Proceedings of GOW 2012, pp. 133 – 136.

Biased random-key genetic algorithm for bound-constrained global optimization

R. M. A. Silva[1], M. G. C. Resende[2], P. M. Pardalos[3], J. F. Gonçalves[4]

[1]*Centro de Informática (CIn), Universidade Federal de Pernambuco, Recife, PE, Brazil,* rmas@cin.ufpe.br

[2]*AT&T Labs Research, Florham Park, NJ, USA,* mgcr@research.att.com

[3]*Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL, USA,* pardalos@ufl.edu

[5]*LIAAD, Faculdade de Economia do Porto, Universidade do Porto, Porto, Portugal,* fgoncal@fep.up.pt

Abstract Global optimization seeks a minimum or maximum of a multimodal function over a discrete or continuous domain. In this paper, we propose a biased random-key genetic algorithm for finding approximate solutions for continuous global optimization problems subject to box constraints. Experimental results illustrate its effectiveness on the robot kinematics problem, a challenging problem according to [7].

Keywords: random-key genetic algorithms, global optimization, metaheuristics.

1. Introduction

Global minimization optimization seeks a solution $x^* \in S \subseteq R^n$ such that $f(x^*) \leq f(x),\ \forall\ x \in S$, where S is some region of R^n and the objective function f is defined by $f : S \to R$. In this paper, we present the BRKGA heuristic for solving continuous global optimization problems subject to box constraints. Without loss of generality, we take the domain S as the hyper-rectangle $S = \{x = (x_1, \ldots, x_n) \in R^n : \ell \leq x \leq u\}$, where $\ell \in R^n$ and $u \in R^n$ such that $u_i \geq l_i$, for $i = 1, \ldots, n$. Therefore, the minimization problem considered in this paper consists in finding $x^* = \operatorname{argmin}\{f(x) \mid \ \ell \leq x \leq u\}$, where $f : R^n \to R$, and $\ell, x, u \in R^n$.

Genetic algorithms with random keys, or *random-key genetic algorithms* (RKGA), were first introduced by [1] for solving combinatorial optimization problems involving sequencing. In a RKGA, chromosomes are represented as vectors of randomly generated real numbers in the interval $[0, 1]$. A deterministic algorithm, called a *decoder*, takes as input a solution vector and associates with it a solution of the combinatorial optimization problem for which an objective value or fitness can be computed.

A RKGA evolves a population of random-key vectors over a number of iterations, called *generations*. The initial population is made up of p vectors of random-keys. Each component of the solution vector is generated independently at random in the real interval $[0, 1]$. After the fitness of each individual is computed by the decoder in generation k, the population is partitioned into two groups of individuals: a small group of p_e *elite* individuals, i.e. those with the best fitness values, and the remaining set of $p - p_e$ *non-elite* individuals. To evolve the population, a new generation of individuals must be produced. All elite individual of the population of generation k are copied without modification to the population of generation $k + 1$. RKGAs implement mutation by introducing *mutants* into the population. A mutant is simply a vector of random keys generated in the same way that an element of the initial population is generated. At each generation, a small number p_m of mutants is introduced into the population. With the p_e elite individuals and the p_m mutants accounted for in population $k + 1$, $p - p_e - p_m$ additional individuals need to be produced to complete the p individuals

that make up the new population. This is done by producing $p - p_e - p_m$ offspring through the process of mating or crossover.

A *biased random-key genetic algorithm,* or BRKGA [4], differs from a RKGA in the way parents are selected for mating. While in a RKGA, [1] selects two parents at random from the entire population; in a BRKGA, each element is generated combining one element selected at random from the elite individuals set in the current population and one from the non-elite individuals set. In some cases, the second parent is selected from the entire population. Repetition in the selection of a mate is allowed and therefore an individual can produce more than one offspring. Since we require that $p_e < p - p_e$, the probability that an elite individual is selected for mating is greater than that of a non-elite individual and therefore the elite individual has a higher likelihood to pass on its characteristics to future generations. Another factor contributing to this end is *parameterized uniform crossover* [5], the mechanism used to implement mating in BRKGAs. Let $\rho_e > 0.5$ be the probability that an offspring inherits the vector component of its elite parent. Let n denote the number of components in the solution vector of an individual. For $i = 1, \ldots, n$, the i-th component $c(i)$ of the offspring c takes on the value of the i-th component $e(i)$ of the elite parent e with probability ρ_e and the value of the i-th component $\bar{e}(i)$ of the non-elite parent $\bar{e}$ with probability $1 - \rho_e$.

When the next population is complete, i.e. when it has p individuals, fitness values are computed for all of the newly created random-key vectors and the population is partitioned into elite and non-elite individuals to start a new generation.

To describe a BRKGA for continuous global optimization problems subject to box constraints, one needs only to show how solutions are encoded as vectors of random keys and how these vectors are decoded to feasible solutions of the problem:

- *Encoding a solution to a vector of random keys*. A solution is encoded as a vector $\chi = (\chi_1, ..., \chi_n)$ of size n, where χ_i is a random number in the interval $[0, 1]$, for $i = 1, \ldots, n$. The i-th component of χ corresponds to the i-th dimension of hyper-rectangle S.
- *Decoding a solution from a vector of random keys*. A decoder takes as input the vector of random keys χ and returns a solution $x \in S$ with $x_i = l_i + \chi_i \cdot (u_i - l_i)$, for $i = 1, \ldots, n$. During all decoder process, the solutions fitness are calculated by the objective function $f : S \to R$ of global optimization problem.

2. Experimental results

All experiments with BRKGA were done on a quad core Intel Core i7 processor (1.60 GHz) with Turbo Boost up to (2.80 GHz) and 16 Gb of memory, running Ubuntu 10.04 LTS released in April 2010. BRKGA heuristic was implemented in C++ and compiled with gcc version 4.4.3. The algorithm used for random-number generation is an implementation of the Mersenne Twister algorithm introduced by [6].

In this paper, we consider a problem from robot kinematics ([7–9]). We are given a 6-revolute manipulator (rigid-bodies, or links, connected together by joints, with each link connected to no more than two others), with the first link designated the base, and the last link designated the hand of the robot. The problem is to determine the possible positions of the hand, given that the joints are movable. In [9], this problem is reduced to solving a system of eight nonlinear equations $f_1(x), \ldots, f_8(x)$ in eight unknowns $x = \{x_1, \ldots, x_8\} \in [-1, 1]^8$:

$$f_1(x) = 4.731 \cdot 10^{-3} x_1 x_3 - 0.3578 x_2 x_3 - 0.1238 x_1 + x_7 - 1.637 \cdot 10^{-3} x_2 - 0.9338 x_4 - 0.3571 = 0$$

$$f_2(x) = 0.2238 x_1 x_3 + 0.7623 x_2 x_3 + 0.2638 x_1 - x_7 - 0.07745 x_2 - 0.6734 x_4 - 0.6022 = 0$$

$$f_3(x) = x_6 x_8 + 0.3578 x_1 + 4.731 \cdot 10^{-3} x_2 = 0$$

$$f_4(x) = -0.7623 x_1 + 0.2238 x_2 + 0.3461 = 0$$

$$f_5(x) = x_1^2 + x_2^2 - 1 = 0$$

$$f_6(x) = x_3^2 + x_4^2 - 1 = 0$$

$$f_7(x) = x_5^2 + x_6^2 - 1 = 0$$

Table 1. Roots of system in $[-1,1]^8$ found by running BRKGA with seed=270001. For each root, the time in seconds and the value of objetive function $F(\cdot)$ are shown in the first collumn, as so as the components (in parenthesis) of known roots described in [7, 8].

time(sec.) $F(x)(10^{-5})$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
4.95	0.1658	−0.9851	0.7153	−0.6950	−0.9975	0.0638	−0.5251	−0.8557
9.79321	(0.1644)	(−0.9864)	(0.7185)	(−0.6956)	(−0.9980)	(0.0638)	(−0.5278)	(−0.8494)
7.5	0.1619	−0.9851	0.7182	−0.6946	−0.9979	−0.0616	−0.5232	0.8503
7.19678	(0.1644)	(−0.9864)	(0.7185)	(−0.6956)	(−0.9980)	(−0.0638)	(−0.5278)	(0.8494)
13.19	0.1731	−0.9827	0.7181	−0.6946	0.9973	−0.0686	−0.5195	0.8544
9.54526	(0.1644)	(−0.9864)	(0.7185)	(−0.6956)	(0.9980)	(−0.0638)	(−0.5278)	(0.8494)
5.95	0.6729	0.7394	−0.6480	−0.7641	−0.9623	−0.2706	−0.4333	0.9027
9.76283	(0.6716)	(0.7410)	(−0.6516)	(−0.7586)	(−0.9625)	(−0.2711)	(−0.4376)	(0.8992)
6.86	0.6736	0.7383	−0.6505	−0.7553	0.9634	0.2696	−0.4333	−0.9010
6.49664	(0.6716)	(0.7410)	(−0.6516)	(−0.7586)	(0.9625)	(0.2711)	(−0.4376)	(−0.8992)
6.53	0.6792	0.7328	−0.6555	−0.7553	0.9612	−0.27613	−0.4343	0.9027
9.23596	(0.6716)	(0.7410)	(−0.6516)	(−0.7586)	(0.9625)	(−0.2711)	(−0.4376)	(0.8992)
11.05	0.6768	0.7358	0.9502	−0.3132	−0.9623	0.2708	0.4002	−0.9162
9.68334	(0.6716)	(0.7410)	(0.9519)	(−0.3064)	(−0.9638)	(0.2666)	(0.4046)	(−0.9145)
15.24	0.6674	0.7427	0.9508	−0.3132	0.9661	−0.2620	0.4002	0.9156
9.81702	(0.6716)	(0.7410)	(0.9519)	(−0.3064)	(0.9638)	(−0.2666)	(0.4046)	(0.9145)
9.16	0.6792	0.7362	−0.6564	−0.7553	−0.9617	0.2745	−0.4343	−0.9008
9.1171	(0.6716)	(0.7410)	(−0.6516)	(−0.7586)	(−0.9625)	(0.2711)	(−0.4376)	(−0.8992)
98.98	0.6707	0.7462	0.9530	−0.3041	0.9644	0.2631	0.4079	−0.9107
8.55693	(0.6716)	(0.7410)	(0.9519)	(−0.3064)	(0.9638)	(0.2666)	(0.4046)	(−0.9145)
135.02	0.6646	0.7490	0.9551	−0.3015	−0.9652	−0.2625	0.4101	0.9114
9.82556	(0.6716)	(0.7410)	(0.9519)	(−0.3064)	(−0.9638)	(−0.2666)	(0.4046)	(0.9145)
354.76	0.1604	−0.9891	−0.9505	−0.3167	−0.9979	−0.0581	0.4111	0.9090
9.32723	(0.1644)	(−0.9864)	(−0.9471)	(−0.3210)	(−0.9982)	(−0.0594)	(0.4110)	(0.9116)
360.76	0.1680	−0.9844	−0.9514	−0.3167	0.9998	−0.0602	0.4098	0.9124
9.70348	(0.1644)	(−0.9864)	(−0.9471)	(−0.3210)	(0.9982)	(−0.0594)	(0.4110)	(0.9116)
409.27	0.1606	−0.9855	−0.9481	−0.3183	−0.9976	0.0554	0.4138	−0.9076
7.28536	(0.1644)	(−0.9864)	(−0.9471)	(−0.3210)	(−0.9982)	(0.0594)	(0.4110)	(−0.9116)
1204.24	0.1712	−0.9850	−0.9427	−0.3275	0.9976	0.0621	0.4052	−0.9143
8.21721	(0.1644)	(−0.9864)	(−0.9471)	(−0.3210)	(0.9982)	(0.0594)	(0.4110)	(−0.9116)
1369.81	0.1718	−0.9837	0.7178	−0.6947	0.9943	0.0687	−0.5246	−0.8519
8.63659	(0.1644)	(−0.9864)	(0.7185)	(−0.6956)	(0.9980)	(0.0638)	(−0.5278)	(−0.8494)

$$f_8(x) = x_7^2 + x_8^2 - 1 = 0$$

With this system, we form the optimization problem

$$\text{Find } x^* = \text{argmin}\{F(x) = \textstyle\sum_{i=1}^{8} f_i^2(x) \mid x \in [-1,1]^8\}. \tag{1}$$

Since $F(x) \geq 0$ for all $x \in [-1,1]^8$, it is easy to see that $F(x) = 0 \iff f_i(x) = 0$ for all $i \in \{1,\ldots,8\}$. Hence, we have the following: $\exists\ x^* \in [-1,1]^8 \ni F(x^*) = 0 \implies x^*$ is a global minimizer of problem (1) and x^* is a root of the system of equations $f_1(x),\ldots,f_8(x)$. From [7, 8], in the given domain, there are 16 known roots to this system. However, solving problem (1) 16 times using BRKGA (or any heuristic) with different starting solutions gives no guarantee of finding all 16 roots. It is entirely possible that some of the roots would be found multiple times, while others would not be found at all.

To avoid this, we modified the objective function $F(x)$, such as proposed by [3]. Suppose that heuristic has just found the k-th root (roots are denoted $x^1,\ldots,x^k$). Then BRKGA will restart, with the modified objective function given by

$$F(x) = \sum_{i=1}^{8} f_i^2(x) + \beta \sum_{j=1}^{k} e^{-\|x-x^j\|}\chi_\rho(\|x-x^j\|), \tag{2}$$

where

$$\chi_\rho(\delta) = 1, if \delta \leq \rho; 0, otherwise,$$

β is a large constant, and ρ is a small constant. This has the effect of creating an area of repulsion near solutions that have already been found by the heuristic.

For this problem, we ran BRKGA five times (a different starting random number seed for each run from 270001 to 270005) with $n = 8$, $p = 10$, $p_e = 0.2p$, $p_m = 0.1p$, $\rho_e = 0.7$, $\rho = 1$, and $\beta = 10^{10}$. At any time during a run, we define the optimality gap by $GAP = |F(x) - F(x^*)|$, where x is the current best solution found by the heuristic and x^* is the known global minimum solution. We then say that the heuristic has solved the problem if $GAP \leq \epsilon$ with $\epsilon = 0.0001$. In each case, the heuristic was able to find all 16 known roots. The average CPU time needed to find the 16 roots was 3623.27 seconds. The Table 1 illustrates one of these solutions: the 16 roots found in 4013.27 seconds by running BRKGA heuristic with seed=270001.

3. Concluding remarks

In this paper, we present the BRKGA heuristic for finding approximate solutions for continuous global optimization problems subject to box constraints. We illustrate the approach using a challenging problem with real-world applications, the robot kinematics, which nonlinear system was solved through a corresponding adaptively modified global optimization problem multiple times, each time using BRKGA with areas of repulsion around roots that have already been found. The promising results shown here illustrate the potential of BRKGA for global optimization problems.

References

[1] J.C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA J. on Computing*, 6: 154–160, 1994.

[2] C. A. Floudas and P. M. Pardalos. A collection of test problems for constrained global optimization algorithms. In G. Goods and J. Hartmanis, editors, *Lecture Notes in Computer Science*, volume 455. Springer Verlag, Berlin, 1990.

[3] M.J. Hirsch, C.N. Meneses, P.M. Pardalos, and M.G.C. Resende. Global optimization by continuous GRASP. In *Optimization Letters*, vol. 1, pages 201–212, 2007.

[4] J.F. Gonçalves and M.G.C. Resende. An evolutionary algorithm for manufacturing cell formation. In *Computers and Industrial Engineering*, 47:247–273, 2004.

[5] W.M. Spears and K.A. DeJong. On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, 1991.

[6] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. In *ACM Transactions on Modeling and Computer Simulation*, pages 3–30, 1998.

[7] C. A. Floudas, P. M. Pardalos, C. Adjiman, W. Esposito, Z. Gumus, S. Harding, J. Klepeis, C. Meyer and C. Schweiger. Handbook of Test Problems in Local and Global Optimization. In *Kluwer Academic Publishers, Dordrecht*, 1999.

[8] R. B. Kearfott. Some Tests of Generalized Bisection. In *ACM Transactions on Mathematical Software*, 13(3): 197–220, 1987.

[9] L. W. Tsai and A. P. Morgan. Solving the Kinematics of the Most General Six- and Five-Degree-of-Freedom Manipulators by Continuation Methods. In *Journal of Mechanisms, Transmissions, and Automation in Design*, 107:189–200, 1985.