USING GENETIC ALGORITHMS FOR ADAPTING APPROXIMATION FUNCTIONS

Marin Golub, Andrea Budin Posavec
Faculty of Electrical Engineering and Computing, University of Zagreb
Department of Electronics, Microelectronics, Computer and Intelligent Systems
Unska 3, 10000 Zagreb, Croatia
e-mail: {golub, andrea}@zemris.fer.hr

Abstract: *This paper describes a different approach: using the genetic algorithm not only for finding the optimum of the given cost function, but for adapting an existing solution of the present problem to the new, modified problem. The genetic algorithm was extended to deal with the dynamic approximation problem: the time series were changed during the optimization process.*

Key words: genetic algorithm, adapting mechanism, approximation functions, time series

1. Introduction

Using genetic algorithms as an optimization method is an attempt at imitating the natural evolution process. A traditional genetic algorithm tries to find an optimum of a cost function which does not change during the optimization process (Goldberg, 1989, Michalewicz, 1994). Individuals in a population represent potential solutions. The cost function of the optimization problem is like the natural environment: an individual continuously tends to adjust to the new environmental conditions. A start population in traditional genetic algorithms is either generated randomly or may be uniform (e.g. all solutions set to zero).

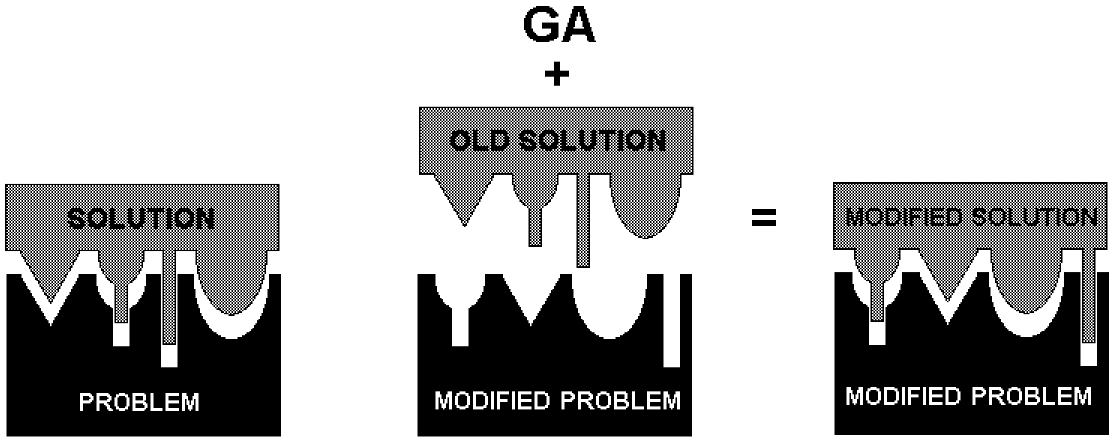


Fig. 1 Using the genetic algorithm as the adapting mechanism

In this paper we consider rephrasing the optimization problem. Let us suppose we have at first used a genetic algorithm to find a solution of an optimization problem. When the problem changes, we try to find a new solution, however, not by starting from the beginning, but by using the known solution of the original problem (Fig. 1). In other words, at the beginning of the optimization process we have a defined problem and a known solution. As time elapses, the problem continuously changes and for each change we have to find a new solution. The new solution is in many cases relatively similar to the previous solution. The genetic algorithm is thus used as an adapting mechanism similar to nature's adapting mechanisms.

2. A test problem

The task to be solved by the genetic algorithm was to interpolate the given function g through the continuously changing time series $T=\{(x_1,y_1),(x_2,y_2),...,(x_S,y_S)\}$. The initial time series consisting of $S=20$ points is (Fig. 2):

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) }.

The approximation functions g (Schoeneburg, 1995) is given as:

$$g(x) = a_0 + a_1 x + \sum_{i=0}^{P-1} \left(b_{3i} \sin\left(b_{3i+1} x + b_{3i+2} \right) \right) + \sum_{i=0}^{R-1} \left(\frac{c_{3i}}{\cosh^2\left(c_{3i+1} x + c_{3i+2} \right)} \right). \qquad (1)$$

The constraint is that the coefficients $a_0,a_1,b_0,b_1,...,b_{P-1},c_0,c_1,...,c_{R-1}$ belong to given intervals as follows:

$a_0 \in [-10,70]$, $a_1 \in [-1,10]$;
$b_{3i} \in [-60,60]$, $b_{3i+1} \in [0,2]$, $b_{3i+2} \in [-10,10]$, where $i=0,1,2,...,P-1$;
$c_{3i},c_{3i+2} \in [-100,100]$, $c_{3i+1} \in [0,5]$, where $i=0,1,2,...,R-1$.

The 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 to be minimized is:

$$f\left(a_0,a_1,b_0,b_1,...,b_{P-1},c_0,c_1,...,c_{R-1}\right) = f\left(\vec{a},\vec{b},\vec{c}\right) = \sum_{i=1}^{S} \left(g(x_i) - y_i \right)^2 . \qquad (2)$$

Each point of the time series has its own velocity v_i and its moving direction which can be *up* ($v_i>0$) or *down* ($v_i<0$) (Fig. 2). The change of the time series with time may precisely be stated as the change of the y_i value of each point according to the following expression:

$$y_i(t) = v_i \cdot t . \qquad (3)$$

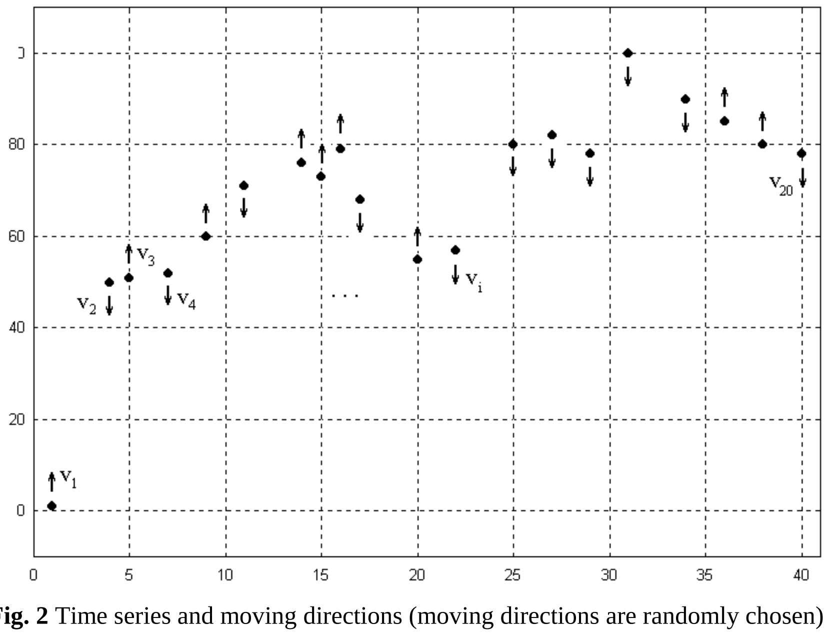
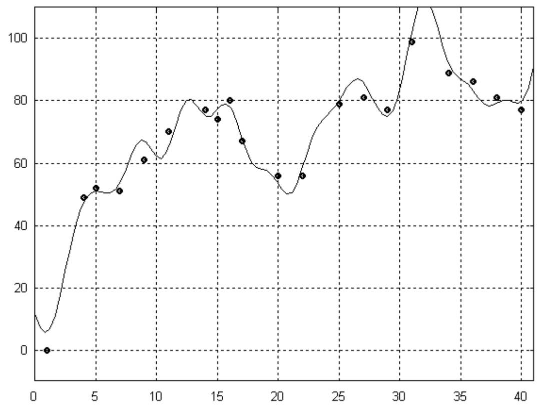


Fig. 2 Time series and moving directions (moving directions are randomly chosen)

The moving direction is randomly chosen. An additional constraint is that the value of each y_i must belong to the interval [0,100]. If a point is moving upwards and its y_i value becomes greater than 99, the moving direction changes to *down*. Analogously, a point moving downwards changes its direction to *up* when its y_i value becomes smaller than 1.

At the beginning of the optimization process we took two different approximation functions: function g_1 with $P=9$ and $R=0$ (containing only the *sin* members) and function g_2 with $P=6$ and $R=3$. The solutions, i.e. functions g_1^0 and g_2^0, were found by the genetic algorithm after several thousands of iterations (Budin, 1996, Golub, 1996) and are shown in Figs. 3 and 4. Those solutions become the start solutions of the optimization process.



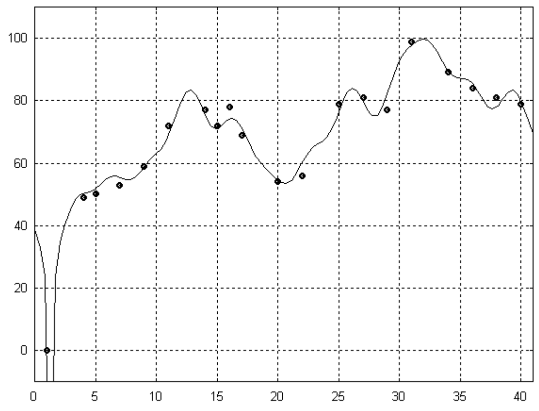


Fig. 3 The start solution - approximation function g_1:

$$g_1^0 = 45.781+1.259\cdot x-2.568\cdot \sin(0.407\cdot x-9.191)-0.544\cdot \sin(1.218\cdot x-6.239)-5.173\cdot \sin(1.173\cdot x-0.277)-1.652\cdot \sin(1.896\cdot x-6.487)+5.526\cdot \sin(0.531\cdot x+0.035)+10.840\cdot \sin(0.315\cdot x-3.627)-5.627\cdot \sin(1.258\cdot x+7.744)-7.830\cdot \sin(0.697\cdot x-5.638)-4.234\cdot \sin(0.872\cdot x+1.447)+9.686\cdot \sin(0.457\cdot x-1.415)-24.478\cdot \sin(0.359\cdot x+0.871)-2.812\cdot \sin(1.649\cdot x-2.715),$$

and the sum of squares of deviations is $f(\vec{a},\vec{b})=201$.

Fig. 4 The start solution - approximation function g_2:

$$g_2^0 = 52.598+0.884\cdot x+4.317\cdot \sin(0.913\cdot x-2.552)+0.191\cdot \sin(0.497\cdot x+7.085)+1.835\cdot \sin(0.147\cdot x+9.993)+1.983\cdot \sin(1.890\cdot x-3.836)+2.842\cdot \sin(1.376\cdot x+9.521)+13.024\cdot \sin(0.331\cdot x-2.483)-3.515\cdot \sin(0.620\cdot x+1.817)+2.760\cdot \sin(0.759\cdot x-9.332)+0.134\cdot \sin(1.524\cdot x-7.498)+60.581\cdot \cosh^{-2}(0.091\cdot x+66.374)-96.677\cdot \cosh^{-2}(0.280\cdot x+66.906)-96.268\cdot \cosh^{-2}(4.999\cdot x-6.275),$$

and $f(\vec{a},\vec{b},\vec{c})=119$.

The further task to be solved by the genetic algorithm was to interpolate the given function $g^{t+\Delta t}$ through the dynamic time series $T^{t+\Delta t}=\{(x_1,y_1(t+\Delta t)), (x_2,y_2(t+\Delta t)),\ldots, (x_S,y_S(t+\Delta t))\}$ based on the previous solution - the function g^t. The problem is N-dimensional, since we have to de-termine N coefficients of the function g for each change of the time series, where $N=2+P+R$.

3. The implemented genetic algorithm

The genetic algorithm with steady-state reproduction with roulette-wheel based bad individual selection has been used. The population was made up of *POP_SIZE* individuals. Steady-state reproduction replaces M individuals, i.e. M bad individuals are selected for elimination at each iteration. Their places are taken by the children of surviving individuals, the parents being chosen randomly. The elitism principle has been used, which means that the best individual is protected from selection and elimination. In other words, the probability of elimination for the best individual equals zero.

The binary chromosome representation has been used, with each chromosome consisting of an array of binary vectors, since the problem is multidimensional Each binary vector contains B bits, so the whole chromosome consists of $B \cdot N$ bits.

Parameter	Description	Value
POP_SIZE	size of population	100
N	dimension of problem	38
B	bits per binary vector	26
M	selected number of individuals for elimination	50
p_m	probability of mutation	0.05
Δk	number of iterations between two changes of time series	200
$v_i=\Delta y/\Delta k$	change velocity	1/200

Table 1 Genetic algorithm and problem parameters

In the first iteration, the genetic algorithm generates an initial population based on the given solution. The initial population consists of the given solution and its mutations. The time series changes every Δk iterations. The values of the parameters of the genetic algorithm (*POP_SIZE*, N, B, M, p_m) and the parameters of the given problem (Δk, v_i) are shown in Table 1. The values of those parameters were chosen as a result of a set of performed experiments.

4. Experimental Results

The results of the optimization processes for the functions g_1 and g_2 are shown in Figs. 5 and 6, respectively. The experiments were performed on a SUN SPARC station 20 with 64 MB RAM. As can be seen in Table 2, somewhat more time was spent for optimization with function g_2. However, the achieved results for this function were superior to those for the function g_1. (the average sum of squares of deviations is smaller). The reason for this lies in the fact that the *cosh* members present in function g_2 enable a steeper change of values. This is rather important when the y_i value of a point significantly deviates from the y_i values of the other points. Such is the case with the first point of the time series in Figs. 3 and 4. As can be seen in Fig. 3, only the *sin* members of the function g_1 do not adequately approximate the first point.

approximation function	g_1	g_2
parameter P	9	6
parameter R	0	3
CPU time in seconds for $\Delta k=200$ iterations	24.9	30.8
average sum of squares of deviations ($\bar{f}$)	695	470

Table 2 CPU times and deviations for both approximation functions

The number of iterations between two changes of the time series (Δk) depends on the maximal velocity $V=max(v_i)$. If Δk is too small, the genetic algorithm cannot follow the changes of the time series. In that case, either Δk must be increased or V must be decreased or both. The number of iterations between two changes of the time series was determined experimentally. If the efficiency of the genetic algorithm is improved, e.g. by adjusting its parameters or changing the whole algorithm, *Δk* can be decreased in order to speed up the optimization process.

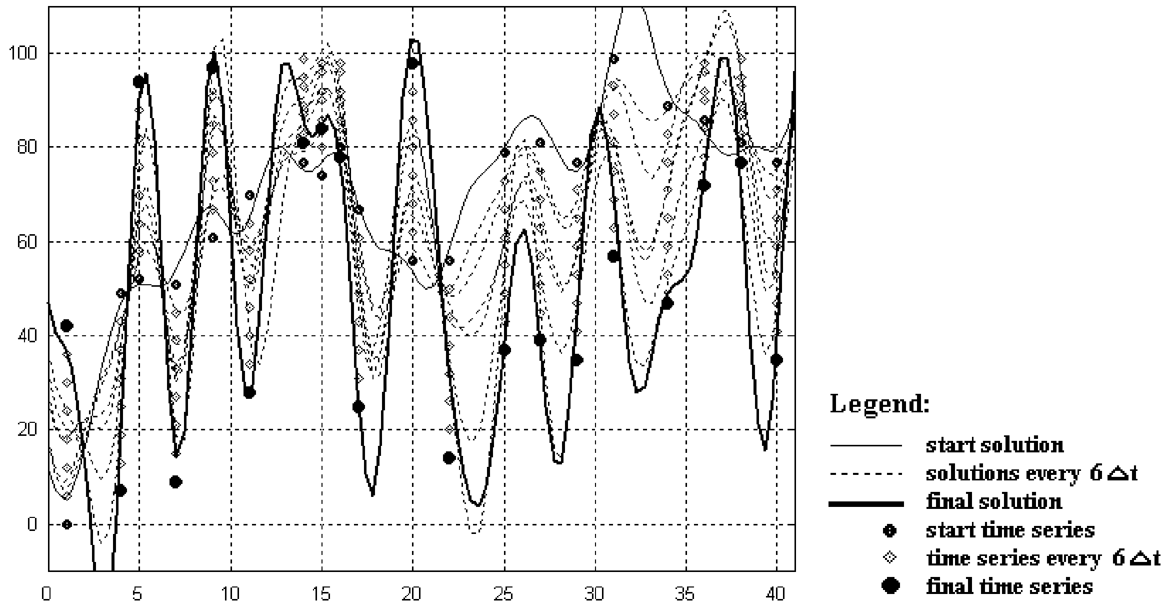


Fig. 5 The adapting process for approximation fuction g_1.The final solution g_1^{42} with $f\left(\vec{a},\vec{b}\right)=983$ is:

g_1^{42} =45.11+0.375·x-8.727·sin(0.974·x-7.920)-15.16·sin(1.215·x-7.3)-8.29·sin(1.18·x-0.31)-4.35·sin(1.88·x-5.73)+
+12.4·sin(1.56·x+6.21)+8.85·sin(0.31·x-8.55)-17.43·sin(1.76·x+7.94)-4.2·sin(1.124·x-5.336)-
-4.51·sin(0.62·x+5.13)+17.38·sin(0.84·x-3.79)-8.7·sin(0.255·x+0.82)-8.739·sin(0.156·x-3.125)

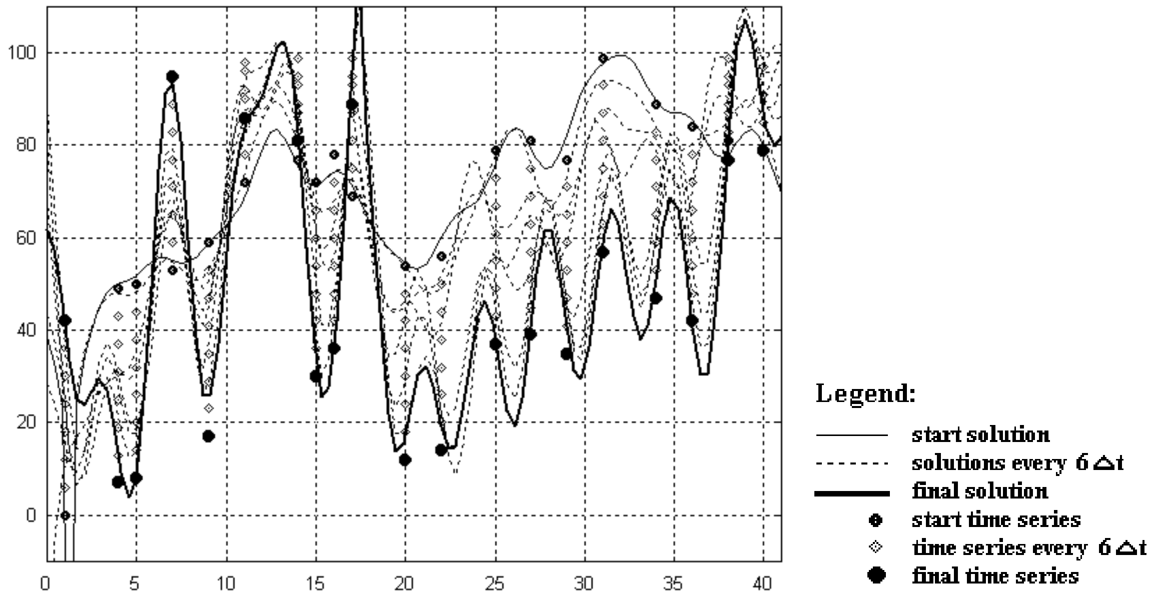


Fig. 6 The adapting process for approximation fuction g_2. The final solution g_2^{42} with $f\left(\vec{a},\vec{b},\vec{c}\right)=342$ is:

g_2^{42} =20+1.75·x-17.43·sin(1.785·x-1.193)-3.05·sin(0.4·x-0.4)+17.45·sin(0.09·x+1.835)+4.98·sin(0.27·x-1.74)-
-17.16·sin(1.14·x-8.99)+6.27·sin(1.35·x-2.52)-11.61·sin(1.0·x+4.38)+0.51·sin(1.92·x+1.1)+
+17.35·sin(0.18·x-0.38)+31.67·cosh^{-2}(3.401·x-58.876)-24.491·cosh^{-2} (1.208·x+32.251)+
+45.503·cosh^{-2} (0.760·x+65.979)

5. Concluding remarks

The main difference between traditional genetic algorithms and the genetic algorithm described in this paper lies in the choice of the start population. Traditional genetic algorithms generate a start population randomly. On the other hand, the implemented genetic algorithm generates a start population based on an already found solution, i.e. the start population consists of the found solution and its mutations.

The start solution was found by the genetic algorithm in about 2000 iterations. Without using the previously found solution in the initial population, several thousand iterations are needed for each change of the time series. If, however, the genetic algorithm is used for adapting an existing solution, the number of needed iterations is about ten times smaller. This means that the time required for the solution of the same problem can be about ten times shorter.

The approximation function can also be changed during the optimization process. As further work, a genetic algorithm should be developed which would change the whole approximation function and not only its coefficients. Special genetic operators should be defined for dealing with parts of approximation functions.

Acknowledgement

This work was carried out within the project 036-014 *Problem-Solving Environments in Engineering*, funded by Ministry of Science and Technology of the Republic of Croatia.

References:

1. Budin, L., Golub, M., Budin, A. (1996), "Traditional Techniques of Genetic Algorithms Applied to Floating-Point Chromosome Representations", KoREMA '96. - *41st Annual Conference*, Opatija, pp. 93-96.
2. Davis, L. (1991), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.
3. Filhio, J.L.R., Treleaven, P.C., Alippi, C. (1994), "Genetic-Algorithm Programming Environments", Computer, Vol. 27-6, pp. 28-43., June 1994.
4. Goldberg, D.E. (1989*), Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley
5. Golub M., "Vrednovanje uporabe genetskih algoritama za aproksimaciju vremenskih nizova", M.S. Thesis, Zagreb, 1996. (in Croatian)
6. Michalewicz, Z. (1994), *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin
7. Schoeneburg, E., Heinzmann, F., Feddersen. S., (1995) *Genetische Algorithmen und Evolutionsstrategien,* Addison-Wesley
8. Srinivas, M., Patnaik, L.M. (1994), "Genetic Algorithms: A Survey", Computer, Vol. 27-6, pp.17-26, June 1994.