

*xxx+x*x+x*x+

6. Conclusions

We have described GAPBNF technique that can map an integer number genotype into phenotype (which is a high level program). Because our mapping technique employs a BNF definition, the system is language independent and can generate complex functions using any language that is wanted to generat the program.

In addition, this technique has many features that distinguish from others:

1. Variable length string may be used to represent each chromosome.
2. In the GAPBNF, the wasted genes are reduced which leads to reducing the storage space.
3. Postfix expression is used to represent the phenotype to the associate genotype, so stack-based implementation is used to generate the phenotype; this type of implementation reduces the time needed to obtain objective value.

Also the GAPBNF technique gives good and correct solution after a small number of generations, and all the encountered problems in GADS technique have been solved.

Reference

1. Allen Kent, and James G. Williams, "Encyclopedia of Computer Science and Technology", In Koza, John R(Ed), Genetic programming, NY: Marcel-Dekker, Volume 39, supplement 24, 1998.
2. Conor Ryan, JJ Collins, Michael O. Neil. "Grammatical Evolution: Evolving Programs for an Arbitrary Language", in Wolfgang Banzhaf, Riccardo Poli, Marc Scheonauer, and Terence C. Fogarty (Eds.), Lecture Notes in Computer Science, on GP, Springer-Verlag London, UK , vol. 1391, pp 83-96. 1998
3. Herbert Schildt. The Complete Reference, Borland C++, McGraw-Hill, 1997.
4. Koza, J. R.Genetic Programming: On the Programming of Computer By means of natural selection, MIT Press, 1992.
5. Kennth E.Kinnear ," Genetic Programming in C++: implementation issues", In Kenneth E. (Eds), Advance in Genetic Programming ,pp 285-310, MIT Press ,1994.
6. Norman R. Paterson, Mike L., "Distinguishing Genotype and Phenotype in Genetic Programming", in Koza, J.R. (Ed.), Late Breaking Papers at the Genetic Programming 1996 Conf. Stanford Univ., pp 141-150. 1996
7. Thomas Haynes, Wainwright R., Sen S., Schoenefeld D, "Strongly Typed Genetic Programming in Evolving Cooperation Strategies", in Eshelman, L.J. (Ed.), Proc. of the

sixth Int. Conf. on GA, Morgan Kaufmann, pp.271-378. 1995.

8. Wolfgang B., Peter N., Robert E. Keller, Frank D. Francone, Genetic Programming, An Introduction on the Automatic Evolution of Computer Programs and its Applications, Morgan Kaufmann and dpunkt. Verlag, 1998.

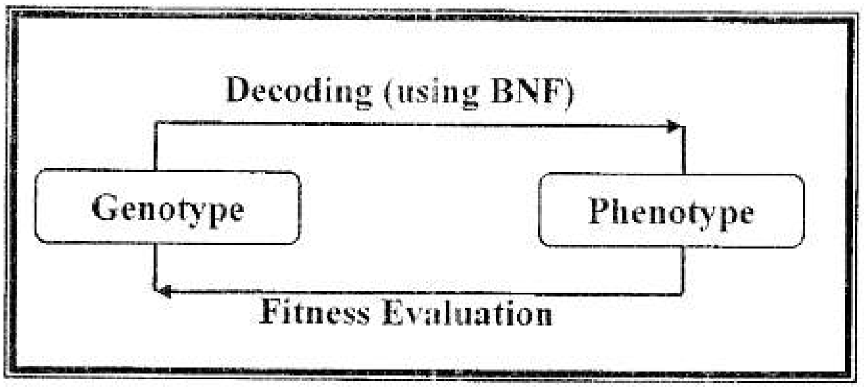

Figure (1) Evolutionary design of GADS

Table (1) The Syntax Rule

Syntax	Rule number	Production number
<sexp> := <application> | <input>	10 11	1
< input> := X | 1	20 21	2
< application> := < sexp>+<sexp> | <sexp>- <sexp> | <exp>% <sexp> | <sexp>* <sexp>	30 31 32 33	3

S->AS1
S1->S3| ES2 |FS2| GS2| H2|λ
S2->AS2 | BS2
S3->C | D

Figure (2) The CFG of the genotypes

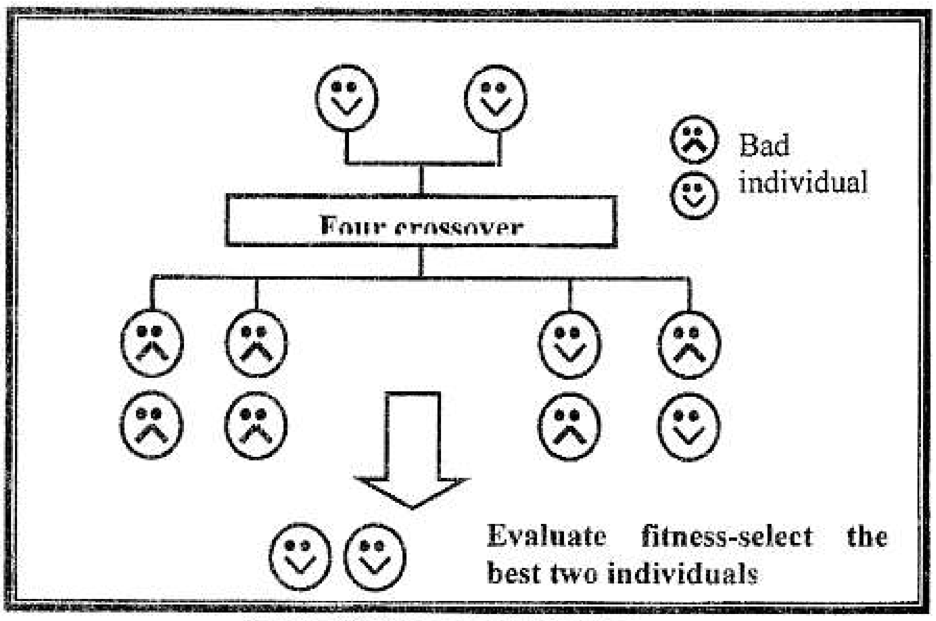

Figure (3) Brood recombination

Table (2) Table for the simple symbolic regression problem

Objective	Find a function of one independent variable and one dependent variable, in symbolic form, that fits a given sample of 20(X_i,Y_i) data points, where the target function is the quartic polynomial($X^4+X^3+X^2+X$).
Terminal set	X(the independent variable)
Function set	+,-,*,%
Raw fitness	The sum, taken over the 20 fitness cases, of the absolute value of difference between value of the dependent variable produced by the S-expression and the target value Yi of the dependent variable.
Standardized fitness	Equal to the raw fitness for this problem.
Adjusted fitness	The adjusted fitness is 1.0/(1.0+raw fitness)
Fitness cases	The given sample of 20 data points (X_i,Y_i) where the X_I comes from the interval[-1,+1].
Parameters	M(population size)=500,G(generation)=51

Table (3) Experimental constants and variables

Value	Description
500	The population is fixed at 500 individuals.
Best-of-run	When an individual has fitness value equal to 1.00, the run is terminated.
AST	Phenotypes are generated as ASTs which can be interpreted.
Variable length	Three individual lengths are compared.
Selection	Tournament selection is used.
Crossover	Brood recombination crossover is used.
0.35	Mutation rate.
1.0	Crossover rate.
<sexp> <application>	Two initial values for generating the phenotype are used.

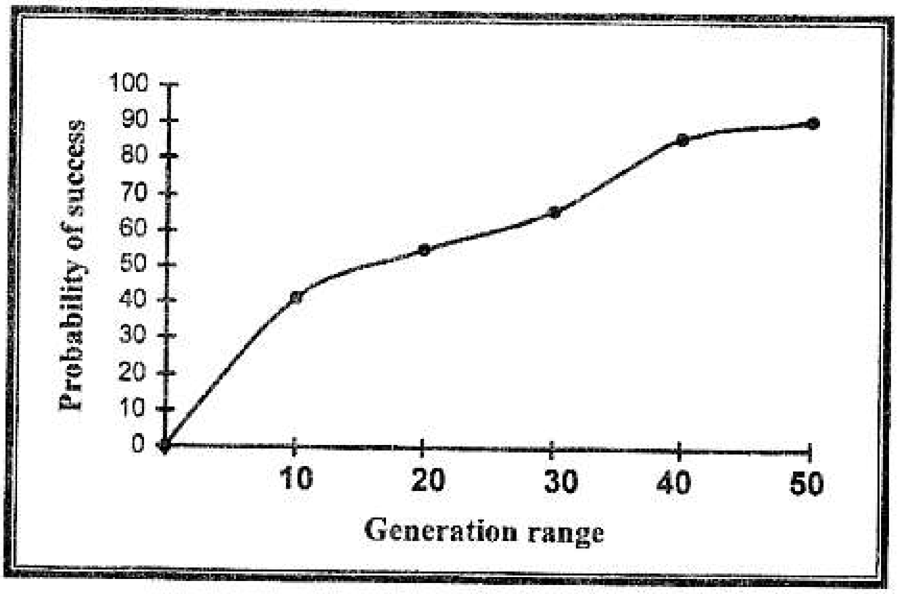

Figure (4) Performance curve for the Symbolic Regression Problem