Operations Research and Cybernetics

DOI :10. 11721/cqnuj20130103

The Application of Genetic Algorithm in Solving Nonsmooth Optimization Problems*

LONG Qiang

(School of SITE, University of Ballarat, Ballarat Victoria 3350, Australia)

Abstract: This paper considers an application of genetic algorithm in solving nonsmooth optimization problems. Nonsmooth optimization devote itself to solve programming problems whose objective function are continuous nondifferentiable. Since the objective function is nondifferentiable, the classical deterministic methods based on gradient may confront numerical difficulties. Therefore, it would be a good choice to use genetic algorithm, in which just information of objective function value but not information of gradient is needed to solve nonsmooth optimization problems. Genetic algorithm is a stochastic method based on the evolutionary process of nature. It firstly codes the original optimization problem by means of binary encoding, Gary encoding or real number encoding. And then the next population generation is generated by applying crossover operator, mutation operator and selection operator. When the iteration time approach a sufficiently large number, the best chromosome in the current population will converge to the optimal solution or approximately optimal solution of the original problem. The genetic algorithm proposed in this paper uses real-number encoding, arithmetic crossover, nonuniform mutation. And it selects the best population size of individuals in the selection step. Some minimax problems, which are nonsmooth optimization problems, are tested and their results are compared with some deterministic nonsmooth optimization methods.

Key words: genetic algorithm; minimax problem; nonsmooth optimization problem

中图分类号: O224 **文献标志码**: A **文章编号**: 1672-6693(2013)01-0012-05

1 Introduction

Genetic algorithm is one of the most important stochastic algorithms in mathematical programming. It was firstly introduced by John Holland in the 1960s, and then developed by his students and colleagues at theUniversity of Michigan in the 1960s and 1970s[1]. In the last two decades, genetic algorithm was increasingly enriched by plenty of literatures[2-6]. And now various genetic algorithms are applied in different areas, such as math programming, combinational optimization, automatic control, image processing, and so on.

The main idea of genetic algorithm is based on biological natural selection and genetic mechanism, and different from traditional deterministic methods, it is a stochastic algorithm. The earliest structure of genetic algorithm was provided by Goldberg in [7]. It firstly randomly generates a series of solutions which is called initial population, and one single individual from the population is called a chromosome. The number of chromosomes in a population is defined as population size. In numerical computation, Chromosomes experience as binary code, Gray code or real-number code. Those chromosomes generate their offspring in two different ways: crossover and mutation. Crossover randomly exchanges some genes (which constitute chromosomes) between two selected individuals. Mutation changes some randomly selected genes of an individual in a certain way. Then the next population is constructed by selecting population size of best chromosomes from the last population and its offspring. The criterion for selecting the next generation is the performance of each chromosome according to fitness function which is normally the objection function value. Those chromosomes

* **Recieved** 04-27-2012 **Accepted** 07-18-2012 **网络出版时间** 2013-01-18 15 05

Foundation: The Science and Technology Project Affiliated to the Education Department of Chongqing Mumicipality(No. KJ120616)

First author biography: LONG Qiang, male, Doctor Student, research area is optimization theory and algorithm, nonsmooth optimization and global optimization, E-mail: 27131375@qq.com

收稿日期: 2012-04-27 **修回日期**: 2012-07-18

资助项目: 重庆市教委科技项目(No. KJ120616)

作者简介: 龙强, 男, 博士研究生, 研究方向为最优化理论与算法、非光滑优化及全局最优化, E-mail: 27131375@qq.com

网络出版地址: http://www.cnki.net/kcms/detail/50.1165.N.20130118.1505.201301.12_026.html

whose fitness is better are kept and whose fitness is worse are eliminated. In this way, as the generation iteration goes on, the algorithm will converge to the best chromosome, which probably is the optimal solution or suboptimal solution of the original optimization problem. In practical computation, we set beforehand a maximal generation time, this maximal generation time further plays a role of stopping criterion.

Suppose that $P(t)$ and $O(t)$ represent the parents and offspring of the t^{th} generation, then the general structure of genetic algorithm can be written in the following pseudo code.

```
1    Initialization
     1.1  Generate the initial population P(0)
     1.2  Set crossover rate, mutation rate and maximal generation time.
     1.3  t←0
2    While the stopping criteria is not satisfied, do
     2.1  Crossover operator: generate O(t)
     2.2  Mutation operator: generate O(t)
     2.3  Evaluate P(t) and O(t): compute the fitness function
     2.4  Selection operator: build the next population
     2.5  t←t+1, go to 2.1
     end
end
```

Fig.1　General Structure of Genetic Algorithm

From the pseudo code, we can see that there are three important operators in general genetic algorithm.

- **Crossover operator**: This operator randomly chooses a locus and exchanges the subsequences before or after that locus between two chromosomes to create two offspring. For example, suppose that strings 10000100 and 11111111 are two parents chromosomes selected to crossover and the third locus is selected, then the crossover operator exchanges the last five genes of those two chromosomes and produces two offspring 10011111 and 1100100. The crossover operator roughly mimics biological recombination between two single chromosome organisms.
- **Mutation operator**: This operator randomly flips some of the bits in a chromosome. For example, the string 00000100 might be mutated in its second position to yield an offspring 01000100. Mutation can occur at each bit position in a string with some probability, usually very small (e.g., 0.01).
- **Selection operator**: This operator selects chromosomes in the population for reproduction. The fitter the chromosome the more time it is likely to be selected to reproduce.

Those three operators are very important in efficiency and robust of genetic algorithm. Better operators not only make algorithms convergent faster, but also reduce the amount of calculation.

2 Real-number encoding genetic algorithm

In this section, we present the real-number encoding genetic algorithm we are going to use in this paper. Firstly, we suppose that the general structure of the problem we are about to solve is

$$\begin{cases} \min f(x) \\ \text{s.t. } x \in [lb, ub] \end{cases} \tag{1}$$

Where f $\mathbf{R}^n \to \mathbf{R}$ is a Lipschitz continuous function, it 's not necessary differentiable. In the following, we call this problem nonsmooth optimization problem.

For the genetic algorithm, we use real-number encoding, which means the initial population is randomly chosen in $[lb, ub]$. Especially, we generate chromosomes by the following formula

$$x = lb + \alpha(ub - lb) \tag{2}$$

Where α is a random number in $[0,1]$.

For the crossover operator, we use arithmetic crossover[8]. For example, suppose x_1 and x_2 are two chromosomes randomly selected to crossover, then we use the following style to form their offspring

$$\begin{aligned} x_1' &= \beta x_1 + (1-\beta) x_2 \\ x_2' &= \beta x_2 + (1-\beta) x_1 \end{aligned} \tag{3}$$

Where β is a randomly chosen number in $[0,1]$.

Nonuniform mutation is applied in mutation operator. For a given parent x, if its component x_k was chosen to mutate, then the offspring should be $x' = (x_1, \cdots, x_k', \cdots, x_n)$, where x_k was randomly changed by the following two approach, or

$$\begin{aligned} x_k &= x_k + d(t, x_k^U - x_k), \text{if } \gamma \leqslant 0.5 \\ x_k' &= x_k + d(t, x_k - x_k^L), \text{if } \gamma > 0.5 \end{aligned} \tag{4}$$

Where $\gamma \in [0,1]$ is a randomly chosen number; x_k^U and x_k^L are upper bound and lower bound of x_k, respectively. The function $d(t,y)$ gives a value in $[0,y]$, which make $d(t,y)$ converge to 0 as t goes larger. For example,

$$d(t,y) = yr(1-\frac{t}{T})^b \tag{5}$$

Where r is a randomly number between 0 and 1, T is the maximal generation number, and b is the parameter for nonuniform degree.

The selection operator selects the population size of best chromosomes to form the next generation. In our numerical test examples, we use function value as fitness.

Algorithm: Real-number encoding genetic algorithm

Step0 Input parameters: population size: *popu_size*; dimension number: *dim*; maximum generation: *max_gene*; crossover rate: *cross_rate*; mutation rate: *mutate_rate*.

Step1 Initialization: use formula (1) to generate the initial population.

Step2 Crossover operator: chose chromosomes according to the crossover rate, then use formula (2) to generate their offspring.

Step3 Mutation operator: chose chromosomes according to the mutation rate, and then randomly chose the component which is about to mutate, then use (3) to generate the offspring.

Step4 Evaluate the current parents and offspring and record the one with the smallest fitness.

Step5 Check the stop criteria: if it is satisfied, then stop and output the chromosome with the smallest fitness as an optimal solution of the problem, otherwise go to step6.

Step6 Selection operator: select the best population size of chromosomes to building the new generation, and then go to step2.

3 Test examples

In this section we test the real-encoding genetic algorithm provided in the previous section by some well known nonsmooth optimization problems. Since genetic algorithm is stochastic algorithm, the results we illustrated in the following tables are average results of 10 times execution of the genetic algorithm. In all the test problems, we set crossover rate and mutate rate 0.4 and 0.1, respectively. Test problems are calculated on an ACER ASPIRE4730Z laptop with 2.16 GHz CPU and 2 GB RAM. And the algorithm code was written in Matlab 2010 environment. First of all, we explain some signs listed in the tables.

- *Dim*—— input parameter, variable dimension of the problem;
- *Popu_size*—— input parameter, size of the population;
- *Max_gene*—— input parameter, the allowable maximum generation, e.g., the maximal iteration time;
- *Objindi*—— output, average time for objective function evaluation;
- *Timespent*—— output, average time spending for one execution of algorithm;
- *Minf*—— output, average minimum value given by genetic algorithm;
- $Minf^*$ —— the theoretic minimum value of the nonsmooth optimization problem.

Example 1 [9] $f(x) = \max\limits_{1\leqslant i\leqslant n} x_i^2, -10\leqslant x_i\leqslant 10, i=1,2,\cdots,n$

Tab. 1 Results for example 1

Input parameters			Output results (average)			$Minf^*$
Dim	*Popu_size*	*Max_gene*	*Objindi*	*Timespent*	*Minf*	
5	10	100	8.832 0E+02	3.620 0E−02	8.500 0E−03	0
10	20	200	4.060 8E+03	1.424 0E−01	1.744 8E−04	0
20	40	500	2.012 1E+04	7.379 0E−01	9.601 7E−06	0
40	80	800	6.424 1E+04	2.914 3E+00	1.222 4E−05	0
80	160	2 000	3.204 8E+05	1.755 3E+01	4.071 9E−05	0
100	200	2 000	4.006 0E+05	2.398 8E+01	7.919 7E−06	0
150	300	3 000	9.009 0E+05	7.635 8E+01	5.630 3E−06	0
200	400	4 000	1.601 2E+06	1.860 2E+02	4.309 6E−06	0
250	500	5 000	2.501 5E+06	2.490 8E+02	3.154 2E−06	0
300	600	6 000	2.501 5E+06	2.902 1E+02	7.496 1E−06	0
350	700	5 000	2.401 8E+06	3.728 4E+02	2.557 7E−05	0
400	800	5 000	3.502 1E+06	5.730 3E+02	1.607 3E−05	0

Tab. 1 shows that the genetic algorithm performs very well on example 1. Even when dimension of the problem is large (like 400), we can still get a good results, but this need to pay a price of larger population size and larger maxi-

mal generation iteration, which extends the computation time.

Example 2[9] $f(x) = \sum_{i=1}^{n-1} \max\{x_i^4 + x_{x+1}^2, (2 - x_i)^2 + (2 - x_{i+1})^2, 2e^{-x_i+x_{i+1}}\}$, $-10 \leqslant x_i \leqslant 10, i = 1, 2, \cdots, n$

Tab. 2　Results for example 2

Input parameters			Output results (average)			*Minf**
Dim	*Popu_size*	*Max_gene*	*Objindi*	*Timespent*	*Minf*	
5	10	100	8.849 0E+02	2.964 5E−02	8.116 0	8
10	20	200	4.060 6E+03	1.274 3E−01	18.283 4	18
20	40	500	2.012 0E+04	7.559 8E−01	38.281 1	38
40	80	800	6.424 1E+04	3.326 5E+00	78.864 8	78
80	250	1 000	2.507 5E+05	1.868 2E+01	160.881 3	158
100	200	2 000	4.006 0E+05	3.561 0E+01	200.047 1	198
150	300	3 000	9.009 0E+05	1.125 5E+02	302.448 8	298
200	400	4 000	1.601 2E+06	2.643 4E+02	406.011 3	398
250	500	5 000	2.501 5E+06	4.634 6E+02	503.242 1	498
300	600	6 000	2.401 8E+06	5.542 7E+02	613.232 2	598
350	700	5 000	3.502 1E+06	9.602 7E+02	716.568 6	698
400	800	5 000	4.002 4E+06	1.250 6E+03	828.942 3	798
500	1 000	2 000	2.003 0E+06	7.839 6E+02	1 260.819 2	998
600	1 000	2 000	2.003 0E+06	9.493 9E+02	1 643.538 8	1 198

From Tab. 2, we can see that as the dimension of the problem goes bigger, the difference between the results getting by genetic algorithm and theoretical optimal solution is becoming larger. And the results do effected by the maximal generation iteration time. For example, in Tab. 2, when the dimension is 500 and 600, evidently, 2 000 times iteration is not enough to get good results.

Example 3[9] $f(x) = \max\left\{\sum_{i=1}^{n-1}(x_i^2 + (x_{i+1} - 1)^2 + x_{i+1} - 1), \sum_{i=1}^{n-1}(-x_i^2 - (x_{i+1} - 1)^2 + x_{i+1} + 1)\right\}$

$-10 \leqslant x_i \leqslant 10, i = 1, 2, \cdots, n$

Tab. 3　Results for example 3

Input parameters			Output results (average)			*Minf**
Dim	*Popu_size*	*Max_gene*	*Objindi*	*Timespent*	*Minf*	
5	10	100	8.811 0E+02	2.567 5E−02	1.616 5E−01	0
10	20	200	4.060 7E+03	1.038 2E−01	1.204 5E−02	0
20	40	500	2.012 1E+04	4.981 3E−01	6.612 9E−03	0
40	80	800	6.424 1E+04	1.994 7E+00	8.294 4E−03	0
80	250	1 000	2.507 5E+05	8.504 7E+00	1.249 0E−02	0
100	200	2 000	4.006 0E+05	1.544 8E+01	1.052 8E−02	0
150	300	3 000	9.009 0E+05	4.707 0E+01	1.189 1E−02	0
200	400	4 000	1.601 2E+06	1.152 9E+02	1.249 9E−02	0
250	500	5 000	2.501 5E+06	1.682 2E+02	2.005 6E−02	0
300	600	6 000	2.401 8E+06	2.162 1E+02	3.628 0E−02	0
350	700	5 000	3.502 1E+06	3.765 7E+02	3.474 2E−02	0
400	800	5 000	4.002 4E+06	4.863 3E+02	4.274 0E−02	0
500	1 000	2 000	2.003 0E+06	3.121 9E+02	2.562 7E−01	0
600	1 000	2 000	2.003 0E+06	3.756 2E+02	4.095 7E−01	0

Compare to example 1 whose theoretic minimizer is still 0, the results of example 3 is not ideal enough, but it is still acceptable and robust. Again, from the last two rows of Tab. 3, we can see that maximal generation iteration time still paly an important role for getting good results.

4 Conclusions

This paper applies real-number encoding genetic algorithm to solve nonsmooth optimization problems. We firstly reviewed the basic idea of genetic algorithm and then designed a real-number encoding genetic algorithm, and then applied this algorithm to solve some minimax problem. From the numerical test, we can see that genetic algorithm is reliable in solving nonsmooth optimization problems. One drawback of genetic algorithm is that when dimension of the problem becomes bigger, the calculation time and time of objective function evaluation increase sharply. How to increase the convergence rate of genetic algorithm but not extremely increase amount of calculation should be the future concerning of this paper.

Reference

[1] Holland J H. Adaptation in natural and artificial systems[M]. 2nd edition. Combridge, MA: University of Michigan Press, 1992.

[2] Goldberg D E. A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing [J]. Complex Systems, 1990(4): 225-460.

[3] Goldberg D E, Korb B, Deb K. Messy genetic algorithms: motivation, analysis, and first results[J]. Complex Systems, 1989 (3): 493-530.

[4] Grefenstette J J. Optimization of control parameters for genetic algorithms[J]. IEEE Transactions on Systems, Man, and Cybernetics, 1986, 16(1): 122-128.

[5] Kitano H. Neurogenetic learning: An integrated method of designing and training neural networks using genetic algorithms [J]. Physica D: Nonlinear Phenomena, 1994, 75(1): 225-238.

[6] Coello C A, Lamont G B, Van Veldhuizen D A. Evolutionary algorithms for solving multi-objective problems[M]. 2nd edition. New York: Springer-Verlag Inc, 2007.

[7] Goldberg D E. Genetic algorithms in search, optimization and machine learning [M]. Boston, MA, USA: Addison-Wesley, Longman Publishing Co Inc, 1989.

[8] Michalewicz Z. Genetic algorithm data structure evalution programs[M]. 2nd edition. New York: Springer-Verlag, 1994.

[9] Haarala H, Miettinen K, Makela M M. New limited memory bundle method for large-scale nonsmooth optimization [J]. Optimization Methods and Software, 2004, 19(6): 673-692.

运筹学与控制论

基因算法在求解非光滑优化问题中的应用

龙 强

(澳大利亚巴拉瑞特大学 科学、信息技术和工程学院, 维多利亚 巴拉瑞特 3350, 澳大利亚)

摘要 本文考虑了基因算法在求解非光滑优化问题中的应用。非光滑优化方法致力于求解目标函数为连续不可微函数的数学规划问题。因为目标函数的不可微性,传统的以梯度为基础的确定性算法在求解非光滑问题时会遇到障碍,所以运用不需要梯度信息而只需要目标函数值信息的遗传算法来求解非光滑问题是一个不错的选择。遗传算法是基于自然界生物遗传变异过程而设计的一种优化算法,它首先对问题的可行解进行编码,编码方法有 0-1 编码,格雷编码和实数编码,然后运用交叉算子,变异算子和选择算子产生下一代种群。当种群迭代达到一定的次数后,种群中的最优染色体就会收敛到原问题的最优解。本文设计的基因算法基于实数编码,算子分别采用算术交叉算子,非一致变异算子,最佳选择算子。

关键词 基因算法;最大最小问题;非光滑优化

(责任编辑 黄 颖)