Traditional Techniques of Genetic Algorithms Applied to Floating-Point Chromosome Representations

Leo Budin, Marin Golub, Andrea Budin
Faculty of Electrical Engineering and Computing
Department of Electronics, Microelectronics, Computer and Intelligent Systems
Unska 3, HR-10000 Zagreb, Croatia
phone: +385.1.61 29 935, fax: +385.1.61 29 653
e-mail: {leo.budin, marin.golub, andrea.budin}@fer.hr

Abstract - The choice of chromosome representation in genetic algorithms depends on the variables of the optimization problem being solved. If the variables are real-valued, the chromosomes can be represented as fixed-point integer values which enables the use of classical genetic operators defined for binary strings. Another possible chromosome representation is using floating-point numbers directly. In this case, genetic operators have to be defined additionaly. This paper presents a different approach: floating-point chromosome representation with traditional genetic operators. In order to achieve fine local tuning of the solutions, a mapping is defined which dynamically changes the operating scope of genetic operators.

I. Introduction

Genetic algorithms have recently emerged as practical and robust optimization methods. One of the most important issues to be considered when trying to solve a particular problem is the choice of adequate chromosome representation. Mostly used chromosome representations are binary strings, character strings, floating-point numbers, arrays of numbers, matrices and other data structures [3,4,5,6,8,10]. For a given problem, there is always a representation which exhibits better optimization results in comparison with other representations. However, the theory of genetic algorithms concentrates mostly on binary representations and has very little to say about nonbinary representations.

Another important issue in genetic algorithm structures, closely related with the choice of chromosome representation, are the encoding and decoding mechanisms which perform transformations between the chromosome representations and the optimization problem's variables [10]. Those mechanisms depend on the nature of the problem variables.

II. Binary chromosome representation

A large number of optimization problems have real-valued continuous variables. We are interested in the minimization of a function of n variables $f(x_1,x_2,...,x_n)$:$R^n \rightarrow R$ with explicit constraints $x_{il} \Lambda x_i \Lambda x_{ih}$, $i \in \{1,2,...,n\}$. The classical genetic algorithms use binary encoding with m bits per variable, i.e. the chromosome is of length $n \cdot m$. Each gene of length m is encoded as:

$$B_i = \frac{x_i - x_{il}}{x_{ih} - x_i} \cdot 2^m \tag{1}$$

Decoding is performed according to the following expression:

$$x_i = x_{il} + B_i \cdot 2^{-m} \cdot (x_{ih} - x_{il}) \tag{2}$$

This method is usually referred to as fixed-point integer encoding. One of the drawbacks of this method is the problem known as Hamming cliffs [8]. Namely, the Hamming distances between binary codes of adjacent integers sometimes tend to be very large. This problem is usually solved by Gray coding.

III. Floating-point chromosome representation

An alternative representation in optimization problems with real-valued continuous variables is the floating-point chromosome representation. With this representation, there is no need for an explicit encoding mechanism. Each member of each population in the genetic algorithm is a floating-point vector. The genetic operators (mutation and crossover) in this case do not handle bit strings and are defined in a different manner. For example, the mutation operation does not randomly change one bit, but randomly chooses a floating-point number within a particular range.

This paper will describe our attempt to develop a different approach to solving optimization problems with real-valued continuous variables. In our approach, classical genetic operators defined for binary strings are used for floating-point chromosome representations.

We first introduce normalized variables:

$$y_i = \frac{x_i - x_{il}}{x_{ih} - x_{il}},\ i \in \{1,\ldots,n\}\text{, i.e.} \tag{3}$$

in $f(x_1,x_2,...,x_n)$ we replace the variables with:

$$x_i = x_{il} + (x_{ih} - x_{il}) \cdot y_i,\ i \in \{1,\ldots,n\} \tag{4}$$

Therefore, the function to be minimized becomes $f(y_i,y_2,...,y_n)$:$[0,1]^n \rightarrow R$, i.e. it's domain is reduced to the interval $[0,1]^n$.

This paper describes an approach for solving optimization problems with real-valued continuous variables which utilizes classical genetic operators (defined for binary strings) over floating-point chromosome representations.

There are various possible floating-point representations of real numbers. However, in order to keep the representation machine-independent, we use the double precision format defined with the IEEE 754-1985 standard [1], where the floating-point number is represented in 64 bits (Figure 1). The first bit is the sign bit, the following 11 bits store the biased exponent (bias=1023). The remaining 52 bits represent the fraction, which is always a value less than one, and the significand is one plus the fraction value. If we denote the sign bit with **s**, the exponent with **e** and the fraction with **f**, the value of a represented number is $(-1)^s \times 1.f \times 2^{e-1023}$.

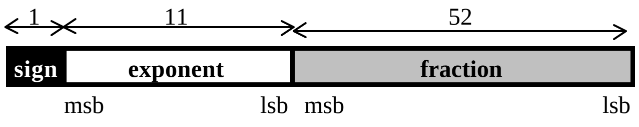

Fig. 1. Double precision IEEE 754-1985 floating-point format

The genetic algorithm treats the floating-point chromosome representations as binary strings. Those binary strings are subject to classical genetic operators. We used uniform crossover - each bit of a child is taken from one of the parents randomly, according to the following expression: CHILD = (A AND B) OR (R AND (A XOR B)), where A and B are parents, and R is a random number. The mutation operator flips randomly selected bits: it changes their values from 0 to 1 or vice versa.

A problem which is rather often encountered is the problem of operators which sometimes generate impossible solutions - solutions not belonging to the solution space. This is the problem of handling constraints and it may be treated as follows. If the frequency of impossible solutions generated by a particular genetic operator is significant, the genetic operator has to be redefined in order to generate possible solutions.

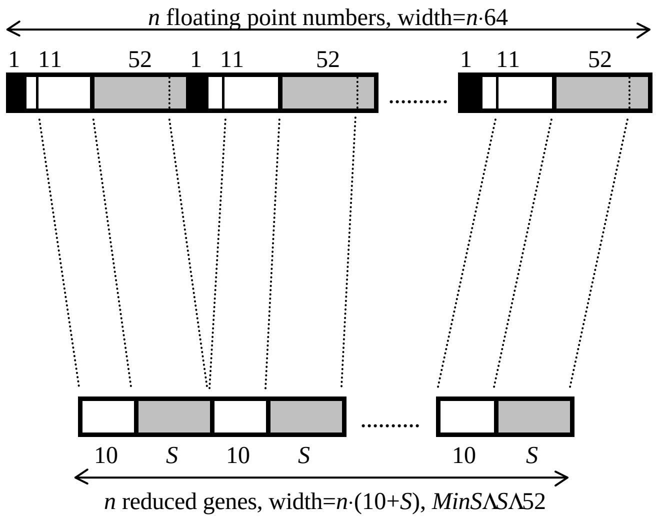
n floating point numbers, width=n·64
1 11 52 1 11 52 1 11 52
....
....
10 S 10 S 10 S
n reduced genes, width=n·(10+S), MinS≤S≤52

Fig. 2. Chromosome mapping

The operating scope or operating length of a genetic operator is the length of the binary string on which the operator performs an operation at a given moment. In our approach, the operating scope (length) of genetic operators is dynamic. It is time-dependent, i.e. it depends on the number of generations or iterations. At each stage of the evolution, the whole chromosome is mapped into a reduced-size chromosome (Figure 2).

Since the domain of the function that has to be minimized has been reduced to the interval $[0,1]^n$, we don't need the sign bit nor the most siginificant bit of the exponent. The start length, i.e. the length of each member of the first population, consists only of 10 bits of the exponent and a chosen number (MinS) of most significant bits of the fraction. During the search process, the length is increased in order to include more most significant bits of the fraction (S). At the end of the search process, the operating length may reach 62 bits (the full length of the floating-point number minus two most sgnificant bits). The goal at the beginning of the process is rough location of the global optimum. As the process develops, increasing the operating scope of the operators enables fine local tuning of the solution.

We found the described approach very appropriate for polynomial, Fourier and wavelet approximation of time series. The simplified encoding - decoding mechanism during the evolution process reduces the computation time.

IV. EXPERIMENTAL RESULTS

As an example, the task to be solved by the genetic algorithm was to interpolate the given function g through the time series $T=\{(x_1,y_1),(x_2,y_2),\ldots,(x_{20},y_{20})\}$, i.e. to determine 29 coefficients $a_0,a_1,a_2,\ldots a_{28}$ of the function g given as:

$$g(x) = a_0 + a_1 x + \sum_{i=1}^{6} \left(a_{3i-1} \sin\left(a_{3i} x + a_{3i+1} \right) \right). \tag{5}$$

Additionally, we introduce the constraint that the coefficients of the function g must belong to given intervals as follows:

$$a_0,a_1,a_2,a_5,a_8,a_{11},a_{14},a_{17} \in [-10,10]$$
$$a_3,a_6,a_9,a_{12},a_{15},a_{18} \in [0,2]$$
$$a_4,a_7,a_{10},a_{13},a_{16},a_{19} \in [-5,5].$$

This problem can be stated as finding the minimal sum of squares of deviations for the given function g and the given time series T. Therefore, the goal function is:

$$f\left(a_0, a_1, \ldots, a_{28}\right) = \sum_{i=1}^{20} \left(g(x_i) - y_i \right)^2 \tag{6}$$

We took the following time series:

T = { (1,1),(4,50),(5,51),(7,52),(9,60),(11,71),(14,76), (15,73),(16,79),(17,68),(20,55),(22,57),(25,80), (27,82),(29,78),(31,100),(34,90),(36,85),(38,80), (40,78) }

In the first iteration, the genetic algorithm generates a random initial population. The best solution in the first iteration is given in Fig. 3.

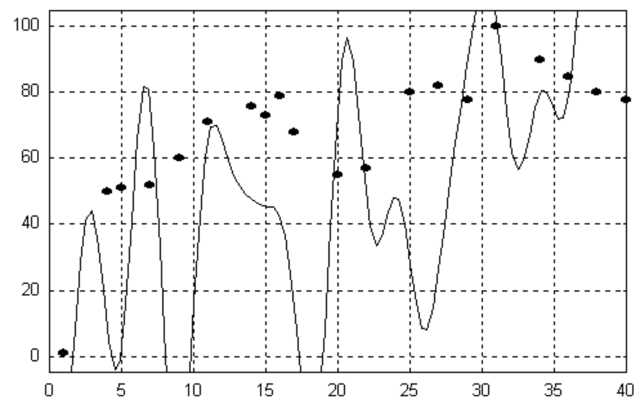

Fig. 3. A randomly generated solution in the first iteration:

$$
\begin{aligned}
g(x) = {} & -8.469+2.768*x-11.575*\sin(1.838*x+4.693)- \\
& -0.220*\sin(1.416*x-4.214)- \\
& -27.042*\sin(1.361*x+7.119)+ \\
& +28.305*\sin(1.406*x-0.974)- \\
& -13.592*\sin(0.702*x+8.645)- \\
& -9.012*\sin(0.229*x-7.921)+ \\
& +15.272*\sin(0.756*x+3.814)+ \\
& +8.445*\sin(0.834*x-3.715)- \\
& -22.951*\sin(0.206*x+8.920)
\end{aligned}
$$

$$f(a_0, a_1, \ldots, a_{28}) = 34066.9$$

During the evolution process, further populations exhibit better solutions, e.g. the best solution in the 114th iteration is shown in Fig. 4.

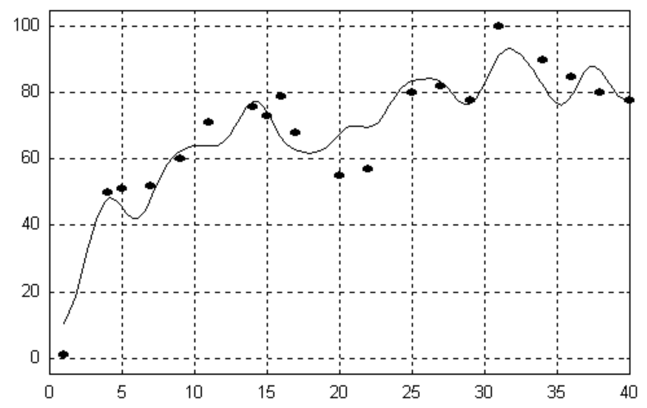

Fig. 4. The solution after 114 iterations:

$$
\begin{aligned}
g(x) = {} & 24.321+2.046*x+5.069*\sin(0.297*x+5.072)- \\
& -13.991*\sin(0.121*x-3.181)+ \\
& +1.088*\sin(1.900*x+6.073)- \\
& -2.783*\sin(0.383*x+0.436)- \\
& -1.542*\sin(0.976*x-0.357)- \\
& -3.979*\sin(0.659*x-4.554)- \\
& -0.583*\sin(1.128*x+2.243)- \\
& -4.182*\sin(1.214*x+7.893)+ \\
& +7.858*\sin(1.140*x+3.279)
\end{aligned}
$$

$$f(a_0, a_1, \ldots, a_{28}) = 922.9$$

The termination criterion for our genetic algorithm was the number of iterations, set to 4000.

In the final solution, depicted in Fig. 5., the sum of squared deviations is 84.6. The obtained solution almost perfectly fits to 16 of the 20 given points of the time series.

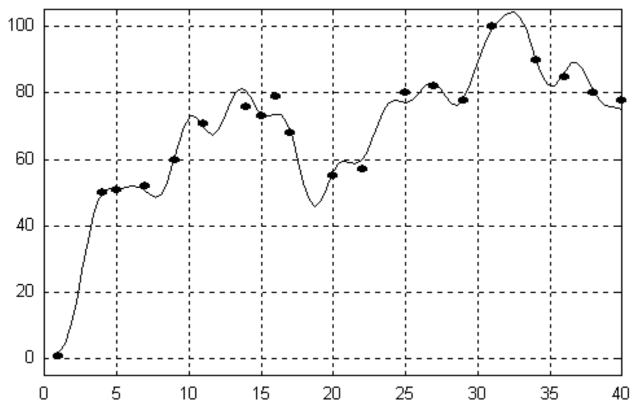

Fig. 5. The final solution after 4000 iterations:

$$
\begin{aligned}
g(x) = {} & 23.906+2.048*x+12.994*\sin(0.288*x+5.045)- \\
& -13.125*\sin(0.121*x-3.164)+ \\
& +3.257*\sin(1.891*x+7.500)- \\
& -2.811*\sin(0.422*x+5.312)- \\
& -3.750*\sin(0.984*x-0.002)- \\
& -7.500*\sin(0.663*x+7.813)- \\
& -1.819*\sin(1.689*x+7.500)- \\
& -3.750*\sin(1.188*x+8.553)+ \\
& +8.438*\sin(1.137*x+3.125)
\end{aligned}
$$

$$f(a_0, a_1, \ldots, a_{28}) = 84.6$$

V. Conclusion

In our approach, the genetic algorithm exhibits good results in fine local tuning because of the variable chromosome length. At the beginning of the optimization process, the chromosome has a certain start length, which increases during the process. In the first part of the process, the solution is subject only to rough tuning. Fine local tuning is performed towards the end of the process, when the chromosomes reach the whole length of the floating-point number.

By implementing the described genetic algorithm, we also achieved better results concerning the execution time in comparison with the traditional genetic algorithms with binary chromosome representations. This improvement is due to the avoidance of the encoding mechanism.

However, some problems have arisen in the implementation of the traditional mutation of exponents. A randomly generated exponents tends to be close to the lower bound of the domain, if the domain is mapped onto the interval [0,1]. The problem has been overridden by using such a mutation of exponents where the random numbers are generated with a uniform probability within a given interval.

References

[1] —, *An American National Standard: IEEE Standard for Binary Floating-Point Arithmetic,* ANSI/IEEE Standard 754-1985.

[2] Davis, L., *Handbook of Genetic Algorithms,* Van Nostrand Reinhold, New York, 1991.

[3] Field, P., *A Multary Theory for Genetic Algorithms: Unifying Binary and Nonbinary Problem Representations*, Doctoral Thesis, University of London, 1995.

[4] Filhio, J.L.R., Treleaven, P.C., Alippi, C., *"Genetic-Algorithm Programming Environments"*, Computer, Vol. 27-6, pp. 28-43, June 1994.

[5] Goldberg, D., *"What Every Computer Scientist Should Know About Floating-Point Arithmetic"*, ACM Computing Surveys, Vol. 23-3, pp. 5-48, March 1991.

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

[7] Golub M., *Vrednovanje uporabe genetskih algoritama za aproksimaciju vremenskih nizova*, M.S. Thesis, Zagreb, 1996. (in Croatian)

[8] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin, 1994.

[9] Schoeneburg E., Heinzmann F., Feddersen S., *Genetische Algorithmen und Evolutionsstrategien*, Addison-Wesley, 1995.

[10] Srinivas, M., Patnaik, L.M., *"Genetic Algorithms: A Survey"*, Computer, Vol. 27-6, pp.17-26, June 1994.